University of Dublin

TRINITY COLLEGE

A Practical Application of Logo Recognition

Dónal Lowry

B.A.(Mod.) Business and Computing

Final Year Project May 2018
Supervisor: Mary Sharp

School of Computer Science and Statistics

O'Reilly Institute, Trinity College, Dublin 2, Ireland

DECLARATION

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other University.

_____ _____

Name Date

ACKNOWLEDGEMENTS

I would like to acknowledge the invaluable guidance given to me by Professor Mary Sharp. As supervisor of the project, Professor Sharp provided advice and direction, throughout the period of execution, which was truly indispensable.

TABLE OF CONTENTS

ABSTRACT

This report describes the development of an Android application which allows users to rent out the back of their laptop as advertising space. The work carried out during the project draws on material from a number of different areas, such as computer vision, relational database design, android application development and human computer interaction. The motivation, design, implementation and testing of the system are discussed in detail in this report. The report concludes with a reflection on the results of the project, as well as consideration of future development and improvement.

Chapter 1 - Introduction

This chapter includes a discussion of the project motivation, objectives and related work, as well as an outline of the report structure.

1.1 Research Motivation

In the domain of offline advertising, targeting specific demographics may be difficult to achieve. A billboard on a roadside, for example, has no method of reaching a certain target market. Additionally, advertising costs can be inaccessibly high, especially to small businesses. The cost of a typical television advertising campaign is €10,000 (Hessinger, 2018).

Word of mouth is an effective form of advertising based on trust between friends (Forbes, 2014). Laptop advertisements can encourage word of mouth and conversations about the brands that are displayed.

1.2 Research Objectives

The primary objective of the project is to design and implement a platform which allows users to rent out the back of their laptop as advertising space. Within the scope of the project, this objective will take the form of an Android application. This section will explain the method by which each of the problems, presented in the section 1.1 will be addressed.

There are three users involved in this system: the *laptop-owner*, the *company* and the *administrator*. The laptop-owner is the person who wishes to rent out their laptop as advertising space, the company is any party who wishes to advertise their logo on laptops, and the administrator is the link between these two parties.

The system allows companies to target their advertisement to specific demographics. For example, a company may specify that they wish to advertise on the laptops of Trinity College students, between the age of 18-24.

The magnitude of the advertising campaigns may be as small or large as needed, making the process accessible to all sizes of company. Additionally, the required fees for the service would be modest, given that the laptop owner involved is obliged to do almost no work in exchange for compensation.

The only requirement of a laptop-owner, involved in an advertising campaign, is periodic verification. Once every two weeks, the user is obliged to verify that they are advertising the logo stickers as

intended. An ideal solution would combine a number of different methods of achieving this. One partial solution to this problem is requesting that the laptop-owner periodically uploads a photograph of the logo displayed on their laptop. This image could then be processed in numerous ways, for example, the geolocation could be cross checked with the laptop-owner's profile details to ensure that it is taken where they have claimed they use their laptop. This method was not implemented within the scope of the current project.

The timestamp of the image could be assessed to ensure that the photograph is taken at the current time. This problem has been solved within the scope of the project due to the fact that users can only upload an image via the app, which provides access to the camera and not the gallery of images stored on the phone. As such, the photograph can only be taken at the present time.

Furthermore, the contents of this image could be analysed to verify the presence of the logo which corresponds to the account of the laptop-owner. This problem was addressed in the project using computer vision techniques.

From the perspective of the laptop-owner, the other main use case of the system is the ability to review, accept and reject offers to become part of an advertising campaign. Additionally, the company user must have the ability to pay fees to laptop-owners in a convenient way.

The original design of the framework of the system involved the laptop-owner and company users only. Following a recommendation from an advisor, an additional user (the administrator) was added to the framework. The administrator role was designed as a manual process which would eventually become automated. The administrator must review all of the *open cases*, which are simply companies that do not have a live advertising campaign. The administrator may convert an open case to an *activated case*. At registration, companies provide a detailed description of the demographic of people who they wish to have advertising their logo. Laptop-owners, who are judged by the administrator to fit this demographic description, are added to the activated case. Requests to join the campaign are then sent to all of the associated laptop-owners. When the required number of laptop-owners have accepted this request, the activated case is converted to a *live campaign*, and the administrator's role is complete. Logo stickers can be posted to the users involved and the payment and verification period begins.

1.3 Background and Related Work

Another example of unconventional advertising, somewhat similar to this project, is coffee cup advertising. Numerous companies have harnessed disposable paper coffee cups as a medium for advertisement.

Recognition software in general has a wide range of practical applications, such as facial recognition as a method of unlocking a phone, or recognising abandoned luggage in an airport.

Feature detection, specifically, can be used to recognise road signs. These types of programs are an important aspect of the software of self-driving cars. Companies such as Uber, who have pioneered the adoption of self-driving cars, face such challenges as recognising road signs that have been vandalised, damaged or weathered. Fortunately, the problem domain of this project is more standardised and constrained. The application instructions insist that images that are submitted by

users are taken in a way that the laptop takes up the majority of the image. This decreases the complexity of the processing and improves the accuracy of the results.

Many companies seek to measure their online presence in order to evaluate overall performance. Much research goes into recognising logos automatically. A number of technology firms, such as the Irish owned LogoGrab (The Irish Times, 2016), develop software to search for company logos across all social media and quantify the results for clients. Often these implementations use neural networks, which have been trained with large datasets, in order to identify logos from a wide range of possibilities. The problem domain of this project, seeks only to match an input image with one potential template logo. Additionally, this operation is only carried out once every two weeks, as opposed to constant processing of a large dataset of social media images, therefore, a neural network implementation is not needed.

1.4 Report Outline

Chapter 2 of this report involves a discussion of how the project description, outlined in section 1.2, was translated into actionable requirements, both functional and non-functional.

Three broad functional requirements formulated during the elicitation process, were achieved within the scope of the project. Each of these requirements will form the basis of discussion in the subsequent three chapters. Chapter 3 outlines the implementation of the verification functionality of the system. Chapter 4 describes the system's supporting database. Chapter 5 details the design and implementation of an interface which can be coherently and logically navigated by users.

Chapter 6 illustrates the process of addressing the non-functional requirements found during elicitation. These non-functional requirements include building a learnable graphical user interface, creating seamless transitions from view to view, and providing the user with continuous feedback.

Conclusions in the final chapter will critique the work carried out in this project. Problems that were encountered during the course of execution will also be addressed, as well as a discussion of future work.

Chapter 2 - Requirements Engineering

Requirements Engineering describes the process of identifying, documenting and analysing the needs of all stakeholders in a software project (Nuseibeh and Easterbrook, 2018). This can be an extremely complex task. It can be difficult for the developer of a project to correctly interpret the requests of the stakeholders. One type of issue which can arise during this process is a *problem of scope*, where stakeholders provide unnecessary technical details that are detrimental to the overall objectives (Christel and Kang, 1992). A hypothetical example of this type of issue would be if a business manager requested that the system should carry out a certain task in under two seconds. A development team might spend weeks refining their work to achieve this, only to find out later that this comment was not supported by any rationale and not essential. Due to the existence of issues like this, much research is carried out into the development of methods of effectively eliciting requirements.

Due to the developer and stakeholder of this project being a single party, issues such as the one previously discussed are avoided. The initial phase of requirements engineering involves the elicitation of requirements. This phase can be done using a range of methods such as questionnaires, observation of workflow and interviews. These methods should be compiled together and result in a detailed description of the entire scope of the project, similar to the project description outlined in Section 1.2. In a normal scenario, this would be followed by a requirements analysis phase. In this phase, certain methods would be employed, such as prototyping, in order to clarify requirements with the stakeholders before development begins. Given that the stakeholder and developer of this project are a single party, this phase can be skipped and requirements specification can begin. Throughout the development cycle, requirements validation will be done, to ensure the project is being implemented in an ideal way.

2.1 Requirements Specification

Requirements specification involves formally documenting the requirements of the project. This includes detailed descriptions of all requirements, both textually and in the form of diagrams.

2.1.1 Functional Requirements

Based on the description of the project in Section 1.2, requirements specification was carried out. This work is summarised by the use case diagram in Figure 2.1. This use case diagram was created using

the specifications of the Unified Modelling Language (UML), a language used to visualise system designs (Booch, Rumbaugh and Jacobson, 2005).

The use cases of the application, which are shared by the laptop-owner user and the company user, are: the ability to register, login, view the user profile, edit profile details and edit profile settings. A company user must be able to pay fees in a convenient way. The laptop owner must be able to review potential offers to be part of an advertising campaign and either reject or accept them. The laptop-owner must also be able to verify that they are advertising as intended.

The administrator must have the ability to review open cases. Open cases are companies that have no current advertising campaign. The administrator can activate these open cases. Activating a case involves adding potential laptop owners, who adhere to the demographic description provided by the company. Requests can then be sent to these laptop owners, which can be accepted or rejected. After the request to join the campaign has been accepted by the required number of laptop owners, the administrator can convert the case to a live campaign.

The requirements described, thus far, encapsulate the information technology aspect of the system. Further requirements relate to manual processes such as posting logo stickers to laptop-owners. In Figure 2.1, the requirements highlighted in red are those which have been fully or partially implemented during this project.

The functional requirements achieved during the project can be categorised into three broad requirements:

1. A verification functionality.
2. A functional supporting database.
3. An interface that can be coherently and logically navigated by users.

Each of these requirements will form the basis of discussion in the subsequent three chapters.

2.1.2 Non-Functional Requirements

A number of non-functional requirements we also outlined in the specification phase. The first of these relates to the application working in a seamless fashion and constant feedback being provided to the user. This requirement involves such things as confirmation messages, when an action has been complete, and progress bars, while the application is carrying out heavy computation. Additionally, the graphical user interface should be aesthetically pleasing and easy for a user to learn. The implementation of non-functional requirements will be discussed in Chapter 6.

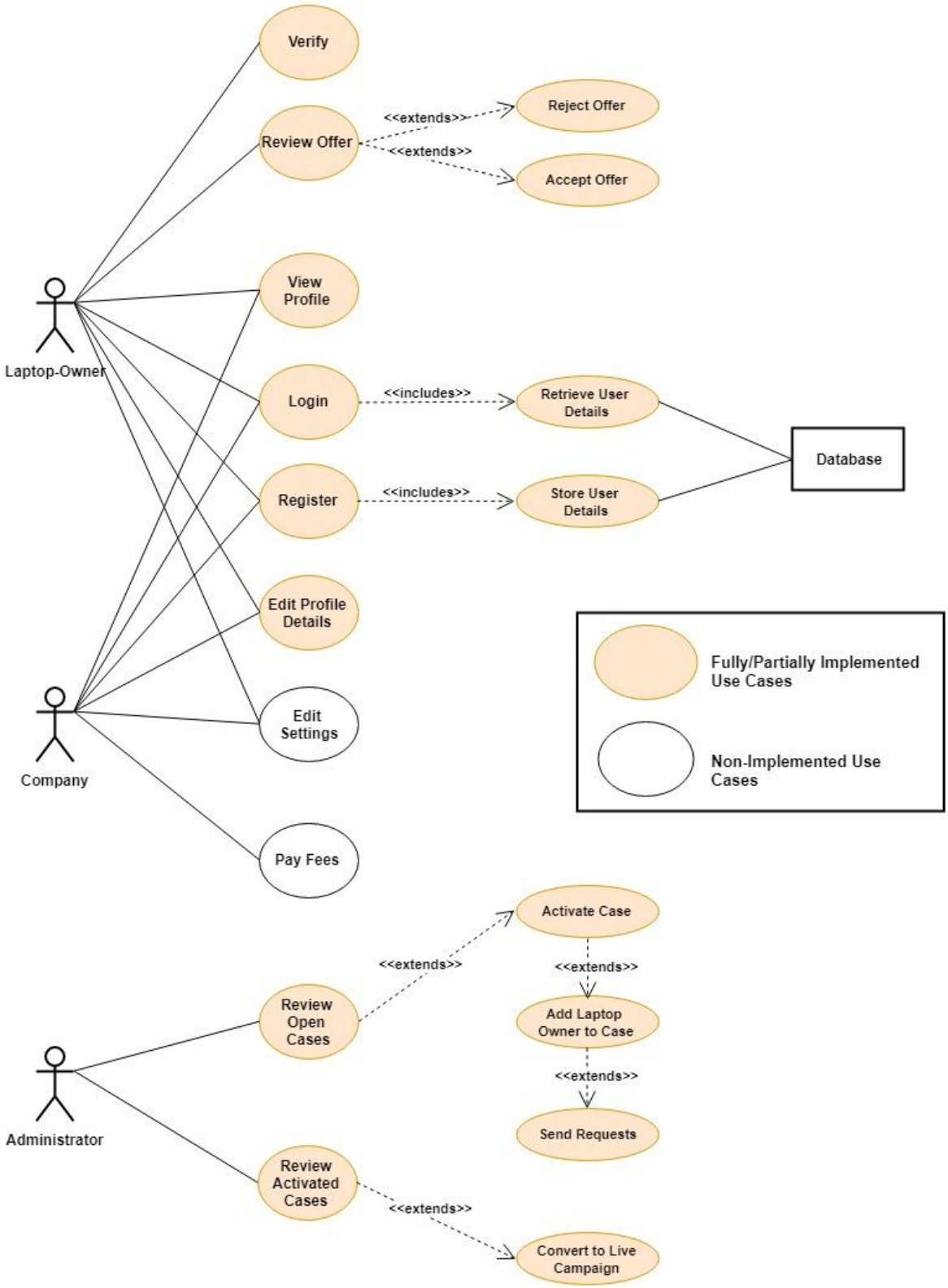


Figure 2.1 *Use case diagram of functional requirements*

Chapter 3 – Verification

In order to justify payment being made to laptop-owners, a system must be devised in order to verify that they are advertising the logo stickers, as intended. Ideally, the final product of this application would combine a number of different methods to achieve this. One partial solution to this problem is periodically requesting that the laptop-owner uploads a photograph, taken via the application, of the back of their laptop. This image should display the logo sticker. Information relating to the image could then be analysed, for example, the geolocation could be cross checked with laptop-owner's profile to ensure it's taken where it should be.

Furthermore, the contents of this image could be analysed to verify the presence of the logo which corresponds to the current laptop-owner. During this project, computer vision techniques were used in order to address the problem of recognising logos in images. Two separate recognition programs were created. One implementation employs histogram comparison, where the colour information of two images is compared. The other implementation uses an algorithm called Scale-invariant Feature Transform (SIFT), which matches distinct features found in two images. Full details of these implementations will follow.

For the first round of testing, it was assumed that the colour of the laptop was the dominant colour in the uploaded image. This requirement can be communicated to the user via message dialogs which instruct them to take the photograph so that the laptop takes up the majority of the image.

Before comparing the two programs' effectiveness, an effort was made to optimise their input parameters to maximise their accuracy. The input parameter being optimised in the histogram comparison program was the *minimum_matching_score* and in the feature matching program it was the *minimum_number_of_matches*. These parameters will be explained in more detail later. In the first round of testing, the two programs were run multiple times using a range of values for the input parameters mentioned. Each time they were run, a number of performance metrics were calculated to measure the programs' effectiveness. These metrics were then analysed, using excel, to identify the optimum value of the input parameters.

Following this, the performance metrics of the two programs were compared and, based on these figures, the feature matching implementation was chosen for the main application. Finally, two more rounds of testing were carried out on the feature matching program.

The computer vision aspects of the project were implemented using OpenCV. OpenCV is a computer vision and machine learning software library. It was created with the intention of providing open source implementations of both basic and high-level algorithms, in order to advance computer vision research as a whole (Pulli et al., 2012). The OpenCV library is cross-platform and provides all the necessary algorithms for the implementation of this project.

