Image Search Engine: Looking for a Needle in a Haystack

A dissertation submitted in partial fulfilment of
the requirements for the degree of
BACHELOR OF *ENGINEERING* in Computer Science
in
The Queen's University of Belfast
by
Adam Cregan
17/04/2023

SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING and COMPUTER SCIENCE

CSC3002 – COMPUTER SCIENCE PROJECT

Dissertation Cover Sheet

A signed and completed cover sheet must accompany the submission of the Software Engineering dissertation submitted for assessment.

Work submitted without a cover sheet will **NOT** be marked.

Student Name: Adam Cregan Student Number: 40261335
Project Title: Image Search Engine: Looking for a Needle in a Haystack

Supervisor: Dr Paul Miller

Declaration of Academic Integrity

Before submitting your dissertation please check that the submission:

1. Has a full bibliography attached laid out according to the guidelines specified in the Student Project Handbook
2. Contains full acknowledgement of all secondary sources used (paper-based and electronic)
3. Does not exceed the specified page limit
4. Is clearly presented and proof-read
5. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance.

   By submitting your dissertation you declare that you have completed the tutorial on plagiarism at http://www.qub.ac.uk/cite2write/introduction5.html and are aware that it is an academic offence to plagiarise. You declare that the submission is your own original work. No part of it has been submitted for any other assignment and you have acknowledged all written and electronic sources used.
6. If selected as an exemplar, I agree to allow my dissertation to be used as a sample for future students. (Please delete this if you do not agree.)

Student's signature *Adam Cregan* *17/04/2023*

Acknowledgements

I would like to acknowledge my supervisor Dr Paul Miller for his advice and guidance surrounding the development of the distance measurements and performance analysis within this project.

Abstract

This project involves creating an image search engine which allows a user to enter a query image, and then the system will search its image database for images that are most similar to the query image in terms of content. The search engine will extract features from the images (colour, shape, and texture), to represent the image. The query image feature representation is then compared to those in the database to find those that provide the best match based on distance measurements. The image database is divided into four categories: Crowds, F1-Cars, Horses, and Landscapes. The final model will have the best performance, accounting for precision and speed of retrieval. The image features should be stored externally to increase speed of retrieval and scalability. This report will cover the theory and analysis of a newly created image search engine.

Git Repository

Follow the URL below to access the git repository which holds the final system developed along with all additional documentation.

https://github.com/acregan04/ISE

Contents

1.0 Introduction and Problem Area

Content-based image retrieval (CBIR) is a technique for retrieving images from a database based on their content, such as colour, texture, shape, and other visual characteristics. In the past, CBIR has been used in various scientific and technological applications. For example, in the field of medicine, CBIR has been used for the classification of different types of tumours in MRI images [1]. In video surveillance, CBIR has been utilized for the recognition of human activities and facial recognition [2]. In the field of remote sensing, CBIR has been used for the classification of land cover [3]. In addition, CBIR has been used in digital libraries for content-based image retrieval and browsing [4]. CBIR technology is currently in a promising state, and researchers are constantly working to boost its efficiency and precision. Advances in computer hardware, machine learning, and deep learning techniques have all played a significant role in improving the performance of CBIR systems. Researchers are also exploring novel methods, such as hybrid approaches combining multiple visual features and machine learning algorithms, to improve the accuracy and efficiency of CBIR systems [5].

CBIR systems often suffer from poor precision and slow response times, which are major issues that limit their effectiveness. In CBIR systems, poor precision refers to the inability to accurately retrieve images that match the user's intended query. This issue arises due to a number of factors, including low-level feature extraction methods, query ambiguity, and a semantic gap between low-level features and high-level semantic concepts [6]. Slow response times in CBIR systems refers to the time it takes to retrieve relevant images from a large image database, which can be caused by factors such as inefficient indexing and retrieval algorithms [7]. Storing image data externally can positively impact the speed and scalability of CBIR systems. The reduced latency and faster access times are two of the most significant benefits of storing image data externally. External storage solutions are intended to provide fast data access and can use advanced caching techniques to reduce data access time [8]. As a result, image retrieval times are reduced, and system performance is improved. Scalability is another critical advantage of storing image data externally. CBIR systems must handle a large amount of data, and internal storage systems may not be able to handle the scale of these databases. These systems can also provide fault-tolerant and highly available data storage, which is critical for CBIR systems that require continuous access to image data.

2.0 System Requirements and Specification

2.1 Solution Description

This application is developed in Eclipse IDE for Java Developers - 2022-12(4.26.0), using Java. The project developed is a content-based image retrieval (CBIR) system that allows users to select a query image and retrieve a set of similar images from a gallery using effective distance measurement techniques for colour, texture, and shape.

The application will be designed with a Graphical User Interface (GUI) that will pop up when executed. The user will interact with the GUI to select a query image, which will be used to perform a search.

A functional requirement is that the image database will be split into two categories: gallery and test, with an 80/20 split ratio [9]. All the of the tests taken will use the test images, and the image database will use the images in the gallery. For this application there are 431 total images that will be used. Therefore, 80% = 334.8 and 20% = 86.2. This is rounded to 345 gallery images and 86 test images. The 4 image categories are Crowds, F1-Cars, Horses, and Landscapes, so the images will be split accordingly. 86 test images/4 categories = 21.5 test images. Therefore, the test folder will include 22 Crowds images, 21 F1-Cars images, 22 Horses images, and 21 Landscapes images.

User Characteristics

This application has been developed for a user with minimal technical skills, as this only requires control of the cursor to navigate the GUI. Moreover, the system is robust enough so that if the user does make mistakes, the system should catch it and allow the user to try again. This is covered in section 2.2 UI Requirements.

2.2 UI Requirements

The UI should be designed in a user-friendly way so that users can easily navigate and interact with the application. The UI should allow the user to select the query image, display the results of the search, update the database, and provide information about any errors or issues that may arise during the search process.

The UI should also include error handling mechanisms to inform the user if they click the "Search" button when no query image has been selected, that they must select a query image. Moreover, a pop-up message should appear if the application cannot access the image database when the user is trying to save/read data and inform the user of the situation. Furthermore, if there are not enough images in the database to populate the results section when the user tries

to search, a pop-up message should inform the user that there are not enough images in the database. Also, if the user selects an invalid file type then a pop-up message should appear telling the user to select a valid file type, and the search function is cancelled.

2.3 Functional Requirements

Colour space and pixel representation:

The images in the database will be stored in an appropriate colour space. HSV has been chosen due to its superior performance in similar applications against the RGB colour space [10, 11]. This CBIR system will also use colour histograms to represent the pixels as it is better than other descriptors like colour moments and colour coherent vectors [12]. Test: The system uses HSV values to generate colour histograms so that distance measures can later be applied.

Query image selection and resizing:

The system shall allow the user to select a query image from either inside or outside the image database. If the query image is not of size 512x512 pixels, the system shall resize it to the required size. Test: The system shall correctly resize the query image to 512x512 pixels and display it in the GUI.

Image search and comparison:

The system shall allow the user to initiate a search for similar images in the gallery by clicking the search button in the GUI. The system will have multiple, manually implemented distance measurement techniques, including Correlation, Chi-Square, Intersection, and Bhattacharyya measures [16], to compare the colour histograms of the query image with those in the gallery. The system will iterate through all images in the gallery and measure the distance to the query image. Then the system shall return the top 15 images from the gallery that are closest to the query image, based on the distance measures. Test: The system will correctly display the top 15 closest images in the GUI after a search is performed.

Graphical user interface (GUI):

The system shall have a GUI that displays buttons for image selection, search, update database, and clear data. The GUI will allow the user to select a query image using a file chooser dialog. The GUI will display the query image and closest images after a search is performed. Test: The GUI correctly displays all buttons and images and allows the user to select a query image and initiate a search.

Scalability:

The system shall improve scalability by calculating features offline. Test: The system shall correctly calculate features and store them externally so they can be accessed at any time.

Sorting:

The system shall implement a sorting method that uses a HashMap to associate and rearrange elements from the list of unsorted images into the order of the sorted distance scores. Test: The system shall correctly sort the images in the gallery based on distance measures using the implemented sorting method.

2.4 Non-Functional Requirements

Timers for Performance Analysis:

The CBIR system should be designed to analyse the performance using timers. The system should display all image results in less to the end-user in less than 5 seconds. Timers will be used to calculate the time taken by the system to display the results. Test: The timers are correctly implemented in the system and that the system provides results to the developer on the console.

Optimize Histogram Bins:

The system will have 4 different distance measuring techniques (Correlation, Chi-Square, Intersection, and Bhattacharyya) where the number of bins in the histograms need to be optimised for precision and time for each measurement. Test: A bins number has been set for each measurement that is optimal based on tests using precision and time and selecting the bins number which performed best.

Pop-up Message for Image Database Update:

The application should generate a pop-up message to inform the user when the image database has been updated. Test: The pop-up message is displayed when the image database is updated, after the user chose to do so.

Determine Best Model from Distance Measures:

The performance of the system should be analysed using generated Precision-Recall curves. Test: Precision-Recall curves are used in selecting the model that performs the best.

Determine Best Data Storage Technique:

Various storage techniques, including using a database manager (SQL Server Management Studio Manager 19) and other simple file formats (.ser, .txt, .bin, .csv), will be tested when finding the best storage technique. The best method will be based on the time taken to update the database. Test: Storage methods mentioned above have been implemented and time tested, thus the best performing method will be deduced. The best technique should update the database in less than 20 seconds.

3.0 Design

3.1 Architecture Description

Fig. 3.1 below demonstrates the overall architecture of the image search engine and illustrates how the external image feature database interacts with the main processes of the system during a normal program execution of the search function.

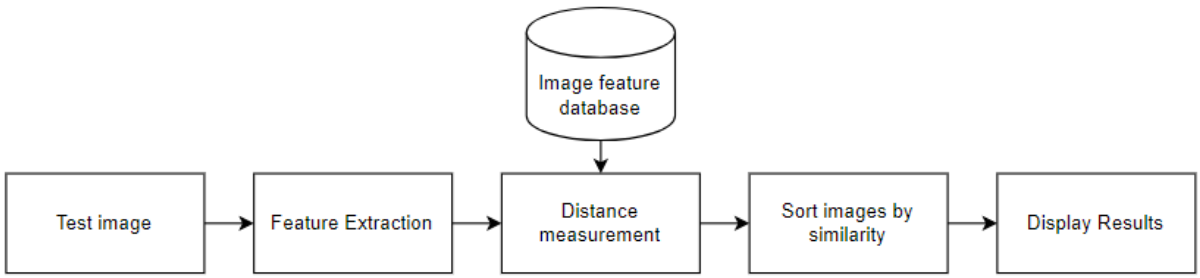

Fig. 3.1 Block diagram of proposed search engine

Additionally, Fig 3.2 below, illustrates in more detail how the four main processes user-triggered functions operate, as it demonstrates the order of the functions and how the data flow always returns to the “Start” point waiting for the next user input (button click)

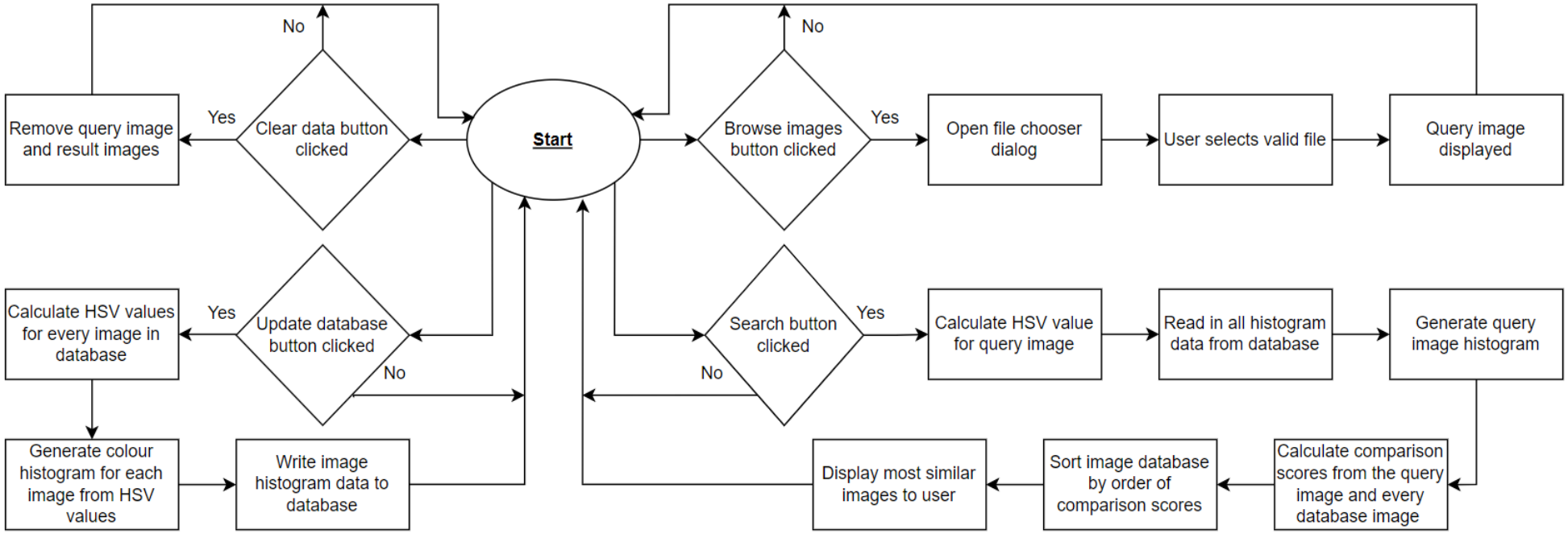

Fig. 3.2 Flowchart showing the overall functionality of the image search engine

3.2 User Interface (UI) Design

Based on the previously mentioned requirements, Fig. 3.3 below shows the proposed UI which would allow the user to update the database, select a query image and search for similar images. The UI will display the 15 most similar images and provide an option below for the user to clear the data. The UI takes a minimalist design as to make the functionality most clear for the end-user. This is supported by the UI being split mainly into two sections, the search section (on the left), and the results section (on the right).

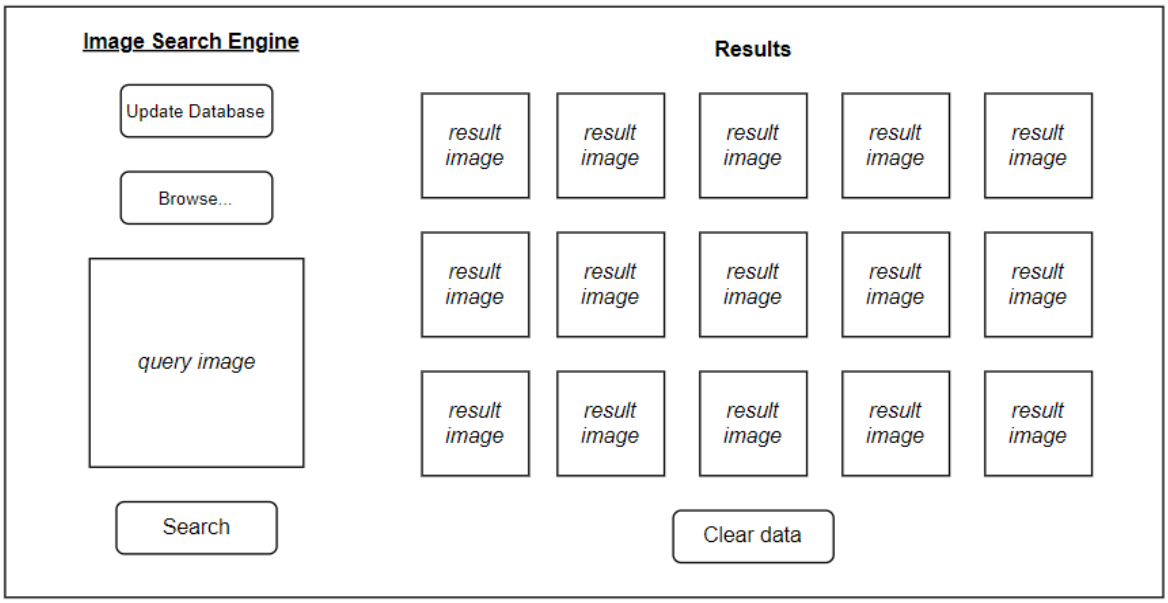

Fig. 3.3 Proposed UI

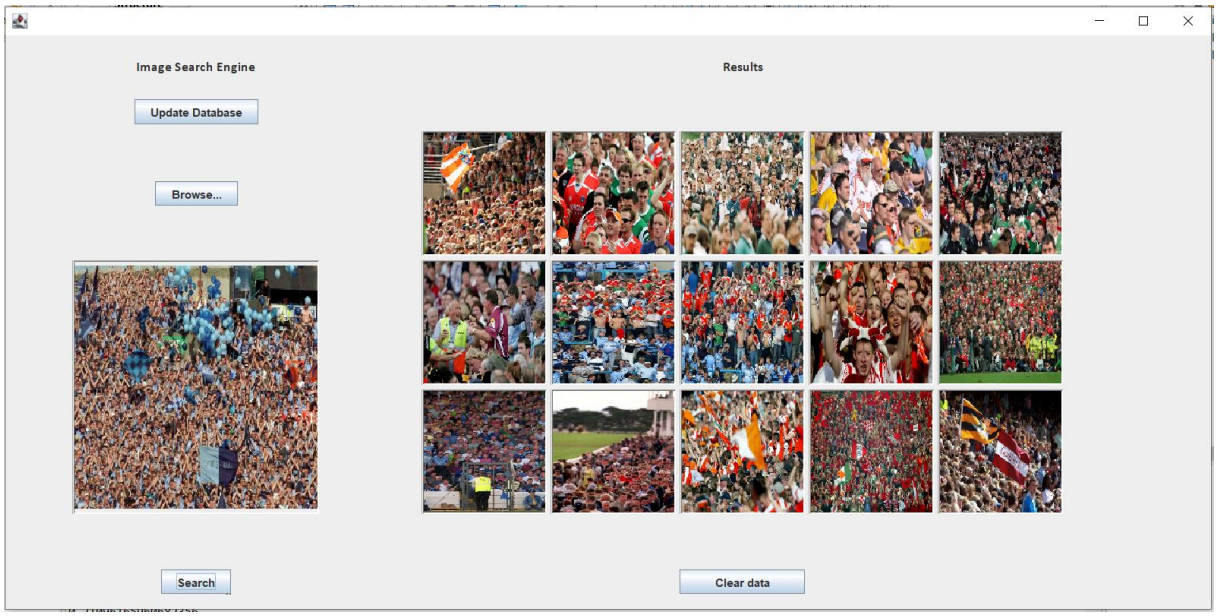

Fig. 3.4 Real Example of UI

Fig. 3.4 shows how the purposed UI in Fig. 3.3 has been visualised in the final application. This this example a query image has already been selected and the search function executed so the 15 most similar images are being displayed. The query image is rescaled to 512x512.

3.3 HSV Calculation

As mentioned in Section 2.3, HSV is the chosen colour space (illustrated in Fig. 3.5). To use HSV the application must convert RGB colour values to the HSV colour model. The HSV model is based on the concepts of hue, saturation, and value. The function takes in three integer arguments representing the red, green, and blue colour components respectively. It then calculates the corresponding HSV values and returns them as a float array.

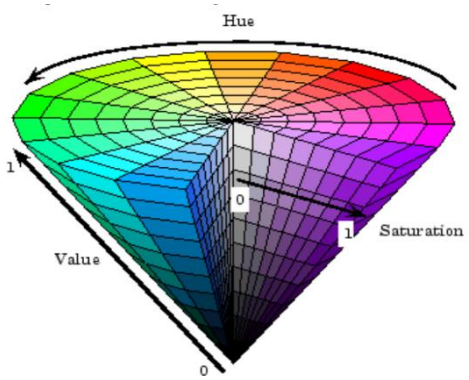

Fig. 3.5 Different Planes of HSV colour space [13]

The algorithm implemented calculates the HSV values using the method mentioned below [10, 14]:

$$R = R/255$$
$$G = G/255$$
$$B = B/255$$

$$Hue = \begin{cases} 60 * \frac{G-B}{MAX-MIN} & (if MAX = R) \\ 60 * \frac{B-R}{MAX-MIN} + 2 & (if MAX = G) \\ 60 * \frac{R-G}{MAX-MIN} + 4 & (if MAX = B) \end{cases}$$

$$Saturation = MAX - MIN$$

$$Value = MAX$$

The function is considered to be a good technique for colour conversion because it provides an efficient and accurate method for converting between the RGB and HSV colour models. Additionally, the algorithm used by the function is widely recognized and has been cited in numerous research papers related to colour image processing and computer graphics [10].

3.4 Histogram Calculation

To generate a colour histogram from a 3D HSV image array, the application uses the method of binning the values in each channel (Hue, Saturation, and Value) into a fixed number of bins, and then counting the number of pixels that are in each bin.

Here is the general algorithm [15, 16]:

(i) Define the number of bins for each channel (e.g.,5, 10, 15). This determines the granularity of the histogram.
(ii) Initialize an empty histogram array with dimensions equal to the number of bins for each channel.
(iii) For each pixel in the image, extract its HSV values and determine which bin it belongs to in each channel based on the range of values covered by each bin.
(iv) Increment the value in the corresponding bin in the histogram array.
(v) Normalize the histogram by dividing each bin value by the total number of pixels in the image.

3.5 Distance Measurement Selection

- Correlation, Chi-Square, Intersection, Bhattacharyya

These methods are distance metrics that can be used to compare image features. Among these distance metrics, correlation and intersection are mainly used for shape features, while chi-square and Bhattacharyya distance can be used for both colour and texture features.

The chi-square distance measures the difference between two histograms, which can be used to compare colour distributions. It is commonly used in object recognition and retrieval applications.

The Bhattacharyya distance is a statistical measure that can be used to compare probability distributions, and it is commonly used to compare image colour and texture features.

Overall, using these distance metrics in this Java-based image search engine application will help perform efficient and accurate image analysis and comparison, ultimately resulting in better image retrieval and recognition capabilities [16, 17, 18].

For each distance measurement, to get an optimal model we want to determine what number of bins to use in our histograms when the images are compared to each other. We also want to record the time these processes take, because it is most desirable for this process to be as efficient as possible and not to sacrifice significant time for a marginal increase in model precision.

Correlation

$$d(H_1, H_2) = \frac{\sum_I (H_1(I) - \bar{H}_1)(H_2(I) - \bar{H}_2)}{\sqrt{\sum_I (H_1(I) - \bar{H}_1)^2 \sum_I (H_2(I) - \bar{H}_2)^2}}$$

where

$$\bar{H}_k = \frac{1}{N} \sum_J H_k(J)$$

H1 and *H2* are the histograms being compared.
N is the total number of histogram bins.

For this method the correlation coefficient is a measure of how similar the two images are. The resulting correlation coefficient ranges from -1 to 1, where values closer to 1 indicate a better match between the template and the search image [17,18]. Fig. 7.10, “Correlation implementation”, illustrates how this has been applied to the image search engine.

Chi-Square

$$d(H_1, H_2) = \sum_I \frac{(H_1(I) - H_2(I))^2}{H_1(I)}$$

where
H1 and *H2* are the histograms being compared.

This distance measurement is based on the idea of finding the minimum of the frequency values of each bin (or category) between the two histograms. The equation sums the minimum value of each bin across all the bins in the histogram, and the resulting value represents the

dissimilarity between the two histograms. The smaller the value of d(H1, H2), the more similar the histograms are, and the larger the value of d(H1, H2), the more dissimilar they are [17,18]. Fig. 7.11, “Chi-Square implementation”, illustrates how this has been applied to the image search engine.

Intersection

$$d(H_1, H_2) = \sum_I \min(H_1(I), H_2(I))$$

where
H1 and *H2* are the histograms being compared.

The equation works by comparing the values in each bin or category of the two histograms and selecting the minimum value between them. For each bin "I", the minimum value between H1(I) and H2(I) is selected and summed across all the bins in the histograms. The resulting value represents the distance between the two histograms. Intuitively, this distance measure captures the idea of finding the "intersection" between the two histograms, by comparing the minimum frequency of each bin. If the two histograms are identical, the distance will be zero, as each bin in both histograms will have the same frequency. On the other hand, if the two histograms are very dissimilar, the distance will be large, as there will be very little overlap/commonality between the bins of the two histograms [17,18]. Fig. 7.12, “Intersection implementation”, illustrates how this has been applied to the image search engine.

Bhattacharyya

$$d(H_1, H_2) = \sqrt{1 - \frac{1}{\sqrt{\bar{H}_1 \bar{H}_2 N^2}} \sum_I \sqrt{H_1(I) \cdot H_2(I)}}$$

where

H1 and *H2* are the histograms being compared.
N is the total number of histogram bins.

The Bhattacharyya distance is a statistical measure of the similarity between two probability distributions. It is calculated based on the overlap between the two distributions, and it considers

both the mean and variance of the distributions. In the above equation the numerator of the equation calculates the covariance between the two histograms, and the denominator of the equation is a normalization factor that scales the covariance value by the product of the means and the number of bins in the histograms. The resulting Bhattacharyya distance is a value between 0 and 1, where 0 indicates a perfect match between the two histograms, and 1 indicates no overlap or similarity between the two histograms. Fig. 7.13, "Bhattacharyya implementation", illustrates how this has been applied to the image search engine.

3.6 Error Handling

In this section covers the error handling implemented into this application so that it will not crash based on what the user has entered, and it will inform the end user what has happened if they have made an error. The pop-up message for each case can be found in Fig. 7.1 – Fig. 7.9.

1. Save Data Failed

Function: Click event listener on "Update Database" button

A try/catch method has been implemented so that if the save data function fails then the application will display a pop-up message informing the user that the database has not been updated. (Fig. 7.1)

2. Image Search Failed

Function: Click event listener on "Search" button

A try/catch method implemented attempts to save the returned data from the "images.details()" function in the "proImgs" ArrayList. If it fails then the application will display a pop-up message to the user informing them that the application has failed to search for the image. (Fig. 7.2)

3. No Image Selected

Function: Click event listener on "Search" button

This try/catch method ensures that the user cannot execute the search query when there is no image selected to query on. If the user does this then the application will display a pop-up message to the user informing them that no search image has been selected. (Fig. 7.3)

4. Failed to Read Images (save)

Function: HSVSaver()

This try/catch method has been implemented so that when the application attempts to access HSV image details from the file and the data is not as expected then the application will display a pop-up message to the user informing them that it has failed to read in the images from the gallery. (Fig. 7.4)

5. Failed to Read Images (read)

Function: readHSV()

This try/catch method implemented attempts to read in the HSV data from the file and save it in the 3D float array. If it fails then the application will display a pop-up message to the user informing them that the application has failed to read in the data from the file. (Fig. 7.5)

6. Failed to Save Images

Function: HSVSaver()

This try/catch method has been implemented so that when the application attempts to write the newly calculated data to the file, and if it fails then the application will display a pop-up message to the user informing them that it has failed to save the data to the file. The code listing and pop-up message are displayed below. (Fig. 7.6)

7. Failed to Save Images

Function: HSVSaver()

This try/catch method has been implemented so that when the application attempts to write the newly calculated data to the file, and if it fails, then the application will display a pop-up message to the user informing them that it has failed to save the data to the file. (Fig. 7.7)

8. Not enough images to populate the UI

Function: Click event listener on “Search” button

This try/catch method has been implemented so that when the application attempts to populate the UI with the most similar images. If there are not enough images in the gallery then the images will not be displayed, and a pop-up message will be displayed informing the user that there are not enough images in the gallery. (Fig. 7.8)

Update Database Message

Function: HSVSaver()

This pop-up is generated after the image data has been saved externally. The pop-up message confirms to the user that the data has been successfully saved to the file after they have clicked the “update Database” button. (Fig. 7.9)

4.0 Implementation

This application was developed using Java for a variety of reasons. Firstly, because of its platform independence. Java is a platform-independent programming language, which means that Java code can run on any platform that has a Java Virtual Machine (JVM) installed. As a result, the CBIR system can be run on a variety of platforms, including Windows, Mac, and Linux. Moreover, Java provides several GUI libraries, including Swing, which can be used to develop a Graphical User Interface (GUI) for the CBIR system. This allows users to interact with the system in a more intuitive and user-friendly way.