3.1 Histogram Comparison Implementation

The first program attempts to locate the logo in the inputted image and compare it to a stored template image of the logo, which corresponds to the account of the laptop-owner in question. It does so using the techniques described in this section. The test input shown in Figure 3.1 will be used to illustrate this process.

Figure 3.1 *Example test input*

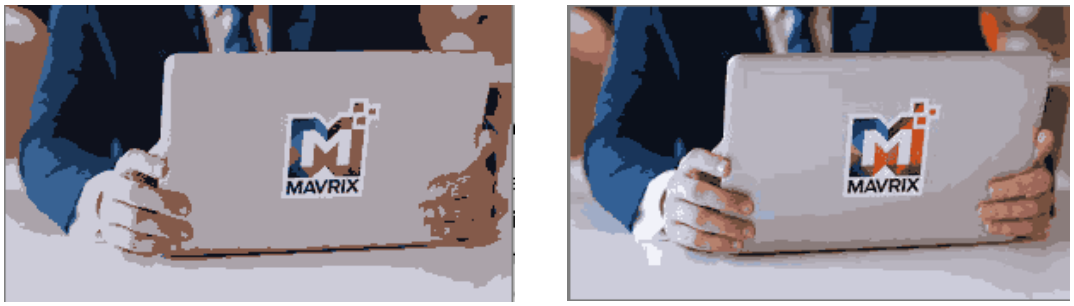
3.1.1 K-means Clustering

Before an attempt is made to locate the logo, some pre-processing techniques are applied to the input image. The first of these techniques is k-means clustering. This is a method of reducing the amount of colours in an image by clustering similar colours as one (Dawson-Howe, 2014). Given that the colour value of two pixels can vary by very small amounts, it can be useful to assign the same colour value to pixels with similar values. This can lead to effective segmentation of an image into its component parts. The inputted value, k, affects the amount of colours which will represent the entire image. For example, the image shown in Figure 3.2 (a) was clustered using 6 starting exemplar colours, whereas the image in Figure 3.2 (b) was clustered using 15 starting exemplar colours.

The algorithm works as follows:

- The first pass iterates through every pixel and assigns each one to the closest exemplar colour. Each time an exemplar colour has a pixel assigned to it, its value is recalculated to be the centre of gravity of all of the pixels in that cluster.
- The second pass iterates through each pixel again, this time keeping the exemplar colour values as constant. It reassigns each pixel to the exemplar colour which it is closest to. These allocations may be different to those in the first pass due to the constant recalculation of exemplar colour values in the first pass (Dawson-Howe, 2014).

A value of 4 was inputted into the algorithm for k, in this program. The result of k-means clustering is shown in Figure 3.3 (b).



(a) *6 starting exemplar colours (k=6)* ***(b)*** *15 starting exemplar colours (k=15)*

Figure 3.2 *Comparing cluster images with different values for k*

(a) *Input image* ***(b)*** *Result Image*

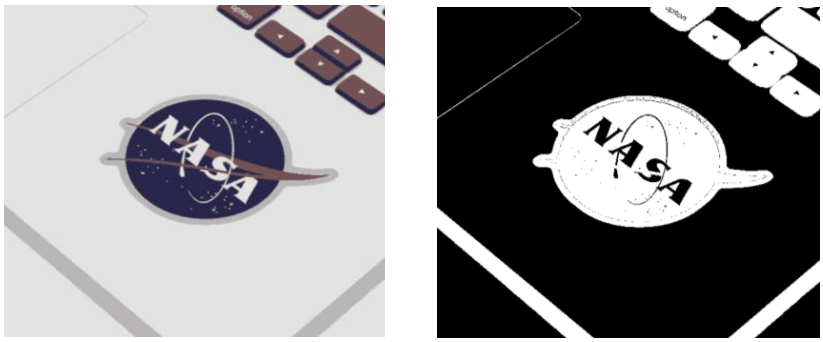
Figure 3.3 *K-means clustering*

3.1.2 Creating a Mask of the Laptop Region

With the assumption that the inputted photograph was taken so that the laptop takes up the majority of the screen, the largest region found by k-means clustering should be the laptop. With this in mind, a binary image mask can be created which displays “holes” in the laptop, shown in white. This idea is illustrated in Figure 3.4 (b). The largest of these white “holes” should be the logo sticker.

This mask can be created by converting the three-channel, colour image to a binary image, where all pixels are either white (foreground) or black (background). This binary image is created by assigning a value of 0 to each pixel which has the label of the most prominent exemplar colour (laptop pixel) and

a value of 255 to all other pixels. This example considers a single channel scale where a value of 0 is black, 255 is white and all values in between are a shade of grey.



(a) *Input Image* ***(b)*** *Result Image*

Figure 3.4 *Creating a binary image mask to display the "holes" in the laptop*

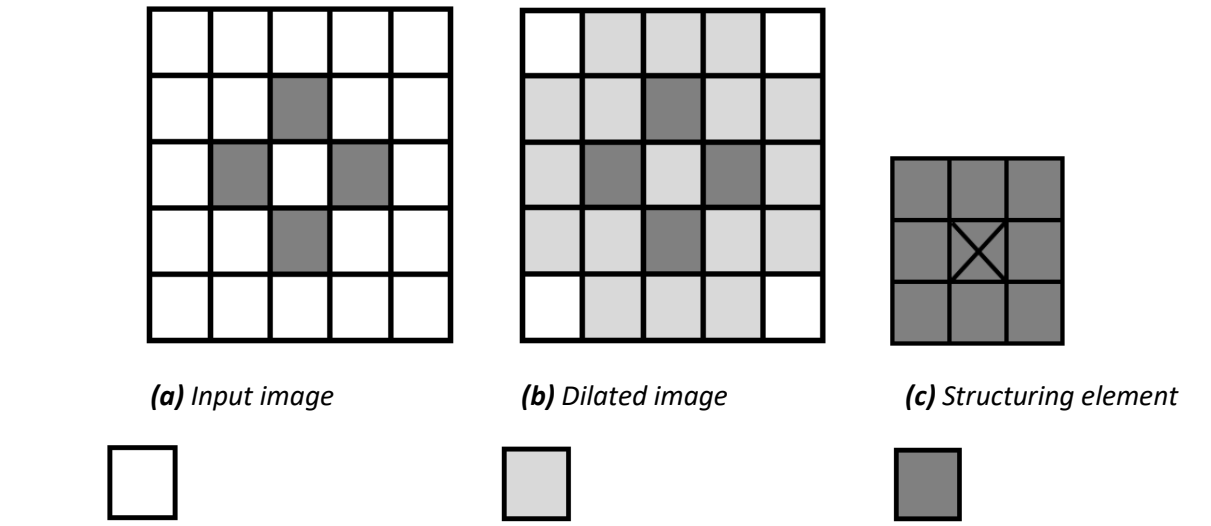
3.1.3 Mathematical Morphology - Opening and Closing

Mathematical morphology describes a set of operations which involve placing a structuring element in all possible positions in the image and performing a logical test between the pixels in the structuring element and the pixels in the current section of the image (Najman and Talbot, 2010). Examples of these operations will follow.

Opening and closing are useful techniques for enhancing binary images before attempting to extract its regions. These two techniques are composed of two morphological operations, dilation and erosion.

Dilation

Consider the 5x5 pixel binary image in Figure 3.5 (a), where foreground pixels are shown in grey and background pixels are shown in white. Additionally, consider the 3x3 structuring element in figure 3.5 (c), where all pixels are foreground pixels. Placing the centre of the structuring element, marked with an X, at each of the foreground pixels in (a); if any of the surrounding points are not foreground, they will become foreground in the output dilated image (Dawson-Howe, 2014). The result of performing this logical test is shown in Figure 3.5 (b).



(a) *Input image* ***(b)*** *Dilated image* ***(c)*** *Structuring element*

(d) *Background pixel* ***(e)*** *Foreground pixel after dilation* ***(f)*** *Original foreground pixel*

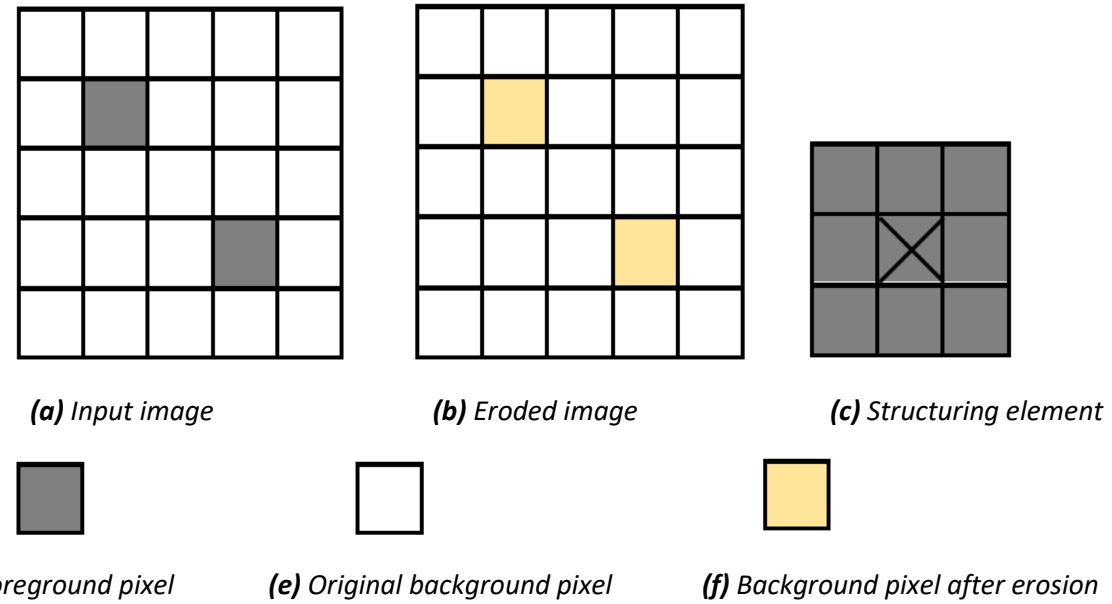
Figure 3.5 *Illustration of dilation*

Erosion

Now, consider the 5x5 binary image in Figure 3.6 (a) and the 3x3 structuring element in Figure 3.6 (c). Placing the centre of the structuring element, at each of the foreground pixels in Figure 3.6 (a); if there is not a perfect logical match i.e. if any of the surrounding pixels are background, the current point at the centre of the structuring element will be recorded as a background pixel in the outputted image (Dawson-Howe, 2014). The result of erosion is shown in Figure 3.6 (b).

Closing

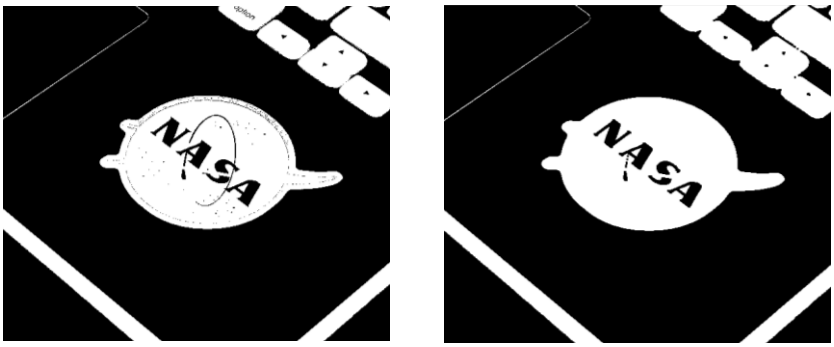
The closing technique involves performing a dilation followed by an erosion. This operation was carried out on the inputted binary image in order to fill in small holes and bridge gaps. The result of this effect can be observed in Figure 3.7 (b). Notice that the gaps inside the logo, such as the outer rim, have been bridged. This makes the region more connected and more of an individual component.



(a) *Input image* ***(b)*** *Eroded image* ***(c)*** *Structuring element*

(d) *Foreground pixel* ***(e)*** *Original background pixel* ***(f)*** *Background pixel after erosion*

Figure 3.6 *Illustration of erosion*



(a) *Input image* ***(b)*** *Result image*

Figure 3.7 *Closing*

Opening

An opening involves performing an erosion followed by a dilation. This operation is performed in order to remove small regions of noise that are too small to be of interest. This effect can be observed in Figure 3.8 (b). Notice the removal of the trackpad line at the top left of the image. This

will result in less irrelevant regions being found during connected components analysis in the next stage.

(a) *Input image* ***(b)*** *Result image*

Figure 3.7 *Opening*

3.1.4 Connected Components Analysis

Connected components analysis identifies regions in a binary image by assigning a label to all pixels that are connected.

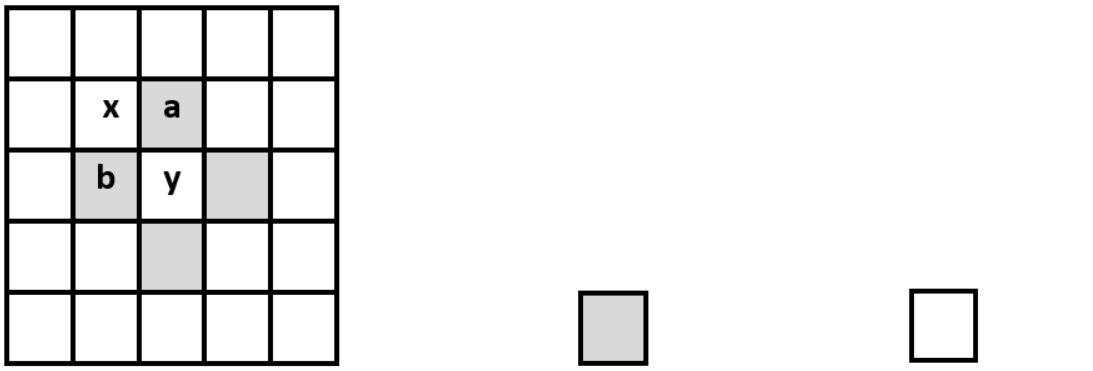
Defining the connectedness of pixels can be done using either 8-adjacency, where all 8 neighbours are considered to be connected, or 4-adjacency, where only north, south, east and west pixels are considered to be connected (Dawson-Howe, 2014). The choice of whether to use 8-adjacency or 4-adjacency presents some connectedness problems.

Consider the 5x5 binary image in Figure 3.8. If 8-adjacency was used to define connectedness, the pixels, *x* and *y*, would be considered connected, whereas they shouldn't be. If 4-adjacency was used, the pixels, *a* and *b*, would not be considered connected, whereas they should be. These problems can be overcome by using a combination of 8 and 4-adjacency. The OpenCV implementation of connected components analysis overcomes this problem by only finding connected foreground regions and not considering background regions (Dawson-Howe, 2014).

The algorithm works as follows:

- The first pass iterates through the image row by row, column by column.
  For each foreground pixel:
  If all of its previous neighbours are background pixels:
  A new label is assigned to that pixel.
  Otherwise:
  The label of one of its previous neighbours is assigned to that pixel. If any of its previous neighbours have differing labels, these labels will be marked as equivalent.

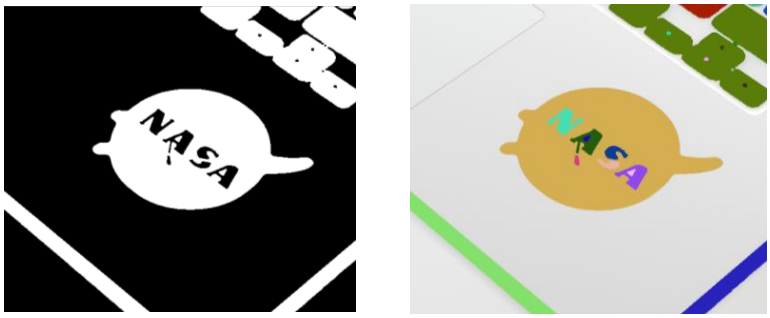
- The second pass iterates through the image again and sets all equivalent labels to be the same label (Dawson-Howe, 2014).