Eclipse IDE for Java Developers - 2022-12 (4.26.0) was used to develop the application because it is specifically designed for Java development, which is the programming language that was used It offers various tools for Java development, such as code highlighting, debugging, and refactoring, , which helps write high-quality code more efficiently. Furthermore, the Eclipse IDE has a user-friendly interface that allows for effective project management and testing/debugging.

Java Swing is used for the UI because it provides a variety of components such as buttons, labels, text fields, that are used in this application. Furthermore, Swing follows an event-driven programming model, which means that GUI components e.g. buttons; respond to user actions such as mouse clicks, button presses, and key presses. This enables this CBIR application to be highly interactive and responsive to user input. MigLayout is used as a versatile and flexible layout manager for Swing that provides a simple yet powerful way to create user interfaces (UIs) that can adapt to different screen sizes and resolutions. Moreover, MigLayout makes it easy to create a responsive and visually appealing UI that can adapt to different screen sizes and resolutions. This is particularly important for image retrieval applications, where users may have different screen sizes and resolutions.

Class Setup

This application is setup using 4 Java classes:

1. gui.java

This class represents the front-end. It creates the GUI and holds the event listeners on all the buttons for the application to function.

2. images.java

This class is used as the backbone of the back end, as it pulls together the saved data and calls the histograms to be generated and uses the comparison measures on those histograms. It sorts the images by the comparison scores, so the most similar images are returned

3. compareHist.java

This class is used for generating distance measurement scores for comparing the query image to every other image in the database

4. saveData.java

This class is used to save data to an external file, and to read the data in from the external file

5.0 Testing and Model Performance

Firstly it is important to setup how the performance of each of the distance measurements will be monitored during testing and analysis. Precision-Recall curves will be developed, and the equations below illustrate how the data will be calculated [29].

$$Precision = \frac{n_k}{L}$$

$$Recall = \frac{n_k}{N}$$

where

n_k = the number of relevant images from the returned images

L = the number of retrieved images

N = the number of all relevant images in the image database

5.1 Correlation

Sample Results

The application will return the top 15 most similar images to the UI, based on the query image selected by the user.

For this simple test:

Bins used: *8*

Query Image: *Crowds001.jpg*

Fig. 7.14, “Correlation Results” shows all the scores calculated for every comparison image in the available database of images, sorted from highest to lowest scoring. From this it is evident that the function is working as all the scores fit the range from -1 to 1. Moreover, the resulting images have a precision of 100%. Fig. 5.1 below shows the output from this. The 15 images displayed are represented in Fig. 7.14 are highlighted in green.

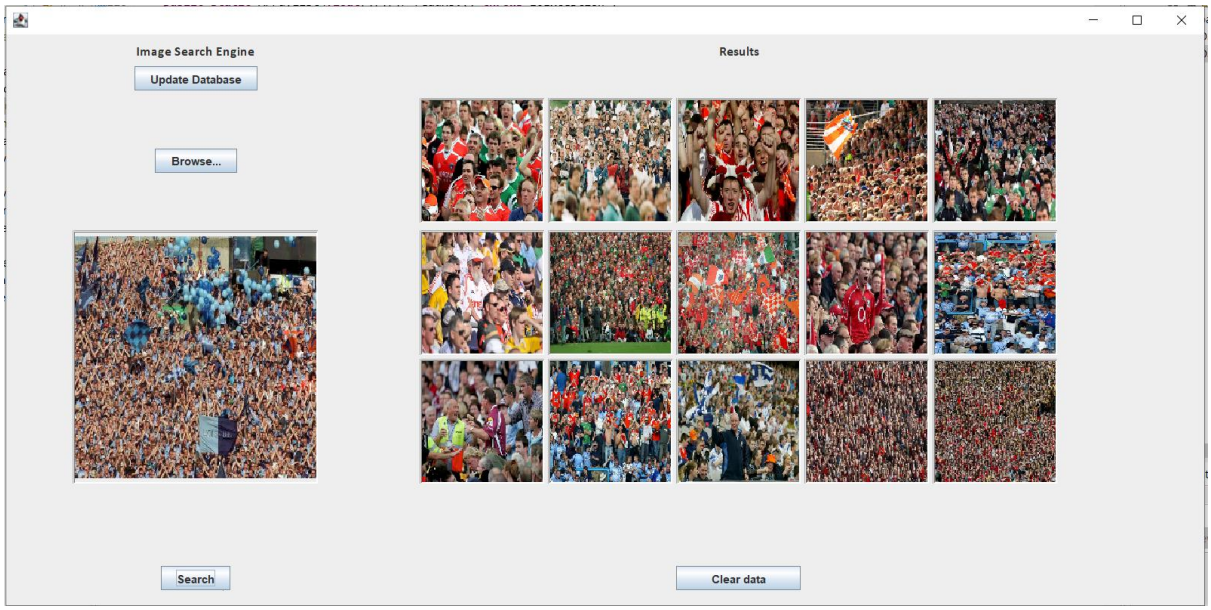

Fig. 5.1 Correlation UI results

Simple Model

A starting point of 8 bins was chosen for this baseline model. A small sample of 8 images will be used to find the precision and calculate an average time taken.

Query Image	Results	Time (s)
Crowds001.jpg	100%	2.615224
Crowds10.jpg	100%	2.463153
F1-Cars001.jpg	7%	2.394913
F1-Cars10.jpg	40%	2.285437
Horses001.jpg	67%	2.302323
Horses10.jpg	53%	2.295571

Precision (%)	Mean Time (s)
58	2.361017363

Landscapes001.jpg	80%	2.267539
Landscapes10.jpg	20%	2.263979

Fig. 5.2 is a curve which shows the average precision for each of the images in the table above over a sample of 5, 10, 15, 20, 25 and 30 images returned.
This gives a good base as to how the model might perform once the bins have been optimised.

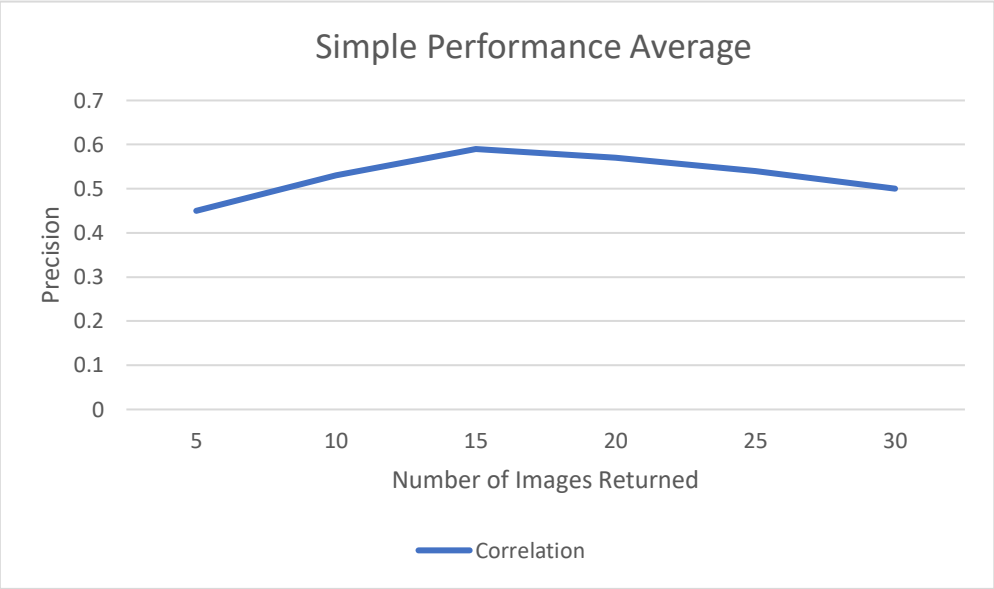

Fig. 5.2 Correlation simple Performance Average

Simple Precision-Recall Curve

This performance is backed up in the Precision-Recall Curve shown below in Fig. 5.3. Precision peaks at about 0.6 and drops to the baseline of 0.25 when recall hits 0.5. The data for this graph can be seen in Fig. 7.15. This data was calculated using 8 histogram bins and returning 15 images.

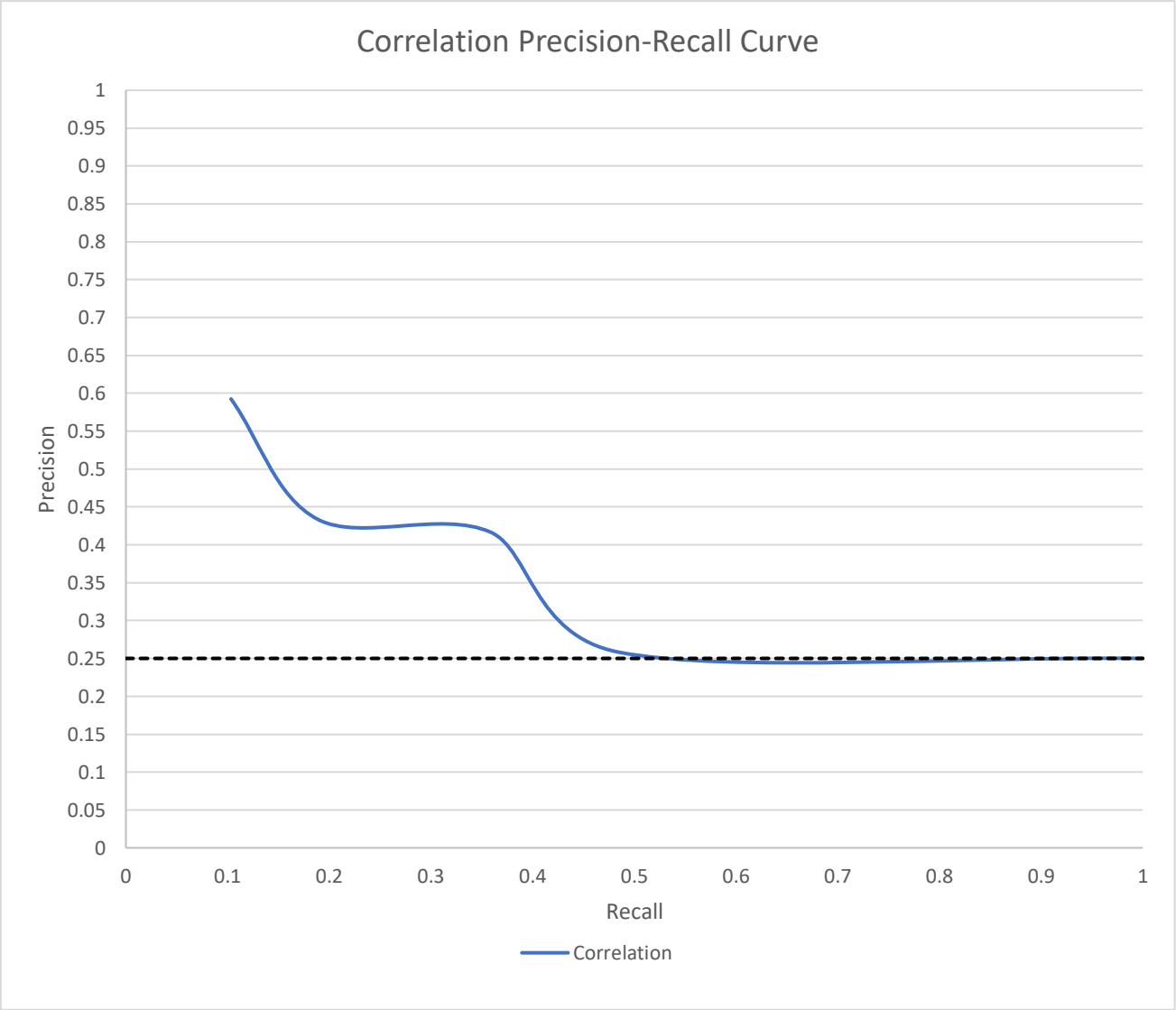

Fig. 5.3 Correlation simple Precision-Recall Curve

Bin Optimisation

For these tests, 15 images will be returned. Moreover, bin intervals of 5 will be used (starting at 5), until the processing time is too great (greater than 5 seconds).

Bin Number	**Average Results (%)**	**Average Time (s)**
5	49.17	0.775169038
10	51.00	4.576087713
15	49.17	14.30001306

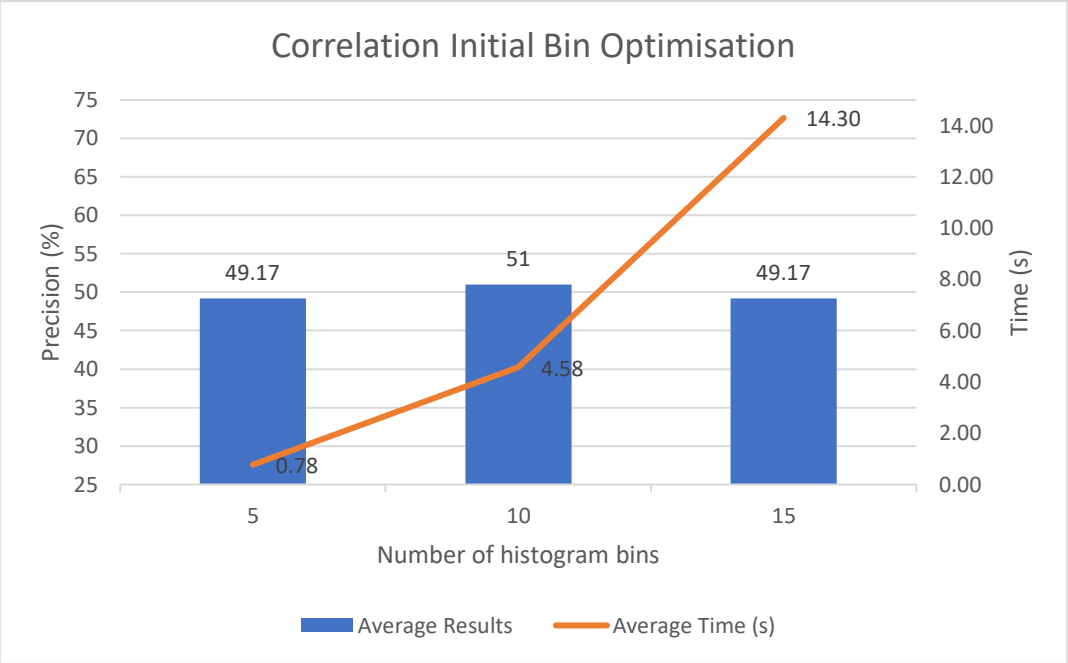

Fig. 5.4 Correlation Initial Bin Optimisation

Fig 5.4 shows that when the number of bins is increased, the processing time increases exponentially. This test was stopped at 15 bins, as any processing time over 14s would clearly be undesirable. Based on the results from Fig. 5.4, a focused test is needed for each bin around the best performing interval. In this case this will be bins 5,6,7,8, and 9. This is because in Fig. 5.4, it appears that going up to bins around 15 doesn't offer a boost to precision and the processing time is much too high. Moreover, bins <5 would offer very poor performance despite the low processing time.

Bin Number	Average Results (%)	Average Time (s)
5	49.17	0.775169
6	53.33	1.160116
7	56.67	1.795463
8	57.5	2.514298
9	54.17	3.52963
10	51.00	4.576088
11	59.17	5.819499

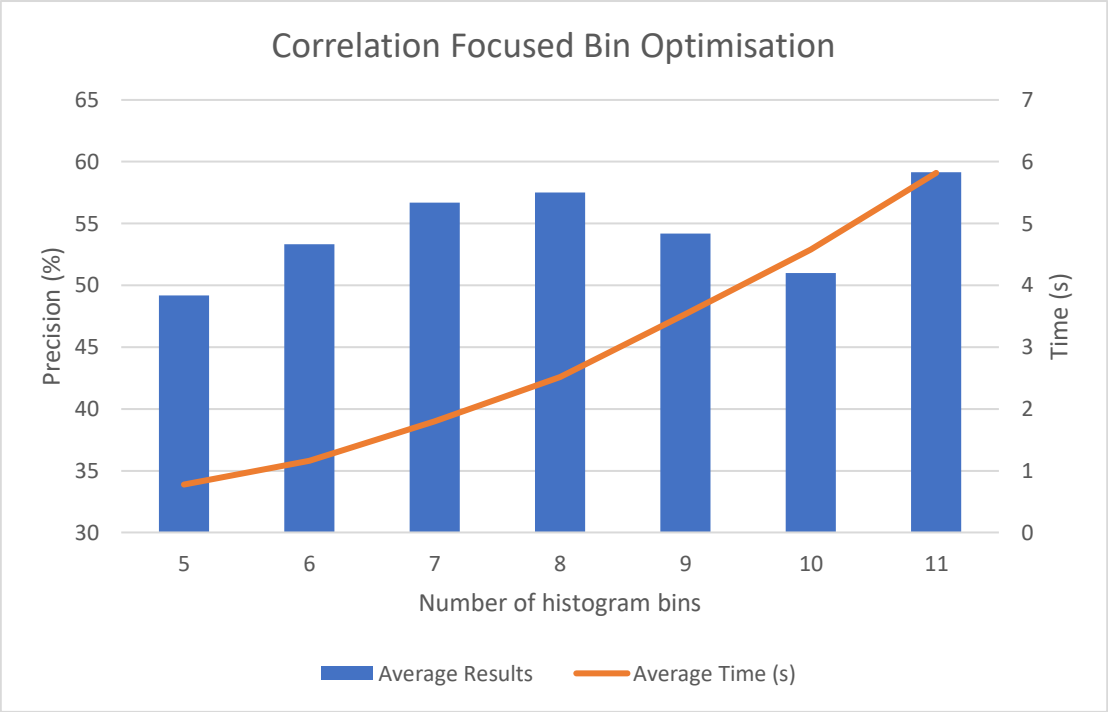

Fig. 5.5 Correlation Focused Bin Optimisation

The chosen bins proved to be a good range as it tests the upper limits of the time constraint (5 seconds). From this it is conclusive that 8 bins would the best to use going forward as it offers the highest precision (57.5%), under 5 seconds.

Using 11 bins was a possibility but there's a negative trade-off between a small increase in precision and relatively large increase in processing time.

Full Performance Results

Due to this method operating best using 8 bins, this test will operate with this setting. Fig. 7.16 shows the results from a series of tests on each available test image. The precision was tested on each image when 5, 10, 15, 20, 25 and 30 images are returned. These numbers were chosen because this sample size will give a very strong indication as to how the application will perform using this method, and how its performance would change. Moreover, testing every image will give the most accurate and comprehensive results. Before running these tests, based on the results from the simple model already tested, the precision should gradually decrease as the number of images returned is increased. Fig. 5.6 below shows the averages from the full results table above. The trendline has been calculated from the average (mean) precision of each image group (Crowds, F1-Cars, Horses, and Landscapes). The results from this table are displayed on the line graph below (Fig. 5.7).

Num of Images Returned	*Image Group*				
	Crowds	**F1-Cars**	**Horses**	**Landscapes**	**TRENDLINE**
5	0.818181818	0.295238095	0.645454545	0.628571429	0.596861472
10	0.786363636	0.280952381	0.581818182	0.595238095	0.561093074
15	0.757575758	0.295238095	0.536363636	0.536507937	0.531421356
20	0.731818182	0.280952381	0.525	0.523809524	0.515395022
25	0.729090909	0.272380952	0.514545455	0.478181818	0.498549784
30	0.706060606	0.271428571	0.510606061	0.480952381	0.492261905

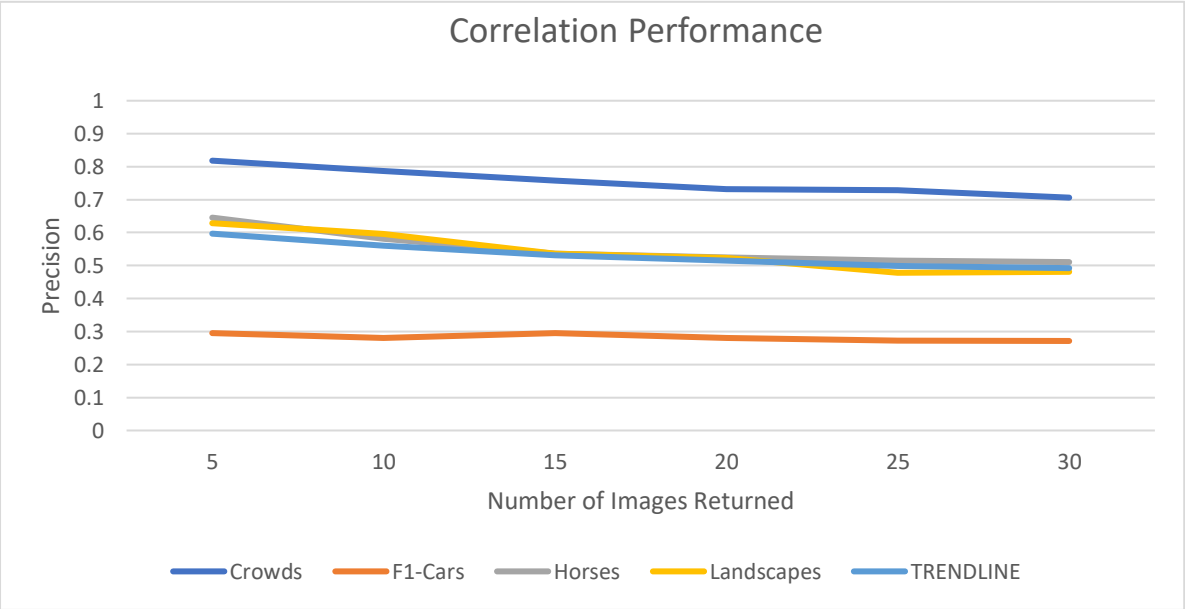

Fig. 5.7 Correlation Performance

The trendline shows this model's results are strongest when the number of images returned is 5 (min), as it has a precision of 59.68% (2dp), and it is at its weakest when the number of images returned is 30 (max), as its precision is 49.23% (2dp). This is a difference of 10.45% which is very significant and follows the trend initially set out by the simple model. This means that for this Correlation model, as the number of images increases, precision decreases.

5.2 Chi-Square

Sample Results

The application will return the top 15 most similar images to the UI, based on the query image selected by the user.

For this example:
Bins used: *8*
Query Image: *Crowds001.jpg*

Fig. 7.17, "Chi-Square Results" shows all the scores calculated for every comparison image in the available database of images, sorted from lowest to highest scoring. From this it is evident that the function is working as all the scores fit the range from 0 to 1. Moreover, the resulting images have a precision of 100%. Fig. 5.8 below shows the output from this. The 15 images displayed are represented in Fig. 7.17 are highlighted in green.

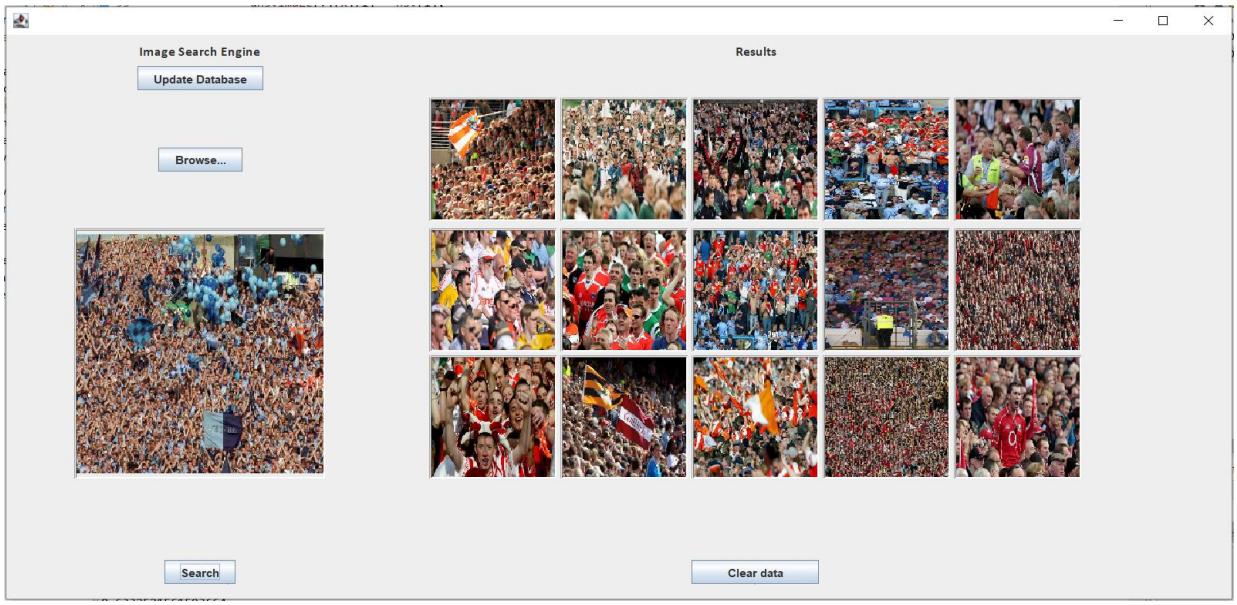

Fig. 5.8 Chi-Square UI results

Simple Model

A starting point of 8 bins was chosen for this baseline model. A small sample of 8 images will be used to find the precision and calculate an average time taken.

Query Image	Results	Time (s)
Crowds001.jpg	100%	3.9969346
Crowds10.jpg	100%	3.708615
F1-Cars001.jpg	80%	3.7084899
F1-Cars10.jpg	47%	3.4998552

Precision (%)	Mean Time (s)
68.38	3.685946725

Horses001.jpg	73%	3.5586724
Horses10.jpg	40%	3.705902
Landscapes001.jpg	80%	3.7998657
Landscapes10.jpg	27%	3.509239

Fig. 5.9 is a curve which shows the average precision for each of the images represented above over a sample of 5, 10, 15, 20, 25 and 30 images returned.

This gives a good base as to how the model performs, and how once the bins have been optimised this will improve.

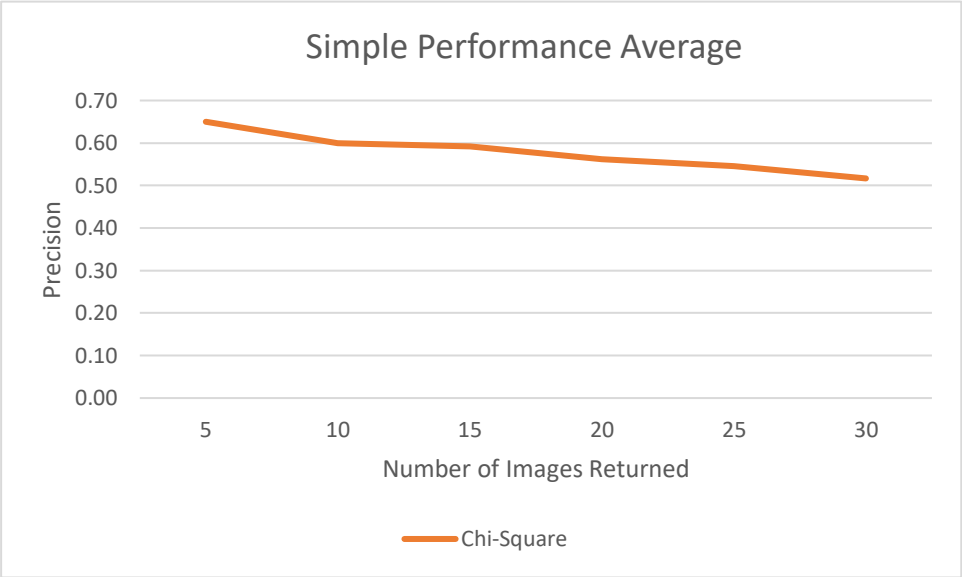

Fig. 5.9 Chi-Square simple Performance Average

Simple Precision-Recall Curve

This performance is backed up in the Precision-Recall Curve shown below in Fig. 5.10. Precision peaks at about 0.6 and is good at keeping its precision high as it doesn't drop to 0.25 until the recall is 1. The data for this graph can be seen in Fig. 7.15. This data was calculated using 8 histograms and returning 15 images.