(a) *5x5 binary image* ***(b)*** *Foreground pixel* ***(c)*** *Background pixel*

Figure 3.8 *An illustration of connectedness problems*

Each connected component found in the inputted binary image is illustrated by a new colour shown in Figure 3.9 (b).



(a) *Input image* ***(b)*** *Result image*

Figure 3.9 *Connected components analysis*

3.1.5 Extracting the Logo Region

As is observable in Figure 3.9 (b), the largest region found by connected components analysis should be the logo. For this reason, the remaining regions were analysed and the one with the greatest area was labelled as the logo. The area of a region can be calculated by simply counting all of the pixels which are bounded by the outer contour of the region. This can also be done in a less computationally expensive way, by using the following formula:

$$area = |\sum_{k=1}^{n}(i_{k+1}.j_k - i_k.j_{k+1})|$$

Given n vertices of the region, where (i_k, j_k) is the current vertex (Dawson-Howe, 2014).

3.1.6 Creating a Bounding Rectangle

A subimage, which contains only the logo will be histogrammed, and compared to the template image of the logo. In order to create this, it is necessary to define the rectangle which will be the outer boundaries of the subimage. In this example, the rectangle needed is the one which bounds the logo region within it.

To define a rectangle, which bounds a region, two coordinates are required, the top left and the bottom right point of the rectangle. These two points can be attained using the following algorithm:

- Iterate through all points in the outer contour of the region:
  - To find the top-left point of the rectangle (x_1, y_1):
    - x_1: use the x-coordinate of the left-most point (the smallest x-coordinate)
    - y_1: use the y-coordinate of the top most point (the smallest y-coordinate)
  - To find the bottom-right point of the rectangle (x_2, y_2):
    - x_2: use the x-coordinate of the right-most point (the greatest x-value)
    - y_2: use the y-coordinate of the bottom-most point (the greatest y value)

An illustration of the bounding rectangle is drawn onto the original test input image, shown in Figure 3.10.

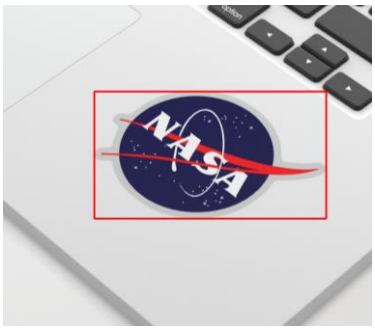
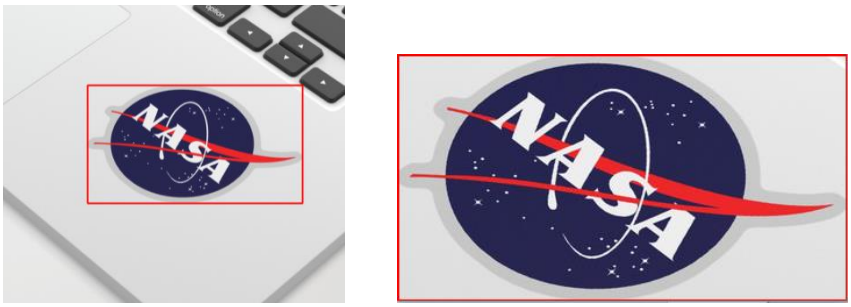


Figure 3.10 *Bounding Rectangle*

3.1.7 Creating a Subimage

A new image, of the same dimensions as the previously found bounding rectangle, is created. The colour value of each pixel within the rectangle in the input image is copied to the result image. The resulting subimage is shown in Figure 3.11.



(a) Input image *(b) Result image*

Figure 3.11 *Creating a subimage*

3.1.8 Histogram Comparison

In order to perform histogram comparison, it is first necessary to create a colour histogram of the two images being compared. The images being histogrammed are shown in Figure 3.12. A common method of representing colour data is by storing a number for the amount of red, green and blue in a particular colour (RGB). A histogram can be created using the following algorithm:

- Create a 3-dimensional histogram of the RGB channels. Maintaining a bin for every possible RGB value would amount to millions of bins. A simpler 8x8x8 histogram can be used instead.
- Initialise all bins to have a count of 0.
- Iterate through every pixel and add 1 to the bin which corresponds to the colour of the current pixel (Dawson-Howe, 2014).

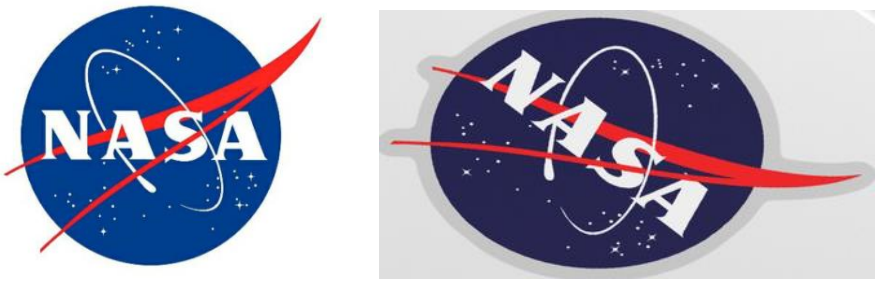
The two histograms can then be compared using a number of methods. The metric used in this project was Correlation:

$$Correlation(h_1, h_2) = \frac{\sum_i \left(h_1(i) - \overline{h_1}\right) . \left(h_2(i) - \overline{h_2}\right)}{\sqrt{\sum_i \left(h_1(i) - \overline{h_1}\right)^2 . \sum_i \left(h_2(i) - \overline{h_2}\right)^2}}$$

Where:

- $\overline{h_1} = \frac{\sum_i (h_1(i))}{N}$
- N is the number of bins in the histogram.
- $h_1(i)$is the i^{th} colour bin of the histogram h_1.

The result of this calculation will be a number between 0.0 and 1.0 and can act as a *matching score* corresponding to the two images.



(a) *Template image* ***(b)*** *Processed input image (subimage)*

Figure 3.12 *The images that will be histogrammed and compared*

3.2 Feature Matching Implementation

The second solution that was implemented, to solve the verification problem, involves feature matching. A *feature* or a *keypoint* is a corner that has been detected in an image. A keypoint *descriptor* is a complex description of the gradients of all of the pixels in the local area around a keypoint. Comparisons can be made between keypoint descriptors in two different images to determine whether an element of one image is present in another. This type of descriptor matching will be used to determine whether the template logo is present in the image inputted by the user.

3.2.1 Greyscaling

A greyscale image only stores the value of the luminance channel for each pixel. The input images are first converted to greyscale. This is a necessary step because the corner detector which will be used in the next step, detects corners based on luminance values alone. That is, it compares different pixels, to assess their cornerness, based on just one channel, rather than on all three RGB channels. An RGB image can be converted to greyscale using the following formula:

$$Luminance = 0.299R + 0.587G + 0.114B$$

Figure 3.13 shows the result of greyscaling the input images.

(a) *Input image* ***(b)*** *Result image*

Figure 3.13 *Greyscaling*

3.2.2 Feature Matching

As mentioned, features or keypoints, are corners found in an image. After finding a keypoint, a keypoint descriptor can be derived, which is a description of the area around the keypoint. These keypoint descriptors can then be compared and matched with other descriptors from different images (Dawson-Howe, 2014).

A traditional corner detector such as the Features from Accelerated Segment Test (Rosten, 2006) works as follows. Consider the circle of pixels, in Figure 3.14, with a radius of 3 around the central pixel, shown in blue. If an arc of 9 pixels, shown in orange, exists which are brighter or darker than the central pixel by a certain threshold, a corner is detected at the location of the central point.

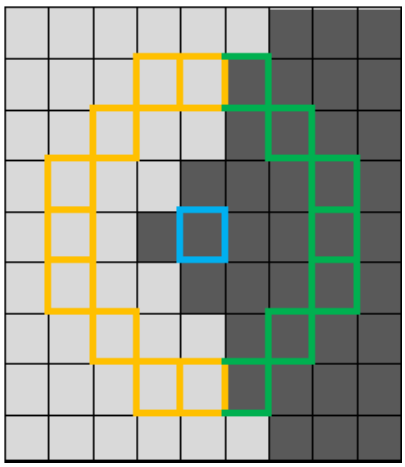


Figure 3.14 *Illustration of how a corner detector works*

While detecting corners in an image, an issue can arise which is illustrated in Figure 3.15. When the area inside the red circle in Figure 3.15 (a) is analysed, a corner is detected. When the shape is scaled

up, as in Figure 3.15 (b), an edge is detected instead of a corner. Scale-invariant Feature Transform (SIFT), solves this problem by detecting corners in an image at a range of different scales.

Figure 3.15 *An illustration of a corner detection error*

SIFT works by, first, creating a number of octaves of the image at different scales. Different level of blur is applied to each of the images in an octave. Difference images (subtracting one image from another) are created from the blur images in each octave. Local minima and maxima are then taken from these difference images and marked as keypoints. Effectively, the algorithm extracts keypoints which are significant at all levels of scale and all levels of blur. Figure 3.16 displays an image with the keypoints that have been found, marked by coloured dots.

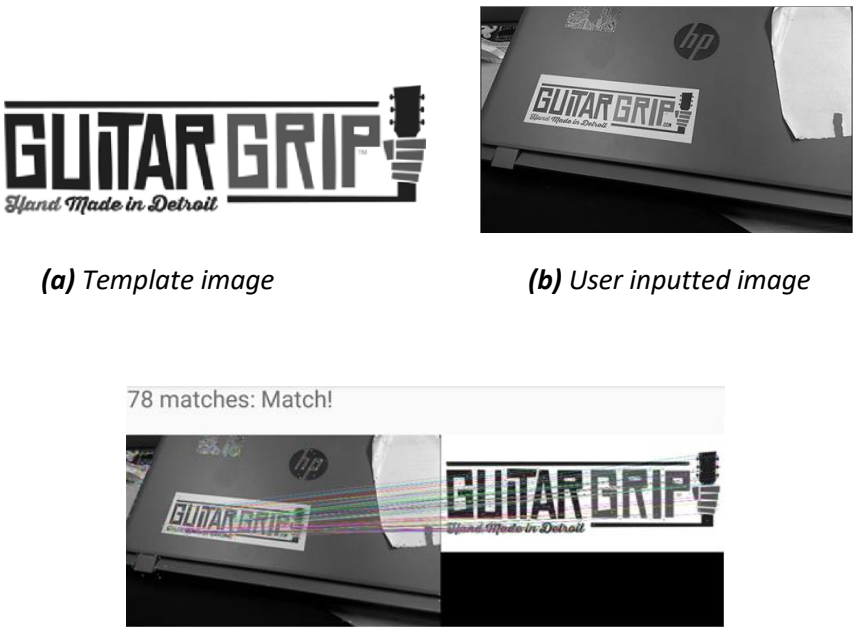
Figure 3.16 *The keypoints found by SIFT*

Keypoint descriptors can then be created, which describe the gradient of the points in the area around the keypoint. These descriptors can be compared to other descriptors in other images.

The implemented program detects descriptors of the user inputted image and the template logo image. Matches are found between the descriptors of the two images. The difference between two keypoint descriptors in a match can be expressed as the Euclidian distance, therefore the *quality* of a match is determined by how small its Euclidean distance is. Many matches are likely to be found in this process, so in order to discard low quality matches, the array of matches is iterated through and a match is discarded unless it is less than a certain proportion of the distance of the following match:

```
if (match1.distance > match2.distance * multiplier_parameter) {
      discard match1;
}
```

The optimum value for the variable: *multiplier_parameter*, seen above, was found during the testing period which will be discussed in the next section. Following this, the number of remaining matches is compared to the minimum threshold for the number of matches that warrants a positive result and the program outputs this result. Figure 3.17 displays a test input which the program has correctly outputted as a positive result.



(a) *Template image* ***(b)*** *User inputted image*

(c) *Result image*

Figure 3.17 *Correctly classified test input*

3.3 Testing

Before the testing process began, a dataset of test inputs was compiled. The dataset was compiled of images taken by classmates of the author and images found online. The entire data set was comprised of equal parts positive and negative test inputs. Positive test inputs are those which should be returned as positive i.e. if the template logo is present in the input image. Conversely, negative test inputs are those which should be returned as negative i.e. if the logo is not present in the input image.

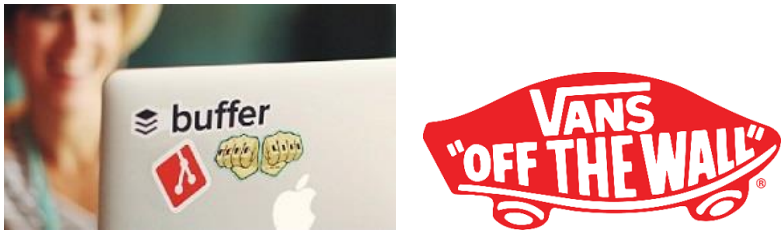
(a) *Positive test input*

(b) *Negative test input*

Figure 3.18 *Illustration of a positive and negative test input*

Half of the negative test inputs used were artificially created by removing the object of interest from the input image. The other half of the negative tests simply involved comparing a template image to a completely different input image. Both of the test inputs in Figure 3.19 should return a negative result.

(a) *Artificially created negative input*



(b) *Normal negative test input*

Figure 3.19 *Illustration of negative test input types*

3.3.1 Threshold Optimisation

Before comparing the two implementation methods, testing was carried out on both programs in order to optimise their input parameters. For the histogram comparison program, the parameter being optimised was the *minimum_matching_score*. The *matching_score* is a rating of the similarity of the two image's histograms. The *minimum_mathcing_score* is the minimum threshold required for a positive result to be returned. For example, if the *matching_score* was 0.8 and the *minimum_matching_score* was 0.75, the inputted image will be successfully verified.

For the feature matching program, the parameter being optimised was the *minimum_number_of_matches*. This is the minimum number of keypoint matches required for a positive result to be returned. For example, if 17 matches are found and the *minimum_number_of_matches* is 12, the inputted image will be verified to contain the logo.

The two programs were run on 15 test inputs, using a range of different values for the input parameters mentioned. The histogram comparison program was run 30 times where the *minimum_matching_score* parameter was in the range of 0.70 – 0.99. The feature matching program was run 16 times where the *minimum_number_of_matches* parameter was in the range of 1-16.

Performance Metrics

A number of performance metrics were calculated for the programs during the testing process. All of these performance metrics are based on the following 4 numbers:

- True positives (TP): the number of test inputs that have been correctly identified as positive.
- False positives (FP): the number of test inputs that have been incorrectly identified as positive.
- True negatives (TN): the number of test inputs that have correctly been identified as negative.
- False negatives (FN): the number of test inputs that have incorrectly been identified as negative.

The main performance metrics which were used in the testing process are as follows:

1. Recall (True Positive Rate) is the percentage of positive inputs which have been successfully located.

$$Recall = \frac{TP}{TP + FN}$$

2. Specificity (True Negative Rate) is the percentage of negative inputs which have been successfully located.

$$Specificity = \frac{TN}{FP + TN}$$

3. Precision is the percentage of positive classifications which are correct.

$$Precision = \frac{TP}{TP + FP}$$

4. Accuracy is the percentage of total samples which are correct.

$$Accuracy = \frac{TP + TN}{Total\ Samples}$$

5. F_β is a weighted measure of both precision and recall. It can be used to put different emphasis on precision and recall depending on the domain of an application. For example, F_1 puts equal weighting on precision and recall, $F_{0.5}$ puts higher weight on recall and F_2 puts higher weight on precision. The formula is as follows:

$$F_{\beta} = (1 + \ \beta^2) \ . \ \frac{Precision \ . \ Recall}{(\beta^2 \ . \ Precision) + Recall}$$

(Altman and Bland, 1994; Dawson-Howe, 2014).