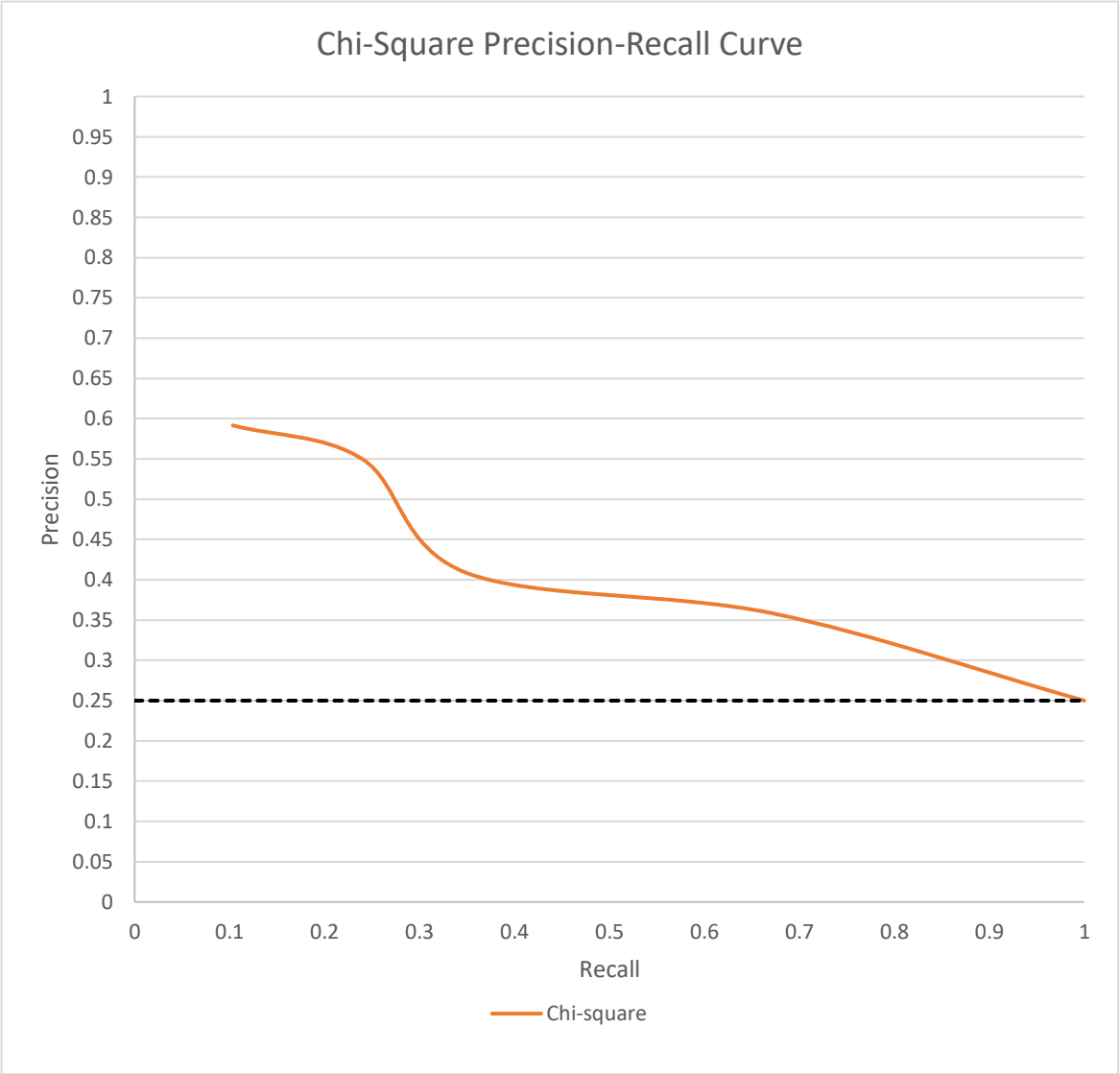

Fig. 5.10 Chi-Square simple Precision-Recall Curve

Bin Optimisation

For these tests, 15 images will be returned. Moreover, bin intervals of 5 will be used (starting at 5), until the processing time is too great (greater than 5 seconds).

Bin Number	**Average Results (%)**	**Average Time (s)**
5	49.17	2.26
10	50	5.72
15	45	15.90

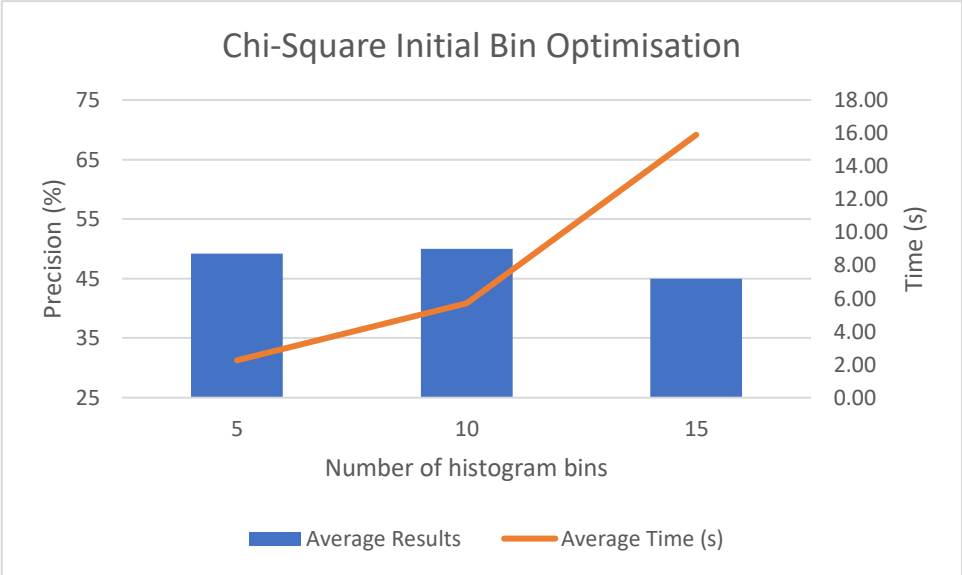

Fig. 5.11 Chi-Square Initial Bin Optimisation

Fig. 5.11 shows that when the number of bins is increased from 5 to 10 the precision increases slightly. Bins greater than 10 should be discarded as the processing time would be too high and based on this test would not offer an increase in precision. This test stopped at 15 bins as any processing time above 15.9s would clearly be undesirable for this application. Based on the results from Fig. 5.11, a focused test is needed for each bin around the best performing interval. In this case this will be bins 6,7,8, and 9. It appears that going above 10 bins doesn't offer a boost to precision and the processing time is too high. Moreover, bins <5 would offer very poor performance despite the low processing time.

Bin Number	**Average Results (%)**	**Average Time (s)**
4	50.83	1.804158
5	49.17	2.258742
6	54.17	2.40748
7	57.5	3.070594
8	59.17	3.569663
9	55	4.955153

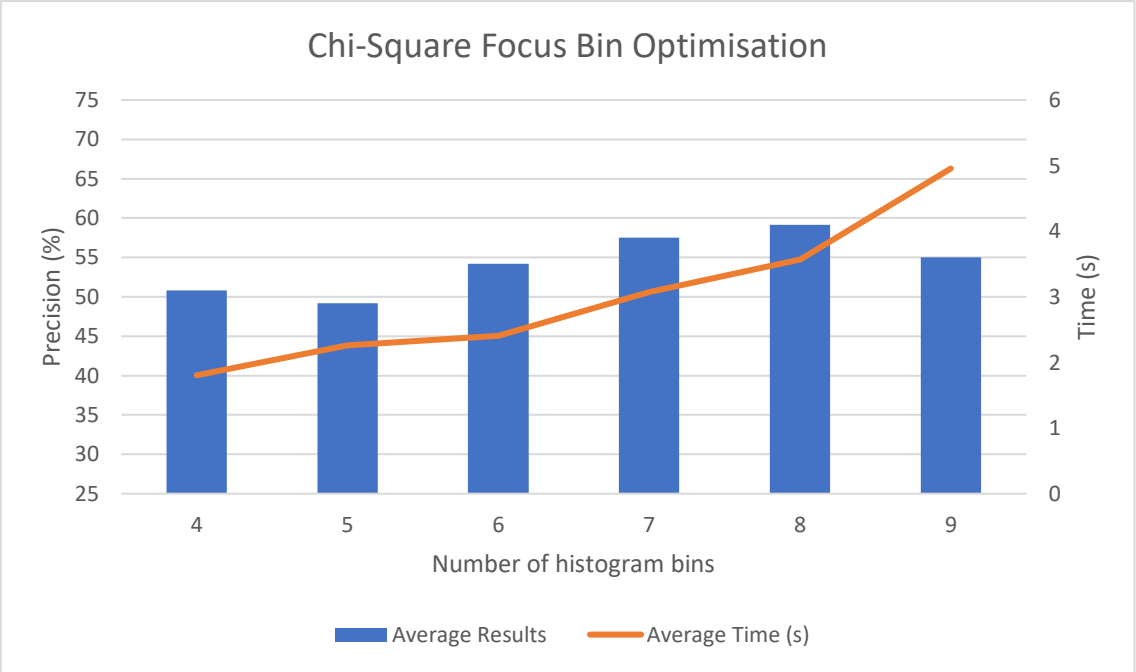

Fig. 5.12 Chi-Square Focused Bin Optimisation

The chosen bins proved to be a good range as it tests the upper limits of the time constraint (5 seconds). From this it is conclusive that 8 bins would the best to use going forward as it offers the highest precision (59.17%), under 5 seconds.

Full Performance Results

Due to this method operating best using 8 bins, this test will operate with this setting. Fig. 7.18 shows the results from a series of tests on each available test image. The precision was tested on each image when 5, 10, 15, 20, 25 and 30 images are returned. These numbers were chosen because this sample size will give a very strong indication as to how the application will perform using this method, and how its performance would change. Moreover, testing every image will give the most accurate and comprehensive results. Before running these tests, based on the results from the simple model already tested, the precision should gradually decrease as the recall is increased. Fig. 5.13 below shows the averages from the full results table above. The trendline has been calculated from the average (mean) precision of each image group (Crowds, F1-Cars, Horses, and Landscapes). The results from this table are displayed on the line graph below (Fig. 5.14).

Num of Images Returned	*Image Group*				
	Crowds	**F1-Cars**	**Horses**	**Landscapes**	**TRENDLINE**
5	0.981818182	0.419047619	0.627272727	0.542857143	0.642748918
10	0.95	0.39047619	0.6	0.476190476	0.604166667
15	0.936363636	0.368253968	0.578787879	0.428571429	0.577994228
20	0.918181818	0.347619048	0.540909091	0.383333333	0.547510823
25	0.883636364	0.321904762	0.516363636	0.36952381	0.522857143
30	0.857575758	0.314285714	0.509090909	0.357142857	0.50952381

Fig. 5.13 Chi-Square condensed results

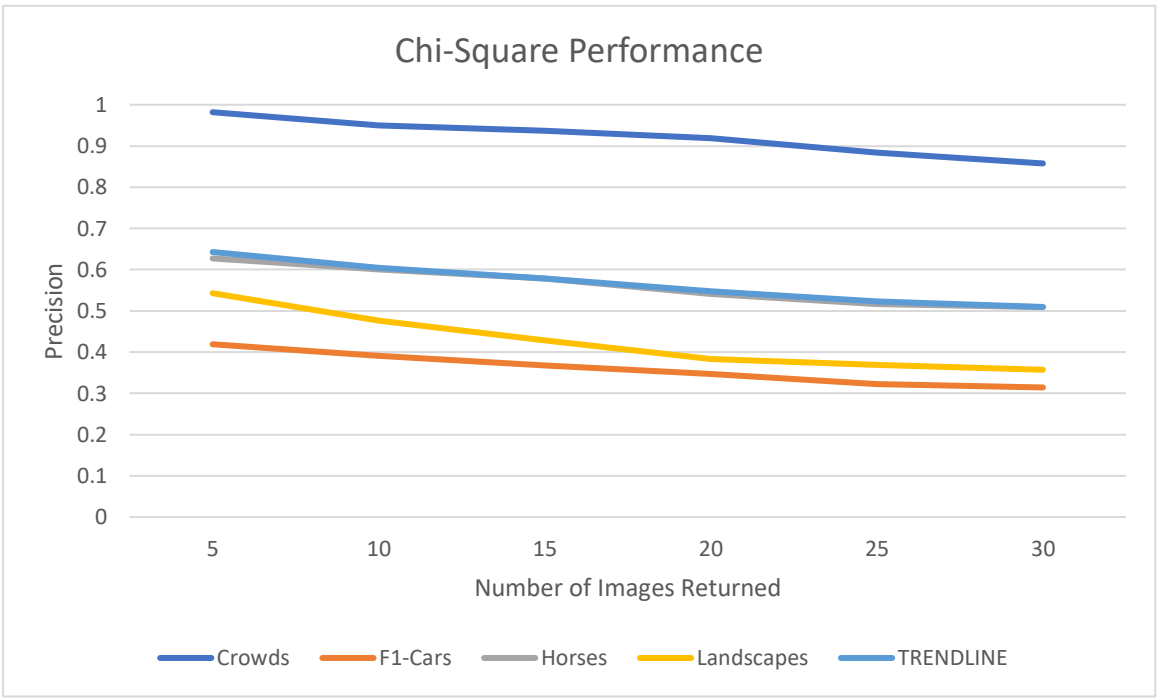

Fig. 5.14 Chi-Square Performance

The trendline shows this model's results are strongest when the number of images returned is 5 (min), as it has a precision of 64.27% (2dp), and it is at its weakest when the number of images returned is 30 (max), as its precision is 50.95% (2dp). This is a difference of 13.32% which is very significant and follows the trend initially set out by the simple model. This means that for this Chi-Square model, as the number of images increases, precision decreases.

5.3 Intersection

Sample Results

The application will return the top 15 most similar images to the UI, based on the query image selected by the user.

For this example:
Bins used: *8*
Query Image: *Crowds001.jpg*

Fig. 7.19, "Intersection Results" shows all the scores calculated for every comparison image in the available database of images, sorted from lowest to highest scoring. From this it is evident that the function is working as all the scores fit the range from 0 to 1. Moreover, the resulting images have a precision of 100%.
Fig 5.15 below shows the output from this. The 15 images displayed are represented in Fig. 7.14 are highlighted in green.

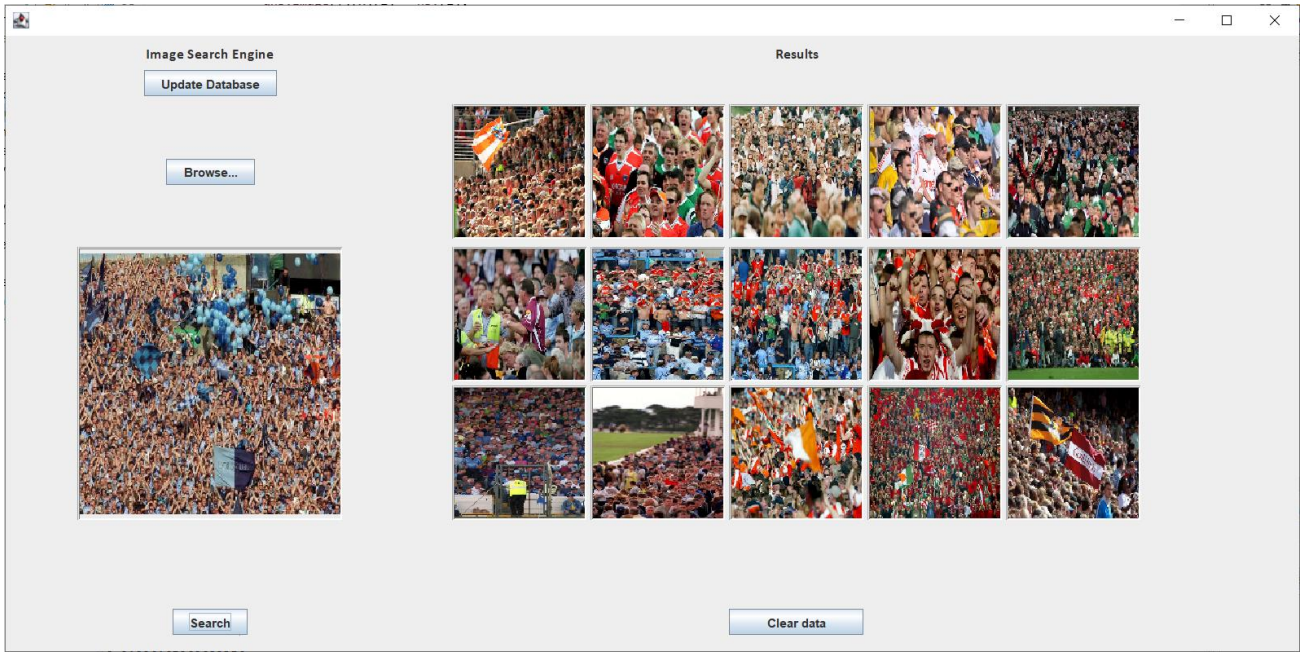

Fig. 5.15 Intersection UI results

Simple Model

A starting point of 8 bins was chosen for this baseline model. A small sample of 8 images will be used to find the precision and calculate an average time taken.

Query Image	Results	Time (s)
Crowds001.jpg	100%	2.5834942
Crowds10.jpg	100%	2.3988715
F1-Cars001.jpg	7%	2.3432522
F1-Cars10.jpg	60%	2.4349141
Horses001.jpg	73%	2.329884
Horses10.jpg	40%	2.3156995
Landscapes001.jpg	60%	2.3262559
Landscapes10.jpg	27%	2.356708

Precision (%)	Mean Time (s)
58.38	2.386134925

Fig. 5.16 is a curve which shows the average precision for each of the images above over a sample of 5, 10, 15, 20, 25 and 30 images returned.

This gives a good base as to how the model performs, and how once the bins have been optimised this will improve.

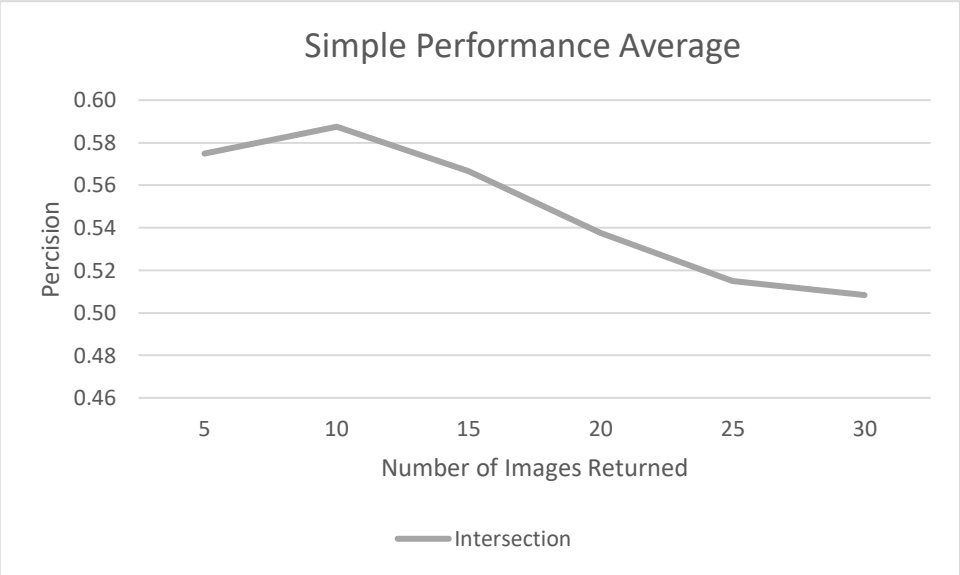

Fig. 5.16 Intersection simple Performance Average

Simple Precision-Recall Curve

This performance is backed up in the Precision-Recall Curve shown below in Fig. 5.17. Precision peaks at around 0.57 and drops at a relatively consistent rate down to 0.25 when the recall is 1. The data for this graph can be seen in Fig. 7.15. This data was calculated using 8 histogram bins and returning 15 images.

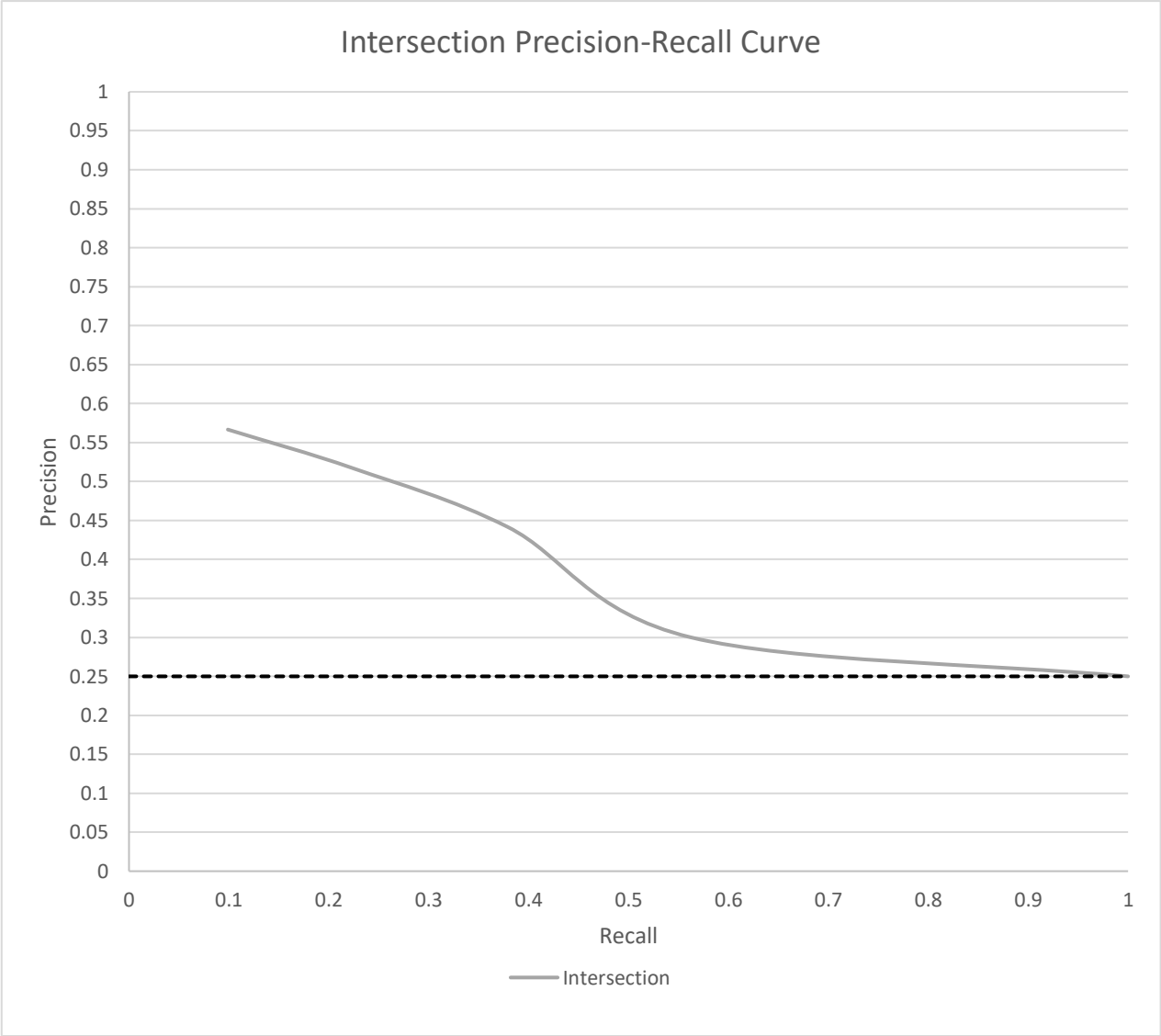

Fig. 5.17 Intersection simple Precision-Recall Curve

Bin Optimisation

For these tests, 15 images will be returned. Moreover, bin intervals of 5 will be used (starting at 5), until the processing time is too great (greater than 5 seconds).

Bin Number	Average Results (%)	Average Time (s)
5	48.33	0.77
10	51.67	4.40
15	50.83	14.49

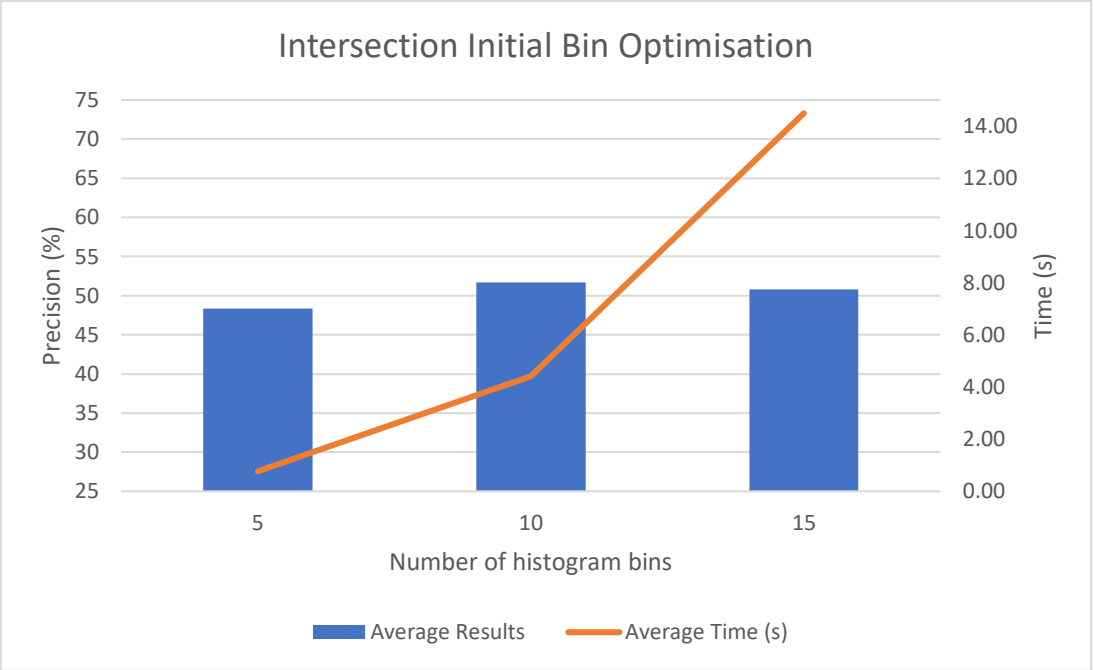

Fig. 5.18 Intersection Initial Bin Optimisation

Fig. 5.18 shows that when the number of bins is increased from 5 to 10 the precision and processing time increases. Based on this graph it seems that 11 bins may be able to stay below the 5 second processing but any more would be unusable. This test stopped at 15 bins as any processing time over 14.49s would clearly be undesirable for this application.

Based on the results from Fig. 5.18, a focused test is needed for each bin around the best performing interval. In this case this will be bins 7,8,9, and 11. Fig. 5.18 suggests bins less than 5 would offer very poor performance despite the low processing time.

Bin Number	**Average Results (%)**	**Average Time (s)**
7	58.33	1.75
8	59.17	2.39
9	55.83	3.34
10	51.67	4.400278
11	58.33	5.971803

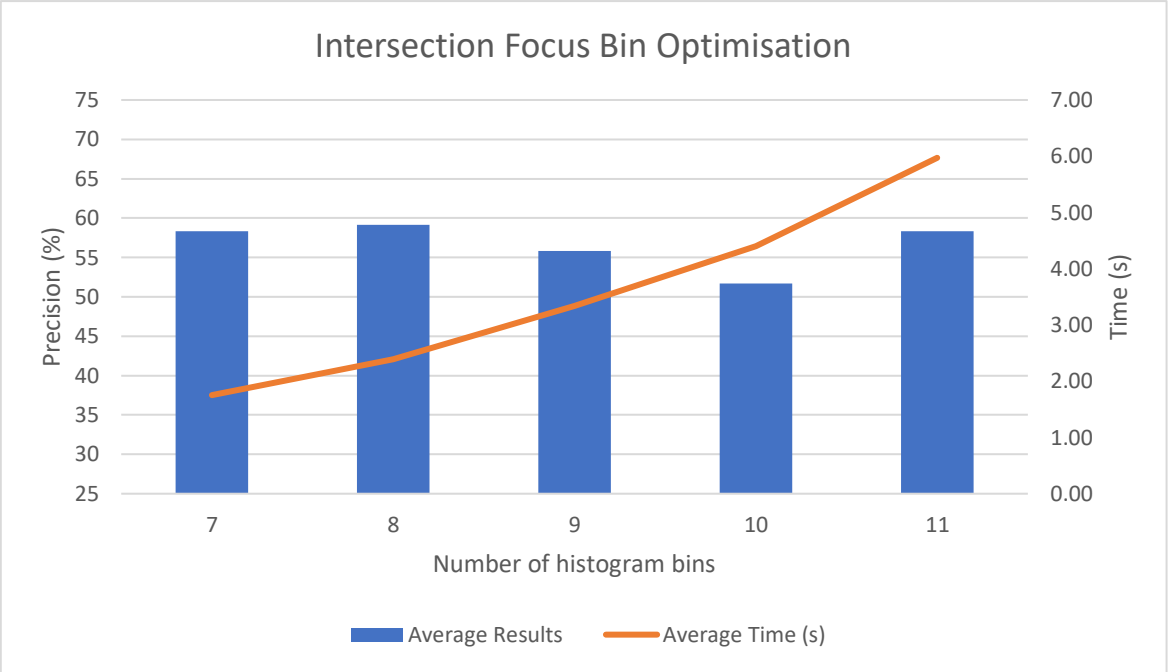

Fig. 5.19 Intersection Focused Bin Optimisation

The chosen bins proved to be a good range as it tests the upper limits of the time constraint (5 seconds). From this it is conclusive that 8 bins would the best to use going forward as it offers the highest precision (59.17%), under 5 seconds.

Full Performance Results

Due to this method operating best using 8 bins, this test will operate with this setting. Fig. 7.20 shows the results from a series of tests on each available test image. The precision was tested on each image when 5, 10, 15, 20, 25 and 30 images are returned. These numbers were chosen because this sample size will give a very strong indication as to how the application will perform using this method, and how its performance would change. Moreover, testing every image will give the most accurate and comprehensive results. Before running these tests, based on the results from the simple model already tested, the precision should gradually decrease as the recall is increased. Fig. 5.20 below shows the averages from the full results table above. The trendline has been calculated from the average (mean) precision of each image group (Crowds, F1-Cars, Horses, and Landscapes). The results from this table are displayed on the line graph below (Fig. 5.21).

Num of Images Returned	*Image Group*				
	Crowds	**F1-Cars**	**Horses**	**Landscapes**	**TRENDLINE**
5	0.972727273	0.428571429	0.636363636	0.495238095	0.633225108
10	0.945454545	0.40952381	0.627272727	0.452380952	0.608658009
15	0.93030303	0.380952381	0.575757576	0.415873016	0.575721501
20	0.9	0.354761905	0.547727273	0.376190476	0.544669913
25	0.865454545	0.339047619	0.538181818	0.36952381	0.528051948
30	0.842424242	0.33015873	0.528787879	0.352380952	0.513437951

Fig. 5.20 Intersection condensed results

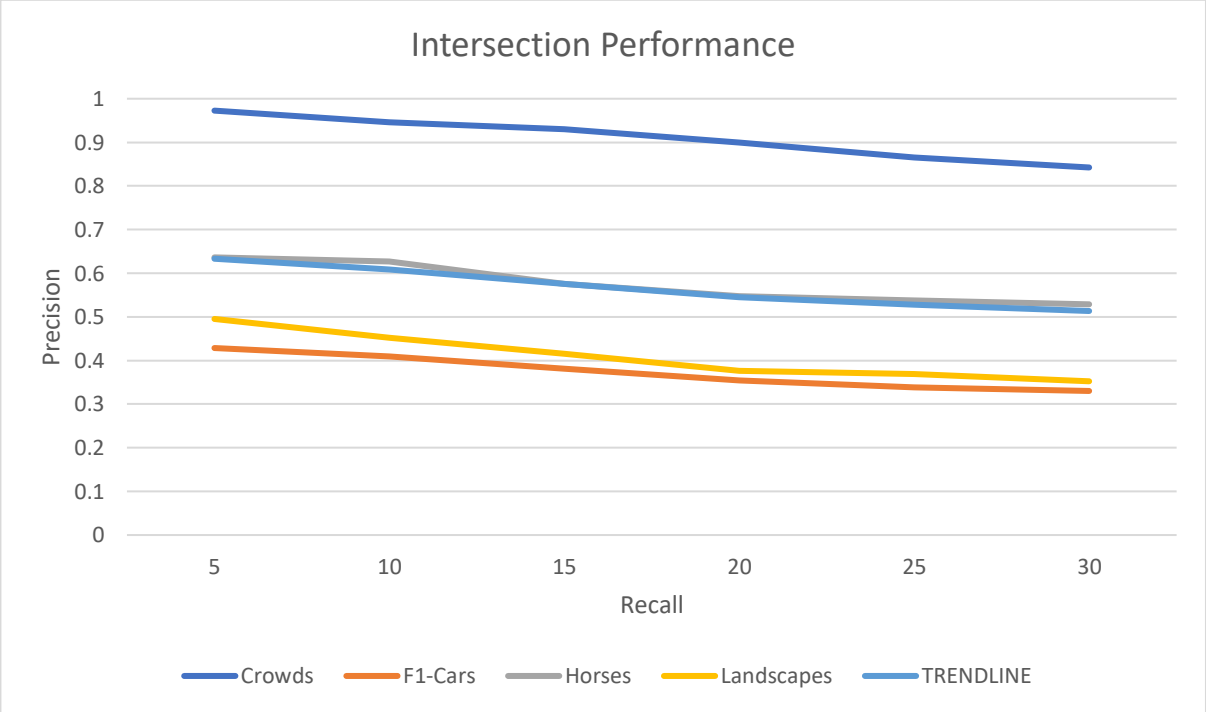

Fig. 5.21 Intersection Performance

The trendline shows this model's results are strongest when the number of images returned is 5 (min), as it has a precision of 63.32% (2dp), and it is at its weakest when recall is 30 (max), as its precision is 51.34% (2dp). This is a difference of 11.98% which is very significant and follows the trend initially set out by the simple model. This means that for this Intersection model, as the number of images returned increases, precision decreases.

5.4 Bhattacharyya

Sample Results

The application will return the top 15 most similar images to the UI, based on the query image selected by the user.

For this example:
Bins used: *8*
Query Image: *Crowds001.jpg*

Fig. 7.21, "Bhattacharyya Results" shows all the scores calculated for every comparison image in the available database of images, sorted from lowest to highest scoring. From this it is evident that the function is working as all the scores fit the range from 0 to 1. Moreover, the resulting images have a precision of 100%.
Fig 5.22 below shows the output from this. The 15 images displayed are represented in Fig. 7.21 are highlighted in green.

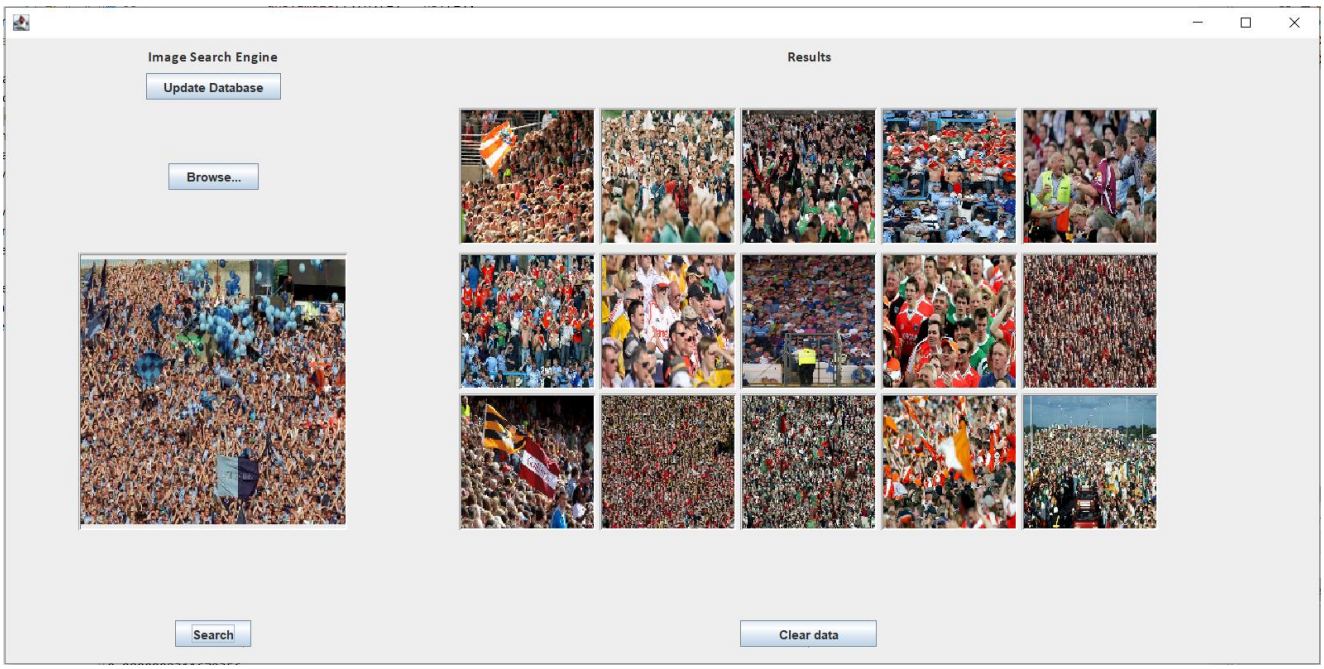

Fig. 5.22 Bhattacharyya UI results

Simple Model

A starting point of 8 bins was chosen for this baseline model. A small sample of 8 images will be used to find the precision and calculate an average time taken.

Query Image	Results	Time (s)
Crowds001.jpg	100%	4.0466897
Crowds10.jpg	100%	6.378527
F1-Cars001.jpg	7%	3.5315454
F1-Cars10.jpg	47%	3.6238344
Horses001.jpg	60%	3.6727772
Horses10.jpg	33%	3.6193182
Landscapes001.jpg	73%	3.4608025
Landscapes10.jpg	20%	3.4783416

Precision (%)	Mean Time (s)
55	3.9764795

Fig. 5.23 is a curve which shows the average precision for each of the images above over a sample of 5, 10, 15, 20, 25 and 30 images returned.

This gives a good base as to how the model performs, and how once the bins have been optimised this will improve.

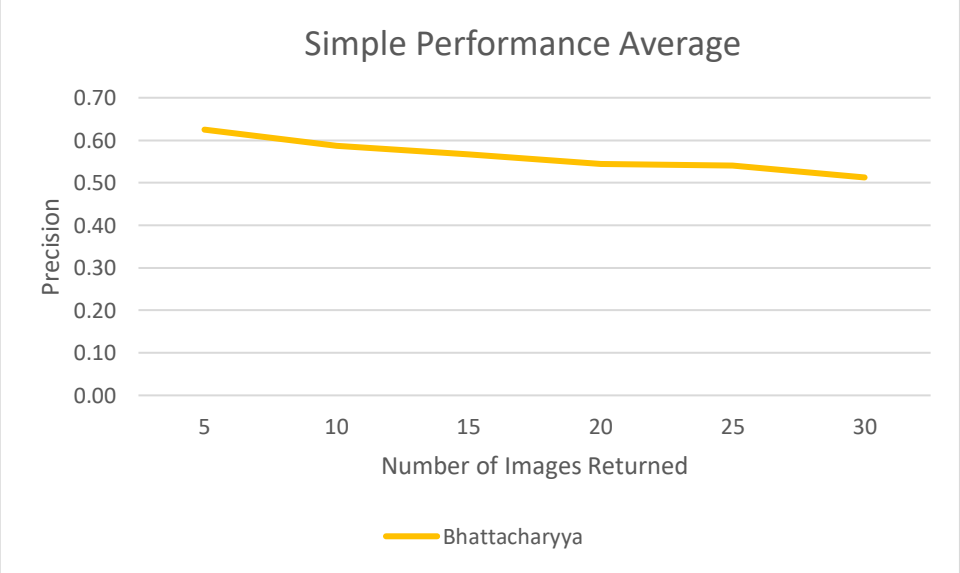

Fig. 5.23 Bhattacharyya simple Performance Average

Simple Precision-Recall Curve

This performance is backed up in the Precision-Recall Curve shown below in Fig. 5.24. Precision peaks at about 0.57 and drops steadily to the baseline of 0.25 when the recall is 1. The data for this graph can be seen in Fig. 7.15. This data was calculated using 8 histogram bins and returning 15 images.

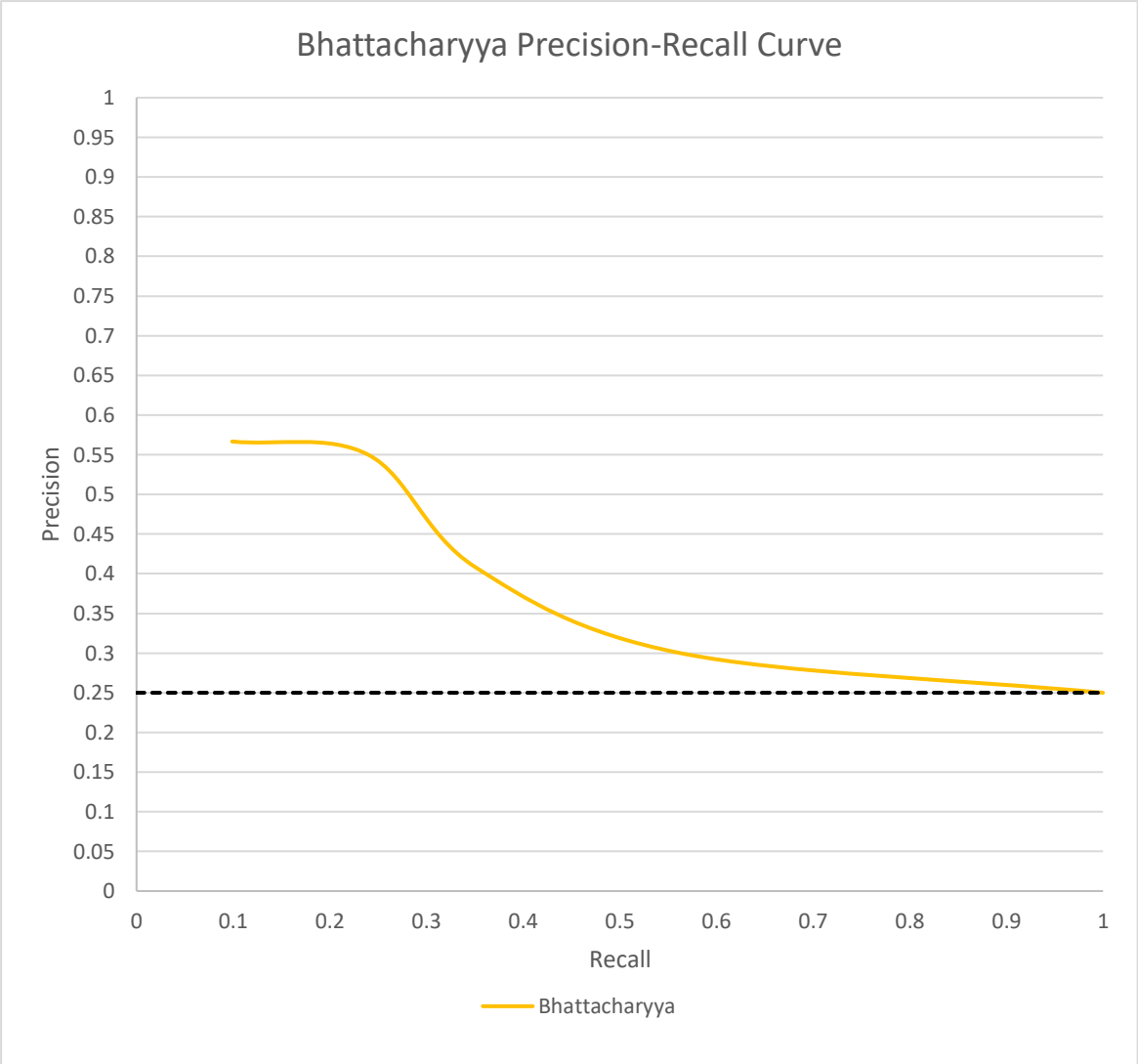

Fig. 5.24 Bhattacharyya simple Precision-Recall

Bin Optimisation

For these tests, 15 images will be returned. Moreover, bin intervals of 5 will be used (starting at 5), until the processing time is too great (greater than 5 seconds).

Bin Number	Average Results (%)	Average Time (s)
5	52.5	1.97
10	48.33	5.74
15	45	15.75

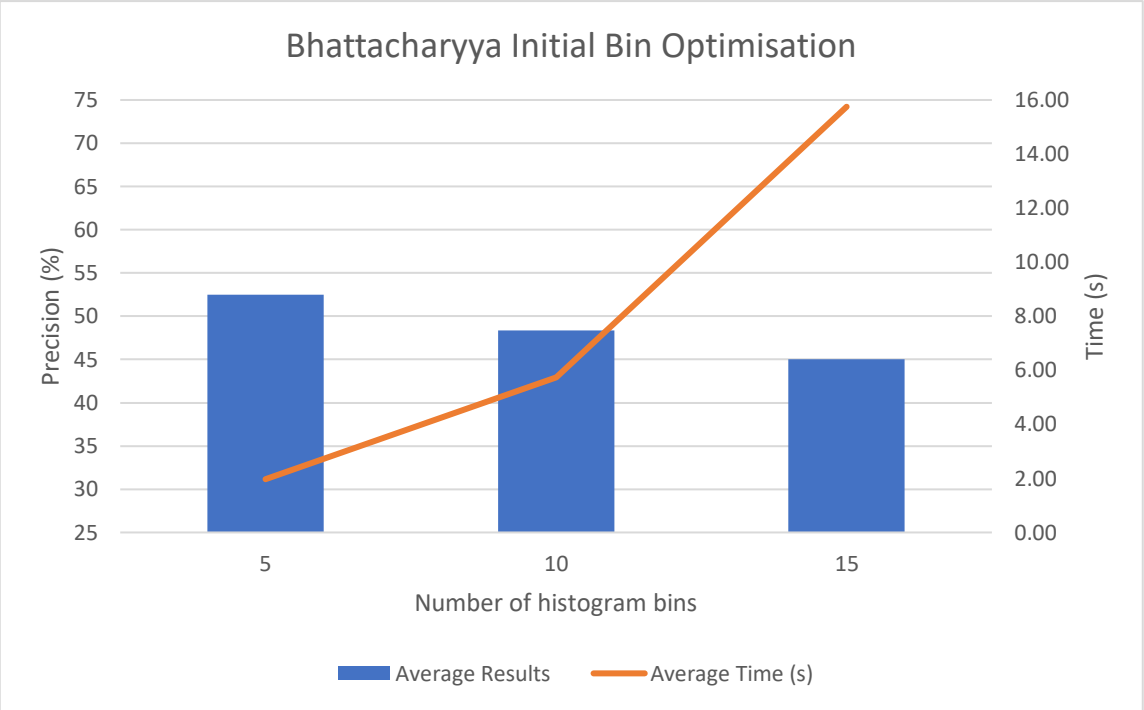

Fig. 5.25 Bhattacharyya Initial Bin Optimisation

Fig. 5.25 differs from the initial bin optimisation for the other models as it appears that the precision decreases as the bin number is increased. Despite this, processing time acts normally as it clearly increases.

Based on this graph it seems that any more than 10 bins would be unusable, as it would breach the 5 second processing limit. This test stopped at 15 bins as any processing time over 15.75 would clearly be undesirable for this application.

Based on the results from Fig. 5.25, a focused test is needed for each bin around the best performing interval. In this case this will be bins 4,6,7,8, and 9.

Bin Number	**Average Results (%)**	**Average Time (s)**
4	50.83	1.74
5	52.5	1.97
6	56.67	2.52
7	60.83	2.782253
8	56.67	3.672113
9	56.67	4.51194095

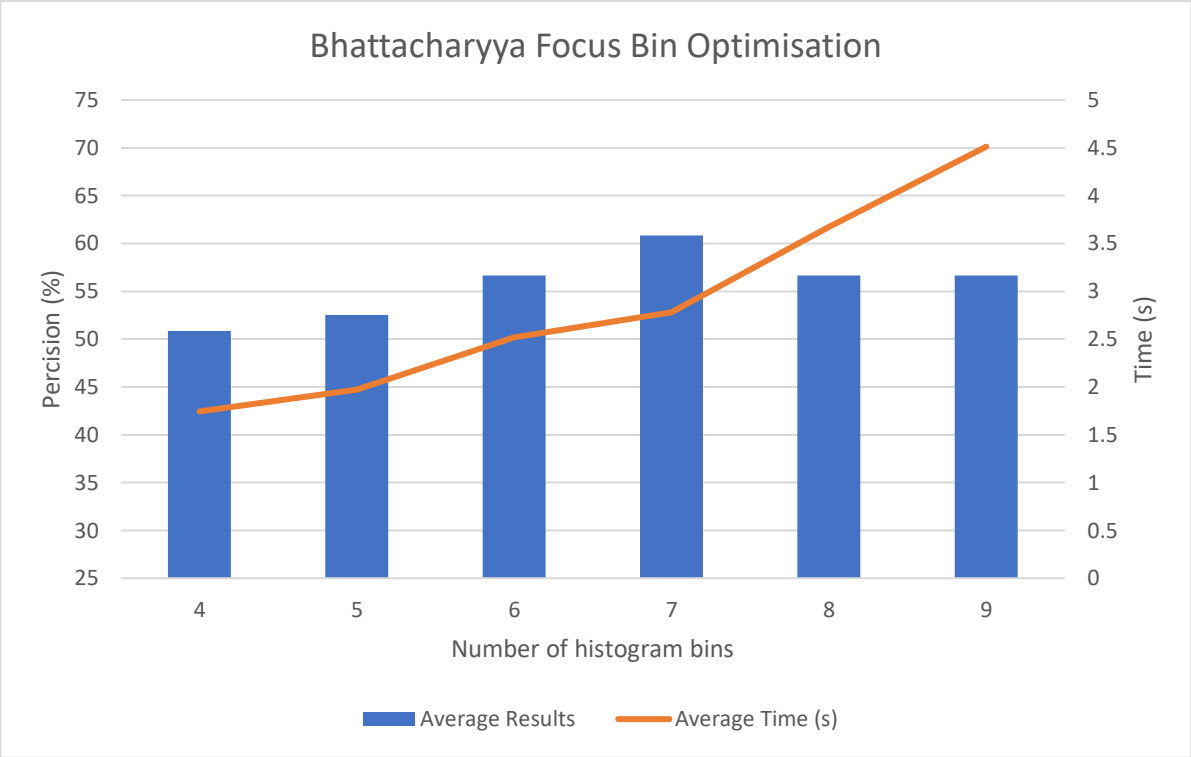

Fig. 5.26 Bhattacharyya Focused Bin Optimisation

The chosen bins proved to be a good range as it tests the upper limits of the time constraint (5 seconds). From this it is conclusive that 7 bins would be the best to use going forward as it offers the highest precision (56.67%), under 5 seconds.

Full Performance Results

Due to this method operating best using 7 bins, this test will operate with this setting. Fig. 7.22 shows the results from a series of tests on each available test image. The precision was tested on each image when 5, 10, 15, 20, 25 and 30 images are returned. These numbers were chosen because this sample size will give a very strong indication as to how the application will perform using this method, and how its performance would change. Moreover, testing every image will give the most accurate and comprehensive results. Before running these tests, based on the results from the simple model already tested, the precision should gradually decrease as the number of images is increased. Fig. 5.27 below shows the averages from the full results table above. he trendline has been calculated from the average (mean) precision of each image groups (Crowds, F1-Cars, Horses, and Landscapes). The results from this table are displayed on the line graph below (Fig. 5.28).

Num of Images Returned	*Image Group*				
	Crowds	**F1-Cars**	**Horses**	**Landscapes**	**TRENDLINE**
5	0.936363636	0.447619048	0.672727273	0.552380952	0.652272727
10	0.931818182	0.371428571	0.595454545	0.452380952	0.587770563
15	0.912121212	0.352380952	0.584848485	0.406349206	0.563924964
20	0.884090909	0.326190476	0.572727273	0.376190476	0.539799784
25	0.870909091	0.331428571	0.554545455	0.346666667	0.525887446
30	0.854545455	0.328571429	0.537878788	0.323809524	0.511201299

Fig. 5.27 Bhattacharyya condensed results

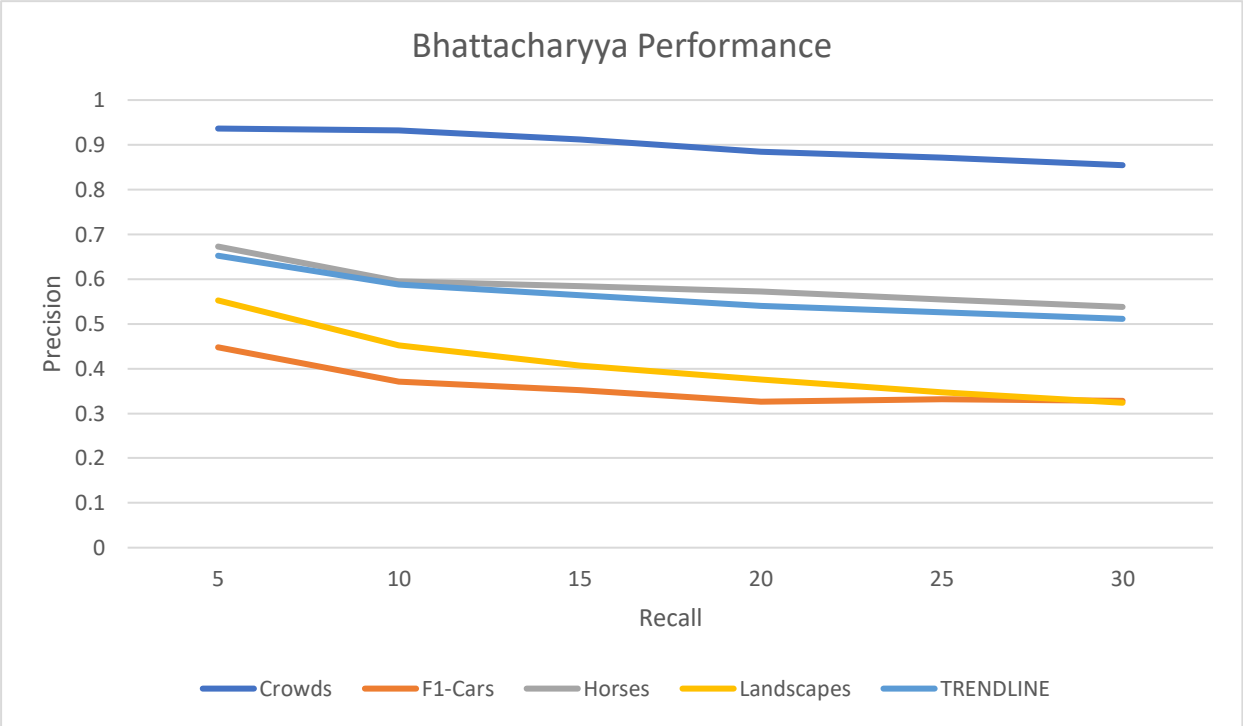

Fig. 5.28 Bhattacharyya Performance

The trendline shows this model's results are strongest when the number of images returned is 5 (min), as it has a precision of 65.23% (2dp), and it is at its weakest when the number of images returned is 30 (max), as its precision is 51.12% (2dp). This is a difference of 14.11% which is very significant and follows the trend initially set out by the simple model. This means that for this Bhattacharyya model, as the number of images returned increases, precision decreases.

6.0 Overall Results

6.1 Simple Models

Fig. 7.18 shows the results from a very basic initial test to help give a better understanding as to how each of the models perform in comparison to each other. In this test, 15 images are being returned and the optimal bin number is used for each comparison method.