Regarding the F_{β} measure, in the context of this application, a slight bias is put on precision over recall. A situation where a laptop-owner is verified when they shouldn't be, is unfavourable. For example, if they are not advertising as intended, but the system thinks that they are. If the system is returning false negatives, however, i.e. the laptop-owner is advertising as intended but the program can't recognise this, the user may complain to a human administrator who will review the case. The second type of error, described here, is more favourable and therefore a slight emphasis is put on precision over recall. A β value of 0.9 is used to reflect this:

$$F_{0.9} = (1 + \ 0.9^2) \ . \ \frac{Precision \ . \ Recall}{(0.9^2 \ . \ Precision) + Recall}$$

For the first round of testing, the two programs were run where the input parameters were equal to all of values in the previously mentioned ranges. Precision and recall of the programs were automatically calculated each time the programs were run and the results were outputted to an excel file. A sample of the outputted data is shown in Figure 3.20, and the entirety of the outputted data is displayed in Appendix 1.1.

Threshold	Precision	Recall
0.77	0.5	0.25
0.78	0.5	0.25
0.79	0.666667	0.25
0.8	0.666667	0.25
0.81	0.666667	0.25
0.82	0.666667	0.25
0.83	0.666667	0.25

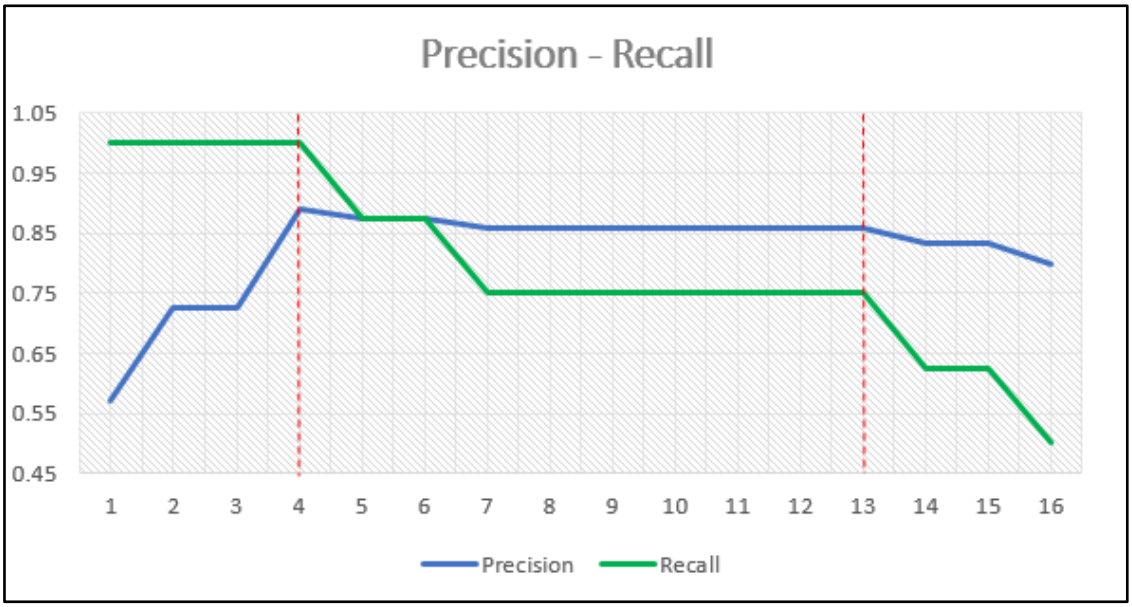
(a) *Histogram comparison sample data*

Threshold	Precision	Recall
1	0.571428571	1
2	0.727272727	1
3	0.727272727	1
4	0.888888889	1
5	0.875	0.875
6	0.875	0.875
7	0.857142857	0.75

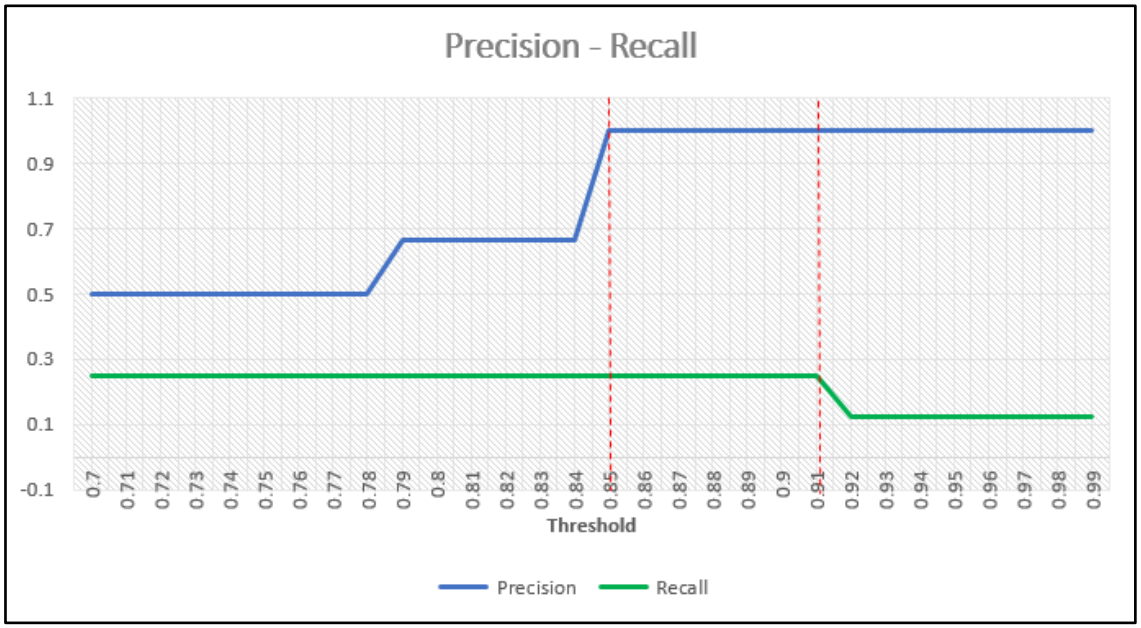
(b) *Feature matching sample data*

Figure 3.20 *Threshold optimisation: Precision-recall data*

A graphical summary of the precision-recall data for both programs can be observed in Figure 3.21.



(a) *Feature matching program*



(b) *Histogram comparison program*

Figure 3.21 *Precision-recall graphs*

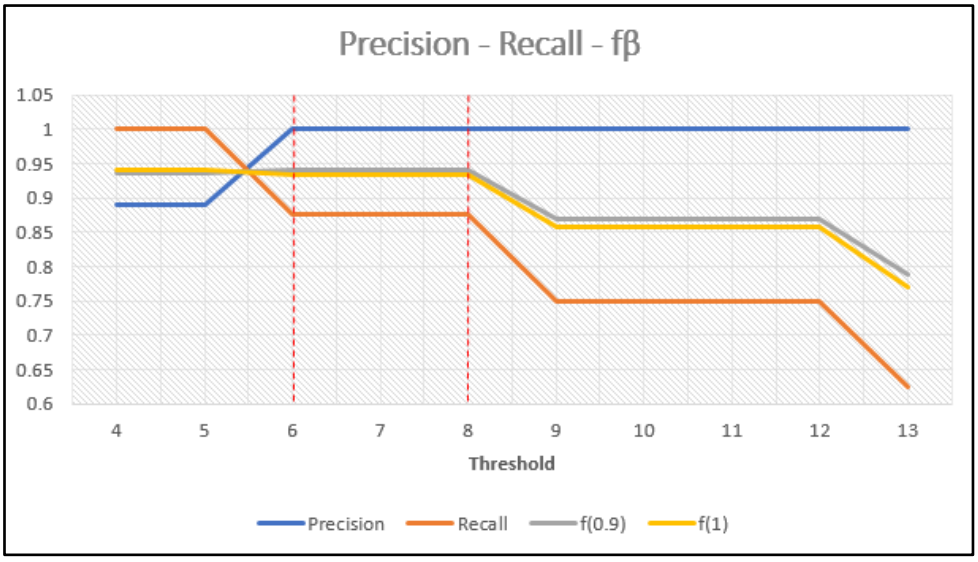
This data was used to derive a relevant range of values for the variable input parameters, within which the value of $F_{0.9}$ would be calculated. The result of the $F_{0.9}$ calculation will be the primary method of choosing the optimum value for the thresholds in question: *minimum_matching_score* and *minimum_number_of_matches*.

Feature Matching Program Optimum Threshold

From observing the graph in Figure 3.21 (b), it can be seen that when the threshold is below 4 or above 13, precision and recall, fall steeply. The feature matching program was run an additional 10 times where the value of the threshold was in the range of 4-13. $F_{0.9}$ was calculated each time. The results were outputted to an excel file and graphed. A sample of the output data can be seen in Figure 3.22 (a) and full details are recorded in Appendix 1.2. The graph of this data can be observed in Figure 3.22 (b).

			fβ	
Threshold	**Precision**	**Recall**	***f(0.9)***	***f(1)***
4	0.888888889	1	0.935400517	0.941176471
5	0.888888889	1	0.935400517	0.941176471
6	1	0.875	0.939910979	0.933333333
7	1	0.875	0.939910979	0.933333333
8	1	0.875	0.939910979	0.933333333
9	1	0.75	0.870192308	0.857142857

(a) *Feature matching program F_β sample data*



(b) *Feature matching program F_β graph*

Figure 3.22 *Threshold optimisation: F_β data*

The threshold that is chosen to be the optimum value was the one which corresponds to the maximum value of $F_{0.9}$ and precision. These metrics are both maximised within the range of 6-8. The median value of this range, 7, was therefore chosen as the optimum threshold.

Histogram Comparison Program Optimum Threshold

Observing Figure 3.21 (b), it can be seen that precision and recall, for the histogram comparison program, are both at their maximum when the threshold value is between 0.85 and 0.91. The $F_{0.9}$ value would be calculated in this range, however, the precision and recall do not change between 0.85 and 0.91, so the $F_{0.9}$ value will remain constant, at 0.43. The median value of the range, 0.88, is therefore chosen as the optimum threshold.

3.3.2 Comparing the Programs

With the input parameters optimised, the additional performance metrics, specificity and accuracy are calculated for each program. Based on the results of these calculations, which are summarised in Figure 3.23, the feature matching program was chosen for the main application. The inputs and results of all tests in this phase of testing can be seen in Appendix 2.1.

				fβ			
Implementation	**Threshold**	**Precision**	**Recall**	***f(0.9)***	***f(1)***	**Accuracy**	**Specificity**
Histogram Comparison	0.88	1	0.25	0.426887	0.4	0.6	1
Feature Matching	8	1	0.875	0.939910979	0.933333333	0.933333333	1

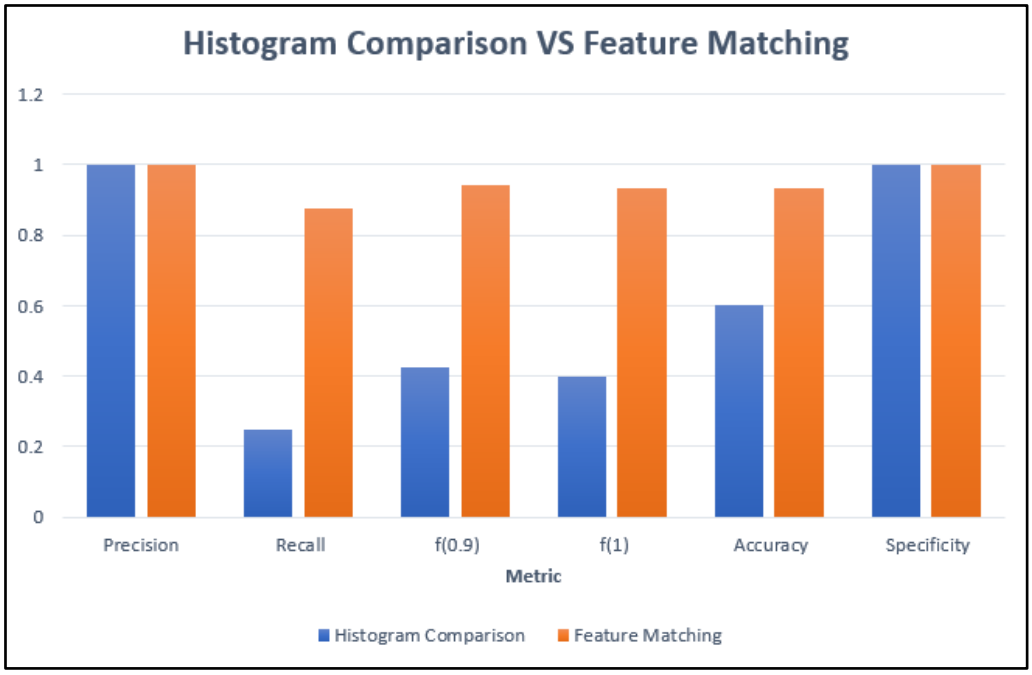


Figure 3.23 *Comparison of the programs*

3.3.3 Additional Threshold Optimisation

Further testing was carried out on the feature matching program only. From manual observation of the results of an additional 21 tests, it became clear that there was a problem involving the discovery of false positives with the program's current configuration. The example shown in Figure 3.24 displays a test input which is correctly returned as positive. This is only the case, however, because the *quality* of the matches accepted by the program was low. This meant that it would return almost every input as positive.

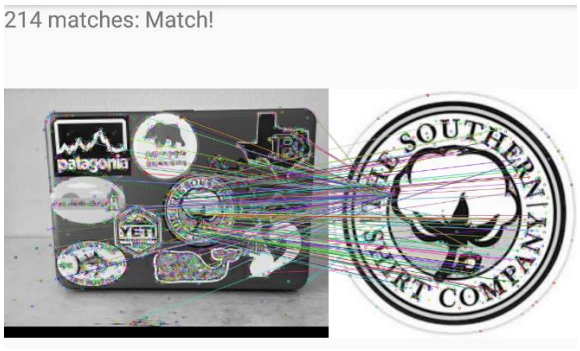


Figure 3.24 *Low quality of matches accepted, results in false positives*

The input parameter which determines the *quality* of matches accepted by the program is the *multiplier_parameter*. The array of matches is iterated through, and a match is discarded unless it is less than a certain proportion of the distance of the next match:

```
if (match1.distance > match2.distance * multiplier_parameter) {
      discard match1;
}
```

With the intention of decreasing false positives, while keeping the true positives rate unaffected, the program was run on an array of test inputs, using a range of values for the *multiplier_parameter*. This was to discern the optimum value for this parameter.

Additional Note on Performance Metrics

Before attempting to optimise the value of the *multiplier_parameter*, the performance of the program in its current configuration should be recorded, so that improvement can be quantified. Where the

multiplier_parameter is equal to 0.7 and *minimum_number_of_matches* is equal to 7, the performance values were as follows:

- Recall = 1
- Precision = 0.76
- Accuracy = 0.81
- Specificity = 0.5
- $F_{0.9}$ = 0.851

The current problem with the program relates to false positives and a lack of true negatives. In order to improve on it, by optimising the *multiplier_parameter*, it would be desirable to devise an additional metric which takes these factors into account. This new metric should, in a single number, encapsulate:

1. The percentage of positive classifications which are correct (precision)
2. The percentage of negative inputs which have been successfully located (specificity)

To encapsulate these two factors, let a slight variation of F_β be defined as:

$$N_\beta = (1 + \beta^2) \cdot \frac{Precision \cdot Specificity}{(\beta^2 \cdot Precision) + Specificity}$$

Putting equal weights on precision and specificity:

$$N_1 = 2 \cdot \frac{Precision \cdot Specificity}{Precision + Specificity}$$

Using the current configuration of input parameters:

- N_1 = 0.6

With the aim of maximising N_1, while leaving other metrics largely unaffected, the program was run on the test set 9 times where the *multiplier_parameter*, was in the range of 0.30 - 0.70. The results, which were outputted to an excel file and graphed, can be viewed in Figure 3.25.