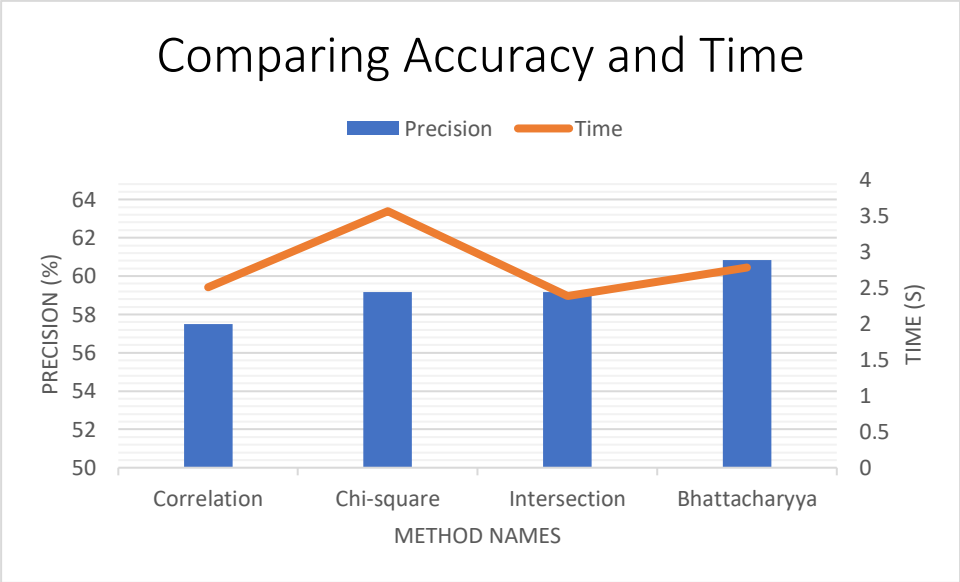

Fig. 6.1 Comparing accuracy and time for simple models

As all of these models have optimal bins already chosen, time should not be a major issue and it is primarily the best precision that matters the most. It is clear to see that Correlation is the worst performing model as its precision is the lowest at only 57.5% despite the fact its processing time is relatively low at just 2.51 seconds. Chi-Square and Intersection have both performed the same in terms of precision in this test, recording 59.17%. Despite this Intersection would still be preferable because it has a lower processing time. 3.569663 seconds compared to 2.389432 seconds. Bhattacharya initially appears to be the best performing model as it has the highest precision (60.83%). It does have the second highest processing time behind Chi-Square, but this should not be an issue as this time is acceptable.

Method	**Precision (%) / time (s) (2dp)**
Correlation	22.87
Chi-Square	16.58
Intersection	24.76
Bhattacharyya	21.86

Fig. 6.2 Simple Model Efficiency

Looking deeper into the efficiency of these models (as seen in Fig. 6.2), we can see that despite Bhattacharyya being the most precise, it has the second lowest efficiency according to this measure (behind Chi-Square). Moreover, it is also important to note that Intersection has the highest efficiency at "24.76" so it might perform better when we explore these methods and results further.

Simple Performance Averages

Recall	Correlation	Chi-Square	Intersection	Bhattacharyya
5	0.45	0.65	0.58	0.63
10	0.53	0.60	0.59	0.59
15	0.59	0.59	0.57	0.57
20	0.57	0.56	0.54	0.54
25	0.54	0.55	0.52	0.54
30	0.5	0.52	0.51	0.51

Fig. 6.3 Simple Precision-Recall Averages Table

The data for Fig. 6.3 and Fig. 6.4 is taken from the trend line from each of the simple performance curves mentioned previously in each of the methods sections (Section 5.1, 5.2, 5.3, 5.4).

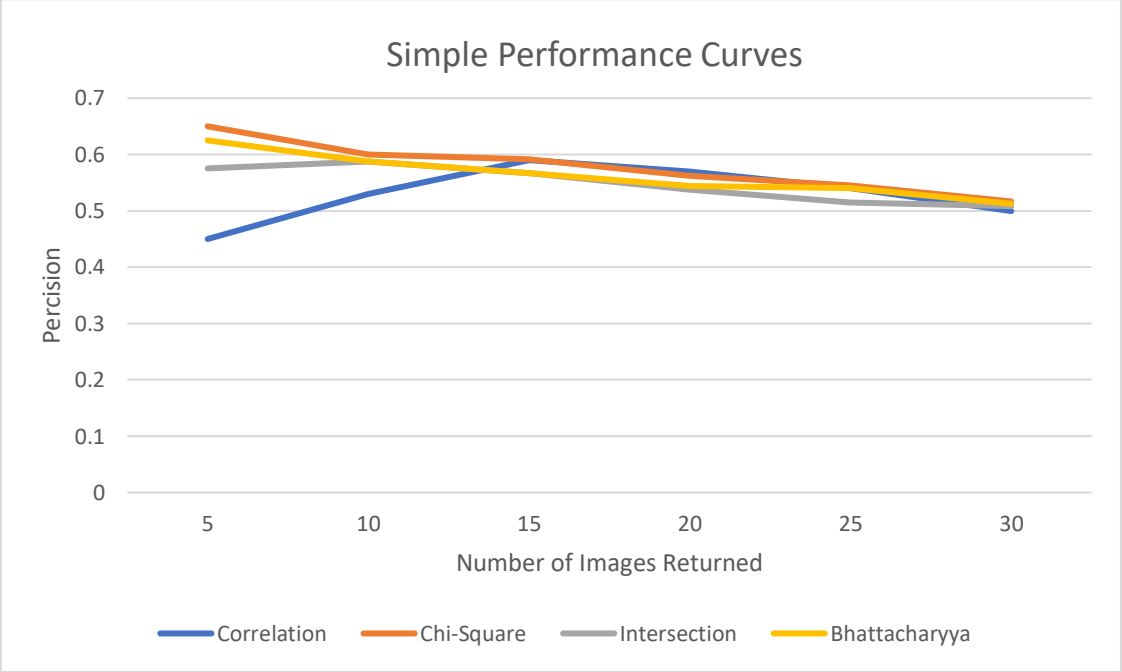

Fig. 6.4 Simple Performance Curves

These results are conclusive that the precision is highest when the number of images retuned is lowest (at the start) and then there is a slight downward trend in precision when the number of images returned is increased. Each method follows this trend except for Correlation, its precision starts unusually low in comparison to the other methods. From this we might suggest that Chi-Square and Bhattacharyya have the best performance as their corresponding values start high relative to the other models and finish with about the same precision as the other models (there is no definitively superior model according to this test when 30 images are returned). In addition, the graph suggests that as the number of images increases, the difference between the models decreases. This means that if the recall was increased even more, the difference between the models would be reduced so that they could be indistinguishable.

This theory is backed up by Fig 6.5 which shows the same results in terms of drop off in performance. The data for this curve can be found in Fig. 7.15 and 7.23. For these tests the gallery size was changed. The standard gallery has 345 images. To observe the changes, database was reduced to 138 images, 69 images, 32 images, and 15 images. 8 test images were used at each gallery size. 2 images from each of the image categories (Crowds, F1-Cars, Horses, and Landscapes). The plotted precision-recall is the average from these results at each gallery size. Moreover, 15 images were returned for each test and optimal bins for each distance measurement.

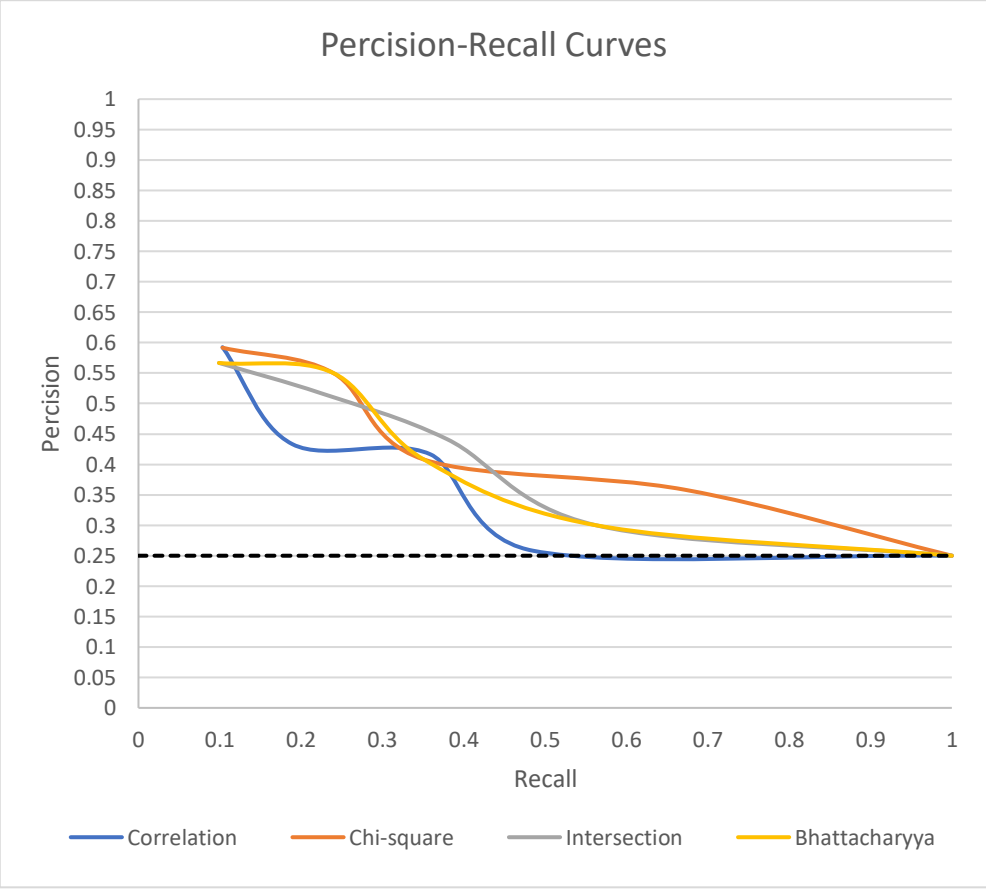

Fig. 6.5 Full Precision-Recall

Full Performance Averages

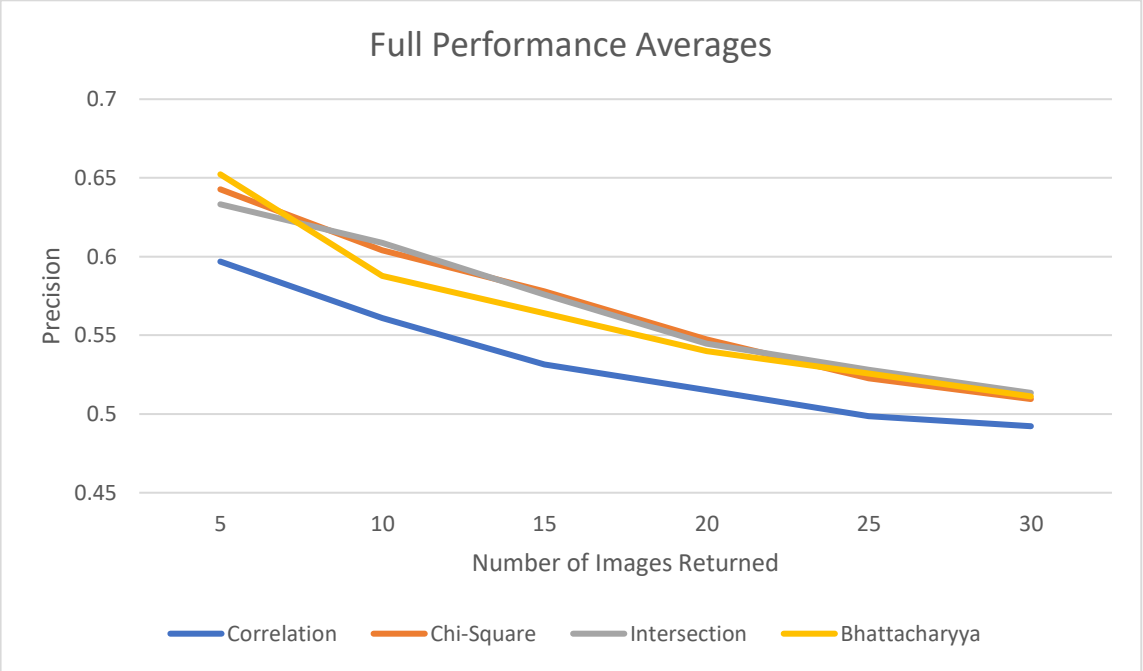

Fig. 6.6 Full Performance Averages

Recall	Correlation	Chi-Square	Intersection	Bhattacharyya
5	0.5968615	0.6427489	0.63322511	0.652272727
10	0.5610931	0.6041667	0.60865801	0.587770563
15	0.5314214	0.5779942	0.5757215	0.563924964
20	0.515395	0.5475108	0.54466991	0.539799784
25	0.4985498	0.5228571	0.52805195	0.525887446
30	0.4922619	0.5095238	0.51343795	0.511201299

Initially from Fig. 6.6 it is easy to determine that Correlation is the worst performing model as it has the lowest precision for each recall value. Therefore, this method will not be used in the final best performing model.

In terms of the other 3 methods, finding the best performing is more difficult to determine. A more in-depth study shows that Bhattacharyya has the best performance when the recall is 5. This is because it has a precision of 65.28% (2dp), and this is an increase of 0.96% (2dp) from Chi-Square which is the second-best performing method at this recall. This increase is very marginal, so it is important to observe how their performance changes as the number of images increases.

Bhattacharyya has a steeper decline in performance than Chi-Square and Intersection. Removing Correlation from the equation, Bhattacharyya went from the best performing with 5 images to the worst performing with 10 images, leaving Chi-Square and Intersection the two best performing. These positions remain consistent throughout the performance curve, although when 30 images are returned, the difference is very marginal between all methods and all methods perform poorly, with an average of 50.66% (2dp).

Furthermore, the drop-off rate is very similar for all methods. Fig. 6.7 below quantifies how the performance drops when the recall is increased. It is ideal to keep the precision decrease as low as possible.

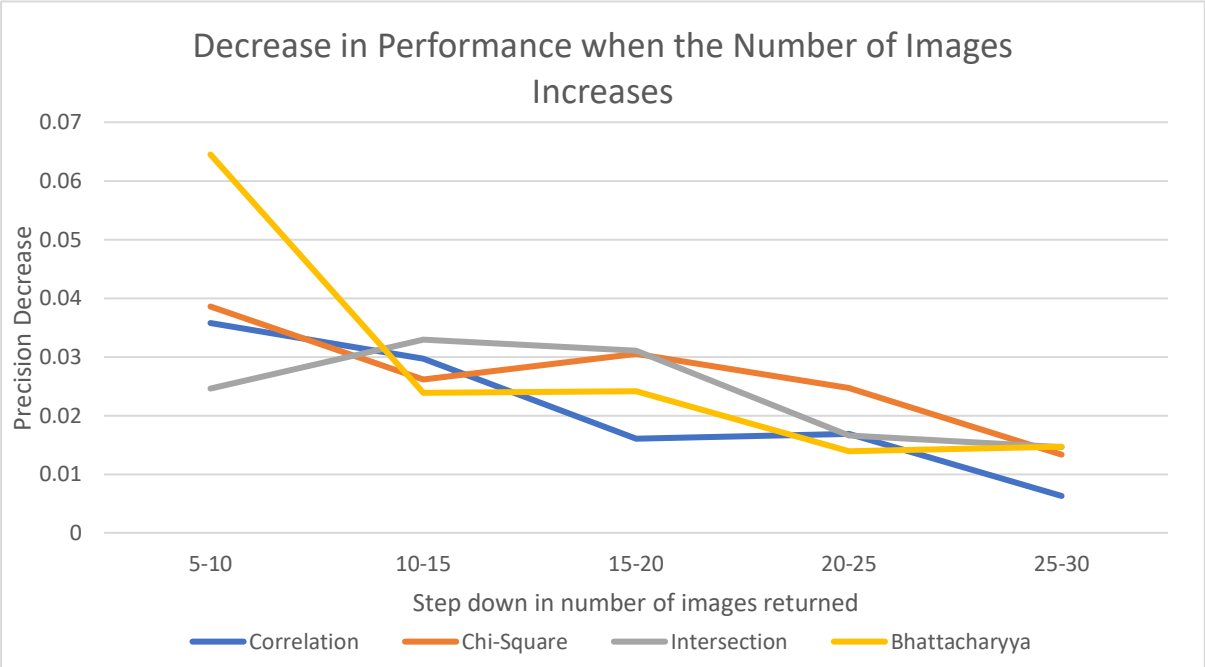

Fig. 6.7 Decrease in Performance

Recall Difference	Correlation	Chi-Square	Intersection	Bhattacharyya
5-10	0.035768398	0.038582251	0.0245671	0.064502165
10-15	0.029671717	0.026172439	0.032936508	0.023845599
15-20	0.016026335	0.030483405	0.031051587	0.02412518
20-25	0.016845238	0.02465368	0.016617965	0.013912338
25-30	0.006287879	0.013333333	0.014613997	0.014686147

From Fig. 6.7 the severe decrease for Bhattacharyya moving from 5 to 10 images returned is observable again. Moreover, based on this graph one could hypothesise that Correlation is the best performing model, as it overall decreases the least as the number of images returned is

increased, and could be viewed as the most consistent model. This is because the decrease from 15-20 is much lower than the other methods (0.016026335) and the decrease from 25-30 is much lower than the other methods (0.006287879). However, as previously discussed, this model will be discarded as its overall precision is definitively much lower than the other methods.

Looking back to the Full Performance Averages graph (Fig. 6.6), the results are conclusive that the precision is highest when the number of images returned is lowest (at the start) and then there is a slight downward trend in precision when the number of images returned is increased. This theory does not change from the simple model put forward earlier. However, what does change is the performance of Correlation. In this simple model, its performance (precision) starts low (45%) but then jumps up in line with the other methods and its performance is not as bad as it is in Fig. 6.6, because there its performance maintains a very low level throughout.

6.2 The Best Performing Method

Based on the results discussed above, the two contending methods are Chi-Square and Intersection. The table and graph below compare these models' side-by-side.

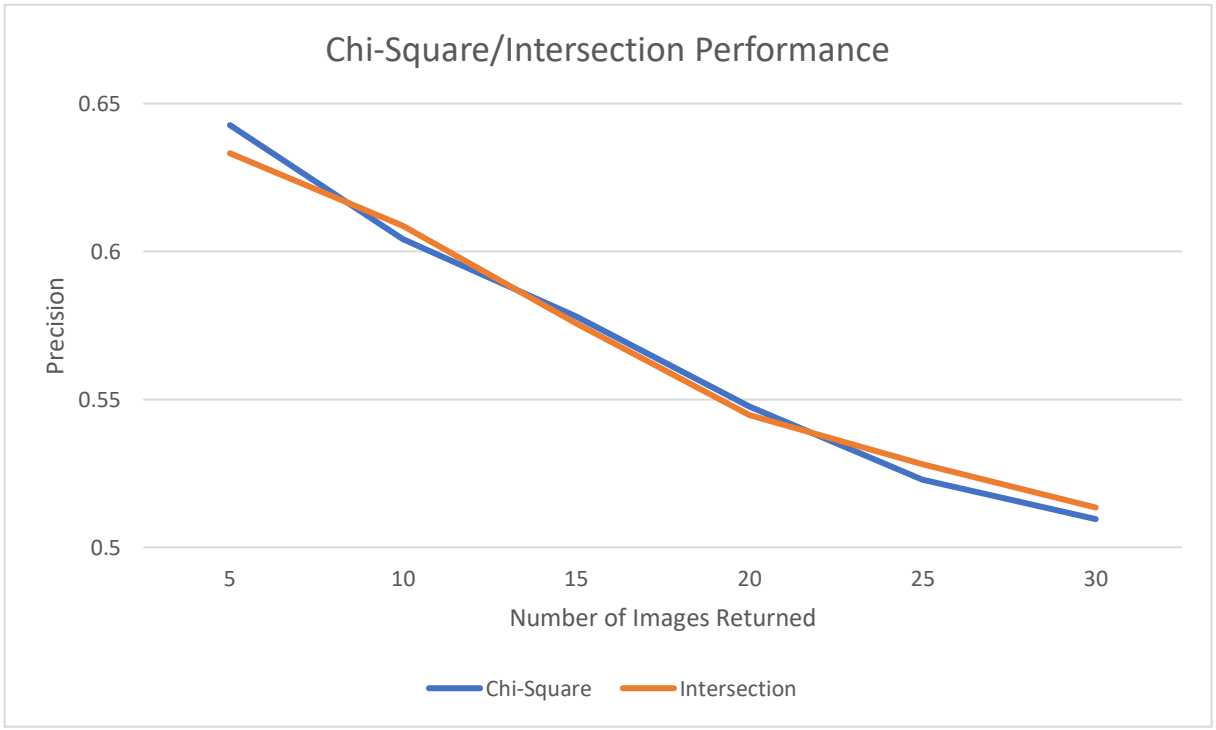

Fig. 6.8 Chi-Square/Intersection Performance

Recall	Chi-Square	Intersection
5	0.642748918	0.633225108
10	0.604166667	0.608658009
15	0.577994228	0.575721501
20	0.547510823	0.544669913
25	0.522857143	0.528051948
30	0.50952381	0.513437951

Fig. 6.8 shows that there is not a reasonable enough difference between these methods to suggest that one is better than the other. This is because the lines overlap for the most part and the results are both very consistent with each other throughout.

	Average Time (s)
Chi-Square	3.685116562
Intersection	2.397183574

Fig. 6.9 Full time averages

Fig. 7.26 and Fig. 7.27 shows the results for the basic tests which were calculated and used earlier for bin optimisation. Interestingly, you can see that with this small sample of 8 images, the average precision recorded in both models (2dp) are exactly the same (59.17%). However, there is initially a substantial difference in average processing time between the models. Intersection seems much more favourable as its processing time is only 2.39s (2dp), in comparison to Chi-Squares 3.57 (2dp). This is a 1.18s difference, which makes Intersection substantially faster.

As the tests illustrated in Fig. 7.27 and Fig. 7.27 only samples 8 images. This test will be expanded and the time for every test image for Chi-Square and Intersection will be recorded using 8 bins (optimal) and returning 15 images. This way the final results and conclusions will be more accurate. All of these test results can be seen in Fig. 7.25.

Fig. 6.9 above gives the averages from the results in Fig. 7.25. These results show the same trend that was determined from the simple model. This is that Chi-Square takes a longer time to process than Intersection. The time difference of 1.28793299s, shows that Intersection is the best performing method, and this will be used in the final model.

6.3 Database Read/Write Results

There are many reasons why data should be stored externally for this application.

This includes data backup and recovery [19]. External storage facilitates data backup and data recovery in case of a hardware failure, power outage, or any other unforeseen event that might cause data loss. Moreover, in terms of scalability, external storage can be easily scaled up or

down depending on the needs of the application [20]. As the database grows, more storage space can be added without having to worry about limitations or slowing down the system.

Also for this application improved performance is very important. By storing data externally, this will significantly improve the performance of the application. Keeping a database of image features, so that they do not need to be calculated at every run time will substantially improve the speed of this application as the bulk of the processing time is spent calculating the 3D HSV array from each of the images in the database.

The application has a button on the frontend that offers the user to update the database. This button calls a function that calculates all of the 3D HSV arrays and stores it externally. This means that for every run time the application will read the data from this file instead of calculating it every time. At this stage it is useful to try a range of different techniques on how to do this so that the application can adapt the best performing method. The best performing method should offer the fastest read/write speeds.

Initially, Microsoft SQL Server Management Studio was tested [21]. This meant creating a database and tables, using a server which connects to the Java search engine application. Doing this, the function for saving data would have used SQL statements to query the database and fetch all of the required data. The application would have connected to the database by using SQL Server Authentication when a user and login has been created for it.

Fig. 6.10 is a screenshot of Microsoft SQL Server Management Studio showing the server with the database named “SearchEngine” in the Object Explorer section.

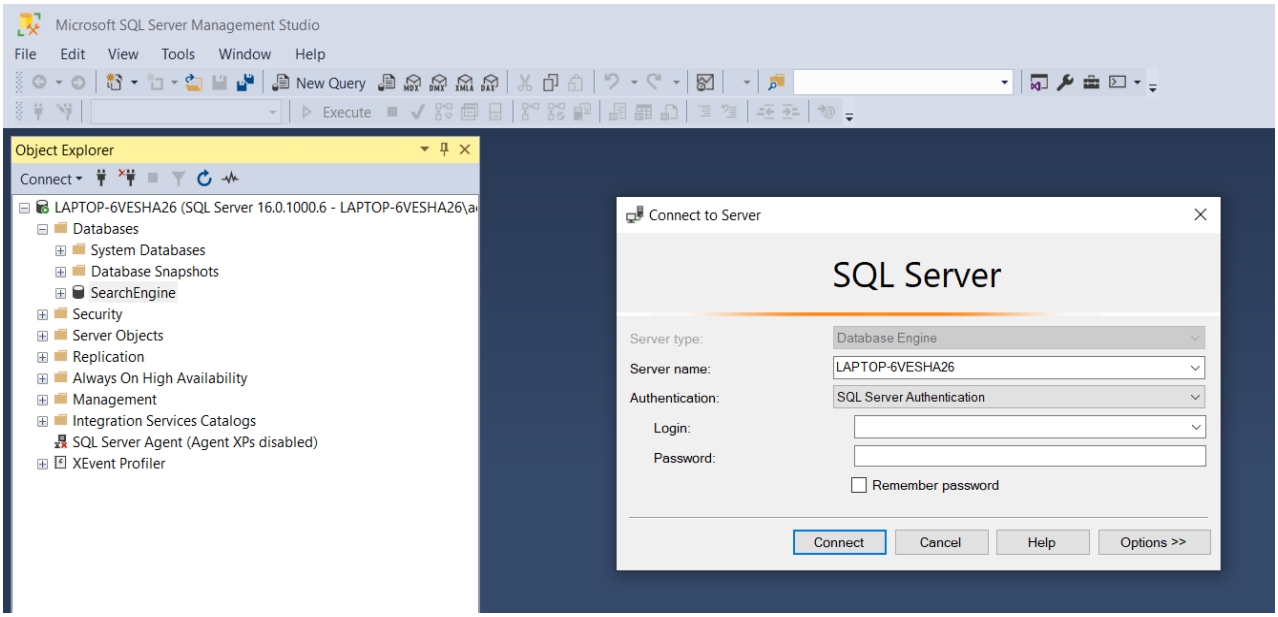

Fig. 6.10 Microsoft SQL Server Management Studio

Through further research, the current scale of this project using this storage method would be overkill and is not best suited to storing relatively small amounts of data [22], due to its high complexity. Other storage options then needed to be explored, .csv then .txt. These are text-based file formats that are human-readable and can be easily edited with a text editor. This can be useful if it is needed to modify the contents of the file manually, without running the application. With both of these methods the same issue arose which is that file took an incredibly long time to update. Having run the update function on both, their results are seen below in Fig. 6.11 and Fig. 6.12.

Fig. 6.11 .csv format

Run Number	Time Taken (mm:ss)
1	45:12
2	43:56
3	47:43
4	45:43

Fig. 6.12 .txt format

Run Number	Time Taken (mm:ss)
1	44.21
2	45.55
3	44.01

Fig. 6.12 and Fig. 6.13 show an average of 45min 38s for CSV file format, and an average of 44min 45s for .txt file format. Fig. 7.28 and Fig. 7.29 show example code snippets of how the application initially tried saving and reading data to a CSV file name “scores.csv” respectively. These methods take way too much time as this is not close to the target time of 20 seconds.

SER

More time-efficient methods are needed so SER and bin file format was chosen.

SER (Simple Extensible Rendering) is a binary file format that is designed for storing scientific data, including 3D arrays. It has a simple header format and supports compression, which can make it efficient for storing large datasets [23].

Expecting the performance to be much better than CSV and .txt file formats the image database, the 3D HSV array, and the image directory references were saved. The results from this are shown below in Fig. 6.13:

Update database (saving SER file)	43 minutes
Loading SER file	46 minutes

Fig. 6.13 SER initial times

It is the function time splits here that are most interesting. Saving the image database and the image directory references took milliseconds each but storing the 3D HSV histogram array took 99+% of the time (this includes calculations and converting the 3D “arraylist” into a serializable format). Due to these less than impressive results, the .bin file format was tested.

BIN

Bin is a generic binary file format that can be used to store any kind of binary data, including 3D arrays. It is often used when performance is a concern, as binary files can be read and written more quickly than text-based formats [23], and therefore would be suited to this application. The .bin file is being used to store only 3D HSV histogram array, as shown in the code listing in Fig. 7.30. This method shows how a new file “src//test.bin” is created and the array is looped through each dimension as it increments through the data writing each element to the file. It is important to note that “imageHistoMatrices” is a 3D float arraylist which contains the histograms for each image.

Loading the data using this method (Fig. 7.31) yielded much better performance results than methods trialled previously. Fig. 7.32 shows a set of results seeing how it performed over a normal range of histogram bins, and cross validated. The average times of those tests are displayed below in Fig. 6.14.

Num Bins	Time (s)
3	15.26439
4	16.97146
5	14.94729
6	16.23525
7	15.22056
8	16.15164
9	17.05334

Fig. 6.14 BIN times average

The data from Fig. 6.14 has been illustrated below in Fig. 6.15, and the trendline shows a slight increase in the processing time when the number of bins is increased. However, this increase is very small when increasing the bins by a small amount. These results are very good and a loading time of around 15 seconds shows a 186x improvement on the loading time from using the SER file format (46 minutes). The code snippet shown in Fig 7.31 shows how the data is loaded into the system from the function just covered. This uses the same strategy as to when the data was written to the file because it loops through each of the dimensions of

the array, iterating and storing the data in a new 3D float ArrayList. This is the final model that was implemented for the final image search engine.

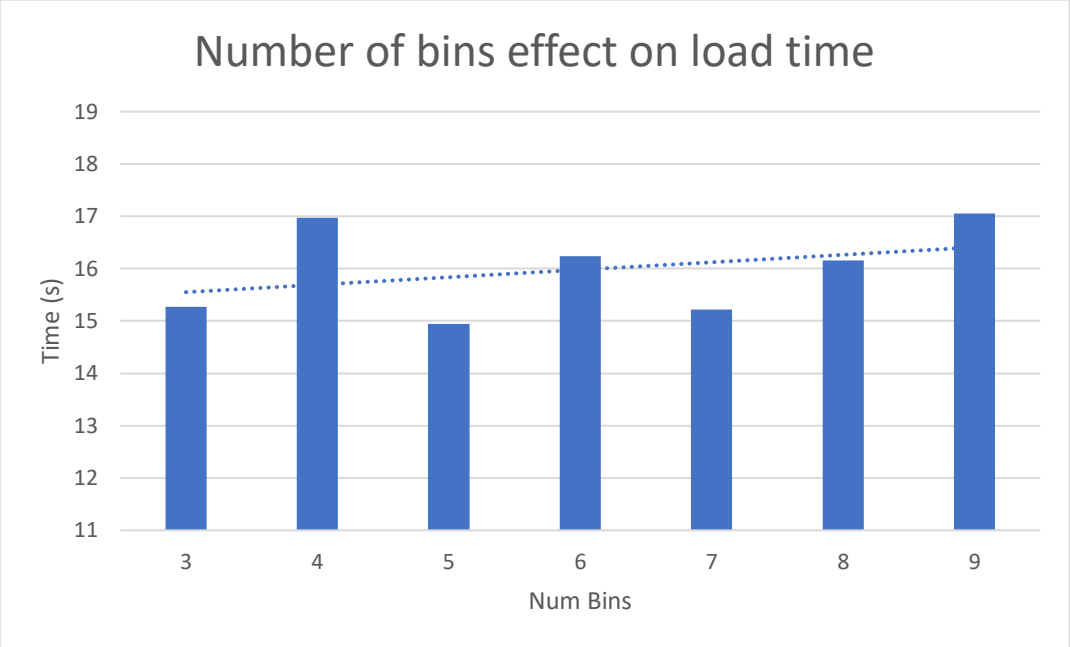

Fig. 6.15 Number of bins effect on load time

7.0 Possible improvements

Machine learning can be used to improve the performance and accuracy of the CBIR system. Using deep learning models to learn feature representations directly from images, rather than handcrafted features for colour, texture, and shape, is one approach. This is because they can capture more complex and abstract patterns and relationships within images and these learned features may be more effective for image retrieval. A convolutional neural network (CNN) is one method that has been shown to achieve cutting-edge results in a variety of computer vision tasks, including image classification and retrieval. The model can learn to extract discriminative features that are relevant for image retrieval by training a CNN on the gallery images [24]. These features can then be used to compare the query image to the gallery images and retrieve the most similar.

Multithreading can also be used to improve the performance of the image search engine by allowing multiple tasks to be executed simultaneously. This could be used to parallelize the image loading process. The application needs to load all the image features into memory. This can be a time-consuming process, especially when there are a lot of images. By using multithreading, the image loading process can be split into multiple tasks that run concurrently,

which can significantly reduce the overall loading time [25]. Moreover, batch processing can be used to perform image feature extraction and indexing in batches, instead of processing each image individually. For example, instead of extracting features and indexing each image one at a time, a batch of images can be processed together, which can be more efficient in terms of processing time and resource utilization [26].

Cloud storage can significantly improve system accessibility and security. The image database can be stored remotely on servers accessible from anywhere with an internet connection using cloud storage, making it easier for multiple users to access and use the system. Furthermore, cloud storage providers typically employ robust security measures such as encryption, access controls, and backups, which can help protect the image database from unauthorised access, data loss, and other security threats. To use cloud storage, the application must be modified to integrate with the API of a cloud storage provider, such as Amazon S3 or Google Cloud Storage. This would entail incorporating code into the application to handle image uploading and downloading to and from the cloud storage provider, as well as managing access to the image database via authentication and authorization mechanisms [27].

Moreover, different similarity methods could be tested which might outperform Correlation, Chi-Square, Intersection and Bhattacharyya. Methods tested in this paper [28], include City Block distance, Canberra Distance, Sum of absolute difference(SAD), Minkowski distance, and Sum of squared absolute Difference(SSAD), as these have not been tested in this application, testing them would mean that there is a chance that the best performing model could change and the precision increase.

Separately, more functionalities could be added to the frontend to improve the user experience. A dropdown list could be added for the user to select how many similar images they want returned. This could range from 1-30 and would mean that the frontend isn't fixed to returning 15 images. Moreover, the UI could include a series of radio buttons that represent each of the programmed distance measurements. This would mean that based on the current system the user would have ability the to choose if they want the results to be based on Correlation, Chi-Square, Intersection, or Bhattacharyya.

8.0 System Evaluation and Experimental Results

8.1 Final model

The final distance measurement chosen is Intersection. Background information about this method can be found in Section 3.5. Intersection was chosen because of its superior performance to the other tested distance measurements. Intersections performance was most similar to the performance of Chi-Square. This is because based on the tests performed, Intersections precision is non-distinguishable to Chi-Square. Both methods recorded a precision of 59.17% (2dp) when using their optimal bins over a test of using 8 test images from the test image database. However, Intersection runs more efficiently as it has a lower processing time. Chi-Square has an average search time of 3.69 (2dp), while Intersection has an average search time of 2.40 (2dp). These times are taken from the average search processing time for every image in the test image database, where the optimal number of bins are used for each method (8 bins). Due to the large test size, these results are conclusive that Intersection performs better than Chi-Square. Chi-Square and Intersections superiority over Correlation and Bhattacharya is explored in Section 6.1.

The final data storage technique is binary file format (.bin). This was explored in Section 6.3. This model has an average of 16.15s (2dp) for update time using 8 histogram bins. This is substantially better than the other methods explored, which include .txt, .csv and .ser.

8.2 Matching requirements

A user-friendly UI was created that allows users to easily navigate and interact with the application. This allows the user to select the query image, update the database, display the results of the search, and provides information about any errors or issues that arise during the search process. More details on the error handling can be seen in Section 3.6.

The chosen colour space and pixel representation works correctly as HSV, and colour histograms have been implemented successfully for comparing image features.

The application allows for query image selection and resizing. When the user selects an image it is resized to 512x512 on the UI. This is illustrated in Section 3.2. Additionally, image search and comparison has been implemented so that the distance measures of Correlation, Chi-Square, Intersection, and Bhattacharyya can rank images by what is most similar and display the top 15 most similar images to the user on the UI. Moreover, this application improves scalability by calculating features offline. These features are stored in a binary file format (.bin). Additionally, timers have been implemented, and based on the averages from the final model (Section 6.2), the image search function takes less than 5 seconds. Furthermore, the histogram bins for each distance measurement have successfully been optimised. This can be seen in more detail in Section 5. Furthermore, when the database has been updated the application generates

a pop-up message informing the user than the database has been updated (Section 3.6). Finally, as discussed in Section 8.1 the best distance measure (Intersection) has been calculated from precision-recall curves. In addition, the best data storage technique (.bin) has also been calculated from timing each of the different methods.

8.3 Comparison

The calculation models covered in this paper [29], are very similar to the best performing model developed in this project. There are 4 models being compared here, with M.E. ElAlami's model showing the best performance (blue line).

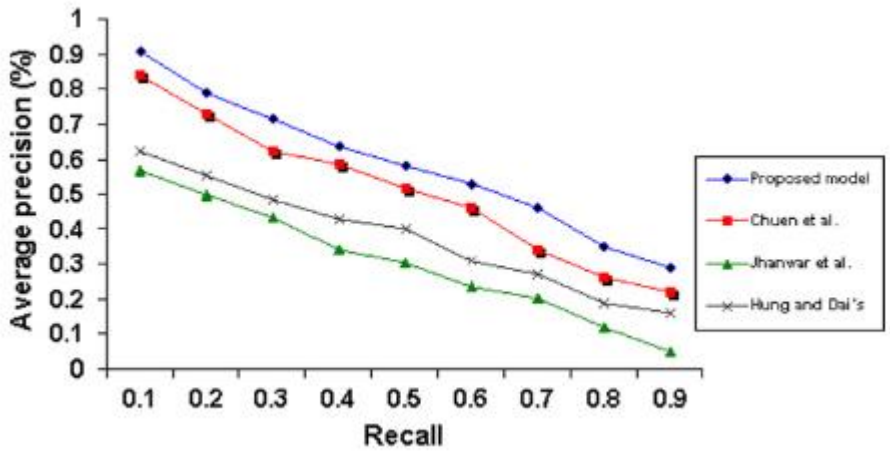

Fig. 8.1 Comparing 4 different models [29]

When comparing the precision-recall curve of the Intersection model illustrated in Fig. 5.17, to these 4 models. It is clear that model represented by "Jhanwar et al." is the most similar to it as precision starts highest at 0.55 for both models and the precision gradually decreases as recall increases. Understanding that the system developed in this project can match the performance of that model is very good as it shows this model can have a real purpose in industry moving forward. If more time was allocated to this project then the additional improvements mentioned in Section 7.0 could be tested and implemented to give a boost in performance. Therefore, the model would likely produce an improved precision-recall curve, more similar to M.E. ElAlami's model [29]. As the model stands, it would have limited real-world application, due to its low precision. However, with the improvements made this could have significant societal and commercial implications. As mentioned in Section 1.0, this CBIR system could be used in the field of medicine, as it could be used for the classification of different types of tumours in MRI images [1]. An improved model here could help medical professionals diagnose the illness

quicker and easier and save more lives as a result. Moreover, in video surveillance, this could be utilized for the recognition of human activities and facial recognition [2]. Economically, this could lead to reduced healthcare costs because catching illnesses before they become more serious would mean less extensive treatments. Furthermore, improving surveillance would help improve security and reduce the risk of criminal activity. This can have a positive economic impact on property values and business activity.

8.4 User feedback

A questionnaire was given out to 5 sample end-users who exhibit the user characteristics mentioned in Section 2.1. The user feedback forms can be found in Fig. 7.33 – Fig. 7.37. The results from these are conclusive that the users are overall very pleased with the software provided. However, some would like the database to update faster and for the results to be more precise. Moreover, they are very satisfied with the number of images returned and the error handling.

8.5 Summary

Overall, this system has been proved to be moderately successful as it has an acceptable performance. This is based on the precision results from the best performing model (Intersection method using 8 bins), and the user feedback. With more time and resources the system could be significantly improved and could be given a real-world application. However, in its current state this does not seem feasible. Moreover, the system implemented has fulfilled the initial requirements, and offers scalability through external storage of image feature data.

9.0 Appendices

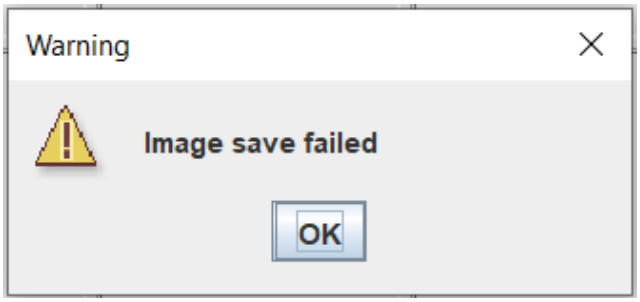

Fig. 7.1 Save Data Failed

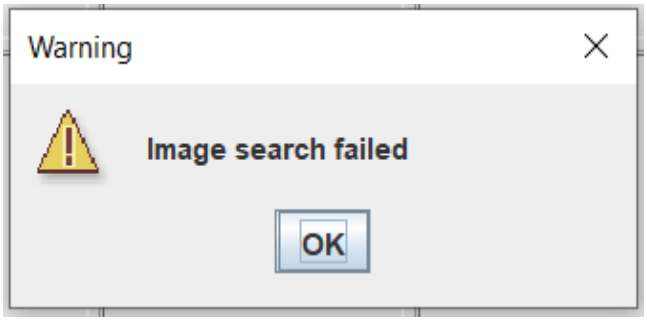

Fig. 7.2 Image Search

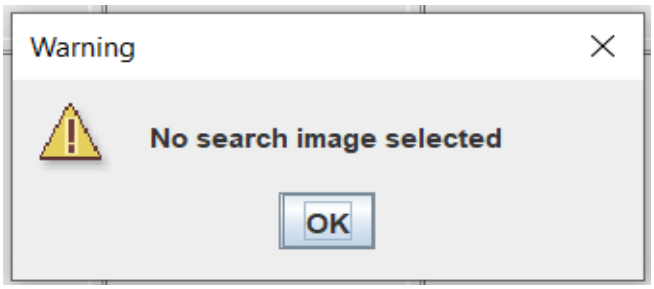

Fig. 7.3 No Image Selected

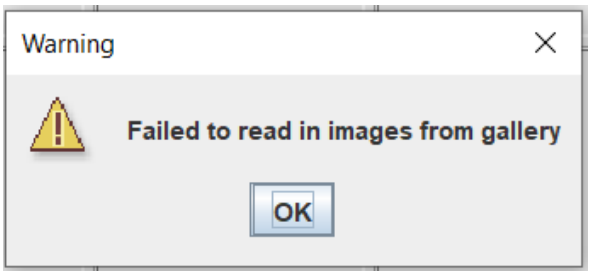

Fig. 7.4 Failed to Read Images (save)

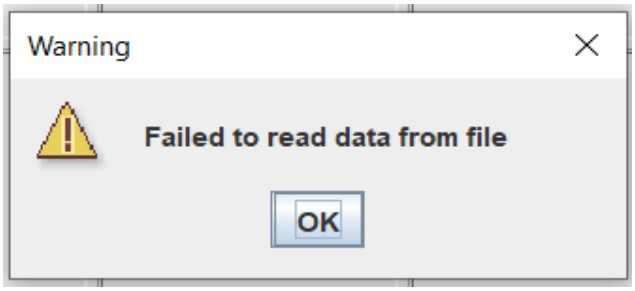

Fig. 7.5 Failed to read images (read)

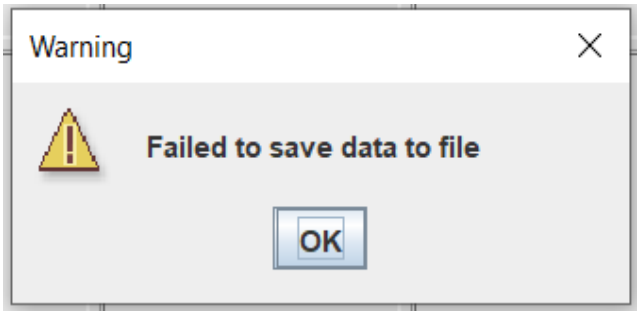

Fig. 7.6 Failed to Save Images

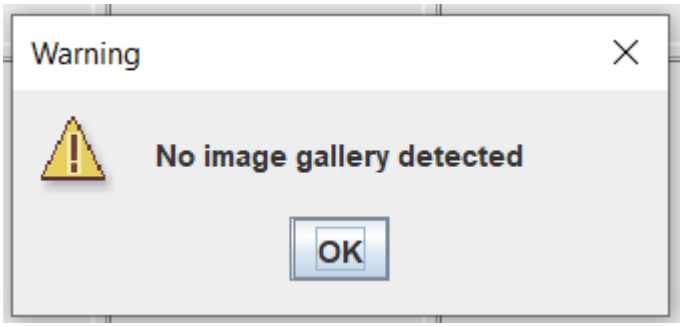

Fig. 7.7 No Image Gallery

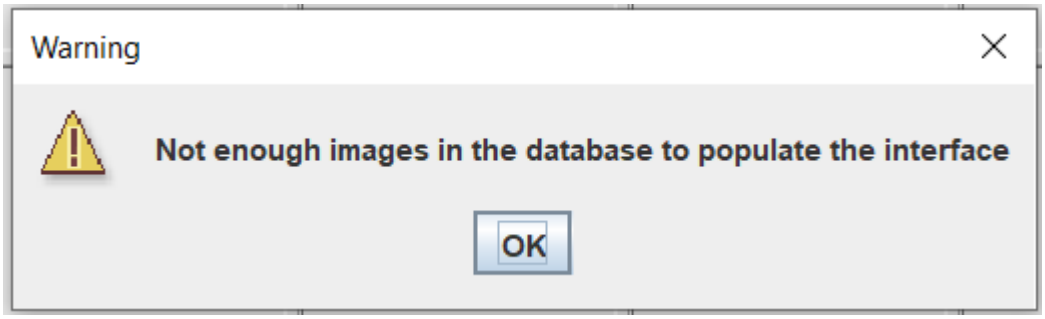

Fig. 7.8 Not enough Images to populate the UI

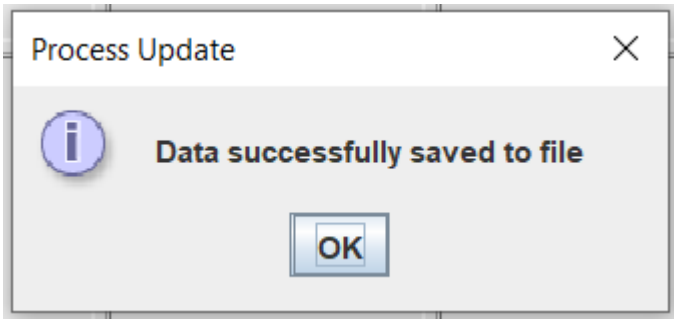

Fig. 7.9 Update database

```
public static double correlation(float[][][] histBase, float[][][] histTest, int bins) {
    int rows = histBase.length;
    int cols = histBase[0].length;
    double sum1 = 0, sum2 = 0, sum3 = 0;
    double mean1 = 0, mean2 = 0;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            for (int k = 0; k < bins; k++) {
                mean1 += histBase[i][j][k];
                mean2 += histTest[i][j][k];
            }
        }
    }
    mean1 /= (rows * cols * bins);
    mean2 /= (rows * cols * bins);
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            for (int k = 0; k < bins; k++) {
                sum1 += (histBase[i][j][k] - mean1) * (histTest[i][j][k] - mean2);
                sum2 += Math.pow(histBase[i][j][k] - mean1, 2);
                sum3 += Math.pow(histTest[i][j][k] - mean2, 2);
            }
        }
    }
    return sum1 / Math.sqrt(sum2 * sum3);
}
```

Fig. 7.10 Correlation implementation

```
public static double chisquare(float[][][] histBase, float[][][] histTest, int bins) {
    double distance = 0;
    int hBins = histBase.length;
    int sBins = histBase[0].length;
    int vBins = histBase[0][0].length;
    for (int h = 0; h < hBins; h++) {
        for (int s = 0; s < sBins; s++) {
            for (int v = 0; v < vBins; v++) {
                double delta = histBase[h][s][v] - histTest[h][s][v];
                double sum = histBase[h][s][v] + histTest[h][s][v];
                if (sum > 0) {
                    distance += delta * delta / sum;
                }
            }
        }
    }
    return distance;
}
```

Fig. 7.11 Chi-Square implementation

```java
public static double intersection(float[][][] histBase, float[][][] histTest, int bins) {
    double intersection = 0;
    int hBins = histBase.length;
    int sBins = histBase[0].length;
    int vBins = histBase[0][0].length;
    for (int h = 0; h < hBins; h++) {
        for (int s = 0; s < sBins; s++) {
            for (int v = 0; v < vBins; v++) {
                intersection += Math.min(histBase[h][s][v], histTest[h][s][v]);
            }
        }
    }
    return intersection;
}
```

Fig. 7.12 Intersection implementation

```java
public static double bhattacharyya(float[][][] histBase, float[][][] histTest, int bins) {
    int rows = histBase.length;
    int cols = histBase[0].length;
    double sum = 0;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            for (int k = 0; k < bins; k++) {
                sum += Math.sqrt(histBase[i][j][k] * histTest[i][j][k]);
            }
        }
    }
    return Math.sqrt(1.0 - sum);
}
```

Fig. 7.13 Bhattacharyya implementation

0.84262344	0.824987434	0.81808332	0.801316703	0.77886204	0.764766727
0.736505927	0.719128692	0.688752314	0.672866897	0.671596902	0.65792797
0.656194644	0.654065127	0.638439513	0.631933075	0.628635661	0.598085047
0.587282513	0.58497173	0.570186753	0.566773157	0.565083048	0.540236183
0.526792272	0.521123343	0.502202935	0.476558587	0.476233572	0.472733289
0.471328041	0.46329864	0.458641672	0.45172543	0.450857056	0.449302965
0.449288812	0.448641871	0.446847079	0.438695024	0.423845276	0.402824346
0.392370555	0.382748833	0.381907916	0.380610703	0.374010195	0.365457501
0.362828153	0.35069428	0.342188379	0.337385222	0.337055568	0.330638523
0.329698592	0.32122675	0.318685518	0.317421815	0.315458732	0.315239366
0.309251386	0.308938355	0.308116762	0.302753944	0.301709561	0.300503704
0.289443456	0.288071181	0.286402738	0.283088251	0.280907469	0.269881831
0.267624255	0.266247765	0.265313802	0.264050294	0.264031792	0.255622495

0.249599115	0.248926477	0.24798834	0.246663108	0.242610199	0.237754996
0.231667315	0.228594344	0.228295256	0.22546458	0.222793611	0.222457817
0.221565658	0.220433594	0.210681811	0.209172044	0.208019091	0.206556288
0.199620017	0.199066401	0.198993215	0.192004676	0.187930091	0.186774765
0.18496253	0.183025781	0.182839281	0.181101801	0.179172399	0.175484334
0.174197568	0.171743801	0.170250849	0.168969315	0.1658904	0.163341378
0.162540758	0.157463242	0.154804362	0.15405846	0.153280946	0.152907262
0.151123101	0.149556672	0.149401706	0.14840628	0.147234933	0.145445699
0.145339196	0.145166165	0.143332828	0.141796777	0.140637069	0.140476242
0.140037075	0.14000401	0.139949272	0.13958114	0.137702433	0.137261534
0.136431326	0.135388863	0.13423435	0.131156478	0.130289927	0.129886344
0.127251238	0.126665459	0.12633979	0.126263799	0.124933834	0.12447742
0.123768074	0.123523653	0.12254895	0.119636786	0.119513821	0.119224944
0.119188814	0.116712533	0.116709551	0.116127171	0.115875846	0.11532518
0.114747784	0.114639722	0.114110961	0.11107077	0.110073825	0.108723136
0.10699626	0.105742922	0.104897557	0.104765208	0.102585779	0.101378501
0.100159244	0.098406057	0.09830978	0.09632434	0.096015695	0.095703153
0.095628246	0.095202184	0.093263907	0.092493167	0.091849458	0.091775367
0.091520235	0.091475985	0.088055172	0.08693757	0.084885737	0.083861223
0.082152887	0.080396442	0.080188451	0.07931637	0.078924189	0.07860063
0.078550155	0.075311426	0.074814067	0.072878409	0.070981542	0.070981136
0.068713817	0.068693565	0.067904053	0.067104779	0.066898646	0.066489554
0.0642638	0.06340873	0.062109305	0.059741183	0.05649781	0.056039862
0.055958206	0.055205092	0.054774101	0.054179942	0.053676489	0.051312904
0.051143917	0.050553687	0.050144938	0.050073118	0.048164584	0.04773922
0.047369396	0.047173639	0.045783837	0.044684247	0.044248082	0.044218808
0.043902328	0.043390092	0.043139348	0.042875828	0.0424868	0.042144341
0.041898207	0.04070462	0.040139586	0.039255739	0.038519027	0.038502692
0.037504892	0.03741361	0.037266846	0.036652401	0.035968057	0.035887568
0.035205474	0.034685187	0.034158993	0.0336975	0.033518572	0.033433097
0.033409909	0.033326976	0.03242287	0.032189744	0.032170598	0.032105934
0.030667795	0.030422539	0.029737893	0.028702833	0.028597086	0.028595995
0.028575037	0.028078629	0.027252646	0.026735551	0.02566026	0.025198552

0.024652512	0.023797735	0.02284173	0.022343716	0.021610981	0.020506262
0.019808231	0.019695633	0.019343366	0.017696677	0.017189186	0.016679305
0.016385607	0.01631217	0.01601723	0.015656283	0.014591648	0.013717717
0.013205954	0.011280957	0.011013653	0.01043529	0.010431729	0.009815149
0.009468652	0.009221347	0.009038603	0.008645476	0.008478442	0.00765197
0.006446356	0.001659077	0.001608657	0.00136622	0.001337174	0.001235642
-0.000933795	-0.001836113	-0.001913736	-0.002235978	-0.002741722	-0.003199447
-0.003261053	-0.004077037	-0.005852751	-0.005881474	-0.007271949	-0.008325626
-0.008536279	-0.009735119	-0.010679274	-0.011420862	-0.014354543	-0.017020388
-0.019184284	-0.02038066	-0.020798229	-0.023327716	-0.023475476	-0.026053736
-0.03029712	-0.030748769	-0.031935674	-0.032337354	-0.032792033	-0.033142769
-0.034440389	-0.034771998	-0.040585627			

Fig. 7.14 Correlation Results

Correlation		**Chi-Square**		**Intersection**		**Bhattacharyya**	
Precision	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**
0.5925	0.1033454	0.59166667	0.103327922	0.566667	0.098934	0.566667	0.098916
0.433333	0.1890756	0.55	0.23960084	0.516667	0.225	0.55	0.239706
0.416667	0.3586601	0.40833333	0.350081699	0.441667	0.379085	0.408333	0.35049
0.258333	0.484375	0.35833333	0.671875	0.3	0.5625	0.3	0.5625
0.25	1	0.25	1	0.25	1	0.25	1

Fig. 7.15 8 bin precision-recall data

Images (.jpg)	**Num of Images Returned**					
	5	***10***	***15***	***20***	***25***	***30***
	Num of Correct Images					
Crowds001	5	10	15	20	25	28
Crowds002	5	10	15	19	22	26
Crowds003	5	10	14	18	20	23
Crowds004	0	1	1	1	3	4