Threshold	Precision	Recall	*F(0.9)*	Accuracy	Specificity	*N(1)*
0.3	1	0.1875	0.334433	0.638889	1	1
0.35	1	0.25	0.426886792	0.666666667	1	1
0.4	1	0.375	0.558863	0.722222	1	1
0.45	1	0.5	0.690839695	0.777777778	1	1
0.5	1	0.625	0.780516	0.833333	1	1
0.55	1	0.75	0.870192308	0.888888889	1	1
0.6	0.888889	0.8125	0.844334	0.861111	0.9	0.894366
0.65	0.7777778	0.875	0.818475452	0.833333333	0.8	0.7887324
0.7	0.666667	0.9375	0.792617	0.805556	0.7	0.683099

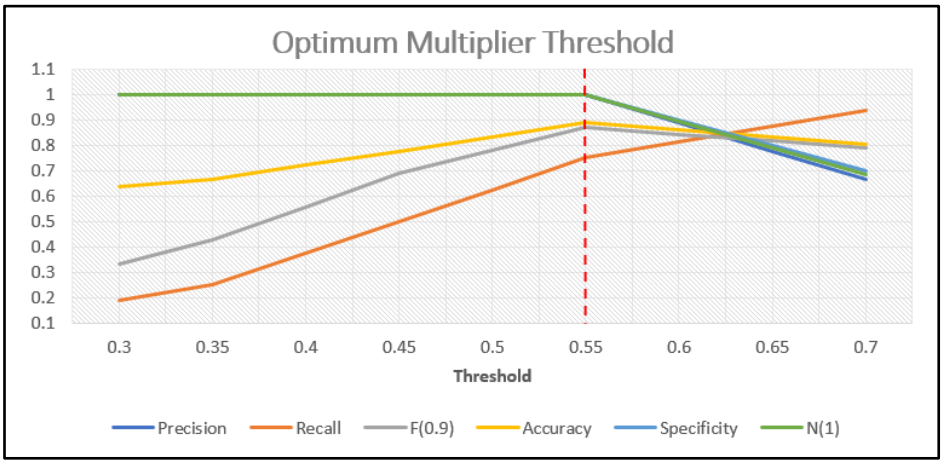


Figure 3.25 *Further threshold optimisation: multiplier_parameter*

From observing Figure 3.25, it can be seen that N_1 is maximised in the range of 0.30 – 0.55. The chosen value for the *multiplier_parameter*, within this range, was the one which maximises all other metrics, 0.55. The tests from the second round of testing were carried out again, with the optimised input parameters, to assess whether there had been an improvement. Figure 3.26 shows the improved output of the example test input discussed previously. A lower volume and higher quality of matches are now accepted, with the majority of the matches localised to the correct logo.

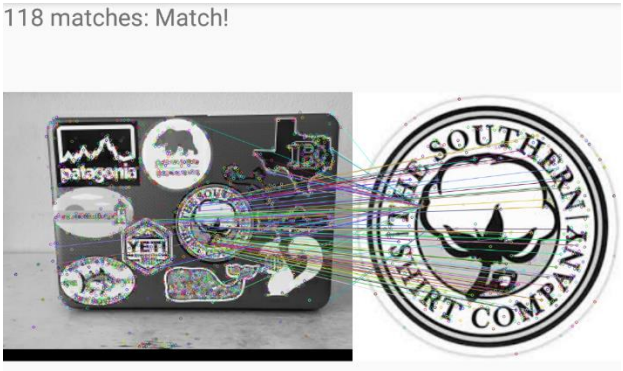


Figure 3.26 *Improvement from increasing quality of matches accepted*

This improvement is also reflected in the performance metrics:

- Recall = 0.923
- Precision = 0.923
- Accuracy = 0.905
- $F_{0.9}$ = 0.923
- Specificity = 0.875
- N_1 = 0.898

An increase in N_1, 0.6 to 0.89, can be observed, as well as an increase in almost all other metrics. One more round of testing, involving 17 inputs, was carried out on the program in its final configuration. The performance metrics for the entire test set were calculated as follows:

- Recall = 0.8
- Precision = 0.952380
- Accuracy = 0.838
- $F_{0.9}$ = 0.878
- Specificity = 0.9167

Thresholds							
matches	*multiplier threshold*	Recall	Precision	Accuracy	Specificity	*f(0.9)*	*n(1)*
7	0.55	0.8	0.95238	0.838	0.9167	0.878	0.9342

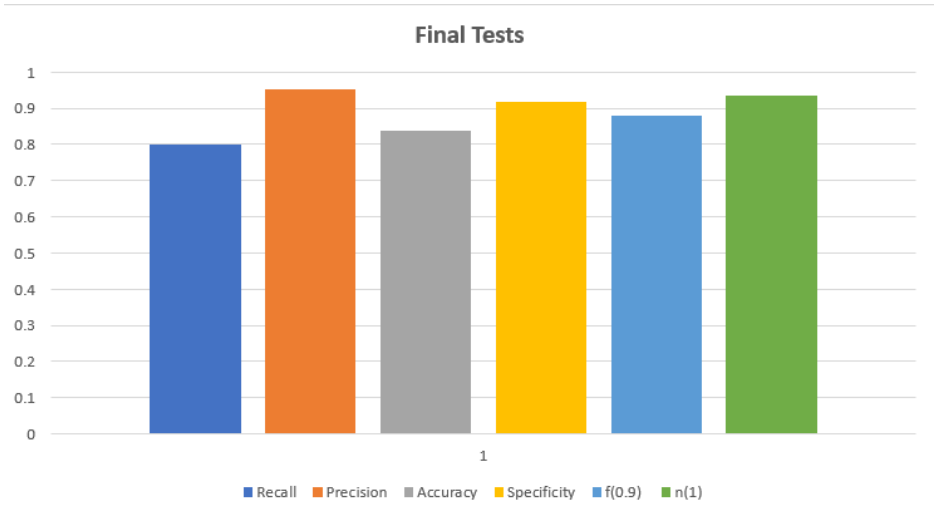


Figure 3.27 *Final test results of the verification program*

A discussion of results, improvements and future work, relating to the verification program, is included in Chapter 7. The output images from the further rounds of testing on the feature matching program are included in Appendix 2.2.

Chapter 4 – Database

The second functional requirement to be addressed was the design and implementation of a database to store necessary information. The design of the database was aided by the techniques: entity relationship mapping and database normalisation. The database was implemented using MySQL, PHP and 000webhost. These techniques will be discussed in detail in this chapter.

4.1 Design

The first stage of designing the database involved identifying the different entities of the system. The two main entities on which information would need to be stored were the laptop-owner and the company. For this reason, the decision was made to design a separate table for each of these entities.

The next step was to identify the information fields that would comprise each table. For both types of user, a name, email, password and unique, automatically incrementing identification number would be stored. The laptop-owner table would also store a user's age range, occupation, address, profile picture and the location where they use their laptop most often. The address data is necessary in order to post logo stickers to laptop-owners. The additional user details stored in the company table were a description of the demographic which they wish to advertise with, and an image displaying the company logo that would be advertised.

In order to verify that the laptop-owner was advertising as intended, it is necessary for the system to retrieve a template logo image which is linked to their account. In the initial design of the database, the laptop-owner table had a field which would store the corresponding template image in each row. This approach stores a large amount of duplicate data, making it redundant. The redundancy was removed by instead storing the company identification number which corresponds to the laptop-owner and retrieving the necessary logo image from the company table instead. In this way, each logo image is only stored once, in the company table, and nowhere else.

4.1.1 Boyce-Codd Normal Form

The next step of the design process was normalisation of the database tables. Creating tables in Boyce-Codd Normal Form relates to a group of rules which prevent issues from arising when inserting and deleting values from a table (Codd, 1974). The steps to ensure that a table is in Boyce-Codd Normal Form are as follows:

- *1st Normal Form:* the domain of each attribute contains only atomic values. In the laptop-owner and company tables, all fields can only have a single value, satisfying 1st Normal Form.

- *2nd Normal Form:* the table should comply with 1st Normal Form and every non-key column must be fully functionally dependant on the primary key. The sample of the laptop-owner table shown in Figure 4.1 illustrates this. The primary key of the table is the unique identifier for each row, which in this case is the *id* column. A non-key attribute is any column which does not have a constraint that the value must be unique. The value of the *id* column is the determinant of the values in the *name* and *email* columns. Hypothetically, if the value in the *id* column occurred twice within the table, the values in the *name* and *email* columns for each of the repeating tuples must be the same. This example illustrates the functional dependencies, *id* → *name,* and *id* → *email,* however *id* duplicates would not occur in the real tables given that the primary key must be unique. All non-key attributes of the laptop-owner and company tables are functionally dependant on the primary key, satisfying 2nd Normal Form.

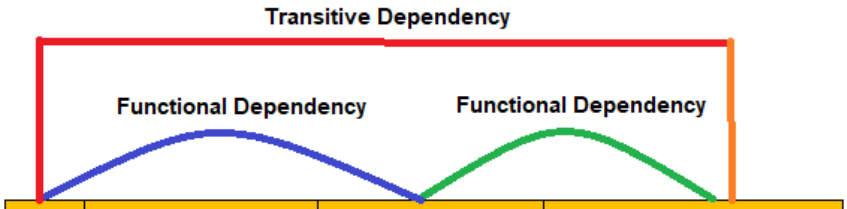
---

id	name	email
1	Puma	puma@gmail.com
2	Deliveroo	deliveroo@yahoo.ie
1	Puma	puma@gmail.com

Figure 4.1 *Illustration of functional dependency*

- *3rd Normal Form:* the table should comply with 2nd Normal Form and no non-key attributes can be transitively dependant on the primary key. The hypothetical sample of the laptop-owner table in Figure 4.2 illustrates a transitive dependency. The value in the *company_name* column displays the name of the company associated with the current laptop-owner. This value is functionally dependant on the *company_id* and the *company_id* is functionally dependant on the *id.* This means that the *company_name* is transitively dependant on the *id.* No non-key attributes in the laptop-owner or company tables are transitively dependant on the primary key. This satisfies 3rd Normal Form.

- *Boyce-Codd Normal Form:* for all functional dependencies, X → Y, that exist in the table; X must be a superkey. A superkey is a set of attributes in a relation whose combined values are unique for every row. Therefore, the superkey is a determinant of non-key attributes. In the table shown in Figure 4.3, an example of a functional dependency is: *name*, *address* → *email*. The combined values *Dónal Lowry* and *Calry, Sligo* are unique within the table. All functional dependencies in the laptop-owner and company tables contain a superkey as their determinant attribute. This satisfies Boyce-Codd Normal Form, which prevents database redundancy issues.



id	name	company_id	company_name
4	Larry Johnson	1	Puma
5	Mary Walsh	2	Deliveroo

Figure 4.2 *Illustration of transitive dependency*

id	name	address	email
1	John Walsh	4 Salthill, Galway	jw@gmail.com
2	Dónal Lowry	Calry, Sligo	lowry@yahoo.ie

Figure 4.3 *Illustration of a superkey*

4.1.2 Special Database Columns

The majority of the columns in the two tables relate to user profile details of a laptop-owner or company. There are a number of fields, however, which handle certain logistics of the system.

The first problem to overcome is determining which companies should be displayed in the different areas of the administrator profile, shown in Figure 4.4. In the company table, the *has_activated_case* field contains a boolean value which indicates whether or not a company has an activated case. The *has_live_campaign* field does the same for live campaigns. When a company with an activated case is converted to a live campaign by an admin, the *has_activated_case* field will be assigned a false value, and the *has_live_campaign* field will be assigned a true value. When both of these fields have a false value, the company in question has an open case. The values of these attributes are relevant when the system carries out a database call to determine which companies are displayed in the different areas of the administrator profile.

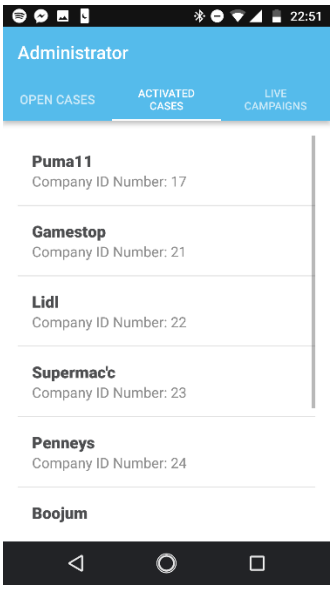


Figure 4.4 *The administrator profile*

The second problem, is dealing with messages being sent to laptop-owners to request that they join a campaign. If a request has been sent to a laptop-owner, by an administrator, the *request_id* field in the laptop-owner table will be assigned the identification number of the company involved in the campaign. If the laptop-owner chooses to accept this request, their *company_id* field is assigned the associated company's identification number and the *request_id* is assigned the default negative value of -1. The *company_id* is relevant when the system carries out a database call to retrieve the company logo during verification.

4.1.3 Entity Relationship Mapping

The diagram in Figure 4.5, illustrates the nature of the relationship between the two entities in the database, company and laptop-owner. The name of the relationship is defined in the diamond and roles of each entity, as well as their cardinality are also shown. The cardinality indicates that a laptop-owner can only have 1 company associated with it, whereas a company can have many laptop-owners associated with it. Additionally, the diagram displays the attributes of each entity, including the, underlined, primary key. The single lines connecting the two entities indicate that both parties may have partial participation in the relationship. This means that a laptop-owner or company may exist without being involved in a relationship. Total participation refers to a scenario where an entity cannot exist without being involved in a relationship, and is denoted by a double line.

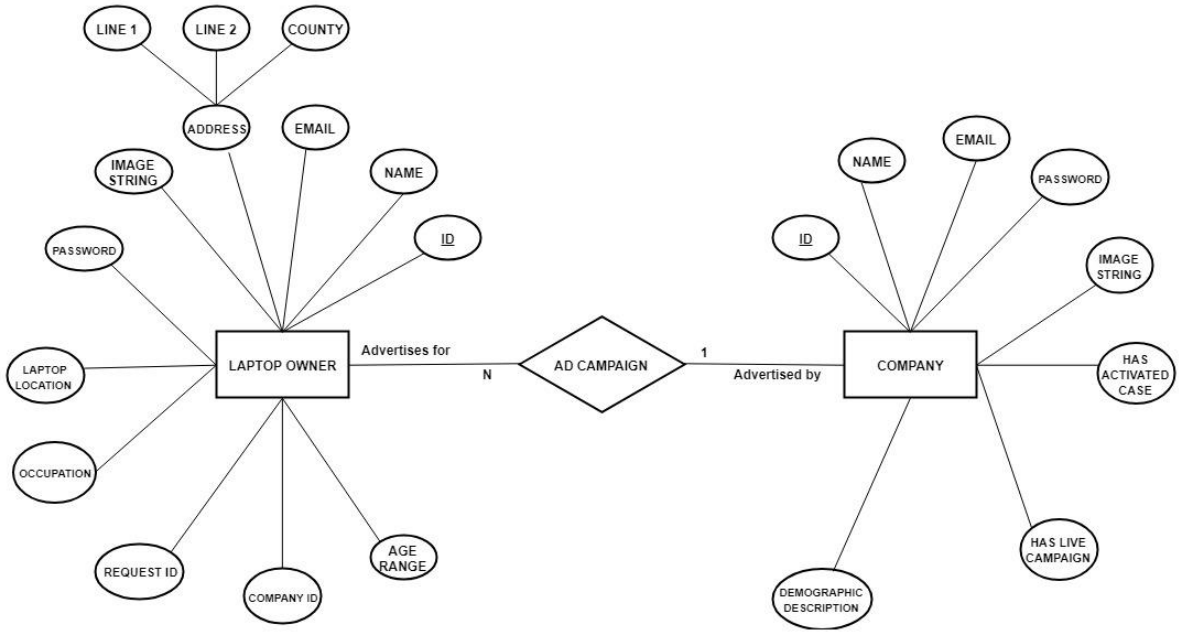


Figure 4.5 *Entity Relationship Diagram*

The entity relationship diagram was then mapped to the tables shown in Figure 4.6.

id	name	email	password	occupation	age_range	laptop_location	image_string

county	address_line_1	address_line_2	request_id	company_id

(a) *Laptop-owner table*

id	name	email	password	image_string	demographic_description	has_activated_case

has_live_campaign

(b) *Company table*

Figure 4.6 *Final table design*

4.2 Implementation

The following software was used to implement the designed database, containing a laptop-owner and a company table, with thirteen and eight columns respectively.

MySQL, an open source database management system, was used for the creation and querying of the database. MySQL uses Structured Query Language commands to create and alter tables. This database was hosted on an online server using a free service called 000webhost.

PHP is a scripting language which facilitates communication between the android application and the database. If a database query simply requires the retrieval of information, without passing any parameters, a GET request object must be created in an android java file. This request navigates to a webpage, which uses PHP to query the database and retrieve the specified information. An example of this type of query would be one which retrieves all laptop-owner rows in the database. If it is necessary to pass a parameter embedded in the request, a POST request must be used instead of a GET request. An example of this type of query would be one which retrieves only laptop-owners whose *company_id* field has a value of 17.

The images in the database were stored as Binary Large Objects, which are strings of binary data that can be used to store complex datatypes such as audio or multimedia.

Chapter 5 - Application Interface

The third functional requirement to be addressed was the design of the application interface. This chapter will include a demonstration, using screenshots, of how the system works. Following this, the interaction between the different classes involved in the system will be discussed.

5.1 User Perspectives

The application interface will be considered from the perspective of each of the three types of user: company, laptop-owner and administrator.

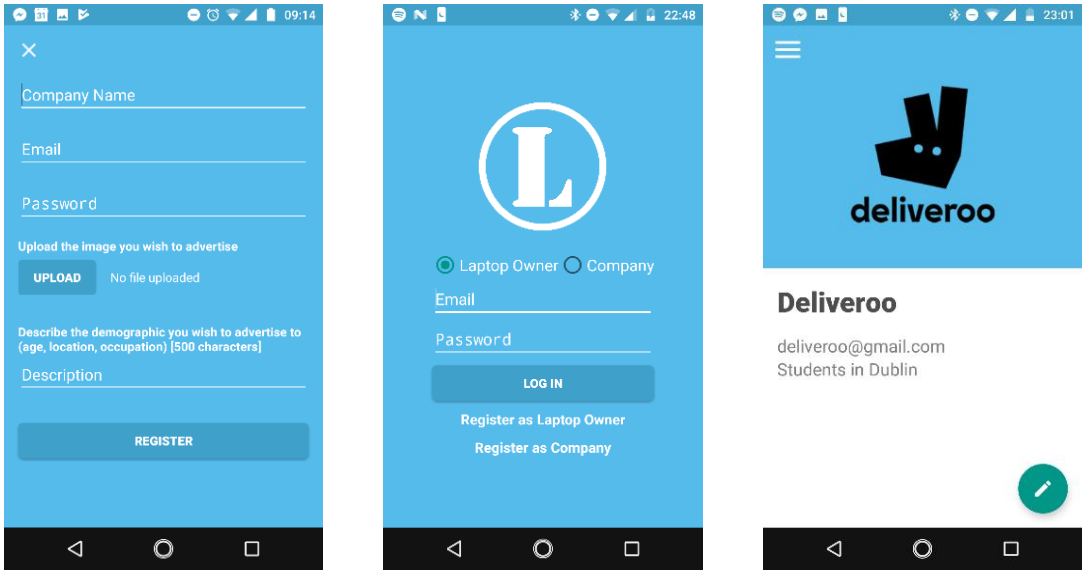
5.1.1 Company

The company user must input their details in the registration form shown in Figure 5.1 (a). The required details are the company name, email, password, a template image of the company logo and a textual description of the company's preference of demographic. These details are stored in the database. Following this, the company can sign into the application using the screen shown in Figure 5.1 (b). The user is then brought to their profile page which displays profile details and a green floating action button in the bottom right hand corner. Clicking this button allows the user to edit their profile details. Other aspects of the company user functionality are, as yet, undeveloped and will thus be addressed in future work.

5.1.2 Administrator

An administrator, who is assigned a special username and password, can sign into the application using the same login screen as the company user. This leads to the administrator profile, seen in Figure 5.2 (b), which is laid out in three separate tabs. The first tab displays a scrollable list of all of the open cases stored in the database. That is, all of the companies with no activated case or live campaign. If the administrator clicks on an open case list item, they will be shown the details of the open case (Figure 5.2 (c)). This page contains the company's identification number, name, email, demographic description and a floating action button which when clicked displays another button, observable in Figure 5.2 (d). When the *Activate Case* button is clicked, the database is updated to reflect this, and the administrator is brought to the screen in Figure 5.2 (e). Here, the administrator may browse a list of available laptop-owners and choose which ones to add to the activated case which has just been created. The administrator may then use the floating action button to send requests to the laptop-owners who have been chosen. If the laptop-owner accepts this request they will be added to the

activated case and eventually be part of the live campaign. If they reject the request they will not be added and the administrator may need to send more requests.



(a) **(b)** **(c)**

Figure 5.1 *Application interface: company perspective*

The company that has been activated is transferred to the list of activated cases, which is the second tab of the administrator profile (Figure 5.2 (i)). If the administrator clicks on an activated case list item, the activated case details will be displayed, as seen in Figure 5.2 (j). The activated case details simply display all of the laptop-owners who have accepted the request to join the campaign. The floating action button, in the bottom right corner, can be used to either send more requests or to convert the case to a live campaign. This case is then viewable in the third tab of the administrator profile, which displays all live campaigns. The live campaign list is not yet clickable and will be addressed in future work. If the application were to scale up to a high volume the administrator process would need to become automated. This issue will be discussed in more detail in the final chapter.

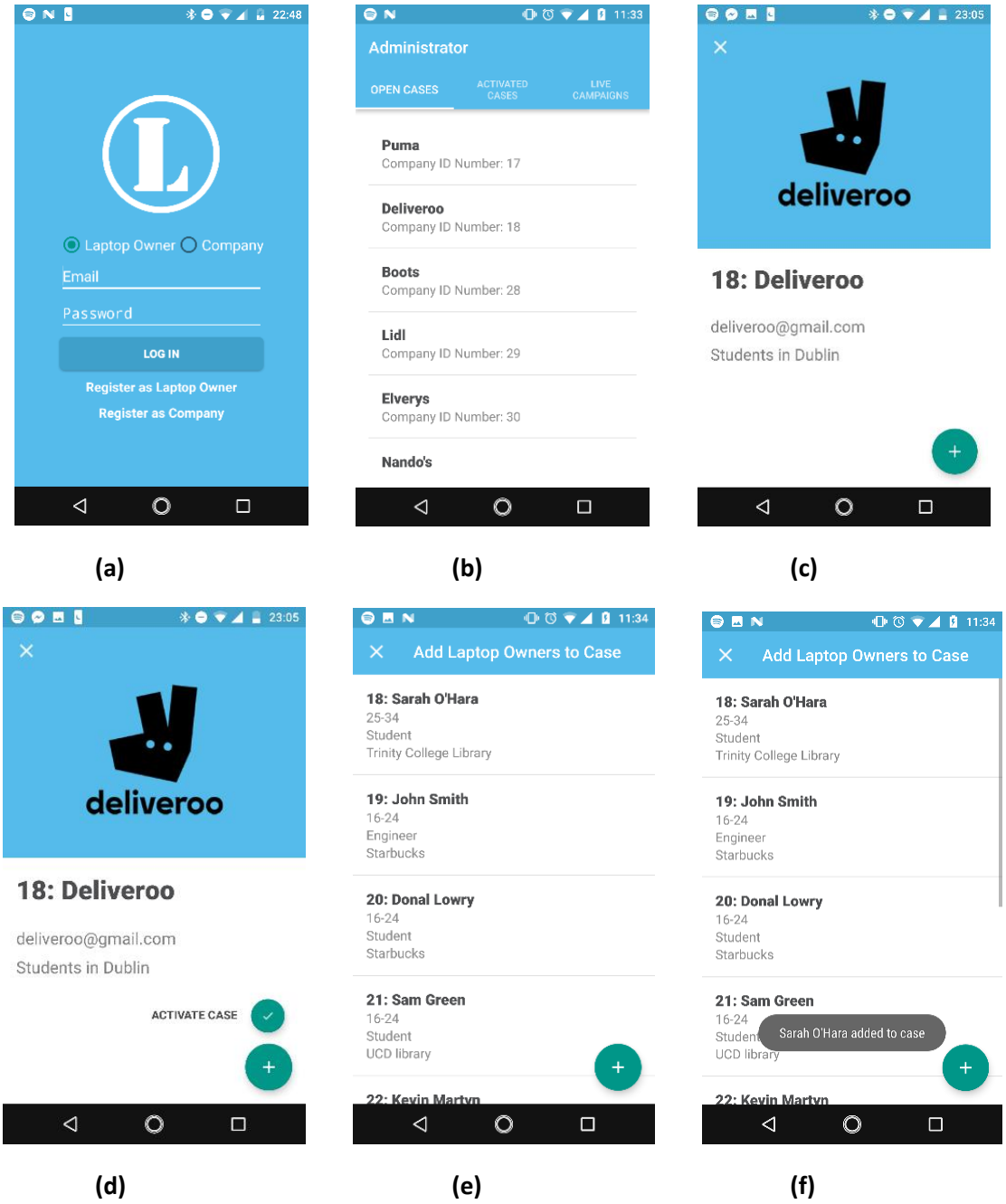


Figure 5.2 *Application interface: administrator perspective*

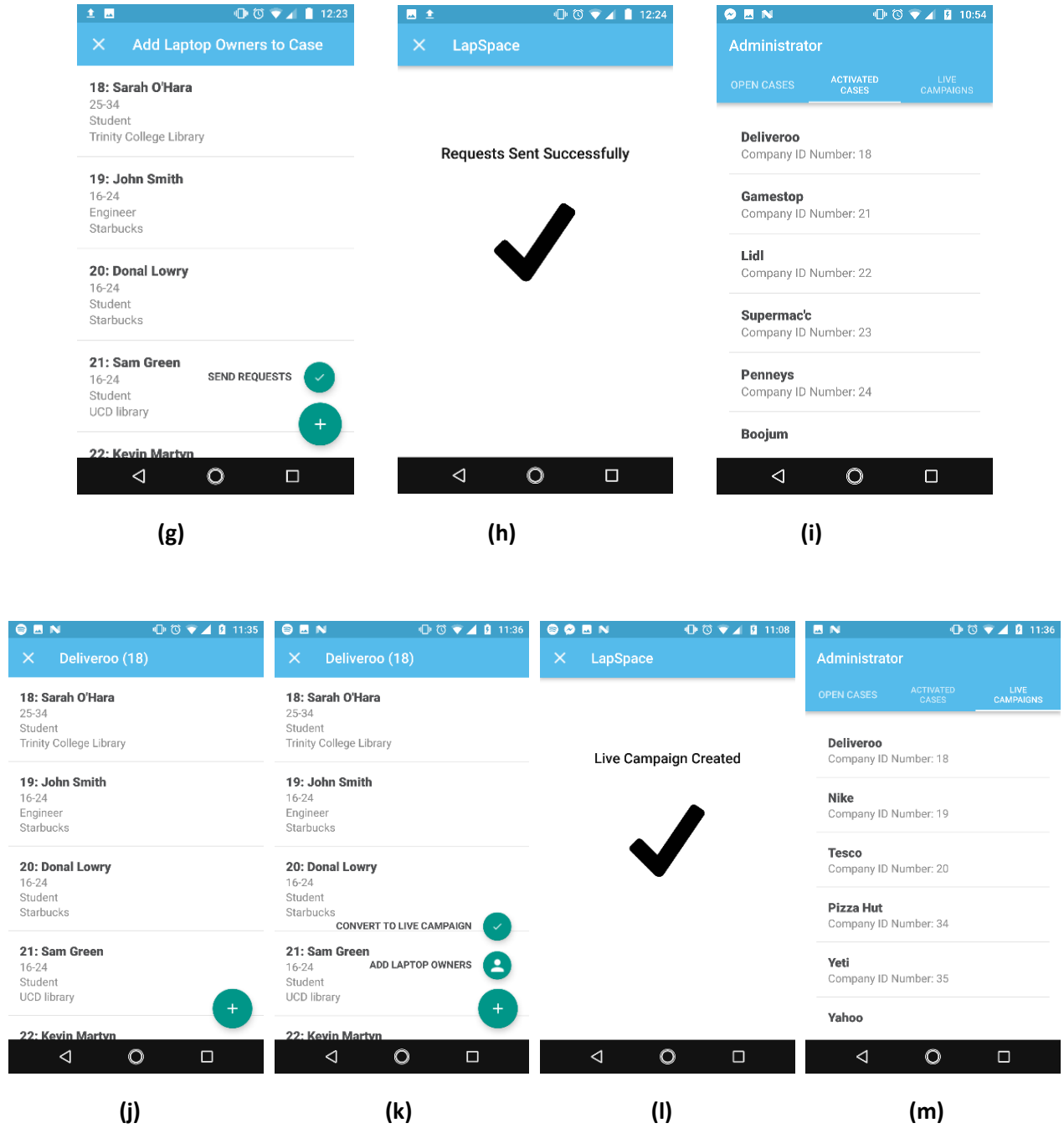


Figure 5.2 *Application interface: administrator perspective*

5.1.3 Laptop-Owner

A laptop-owner user can register using the form shown in Figure 5.3 (a). The necessary information includes the user's name, email, password, age range, occupation, address, the location where they use their laptop most and an optional profile picture. This information is stored in the database. The laptop owner can then sign into the application using the same login form as the company and administrator users. The application then displays their profile, shown in Figure 5.3 (c), displaying all user details.

Upon clicking the icon in the top left corner, a navigation drawer opens containing four buttons, two of which are functional. If the user clicks the *Notifications* button, a new activity displays any requests to join campaigns that the laptop-owner may have. If the user has no current requests they are shown the screen in Figure 5.3 (e). If they do have a request they are shown the screen in Figure 5.3 (f), which contains the company name, the company logo and a button to accept or reject the offer.

If the user clicks the *Verify* button in the navigation drawer, the phone camera launches which allows them to photograph their laptop, in order to verify that they are advertising the logo as intended. The application then retrieves the template logo of the company that is linked to the laptop-owner. The application then performs the verification processing, and if the logo is not present in the input image the screen displayed will be Figure 5.3 (j). If it is present the screen displayed will be Figure 5.3 (k).