Crowds005	5	9	13	18	21	25
Crowds006	3	7	9	12	16	17
Crowds007	4	7	11	15	18	20
Crowds008	2	3	6	7	10	14
Crowds009	5	9	13	16	21	26
Crowds010	5	10	15	18	22	25
Crowds011	5	10	15	20	25	30
Crowds012	5	10	14	19	22	26
Crowds013	4	6	9	11	13	14
Crowds014	5	10	12	16	19	22
Crowds015	4	6	8	8	11	13
Crowds016	5	9	13	16	21	24
Crowds017	5	10	12	17	22	25
Crowds018	5	10	15	18	22	26
Crowds019	5	10	15	20	24	27
Crowds020	5	10	15	18	22	25
Crowds021	1	3	4	7	10	11
Crowds022	2	3	6	8	12	15
Average Precision	0.818	0.786364	0.757576	0.731818	0.729091	0.706061
F1-Cars001	0	1	1	1	1	1
F1-Cars002	0	1	1	1	2	3
F1-Cars003	2	4	5	5	8	9
F1-Cars004	1	3	4	5	6	7
F1-Cars005	1	2	3	6	6	7
F1-Cars006	0	0	0	2	3	4
F1-Cars007	4	4	4	7	9	10
F1-Cars008	1	2	4	5	5	5
F1-Cars009	0	1	1	2	3	5
F1-Cars010	0	2	5	6	8	10
F1-Cars011	2	5	9	11	12	13
F1-Cars012	3	4	8	11	13	17
F1-Cars013	1	1	3	3	4	5

F1-Cars014	1	2	6	7	8	9
F1-Cars015	1	1	1	1	1	1
F1-Cars016	4	5	6	8	9	9
F1-Cars017	1	2	2	2	4	5
F1-Cars018	5	10	14	17	19	23
F1-Cars019	1	3	7	7	8	11
F1-Cars020	0	0	0	0	0	1
F1-Cars021	3	6	9	11	14	16
Average Precision	0.295	0.280952	0.295238	0.280952	0.272381	0.271429
Horses001	2	6	10	13	13	15
Horses002	5	8	10	13	15	18
Horses003	4	7	9	11	14	15
Horses004	5	8	10	13	15	19
Horses005	3	6	9	13	15	16
Horses006	3	7	8	11	14	15
Horses007	5	9	14	19	22	23
Horses008	0	0	1	3	4	7
Horses009	3	5	6	8	10	13
Horses010	1	4	8	11	14	15
Horses011	4	6	9	12	15	19
Horses012	4	4	5	5	7	9
Horses013	0	1	2	3	5	6
Horses014	2	2	3	4	4	4
Horses015	3	5	8	13	16	18
Horses016	4	8	10	12	16	21
Horses017	5	8	12	14	18	22
Horses018	4	8	8	9	11	13
Horses019	2	5	6	10	13	17
Horses020	2	5	5	5	6	8
Horses021	5	8	12	13	17	20
Horses022	5	8	12	16	19	24
Average Precision	0.645	0.581818	0.536364	0.525	0.514545	0.510606

Landscapes001	4	9	12	16	18	19
Landscapes002	3	5	8	10	13	13
Landscapes003	4	9	11	14	14	15
Landscapes004	0	1	1	1	2	4
Landscapes005	3	5	7	9	12	16
Landscapes006	3	4	8	10	13	14
Landscapes007	3	3	3	4	6	6
Landscapes008	4	7	11	14	15	16
Landscapes009	0	3	3	6	6	8
Landscapes010	1	2	3	4	5	7
Landscapes011	4	7	10	14	18	20
Landscapes012	5	9	14	15	16	17
Landscapes013	3	5	6	6	7	9
Landscapes014	5	8	8	11	13	15
Landscapes015	4	8	12	16	21	23
Landscapes016	1	1	1	3	3	6
Landscapes017	3	6	9	12	16	19
Landscapes018	4	8	9	12	13	16
Landscapes019	3	7	9	12	16	21
Landscapes020	4	9	13	17	20	20
Landscapes021	5	9	11	14	16	19
Average Precision	0.629	0.595238	0.536508	0.52381	0.478182	0.480952

Fig. 7.16 Correlation Full Results

0.147198554	0.148594692	0.154034696	0.15676748	0.156990004	0.160475193
0.163228425	0.170389943	0.173957717	0.179043956	0.183769368	0.18728503
0.187364101	0.189503343	0.191980469	0.192485391	0.192964541	0.193424767
0.198962218	0.200548438	0.201671132	0.202986987	0.205520833	0.205894157
0.206728202	0.209259672	0.209501844	0.210236515	0.220428036	0.221153977
0.227993504	0.229566833	0.235518263	0.238827594	0.240618824	0.241889854

0.244555775	0.246915922	0.248578455	0.251529748	0.251601145	0.251691365
0.252254805	0.25576021	0.256159329	0.256890262	0.258346016	0.260196771
0.261466089	0.264023391	0.269366621	0.269379682	0.272756656	0.277390068
0.27882715	0.27901263	0.279944276	0.28171997	0.285461433	0.286976624
0.287083259	0.290863107	0.290945523	0.292846528	0.292868846	0.29373474
0.29477718	0.296741335	0.29987821	0.300380351	0.30169585	0.301778753
0.304940278	0.306141995	0.306143778	0.306222255	0.311621994	0.311966489
0.312694189	0.315976434	0.316862236	0.318233521	0.318957749	0.319008459
0.319187494	0.319334276	0.321783104	0.323335384	0.323535688	0.323976261
0.324456793	0.324834668	0.327430047	0.327824992	0.329190465	0.329757967
0.32993849	0.330101342	0.33086527	0.334073989	0.334696681	0.339299653
0.340177426	0.340393811	0.341662444	0.342034363	0.34240059	0.343054236
0.343186517	0.343478596	0.343652331	0.344731832	0.345486604	0.346827513
0.347088125	0.349099368	0.349425668	0.350468191	0.350894413	0.351407616
0.353622295	0.354616093	0.355412435	0.355974436	0.35707384	0.357793465
0.359227887	0.359496024	0.360818397	0.363151829	0.364007433	0.365356194
0.366522637	0.369046086	0.36968413	0.370525554	0.370895791	0.371067711
0.371203567	0.372569628	0.373951387	0.373996789	0.374436294	0.374567518
0.376094224	0.376900592	0.376912862	0.377646418	0.377736449	0.378456369
0.378544079	0.378578512	0.380432575	0.380685402	0.381564052	0.383005798
0.386281174	0.386557058	0.386612682	0.391252474	0.391898548	0.392305591
0.392453865	0.392941526	0.393179178	0.394860585	0.395384654	0.396196145
0.397063315	0.39752171	0.401195352	0.402726154	0.403142986	0.403156236
0.405195943	0.40529929	0.405316478	0.408809945	0.409982993	0.410916171
0.411680801	0.411730744	0.412314148	0.413138678	0.413148394	0.413672697
0.413717548	0.414128227	0.414311554	0.416209874	0.41715881	0.419171853
0.419382611	0.421413074	0.421767184	0.422159688	0.422444845	0.423755742
0.42533172	0.425671694	0.42690941	0.427321835	0.428445815	0.428774565
0.42888076	0.429408494	0.429580406	0.431026086	0.431354033	0.432242076
0.432326081	0.433014397	0.433662377	0.4342688	0.434275933	0.435677163
0.43603277	0.436331077	0.436380179	0.43673743	0.436807902	0.437329353
0.438035346	0.438398396	0.440347528	0.441816842	0.442383728	0.444798633
0.44486778	0.444880618	0.445024844	0.445657721	0.446254226	0.446900975

0.44746611	0.45043229	0.45052963	0.451517721	0.452278016	0.45266071
0.453124254	0.453329622	0.453676652	0.455250072	0.455653509	0.457157369
0.458005923	0.458164735	0.458400293	0.45942914	0.459505484	0.459682439
0.460185558	0.462658359	0.462745539	0.463035897	0.463200882	0.463431616
0.46430294	0.464712649	0.465036878	0.466062245	0.466711904	0.4667731
0.467118	0.468159288	0.469738509	0.47254708	0.474920949	0.475445658
0.475633674	0.47751989	0.477753676	0.477900742	0.477969112	0.478531231
0.478543925	0.478570914	0.478782753	0.479872407	0.480372822	0.481453714
0.48148805	0.48315875	0.48382811	0.48886258	0.488914963	0.488978048
0.489165647	0.489201363	0.489704693	0.491250899	0.491911881	0.492545143
0.494670697	0.495672017	0.496006515	0.496951566	0.498562612	0.498812269
0.498900503	0.498900719	0.499340836	0.501094223	0.503501856	0.504779864
0.504812668	0.505950194	0.506233791	0.506480916	0.507953826	0.508450214
0.508474754	0.509531263	0.509567685	0.510405248	0.512145268	0.51286646
0.514457898	0.51675439	0.520174843	0.521004758	0.526480502	0.527517748
0.528798537	0.531535443	0.536475032	0.536563908	0.537189437	0.539274281
0.540655308	0.541663914	0.541850002	0.546489787	0.558288064	0.560897167
0.562934947	0.564341754	0.565780552	0.567216507	0.576501257	0.576562763
0.579712104	0.597204841	0.633252156			

Fig. 7.17 Chi-Square Results

Images (.jpg)	**Num of Images Returned**					
	5	***10***	***15***	***20***	***25***	***30***
	Num of Correct Images					
Crowds001	5	10	15	20	24	28
Crowds002	5	10	15	20	24	29
Crowds003	5	9	13	18	21	24
Crowds004	5	10	14	19	23	28
Crowds005	5	10	15	19	23	26
Crowds006	3	8	12	14	17	19
Crowds007	5	9	14	18	20	23

Crowds008	5	8	13	17	19	22
Crowds009	5	10	15	19	24	29
Crowds010	5	10	15	19	24	28
Crowds011	5	10	15	19	24	29
Crowds012	5	10	14	19	24	28
Crowds013	5	9	13	17	19	22
Crowds014	5	10	14	19	23	26
Crowds015	5	8	12	16	18	19
Crowds016	5	10	14	19	24	28
Crowds017	5	10	14	19	24	29
Crowds018	5	10	15	20	25	29
Crowds019	5	10	15	20	25	29
Crowds020	5	10	15	19	22	26
Crowds021	5	9	13	17	19	22
Crowds022	5	9	14	17	20	23
Average Precision	*0.981818*	*0.95*	*0.936364*	*0.918182*	*0.883636*	*0.857576*
F1-Cars001	0	0	1	2	2	2
F1-Cars002	2	4	4	4	5	6
F1-Cars003	2	2	3	4	6	7
F1-Cars004	2	5	6	7	10	12
F1-Cars005	0	2	3	3	3	4
F1-Cars006	0	1	3	4	4	4
F1-Cars007	0	1	4	6	6	8
F1-Cars008	2	3	4	5	6	6
F1-Cars009	2	3	3	4	6	7
F1-Cars010	4	7	7	10	12	15
F1-Cars011	3	7	10	11	12	15
F1-Cars012	4	7	8	10	10	11
F1-Cars013	3	3	5	7	9	12
F1-Cars014	2	3	3	4	6	6
F1-Cars015	0	0	1	1	1	1
F1-Cars016	4	6	9	10	11	13

F1-Cars017	1	2	2	2	2	3
F1-Cars018	5	10	15	20	22	26
F1-Cars019	3	7	9	13	15	16
F1-Cars020	2	4	6	6	7	8
F1-Cars021	3	5	10	13	14	16
Average Precision	*0.419048*	*0.390476*	*0.368254*	*0.347619*	*0.321905*	*0.314286*
Horses001	4	8	11	14	16	17
Horses002	4	8	11	13	15	20
Horses003	5	9	13	15	17	19
Horses004	5	9	13	15	18	23
Horses005	4	7	9	13	17	19
Horses006	5	10	14	16	18	23
Horses007	5	10	14	16	17	19
Horses008	0	0	0	3	3	4
Horses009	2	5	7	9	10	13
Horses010	3	4	6	8	11	13
Horses011	5	9	12	13	18	22
Horses012	2	3	6	8	10	11
Horses013	0	3	3	4	4	5
Horses014	0	0	1	1	2	3
Horses015	3	7	11	14	17	19
Horses016	4	4	8	9	10	14
Horses017	4	9	11	15	19	21
Horses018	1	3	5	5	5	6
Horses019	3	7	9	12	16	18
Horses020	1	2	4	5	5	6
Horses021	4	6	10	12	15	16
Horses022	5	9	13	18	21	25
Average Precision	*0.627273*	*0.6*	*0.578788*	*0.540909*	*0.516364*	*0.509091*
Landscapes001	4	9	13	13	13	13
Landscapes002	2	4	4	4	6	9

Landscapes003	5	6	7	7	8	8
Landscapes004	0	0	0	1	2	3
Landscapes005	2	6	7	9	13	14
Landscapes006	2	4	5	6	8	9
Landscapes007	3	4	4	4	4	5
Landscapes008	2	4	7	8	10	12
Landscapes009	0	0	1	2	2	5
Landscapes010	2	2	5	6	8	8
Landscapes011	4	9	12	15	16	20
Landscapes012	5	8	8	9	11	14
Landscapes013	2	4	6	7	7	8
Landscapes014	3	3	4	4	5	6
Landscapes015	4	7	11	14	15	16
Landscapes016	0	0	2	3	3	4
Landscapes017	2	3	4	6	8	9
Landscapes018	4	6	6	8	11	14
Landscapes019	3	6	8	11	15	17
Landscapes020	4	6	9	12	15	16
Landscapes021	4	9	12	12	14	15
Average Precision	*0.542857*	*0.47619*	*0.428571*	*0.383333*	*0.369524*	*0.357143*

Fig. 7.18 Chi-Square Full Results

0.010299683	0.023466746	0.028322855	0.02897644	0.029860179	0.031131744
0.031205495	0.033870697	0.033912659	0.036047617	0.038022359	0.039609273
0.040112813	0.040566762	0.041027069	0.04162852	0.042077382	0.043102264
0.043543497	0.043621063	0.043641408	0.045459747	0.046005249	0.046731313
0.047590891	0.04909261	0.049922943	0.049978892	0.050331116	0.050412496
0.051836649	0.052346547	0.052436828	0.052687327	0.054059347	0.0542895
0.054506938	0.054601034	0.05480957	0.054948171	0.055019378	0.055233001
0.055437724	0.055768331	0.055781046	0.056056976	0.056191762	0.056435903
0.056512197	0.056625366	0.056714376	0.056971232	0.057239532	0.057657877
0.058231354	0.058795929	0.059131622	0.059505462	0.060001373	0.060198466

0.060451508	0.060559591	0.061439514	0.061481476	0.06161499	0.062104543
0.062526703	0.062689463	0.063284556	0.063423157	0.063721975	0.064423879
0.064636231	0.064856211	0.064879099	0.064966837	0.065724691	0.065865835
0.066037496	0.066799164	0.066936493	0.066958109	0.068042754	0.068180084
0.068401337	0.068843841	0.06887563	0.068976084	0.06935501	0.069653829
0.069746653	0.06982549	0.070134481	0.070217132	0.070435841	0.070486705
0.07070287	0.07089742	0.070983886	0.071153005	0.071371714	0.072448731
0.072731018	0.073066711	0.07314682	0.073397318	0.073654175	0.07423528
0.07452774	0.074709574	0.075088501	0.075215658	0.075228373	0.075578054
0.075785319	0.076110839	0.076148986	0.076501211	0.076615651	0.076978047
0.077419281	0.077925364	0.078062693	0.078235626	0.078479767	0.078564962
0.078699747	0.079938253	0.080111186	0.080214182	0.0802447	0.080266317
0.080403645	0.080457051	0.080557505	0.080640157	0.08070755	0.080949147
0.080979665	0.0811704	0.081563313	0.081573486	0.082027435	0.082191467
0.082688649	0.08275477	0.082883199	0.084273021	0.085268657	0.085489909
0.085534413	0.085586548	0.085840861	0.086125692	0.086716969	0.086744944
0.086772919	0.08690389	0.086907705	0.087284088	0.087861379	0.088064829
0.088134765	0.089071909	0.089731852	0.08986028	0.090601603	0.090602875
0.09082667	0.091147104	0.091546376	0.091810862	0.092351277	0.09250768
0.092894236	0.093199412	0.093420664	0.093611399	0.094445546	0.094484965
0.09464391	0.094969431	0.095637003	0.095938364	0.096539815	0.097722371
0.097886403	0.099203745	0.099259694	0.099268595	0.100280762	0.1003685
0.100397746	0.100963593	0.101006826	0.101062774	0.101107279	0.101648966
0.101815541	0.102390289	0.102750142	0.102884928	0.103696187	0.103713989
0.103900909	0.104230244	0.104911804	0.10543696	0.105806986	0.105809529
0.106380462	0.106433868	0.106685638	0.107116698	0.107643127	0.107856751
0.108690897	0.108720144	0.109279632	0.109348297	0.109450022	0.109545389
0.110123952	0.110459646	0.110984802	0.111003875	0.111846923	0.11251831
0.11267217	0.113623301	0.113883972	0.114058175	0.1142807	0.114354451
0.114496867	0.115328471	0.115918477	0.116841634	0.116970062	0.116984049
0.117158254	0.118770599	0.118905385	0.11900711	0.119116465	0.119326274
0.120855967	0.121081034	0.121720632	0.121772766	0.121836344	0.122450511
0.122624714	0.122714996	0.122891744	0.1236763	0.123685201	0.123797098

0.124365489	0.125334421	0.125765483	0.125862121	0.128223418	0.128369649
0.129936219	0.130208333	0.130554198	0.130869547	0.131444295	0.131896972
0.132125854	0.132441203	0.132471719	0.132663727	0.132704417	0.133363088
0.133926392	0.134848276	0.135735829	0.136330921	0.137914021	0.137989043
0.13831838	0.138483683	0.139575959	0.140280405	0.140892028	0.141058604
0.14136378	0.142042795	0.14206187	0.143361409	0.143992105	0.145366669
0.146737416	0.147335052	0.149004617	0.149426778	0.151124318	0.152222952
0.152286529	0.153355916	0.15380605	0.154257455	0.154490153	0.154588063
0.15618515	0.157449087	0.159510294	0.160049438	0.160162608	0.161191304
0.161674499	0.16676712	0.166983286	0.167794546	0.168028513	0.170899708
0.171440124	0.174892425	0.176076253	0.176120759	0.176222484	0.176315307
0.176320394	0.178910572	0.179880778	0.183583578	0.183715821	0.184140522
0.185376485	0.185646056	0.186153412	0.186553955	0.190457661	0.192638397
0.192774455	0.193851471	0.195364634	0.201718647	0.201817831	0.206745149
0.208287558	0.210079194	0.21096166			

Fig. 7.19 Intersection Results

Images (jpg)	**Num of Images Returned**					
	5	***10***	***15***	***20***	***25***	***30***
	Num of Correct Images					
Crowds001	5	10	15	20	24	29
Crowds002	5	10	15	20	24	26
Crowds003	5	9	13	18	21	25
Crowds004	5	10	15	20	23	26
Crowds005	5	10	15	20	23	27
Crowds006	3	8	11	13	17	20
Crowds007	5	10	13	15	19	21
Crowds008	4	7	12	14	17	19
Crowds009	5	10	15	19	24	28
Crowds010	5	10	15	19	24	28
Crowds011	5	10	15	20	24	29

Crowds012	5	9	14	19	24	28
Crowds013	5	9	13	16	19	23
Crowds014	5	10	14	17	22	27
Crowds015	5	8	12	15	17	19
Crowds016	5	10	14	19	23	27
Crowds017	5	10	14	19	24	29
Crowds18	5	10	15	20	24	29
Crowds019	5	10	15	20	24	28
Crowds020	5	10	15	19	22	25
Crowds021	5	9	13	16	18	21
Crowds022	5	9	14	18	19	22
Average Precision	0.972727	0.945455	0.930303	0.9	0.865455	0.842424
F1-Cars001	0	1	1	3	3	3
F1-Cars002	2	4	6	7	7	7
F1-Cars003	1	2	3	4	7	7
F1-Cars004	2	5	6	8	11	14
F1-Cars005	1	2	3	4	4	4
F1-Cars006	0	2	4	4	4	5
F1-Cars007	0	2	5	6	6	6
F1-Cars008	2	3	4	5	5	6
F1-Cars009	3	3	4	5	7	7
F1-Cars010	3	6	9	11	13	15
F1-Cars011	5	7	9	10	13	16
F1-Cars012	3	6	8	9	9	10
F1-Cars013	2	4	5	6	9	12
F1-Cars014	1	3	4	4	4	4
F1-Cars015	1	1	1	1	1	2
F1-Cars016	4	6	8	8	11	13
F1-Cars017	1	2	2	2	3	5
F1-Cars018	5	10	15	20	23	26
F1-Cars019	3	6	8	11	15	18
F1-Cars020	2	4	5	7	7	9

F1-Cars021	4	7	10	14	16	19
Average Precision	0.428571	0.409524	0.380952	0.354762	0.339048	0.330159
Horses001	4	8	11	13	15	18
Horses002	4	8	11	14	18	21
Horses003	5	9	13	15	18	20
Horses004	5	8	12	14	18	23
Horses005	4	7	9	13	16	18
Horses006	5	10	11	15	18	21
Horses007	5	10	12	16	18	20
Horses008	0	1	3	5	7	8
Horses009	2	5	8	10	11	14
Horses010	2	5	7	9	11	13
Horses011	4	9	12	14	19	23
Horses012	3	7	8	9	11	13
Horses013	1	3	4	4	6	6
Horses014	0	0	0	0	2	3
Horses015	3	8	10	13	18	19
Horses016	4	4	7	9	11	15
Horses017	4	8	11	16	19	21
Horses018	1	3	4	5	6	7
Horses019	3	6	8	12	14	17
Horses020	2	3	5	6	6	8
Horses021	4	7	10	11	13	16
Horses022	5	9	14	18	21	25
Average Precision	0.636364	0.627273	0.575758	0.547727	0.538182	0.528788
Landscapes001	4	9	10	10	12	13
Landscapes002	1	2	2	3	4	5
Landscapes003	4	6	6	6	7	8
Landscapes004	0	0	1	1	3	3
Landscapes005	3	6	8	10	11	13
Landscapes006	1	3	3	5	7	8

Landscapes007	3	3	4	4	5	6
Landscapes008	2	5	8	10	12	14
Landscapes009	0	0	2	2	3	3
Landscapes010	2	2	4	5	6	7
Landscapes011	4	8	12	16	19	20
Landscapes012	3	6	6	8	10	13
Landscapes013	2	2	3	5	7	7
Landscapes014	3	3	5	5	5	6
Landscapes015	4	8	12	13	15	16
Landscapes016	0	1	3	3	4	5
Landscapes017	2	3	4	5	6	9
Landscapes018	3	7	7	9	13	14
Landscapes019	3	5	8	12	14	18
Landscapes020	4	7	11	14	17	18
Landscapes021	4	9	12	12	14	16
Average Precision	0.495238	0.452381	0.415873	0.37619	0.369524	0.352381

Fig. 7.20 Intersection Full Results

0.844687746	0.84486907	0.845155693	0.84540345	0.846937671	0.848905089
0.849337502	0.849969806	0.850310239	0.850454253	0.8517765	0.852494668
0.853439933	0.85399739	0.854391166	0.85446577	0.8546489	0.854931081
0.855146408	0.85543718	0.855712274	0.855982623	0.856093682	0.856732133
0.857033735	0.857372816	0.858212723	0.858788834	0.859986795	0.860485726
0.862474506	0.862893948	0.863399949	0.863545914	0.86362269	0.863685095
0.863704548	0.864597075	0.86518504	0.865689182	0.866598701	0.866706406
0.867068755	0.867215303	0.867724405	0.867879416	0.868441626	0.86899869
0.870005512	0.870491189	0.870844059	0.871227552	0.871262874	0.87155719
0.871603638	0.871905264	0.872391617	0.872452531	0.873039144	0.873439944
0.874071015	0.874526663	0.874529664	0.875149208	0.875318145	0.875963621
0.87624388	0.877743197	0.877995122	0.878255189	0.878439902	0.878567221
0.878603749	0.87866497	0.879103086	0.879807812	0.881455927	0.88170915

0.882038697	0.882450067	0.882974757	0.883132105	0.883341411	0.883368626
0.883459344	0.883543372	0.883638147	0.883771209	0.8839412	0.883947284
0.884249179	0.884326602	0.884416101	0.88465453	0.884680495	0.884724119
0.885105254	0.886155362	0.886331691	0.886546769	0.886814595	0.886914658
0.88757254	0.887701876	0.887931217	0.887997256	0.888028878	0.888295905
0.888529509	0.889077143	0.889382078	0.889667594	0.890091961	0.890547565
0.890571423	0.89071067	0.89074222	0.890756459	0.89119813	0.891586454
0.891675874	0.891747578	0.891909783	0.891963531	0.892364526	0.892759656
0.892763604	0.892954356	0.893364377	0.893402232	0.893590366	0.893665348
0.893810019	0.894256211	0.894287483	0.894833686	0.895313273	0.89547253
0.895486867	0.895844715	0.895939	0.895983966	0.896025321	0.896177618
0.896310462	0.896671989	0.896673665	0.896858399	0.897132254	0.897409943
0.89789598	0.897899905	0.897928527	0.898902082	0.899513251	0.899752249
0.899829125	0.899861939	0.900003554	0.900006376	0.900143575	0.90016131
0.900281795	0.900416062	0.900596367	0.900871465	0.901066844	0.901629561
0.901877369	0.902340069	0.902657055	0.902833685	0.903238494	0.903287144
0.903418751	0.904146749	0.90420107	0.904727017	0.904895613	0.905022572
0.906057213	0.906083	0.906120597	0.906131074	0.906601853	0.907305995
0.907455292	0.907848668	0.907884864	0.907956415	0.908102246	0.908379985
0.90850407	0.908593927	0.90943463	0.909470678	0.909491898	0.909502715
0.910106929	0.910115664	0.910212764	0.910333753	0.910459179	0.910481885
0.910788503	0.910839802	0.910859815	0.91092727	0.91095673	0.911019199
0.911062521	0.911086105	0.911116741	0.911141858	0.911242176	0.911309232
0.91164368	0.911723306	0.911858109	0.912260154	0.913138814	0.913290513
0.913467652	0.913879909	0.913941072	0.914025273	0.91530748	0.915308986
0.915527087	0.915632037	0.915736437	0.91608296	0.916165527	0.916264146
0.916285825	0.916358291	0.916447486	0.916509019	0.916661451	0.917114111
0.917129997	0.917530099	0.917660487	0.918146987	0.918336515	0.918633324
0.918997203	0.919020783	0.919331068	0.919353828	0.919430997	0.919570118
0.919975514	0.920125091	0.920139587	0.920429377	0.920749288	0.920953061
0.921496837	0.921550861	0.922718344	0.922923259	0.923125376	0.923267626
0.923277028	0.923623599	0.923650781	0.923735396	0.923814381	0.924078687
0.924147492	0.924307339	0.924612308	0.925014967	0.925384159	0.925751233

0.925954825	0.926099694	0.926313118	0.926393128	0.926474989	0.926518451
0.926960985	0.927115199	0.927771371	0.928172396	0.928204123	0.928226878
0.928331353	0.928387482	0.928648684	0.928675219	0.928709971	0.92933007
0.929616116	0.930066848	0.930456578	0.930473612	0.930621696	0.930651408
0.930798849	0.931053286	0.931084595	0.931151376	0.931239494	0.931522679
0.932483843	0.932633086	0.93315663	0.933209617	0.93335945	0.933429265
0.933471363	0.933577098	0.934802725	0.936291778	0.936459635	0.93648351
0.93662399	0.937233112	0.937935231	0.938083305	0.93836208	0.938401293
0.939972831	0.940047795	0.943516382	0.943636617	0.943749016	0.944871275
0.94564848	0.946274883	0.947042886	0.948791439	0.950367479	0.952598906
0.952676156	0.952907732	0.953241656	0.956700668	0.958850743	0.959562391
0.962875106	0.965180724	0.980099231			

Fig. 7.21 Bhattacharyya Results

Images (.jpg)	**Num of Images Returned**					
	5	**10**	**15**	**20**	**25**	**30**
	Num of Correct Images					
Crowds001	5	10	14	19	24	28
Crowds002	5	10	15	19	24	28
Crowds003	5	10	14	18	21	24
Crowds004	4	9	14	18	21	26
Crowds005	5	9	14	19	22	26
Crowds006	3	8	11	14	17	21
Crowds007	5	9	13	17	21	25
Crowds008	5	8	12	16	19	22
Crowds009	5	10	14	19	24	29
Crowds010	5	10	15	19	24	28
Crowds011	5	10	15	19	24	28
Crowds012	5	10	15	19	24	29
Crowds013	5	8	12	16	19	22
Crowds014	5	10	15	20	23	27