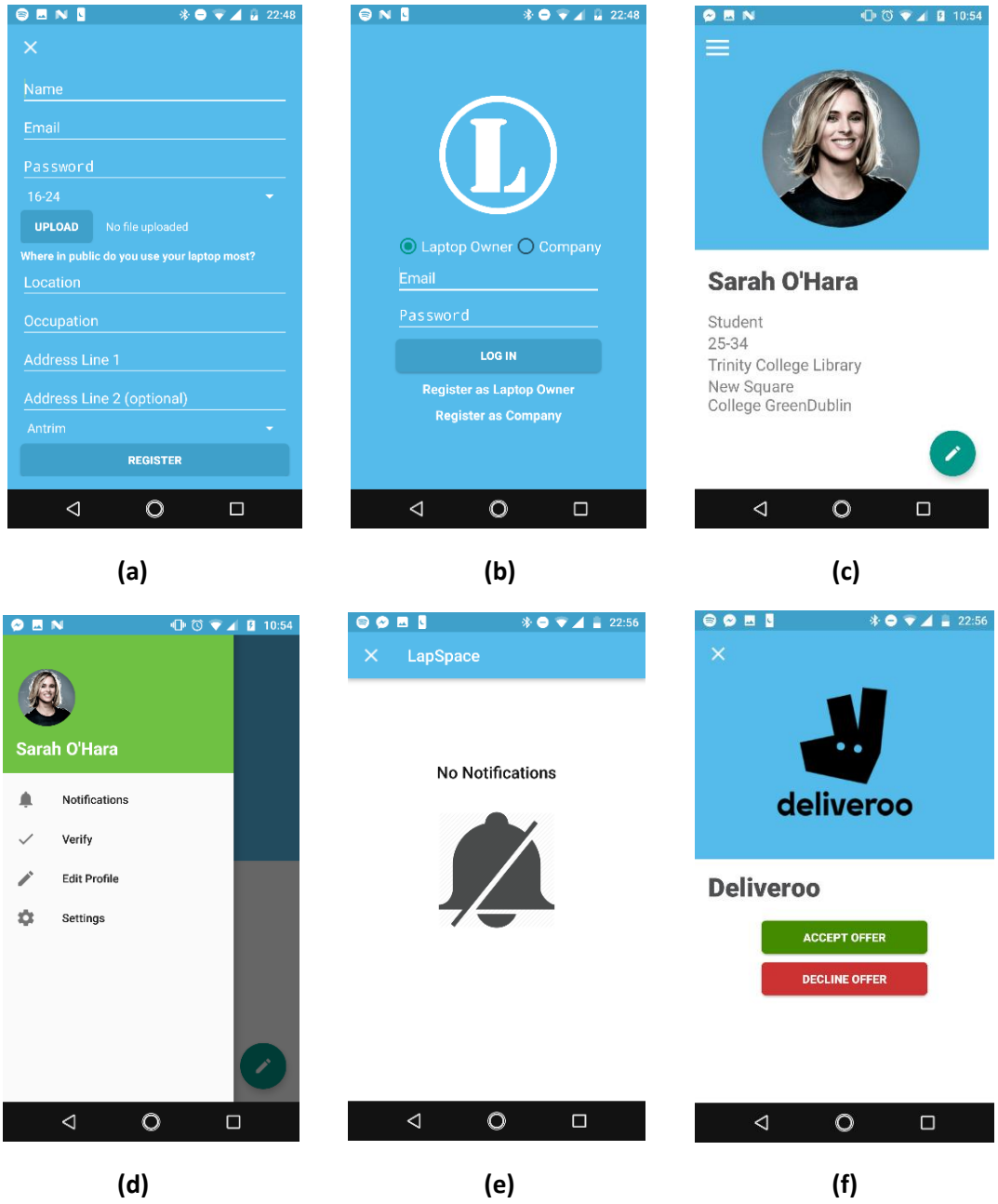


Figure 5.3 *Application interface: laptop-owner perspective*

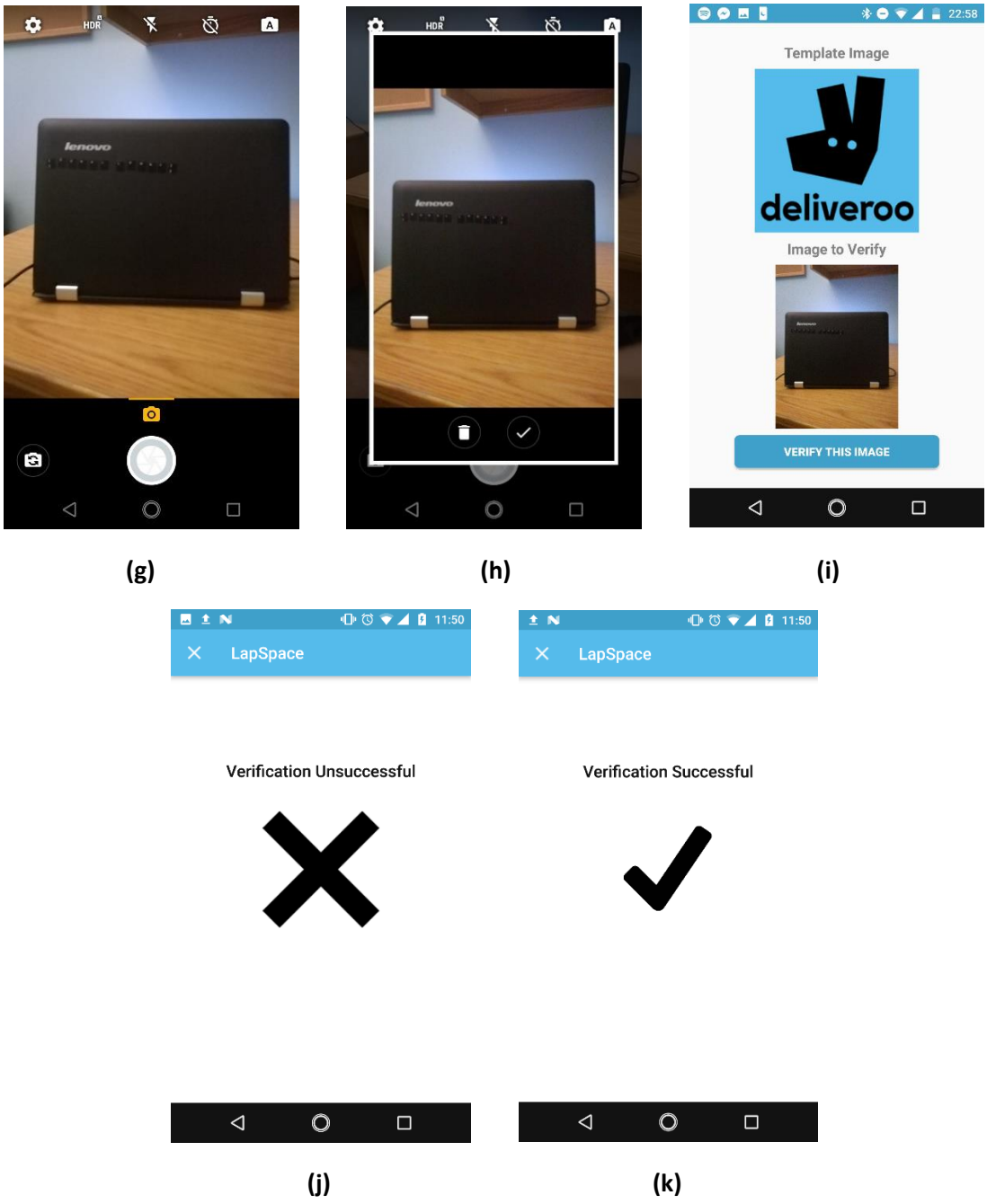


Figure 5.3 *Application interface: laptop-owner perspective*

5.2 System Architecture

In this section, the interaction of the different components of the system will be described with the aid of UML diagrams.

5.2.1 Java Classes

A class is a template file that describes the behaviour of an object of its type. Android classes are written in Java, a widely used, object-oriented programming language (Docs.oracle.com, 2018). This section will discuss some variations of Java class used in the program. An Android activity is a class which dictates the contents and functionality of each screen that is displayed in different areas of an application (Developer.android.com, 2018). An Android fragment represents a portion of an activity. Multiple fragments can be encapsulated in one activity. User interface layouts were created using Extensible Markup Language (XML), which will be discussed in the next chapter.

Figures 5.5 (a), (b), (c) and (d) display the system architecture in terms of the activities, classes and fragments involved in the program. For clarity, the entities and relationships have been grouped into four different diagrams:

- (a): entities related to the registration and login process
- (b): entities related to the company user
- (c): entities related to the laptop-owner user
- (d): entities related to the administrator user

These figures are followed by a discussion of important aspects and concepts involved in the system. The notation used in these graphs is explained in Figure 5.4. Each rectangle represents either an activity, a class or a fragment. The area in the upper section of each rectangle displays the name of the entity, and the lower section lists the entity's methods. The access modifier icon before each method can be a '#' icon for *protected*, a '+' icon for *public* or a '-' icon for *private*. The text within the brackets of a method indicates the name and datatype of any input parameters. The lines connecting the classes shows the relationship between the entities. The relationship line with the black diamond, shown in Figure 5.4, is a *composition*, meaning that the class *DatabseCallTask* is an inner class of the activity *DetailsActivatedCase* and it has a lifecycle dependency on its container. The relationship with the white diamond indicates an *aggregation*, which means that a class does not have a lifecycle dependency on its container class (Booch, Rumbaugh and Jacobson, 2005). A simple arrow with no diamond shape represents an intent from one activity to another, for example a *Login* activity would intent to a *Profile*.

The number displayed above this line indicates the cardinality of an object of the class. For example, the cardinality '0..1' of the *DetailsActivatedCase* activity means that either 0 or 1 instances of that activity may exist. The cardinality '1..1' of the *DatabseCallTask* class means that an instance of this class *must* exist if an instance of the *DetailsActivatedCase* activity exists.