Crowds015	4	8	10	12	15	16
Crowds016	5	10	15	20	25	29
Crowds017	5	10	15	19	24	29
Crowds018	5	10	15	20	25	29
Crowds019	4	9	14	18	23	27
Crowds020	5	9	14	16	21	25
Crowds021	4	9	12	16	20	24
Crowds022	4	9	13	16	19	22
Average Precision	*0.936364*	*0.931818*	*0.912121*	*0.884091*	*0.870909*	*0.854545*
F1-Cars001	0	0	1	2	3	3
F1-Cars002	3	3	4	5	6	7
F1-Cars003	1	2	3	6	7	10
F1-Cars004	2	3	5	5	9	11
F1-Cars005	0	1	2	3	4	6
F1-Cars006	0	0	0	1	3	4
F1-Cars007	1	2	3	5	6	10
F1-Cars008	3	5	5	5	5	7
F1-Cars009	2	3	4	4	8	9
F1-Cars010	4	5	8	8	9	10
F1-Cars011	4	6	10	11	13	14
F1-Cars012	5	7	8	9	11	12
F1-Cars013	1	3	5	9	11	13
F1-Cars014	2	3	5	5	7	7
F1-Cars015	0	0	0	1	1	1
F1-Cars016	4	7	9	11	13	15
F1-Cars017	1	1	2	2	3	5
F1-Cars018	5	10	15	17	20	22
F1-Cars019	3	7	9	10	13	15
F1-Cars020	2	3	4	6	8	8
F1-Cars021	4	7	9	12	14	18
Average Precision	*0.447619*	*0.371429*	*0.352381*	*0.32619*	*0.331429*	*0.328571*

Horses001	3	7	11	13	17	19
Horses002	4	6	9	12	15	17
Horses003	4	8	12	15	16	17
Horses004	5	9	13	15	19	23
Horses005	4	7	8	11	15	16
Horses006	5	10	12	15	19	22
Horses007	4	7	10	13	14	17
Horses008	1	2	4	6	7	8
Horses009	3	5	8	11	13	14
Horses010	3	5	8	11	13	15
Horses011	5	7	10	14	18	22
Horses012	4	5	8	11	14	15
Horses013	1	2	2	5	5	6
Horses014	2	2	3	4	6	9
Horses015	3	8	12	15	17	21
Horses016	3	4	8	9	13	14
Horses017	5	9	11	15	18	22
Horses018	2	3	5	7	8	10
Horses019	3	7	11	13	18	22
Horses020	2	2	4	5	5	6
Horses021	3	6	10	14	15	17
Horses022	5	10	14	18	20	23
Average Precision	*0.672727*	*0.595455*	*0.584848*	*0.572727*	*0.554545*	*0.537879*
Landscapes001	4	9	11	14	14	14
Landscapes002	4	5	7	9	9	9
Landscapes003	4	5	7	8	9	9
Landscapes004	0	0	1	1	4	6
Landscapes005	3	6	10	12	14	17
Landscapes006	3	6	8	8	11	12
Landscapes007	3	3	4	4	4	4
Landscapes008	2	2	5	9	11	14
Landscapes009	0	2	2	4	4	5

Landscapes010	1	3	4	4	5	5
Landscapes011	5	7	9	11	11	12
Landscapes012	4	7	9	11	11	12
Landscapes013	1	3	4	6	6	8
Landscapes014	3	4	4	4	6	6
Landscapes015	4	7	9	11	11	13
Landscapes016	0	1	1	2	4	4
Landscapes017	1	2	3	4	4	4
Landscapes018	4	5	6	7	10	10
Landscapes019	2	4	6	8	10	13
Landscapes020	5	6	8	10	12	14
Landscapes21	5	8	10	11	12	13
Average Precision	*0.552381*	*0.452381*	*0.406349*	*0.37619*	*0.346667*	*0.32381*

Fig. 7.22 Bhattacharyya Full Results

Bhattacharyya (7 bins)	
Precision	**Recall**
0.65	0.113847243
0.525	0.228676471
0.441667	0.379493464
0.341667	0.640625
0.25	1

Fig. 7.23 7 bin Bhattacharyya precision-recall data

Method	**Query Image**	**Results**	**Time (s)**
Correlation (8 bins)	Crowds001.jpg	15.00	2.596444
	Crowds10.jpg	15.00	2.929038
	F1-Cars001.jpg	1.00	2.580701
	F1-Cars10.jpg	6.00	2.614612
	Horses001.jpg	10.00	2.276192
	Horses10.jpg	8.00	2.401694
	Landscapes001.jpg	11.00	2.29146
	Landscapes10.jpg	3.00	2.424246
		57.5	**2.514298**

Chi-square (8 bins)	Crowds001.jpg	15.00	3.841582
	Crowds10.jpg	15.00	3.59576
	F1-Cars001.jpg	1.00	3.495053
	F1-Cars10.jpg	7.00	3.556995
	Horses001.jpg	11.00	3.524728
	Horses10.jpg	6.00	3.427903
	Landscapes001.jpg	12.00	3.59234
	Landscapes10.jpg	4.00	3.522942
		59.17	**3.569663**
Intersection (8 bins)	Crowds001.jpg	15.00	2.409595
	Crowds10.jpg	15.00	2.377954
	F1-Cars001.jpg	1.00	2.284409
	F1-Cars10.jpg	9.00	2.29746
	Horses001.jpg	11.00	2.408053
	Horses10.jpg	7.00	2.479597
	Landscapes001.jpg	9.00	2.467109
	Landscapes10.jpg	4.00	2.391278
		59.17	**2.389432**
Bhattacharyya (7 bins)	Crowds001.jpg	14.00	2.827121
	Crowds10.jpg	15.00	2.967066
	F1-Cars001.jpg	1.00	2.745372
	F1-Cars10.jpg	8.00	2.797245
	Horses001.jpg	11.00	2.759143
	Horses10.jpg	8.00	2.734024
	Landscapes001.jpg	12.00	2.723621
	Landscapes10.jpg	4.00	2.704434
		60.83	**2.782253**

Fig. 7.24 Overall Simple Model Results

Chi-Square

Image (.jpg)	Time (s)
Crowds1	4.605819
Crowds2	3.8403824
Crowds3	3.792076
Crowds4	3.854858
Crowds5	3.7673239
Crowds6	3.6917424
Crowds7	3.8381378
Crowds8	3.6929511
Crowds9	3.7244117
Crowds10	3.7729332
Crowds11	3.5820425

Intersection

Image (.jpg)	Time (s)
Crowds1	2.5209486
Crowds2	2.4966293
Crowds3	2.3949204
Crowds4	2.4127277
Crowds5	2.346658
Crowds6	2.4227307
Crowds7	2.6448609
Crowds8	2.4250527
Crowds9	2.3701604
Crowds10	2.3593713
Crowds11	2.4125805

Crowds12	3.8844728	Crowds12	2.4084617
Crowds13	3.6527416	Crowds13	2.358908
Crowds14	3.6589671	Crowds14	2.3561192
Crowds15	3.6477954	Crowds15	2.3750578
Crowds16	3.8191289	Crowds16	2.3505605
Crowds17	3.7132562	Crowds17	2.3844497
Crowds18	3.8479412	Crowds18	2.3486435
Crowds19	3.9153327	Crowds19	2.4206007
Crowds20	3.7111782	Crowds20	2.4950359
Crowds21	3.9275168	Crowds21	2.3765917
Crowds22	3.6774119	Crowds22	2.3585668
F1-Cars1	3.7945423	F1-Cars1	2.3481163
F1-Cars2	3.6136033	F1-Cars2	2.3783733
F1-Cars3	3.6701126	F1-Cars3	2.3772791
F1-Cars4	3.5823894	F1-Cars4	2.4014641
F1-Cars5	3.6249575	F1-Cars5	2.3820064
F1-Cars6	3.6116416	F1-Cars6	2.5007376
F1-Cars7	3.6269189	F1-Cars7	2.3539912
F1-Cars8	3.5615297	F1-Cars8	2.3619369
F1-Cars9	3.6619709	F1-Cars9	2.3667159
F1-Cars10	3.5649533	F1-Cars10	2.5120698
F1-Cars11	3.6347257	F1-Cars11	2.3578904
F1-Cars12	3.6296782	F1-Cars12	2.4145186
F1-Cars13	3.5892652	F1-Cars13	2.6331553
F1-Cars14	3.7005351	F1-Cars14	2.3588917
F1-Cars15	3.6361051	F1-Cars15	2.3610913
F1-Cars16	3.6265197	F1-Cars16	2.4641219
F1-Cars17	3.6556064	F1-Cars17	2.6909607
F1-Cars18	3.6750368	F1-Cars18	2.336556
F1-Cars19	3.741091	F1-Cars19	2.3630242
F1-Cars20	4.0333199	F1-Cars20	2.3631919
F1-Cars21	3.6485637	F1-Cars21	2.3572737
Horses1	3.6211197	Horses1	2.4004104
Horses2	3.6683815	Horses2	2.3490969
Horses3	3.5511561	Horses3	2.3758614
Horses4	3.6233298	Horses4	2.3689042
Horses5	3.5910799	Horses5	2.3383669
Horses6	3.6373671	Horses6	2.346771
Horses7	3.6196952	Horses7	2.4615119
Horses8	3.6102303	Horses8	2.3597888
Horses9	3.5962933	Horses9	2.3293568
Horses10	3.6407	Horses10	2.3375944
Horses11	3.5841964	Horses11	2.3615289
Horses12	3.6448433	Horses12	2.4300051
Horses13	3.6009644	Horses13	2.3784057
Horses14	3.7578215	Horses14	2.3817592
Horses15	3.5823819	Horses15	2.3443102
Horses16	3.6041123	Horses16	2.6019225

Horses17	3.6524518	Horses17	2.3267915
Horses18	3.7544131	Horses18	2.3699738
Horses19	3.5659453	Horses19	2.3638826
Horses20	3.6268338	Horses20	2.3711815
Horses21	3.5600326	Horses21	2.3517662
Horses22	3.8261317	Horses22	2.3243634
Landscapes1	3.5588364	Landscapes1	2.3871247
Landscapes2	3.649122	Landscapes2	2.3731272
Landscapes3	3.5859241	Landscapes3	2.3488414
Landscapes4	3.6240597	Landscapes4	2.335114
Landscapes5	3.7130128	Landscapes5	2.3482805
Landscapes6	3.6107566	Landscapes6	2.352889
Landscapes7	3.5982741	Landscapes7	2.3702992
Landscapes8	3.6821039	Landscapes8	2.3491287
Landscapes9	3.5991212	Landscapes9	2.356877
Landscapes10	3.7607479	Landscapes10	2.4501269
Landscapes11	3.5448654	Landscapes11	2.5367317
Landscapes12	3.6222644	Landscapes12	2.3661763
Landscapes13	3.6556473	Landscapes13	2.3732675
Landscapes14	3.6467095	Landscapes14	2.3535667
Landscapes15	3.6720493	Landscapes15	2.3453065
Landscapes16	3.6651785	Landscapes16	2.5838183
Landscapes17	3.5797601	Landscapes17	2.3511386
Landscapes18	3.6705763	Landscapes18	2.3539536
Landscapes19	3.7647519	Landscapes19	2.3441794
Landscapes20	3.6700834	Landscapes20	2.4747087
Landscapes21	3.6252114	Landscapes21	2.4065759

Fig. 7.25 Full Chi-square/Intersection time test results

Query Image	**Results (%)**	**Time (s)**
Crowds001.jpg	100	3.841582
Crowds10.jpg	100	3.59576
F1-Cars001.jpg	6.67	3.495053
F1-Cars10.jpg	46.67	3.556995
Horses001.jpg	73.33	3.524728
Horses10.jpg	40	3.427903
Landscapes001.jpg	80	3.59234
Landscapes10.jpg	26.67	3.522942
Averages	**59.17**	**3.569663**

Fig. 7.26 Chi-Square Precision/time

Query Image	Results (%)	Time (s)
Crowds001.jpg	100	2.409595
Crowds10.jpg	100	2.377954
F1-Cars001.jpg	6.67	2.284409
F1-Cars10.jpg	60	2.29746
Horses001.jpg	73.33	2.408053
Horses10.jpg	46.67	2.479597
Landscapes001.jpg	60	2.467109
Landscapes10.jpg	26.67	2.391278
Averages	**59.17**	**2.389432**

Fig. 7.27 Intersection Precision/time

```
public static void saver(ArrayList<Double>[] scores) throws IOException {

    // Create a new file
    File file = new File("src\\scores.csv");

    if (file.exists()) { file.delete();
    // Write the data
    FileWriter writer = new FileWriter(file);
    for (int i = 0; i < scores.length; i++) {
        // Write the ID
        writer.write("ID" + (i+1) + ",");
        //ID1*Correlation*
        //ID2 *Chi-square*
        //ID3*Intersection*
        //ID4*Bhattacharyya*

        // Write the scores
        List<Double> row = scores[i];
        for (Double score : row) {
            writer.write(score + ","); }

        // Move to next row
        writer.write("\n"); }

    // Close the file
    writer.close(); }
}
```

Fig. 7.28 Initial .csv save method

```
public static ArrayList<Double>[] readScores() throws FileNotFoundException {

    String fileName = "src\\scores.csv";

    List<ArrayList<Double>> result = new ArrayList<>();
    Scanner scanner = new Scanner(new File(fileName));
    while (scanner.hasNextLine()) {
        ArrayList<Double> row = new ArrayList<>();
        String[] data = scanner.nextLine().split(",");
        for (int i = 1; i < data.length; i++)
        {
            row.add(Double.parseDouble(data[i]));
        }
        result.add(row);
    }
    scanner.close();
    return result.toArray(new ArrayList[0]);
}
```

Fig. 7.29 Initial .csv read method

```
File file = new File("src\\test.bin");

try (DataOutputStream out = new DataOutputStream(new FileOutputStream(file))) {
    // Write the header
    out.writeInt(imageHistoMatrices.size());

    // Write the data for each image
    for (float[][][] image : imageHistoMatrices) {
        out.writeInt(image.length); // Width
        out.writeInt(image[0].length); // Height

        // Write the hsv values
        for (int i = 0; i < image.length; i++) {
            for (int j = 0; j < image[0].length; j++) {
                for (int k = 0; k < image[i][j].length; k++) {
                    out.writeFloat(image[i][j][k]);
                }
            }
        }
    }
    System.out.println("SAVED!");
    JOptionPane.showMessageDialog(null, "Data successfully saved to file", "Process Update",
            JOptionPane.INFORMATION_MESSAGE);
}
catch (final IOException e) {
    JOptionPane.showMessageDialog(null, "Failed to data to file", "Warning",
            JOptionPane.WARNING_MESSAGE);
    return;
}
```

Fig. 7.30 BIN write method

```java
public static ArrayList<float[][][]> readHSV() throws IOException {

    File file = new File("src\\test.bin");
    ArrayList<float[][][]> data = new ArrayList<>();

    try (DataInputStream in = new DataInputStream(new FileInputStream(file))) {
        // Read the header
        int numImages = in.readInt();

        // Read the data for each image
        for (int k = 0; k < numImages; k++) {
            int width = in.readInt();
            int height = in.readInt();
            int depth = 8;

            // Initialize the image array with the correct dimensions
            float[][][] image = new float[width][height][depth];

            // Read the hsv values
            for (int i = 0; i < width; i++) {
                for (int j = 0; j < height; j++) {
                    for (int d = 0; d < depth; d++) {
                        image[i][j][d] = in.readFloat();
                    }
                }
            }
            data.add(image);
        }
    }
    catch (final IOException e) {
        JOptionPane.showMessageDialog(null, "Failed to read data from file", "Warning",
                JOptionPane.WARNING_MESSAGE);
    }
    return data;
```

Fig. 7.31 BIN read method

Num Bins	Time (s)
3	17.0071149
4	19.9838624
5	14.950811
6	18.8896121
7	18.0683443
8	19.2009816
9	19.3936016

Num Bins	Time (s)
3	15.0725753
4	15.9499971
5	15.5880655
6	13.6463177
7	13.9659532
8	14.4504188
9	17.2094585

Num Bins	Time (s)
3	13.713485
4	14.980522
5	14.302992
6	16.169821
7	13.627377
8	14.803532
9	14.556957

Fig. 7.32 BIN times cross validated

User Feedback Form 1:

Please rate your satisfaction with the following aspects of the content-based image retrieval system using a scale of 1 to 10, where 1 represents the lowest level of satisfaction and 10 represents the highest level of satisfaction.

1) How easy was it to select a query image using the User Interface (UI)?

   1 (not at all easy) - 10 (extremely easy) **Answer:**_8__

2) Did the application retrieve the set of similar images that you expected from the gallery?

   1 (did not retrieve expected images) - 10 (retrieved expected images) **Answer:**_10__

3) How effective was the distance measurement in retrieving similar images?

   1 (not effective at all) - 10 (extremely effective) **Answer:**_7__

4) Was the speed of the image retrieval process acceptable?

   1 (unacceptably slow) - 10 (very fast) **Answer:**_9__

5) Did the application handle errors and exceptions appropriately?

   1 (did not handle errors well) - 10 (handled errors very well) **Answer:**_10__

6) Was the speed of the database update process acceptable?

   1 (unacceptably slow) - 10 (very fast) **Answer:**_5__

7) Did the application meet your expectations for a content-based image retrieval system?

   1 (did not meet expectations) - 10 (exceeded expectations) **Answer:**_8__

Fig. 7.33 User Feedback Form 1

User Feedback Form 2:

Please rate your satisfaction with the following aspects of the content-based image retrieval system using a scale of 1 to 10, where 1 represents the lowest level of satisfaction and 10 represents the highest level of satisfaction.

1) How easy was it to select a query image using the User Interface (UI)?

   1 (not at all easy) - 10 (extremely easy) **Answer:**_9__

2) Did the application retrieve the set of similar images that you expected from the gallery?

   1 (did not retrieve expected images) - 10 (retrieved expected images) **Answer:**_10__

3) How effective was the distance measurement in retrieving similar images?

   1 (not effective at all) - 10 (extremely effective) **Answer:**_7__

4) Was the speed of the image retrieval process acceptable?

   1 (unacceptably slow) - 10 (very fast) **Answer:**_9__

5) Did the application handle errors and exceptions appropriately?

   1 (did not handle errors well) - 10 (handled errors very well) **Answer:**_10__

6) Was the speed of the database update process acceptable?

   1 (unacceptably slow) - 10 (very fast) **Answer:**_5__

7) Did the application meet your expectations for a content-based image retrieval system?

1 (did not meet expectations) - 10 (exceeded expectations) **Answer:**_8__

Fig. 7.34 User Feedback Form 2

User Feedback Form 3:

Please rate your satisfaction with the following aspects of the content-based image retrieval system using a scale of 1 to 10, where 1 represents the lowest level of satisfaction and 10 represents the highest level of satisfaction.

1) How easy was it to select a query image using the User Interface (UI)?

   1 (not at all easy) - 10 (extremely easy) **Answer:**__8_

2) Did the application retrieve the set of similar images that you expected from the gallery?

   1 (did not retrieve expected images) - 10 (retrieved expected images) **Answer:**__10_

3) How effective was the distance measurement in retrieving similar images?

   1 (not effective at all) - 10 (extremely effective) **Answer:**_6__

4) Was the speed of the image retrieval process acceptable?

   1 (unacceptably slow) - 10 (very fast) **Answer:**__10_

5) Did the application handle errors and exceptions appropriately?

   1 (did not handle errors well) - 10 (handled errors very well) **Answer:**__10_

6) Was the speed of the database update process acceptable?

   1 (unacceptably slow) - 10 (very fast) **Answer:**_4__

7) Did the application meet your expectations for a content-based image retrieval system?

1 (did not meet expectations) - 10 (exceeded expectations) **Answer:**_8__

Fig. 7.35 User Feedback Form 3

User Feedback Form 4:

Please rate your satisfaction with the following aspects of the content-based image retrieval system using a scale of 1 to 10, where 1 represents the lowest level of satisfaction and 10 represents the highest level of satisfaction.

1) How easy was it to select a query image using the User Interface (UI)?

   1 (not at all easy) - 10 (extremely easy) **Answer:**__9_

2) Did the application retrieve the set of similar images that you expected from the gallery?

   1 (did not retrieve expected images) - 10 (retrieved expected images) **Answer:**_10__

3) How effective was the distance measurement in retrieving similar images?

   1 (not effective at all) - 10 (extremely effective) **Answer:**__8_

4) Was the speed of the image retrieval process acceptable?

   1 (unacceptably slow) - 10 (very fast) **Answer:**__7_

5) Did the application handle errors and exceptions appropriately?

   1 (did not handle errors well) - 10 (handled errors very well) **Answer:**_10__

6) Was the speed of the database update process acceptable?

   1 (unacceptably slow) - 10 (very fast) **Answer:**__4_

7) Did the application meet your expectations for a content-based image retrieval system?

1 (did not meet expectations) - 10 (exceeded expectations) **Answer:**_8__

Fig. 7.36 User Feedback Form 4

User Feedback Form 5:

Please rate your satisfaction with the following aspects of the content-based image retrieval system using a scale of 1 to 10, where 1 represents the lowest level of satisfaction and 10 represents the highest level of satisfaction.

1) How easy was it to select a query image using the User Interface (UI)?

   1 (not at all easy) - 10 (extremely easy) **Answer:**__10_

2) Did the application retrieve the set of similar images that you expected from the gallery?

   1 (did not retrieve expected images) - 10 (retrieved expected images) **Answer:**_10__

3) How effective was the distance measurement in retrieving similar images?

   1 (not effective at all) - 10 (extremely effective) **Answer:**__6_

4) Was the speed of the image retrieval process acceptable?

   1 (unacceptably slow) - 10 (very fast) **Answer:**__8_

5) Did the application handle errors and exceptions appropriately?

   1 (did not handle errors well) - 10 (handled errors very well) **Answer:**_10__

6) Was the speed of the database update process acceptable?

   1 (unacceptably slow) - 10 (very fast) **Answer:**_6__

7) Did the application meet your expectations for a content-based image retrieval system?

1 (did not meet expectations) - 10 (exceeded expectations) **Answer:**__8_

Fig. 7.37 User Feedback Form 5

10.0 References

[0] https://github.com/acregan04/ISE

[1] M. P. Arakeri and G. Ram Mohana Reddy, "An intelligent content-based image retrieval system for clinical decision support in Brain tumor diagnosis," *International Journal of Multimedia Information Retrieval*, vol. 2, no. 3, pp. 175–188, 2013.

[2] M. Sultana and M. Gavrilova, "A content based feature combination method for face recognition," *Proceedings of the 8th International Conference on Computer Recognition Systems CORES 2013*, vol. 226, 2013.

[3] T. F. Stepinski, P. Netzel, and J. Jasiewicz, "Landex—a geoweb tool for query and retrieval of spatial patterns in land cover datasets," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 7, no. 1, pp. 257–266, Jan. 2014.

[4] D. Edmundon, G. Schaefer, and M. E. Celebi, "Similarity-based browsing of image search results," *2013 IEEE International Symposium on Multimedia*, pp. 502–503, 2013.

[5] M. Arevalillo-Herráez, F. J. Ferri, and S. Moreno-Picot, "A hybrid multi-objective optimization algorithm for content based image retrieval," *Applied Soft Computing*, vol. 13, no. 11, pp. 4358–4369, 2013.

[6] R. Datta, D. Joshi, J. Li, and J. Z. Wang, "Image retrieval: Ideas, influences, and trends of the new age," *ACM Computing Surveys*, vol. 40, no. 2, pp. 1–60, 2008.

[7] S. Gandhani, R. Bhujade, and A. Sinhal, "An improved and efficient implementation of CBIR system based on combined features," *Third International Conference on Computational Intelligence and Information Technology (CIIT 2013)*, 2013.

[8] T. G. Bhat, N. D. Gundi, and H. R. Kulkarni, "A novel approach for content based image retrieval from huge database sets," *Third International Conference on Computational Intelligence and Information Technology (CIIT 2013)*, 2013.

[9] A. Urmaliya and J. Singhai, "Sequential minimal optimization for support vector machine with feature selection in breast cancer diagnosis," *2013 IEEE Second International Conference on Image Information Processing (ICIIP-2013)*, pp. 481–486, 2013.

[10] I. S. P. James, "Face image retrieval with HSV color space using clustering techniques," *The SIJ Transactions on Computer Science Engineering & its Applications (CSEA)*, vol. 01, no. 01, pp. 18–19, 2013.

[11] G.-H. Liu and J.-Y. Yang, *Content-based image retrieval using color difference histogram*, vol. 46, no. 1, pp. 188–198, 2013.

[12] P. Anantharatnasamy, K. Sriskandaraja, V. Nandakumar, and S. Deegalla, “Fusion of colour, shape and texture features for content based image retrieval,” *2013 8th International Conference on Computer Science & Education*, pp. 422–427, 2013.

[13] Y.-S. Qin, S.-F. Sun, X.-B. Ma, S. Hu, and B.-J. Lei, “A shadow removal algorithm for vibe in HSV color space,” *Proceedings of 3rd International Conference on Multimedia Technology(ICMT-13)*, pp. 959–966, Nov. 2013.

[14] T. Hamachi, H. Tanabe, and A. Yamawaki, “Development of a generic RGB to HSV hardware,” *The Proceedings of the 1st International Conference on Industrial Application Engineering 2013*, 2013.

[15] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed. New York, NY: Pearson, 2018.

[16] G. Bradski and A. Kaehler, “Color Histograms,” in *Learning OpenCV*, 1st ed., Sebastopol, CA: O'Reilly, 2008, pp. 218.

[17] “Histogram comparison,” *OpenCV*. [Online]. Available: https://docs.opencv.org/4.x/d8/dc8/tutorial_histogram_comparison.html. [Accessed: 05-Apr-2023].

[18] I. M. Hameed, S. H. Abdulhussain, and B. M. Mahmmod, “Content-based Image Retrieval: A review of recent trends,” *Cogent Engineering*, vol. 8, no. 1, 2021.

[19] S. Nelson, “Introduction to Backup and Recovery,” in *Pro Data Backup and recovery*, New York, NY: Apress L. P, 2011, pp. 1–17.

[20] N. Zhang, G. Zheng, H. Chen, J. Chen, and X. Chen, “HBaseSpatial: A Scalable Spatial Data Storage based on HBase,” *2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 644–651, 2014.

[21] N. H. Minh, *Java connect to Microsoft SQL Server example*, 13-Mar-2020. [Online]. Available: https://www.codejava.net/java-se/jdbc/connect-to-microsoft-sql-server-via-jdbc. [Accessed: 05-Apr-2023].

[22] R. Pijacek, “Microsoft SQL server pros and cons,” *LearnSQL.com*, 17-Jan-2019. [Online]. Available: https://learnsql.com/blog/microsoft-sql-server-pros-and-cons/. [Accessed: 16-Apr-2023].

[23] C. Hummert and D. Pawlaszczyk, *Mobile Forensics - The File Format Handbook: Common File formats and file systems used in mobile devices*. Springer Nature, 2022.

[24] F. Radenovic, G. Tolias, and O. Chum, “Fine-tuning CNN image retrieval with no human annotation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 7, pp. 1655–1668, 2019.

[25] K.-Y. Chen, J. M. Chang, and T.-W. Hou, “Multithreading in java: Performance and scalability on Multicore Systems,” *IEEE Transactions on Computers*, vol. 60, no. 11, pp. 1521–1534, Nov. 2011.

[26] S. Shahrivari, “Beyond batch processing: Towards real-time and streaming Big Data,” *Computers*, vol. 3, no. 4, pp. 117–129, 2014.

[27] G. Boss, P. Malladi, D. Quan, L. Legregni, and H. Hall, “Cloud Computing,” pp. 224–231, Oct. 2007.

[28] F. Malik and B. Baharudin, “Analysis of distance metrics in content-based image retrieval using statistical quantized histogram texture features in the DCT domain,” *Journal of King Saud University - Computer and Information Sciences*, vol. 25, no. 2, pp. 207–218, 2013.

[29] M. E. ElAlami, “A novel image retrieval model based on the most relevant features,” *Knowledge-Based Systems*, vol. 24, no. 1, pp. 23–32, Feb. 2011.