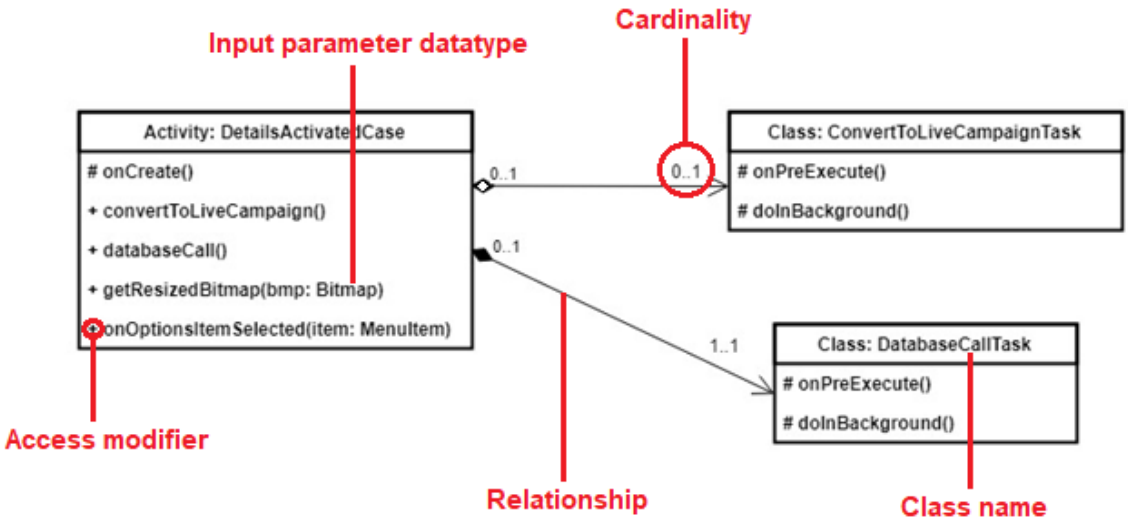


Figure 5.4 *Class diagram key*

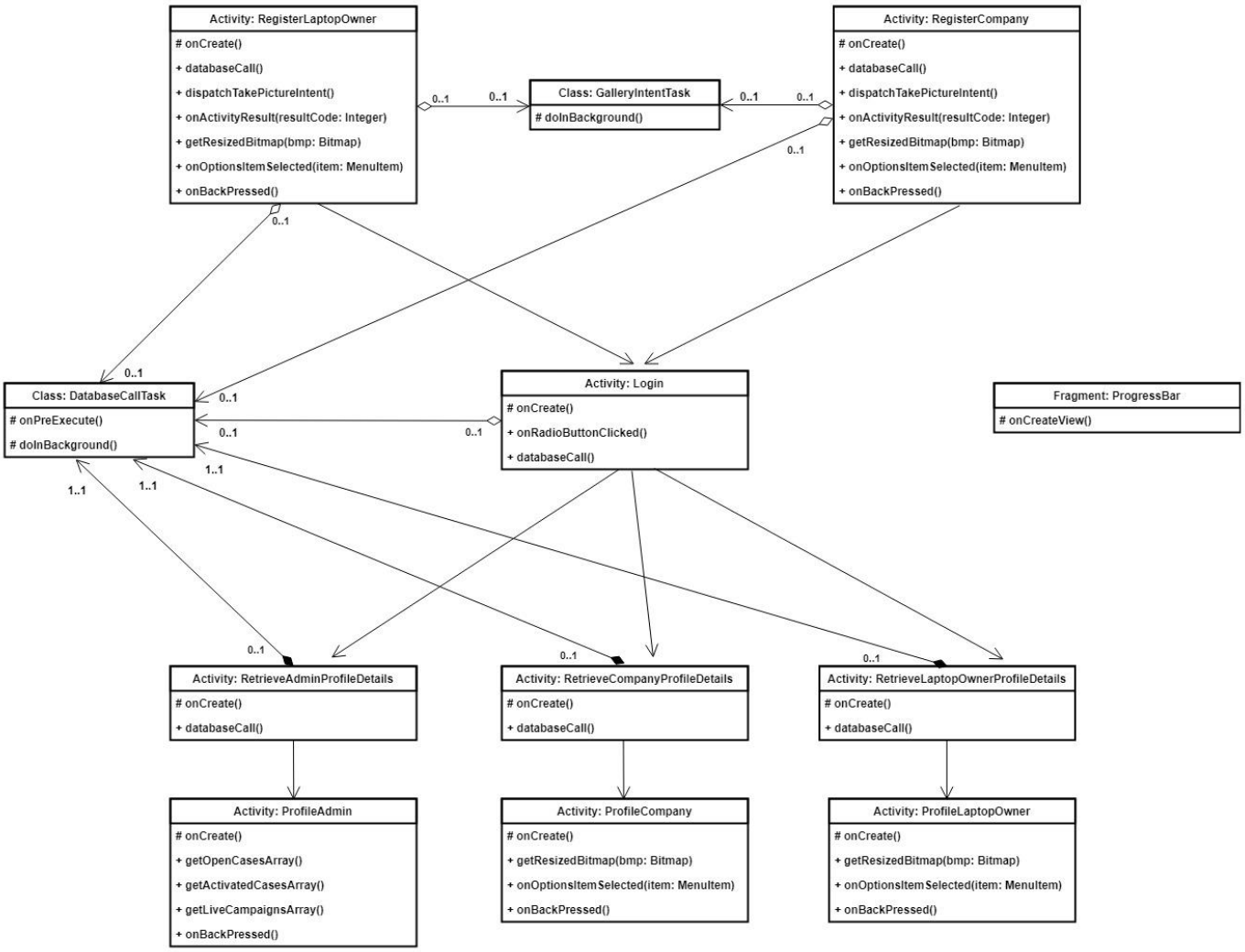


Figure 5.5 (a) *Class diagram: Registration and login*

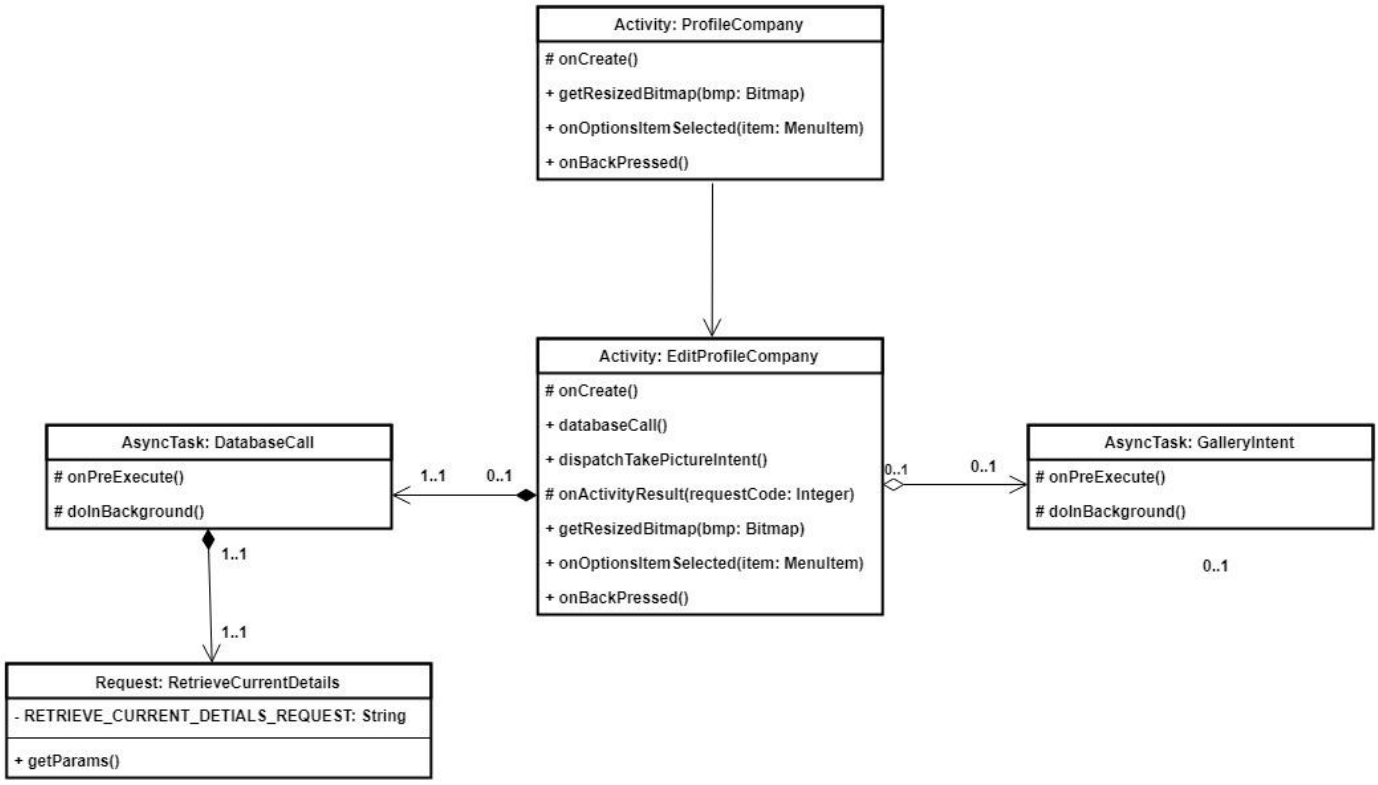


Figure 5.5 (b) *Class diagram: Company user perspective*

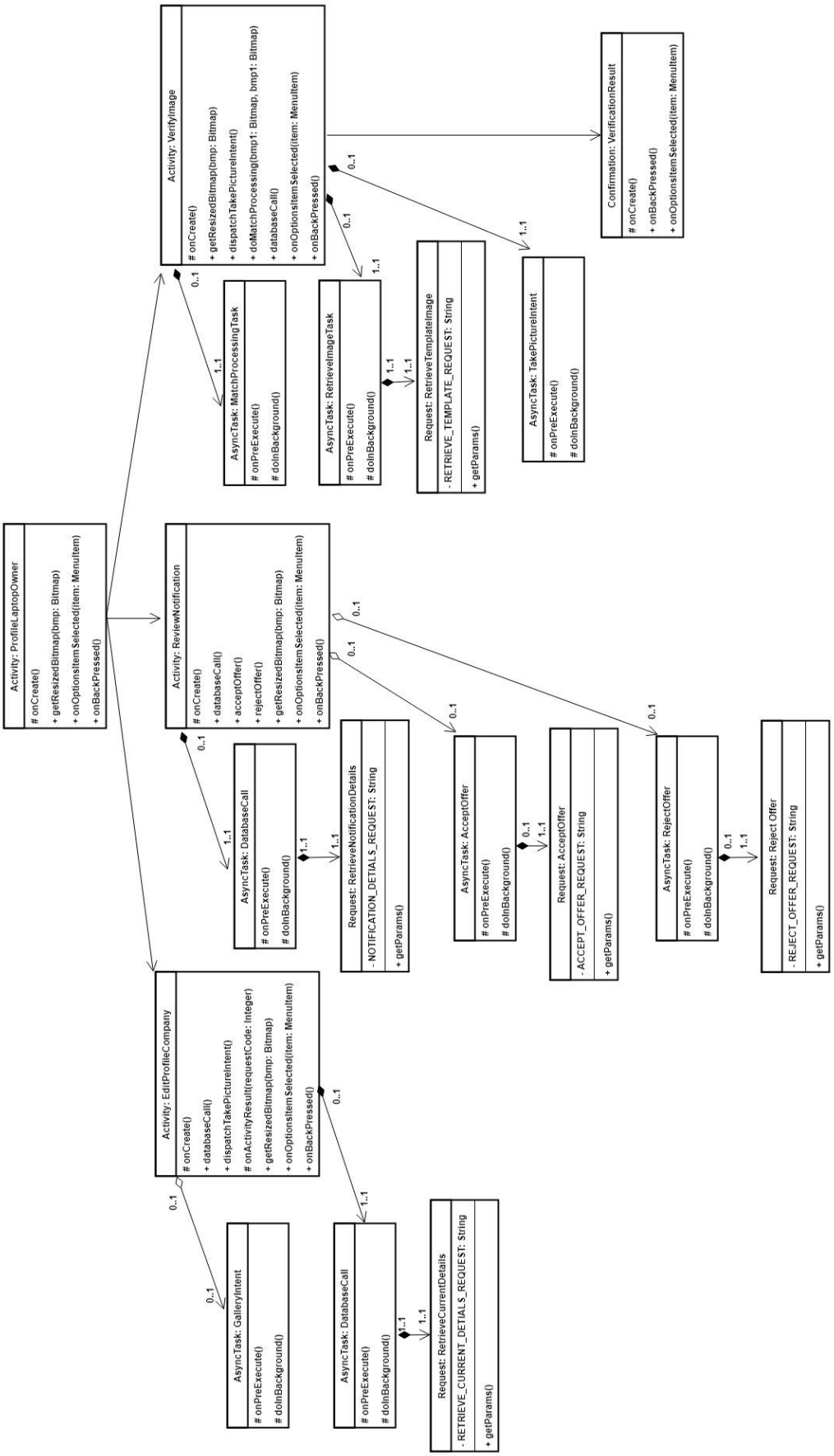


Figure 5.5 (c) *Class diagram: Laptop-owner user perspective*

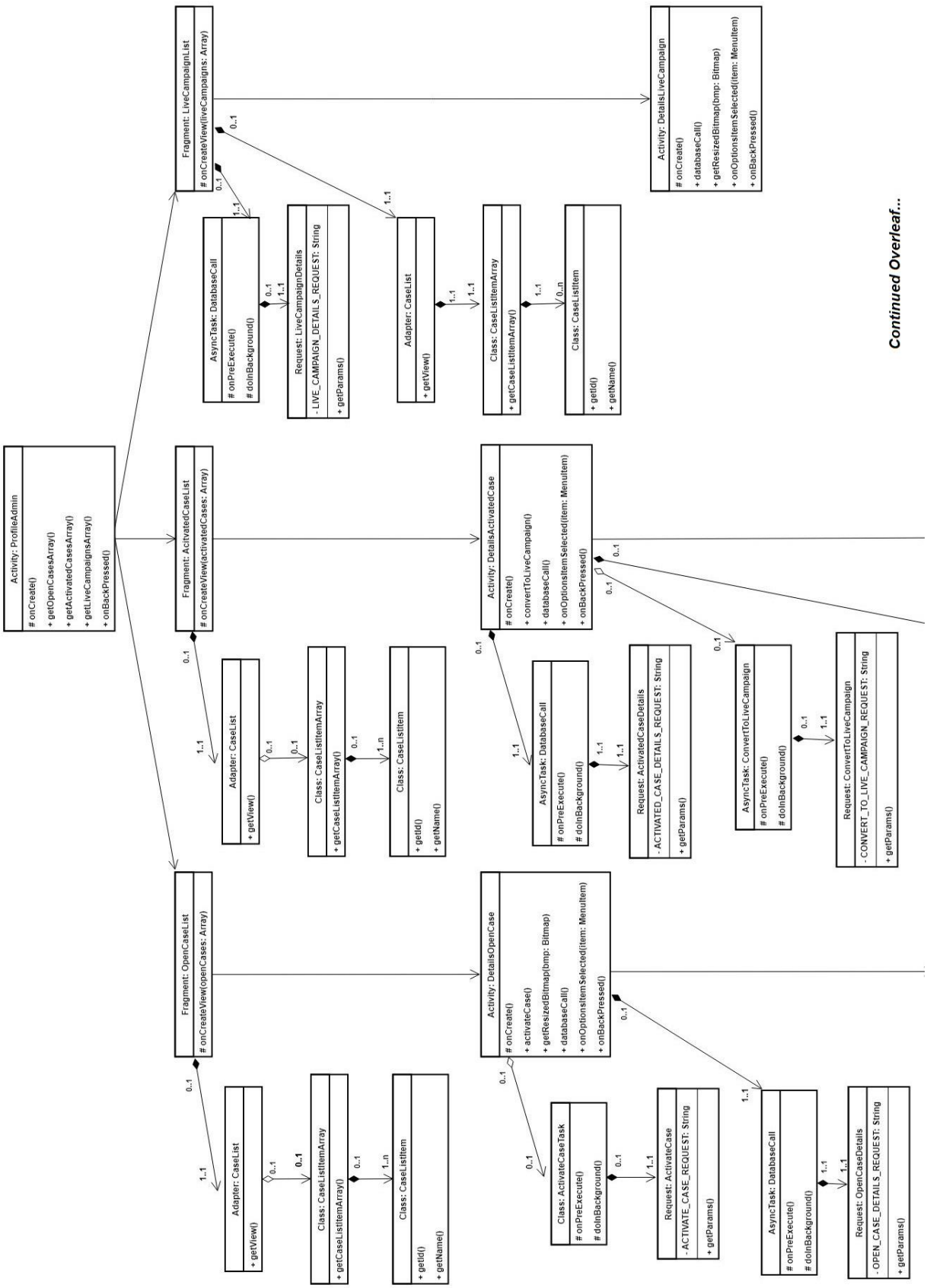


Figure 5.5 (d) *Class diagram: Administrator user perspective*

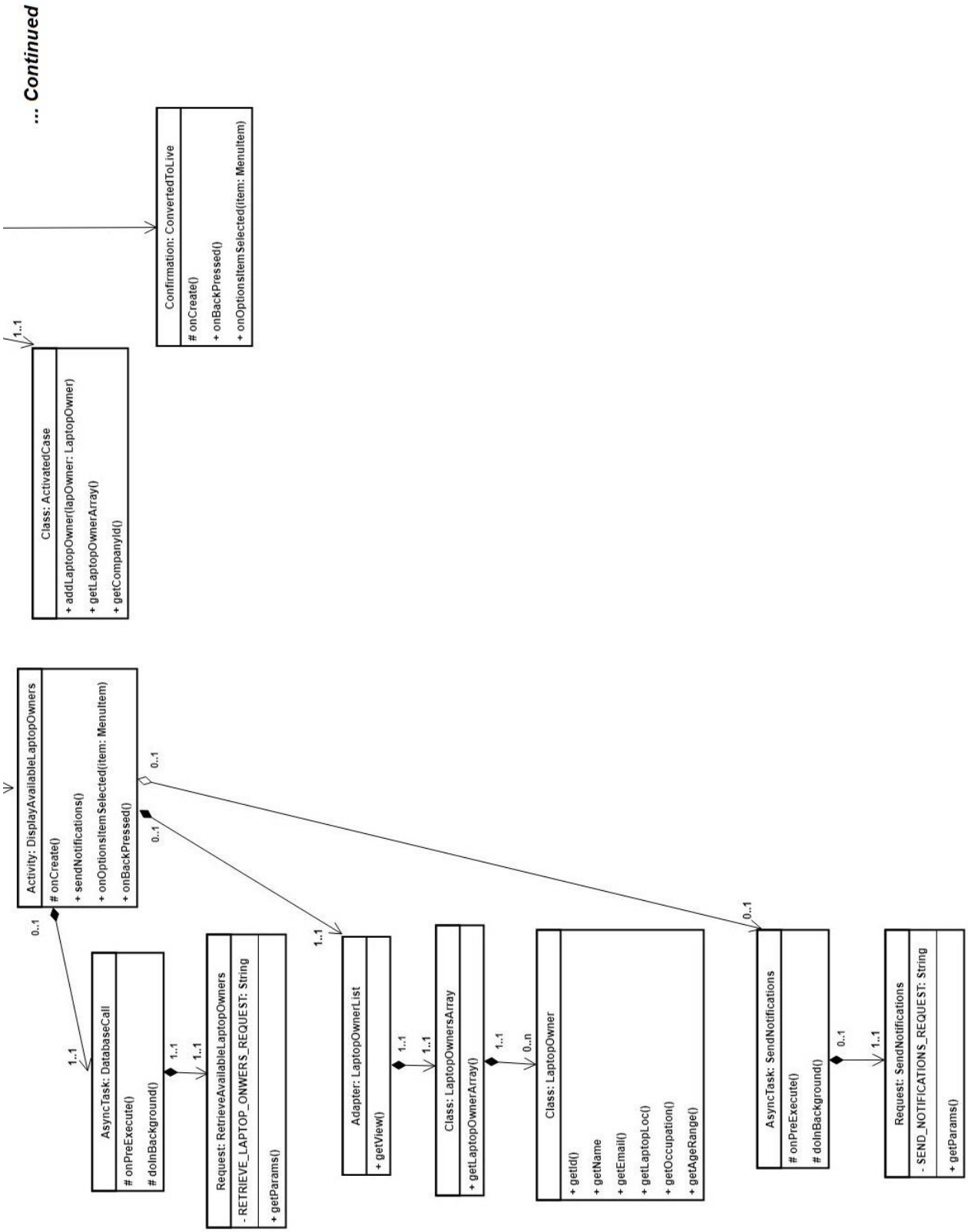


Figure 5.5 (d) *Class diagram: Administrator user perspective*

Asynchronous Tasks

In order to maintain a user interface that transitions from one activity to another smoothly, it may be necessary to perform some operations on a separate thread to the user interface thread. This can be achieved by creating a class which extends the *AsyncTask* class. In this application, the operations that are performed on a separate thread are database calls, capturing images from the phone camera, and performing the match processing involved in verification. As shown in the previous diagrams, *AsyncTasks* are declared as private inner classes within an activity. The three important methods of an *AsyncTask* are *doInBackground(), onPreExecute()* and *onPostExecute().* The *doInBackground()* method encapsulates the operation which is carried out on a separate thread to the user interface thread. *onPreExecute()* may contain operations which will be executed before *doInBackground()* and *onPostExecute()* has operations which will be executed after *doInBackground().* These methods can be useful in implementing a progress bar, for example. A progress bar may be initiated in the *onPreExecute()* method and terminated in the *onPostExecute()* method, after the background processing has been complete.

```
private class GalleryIntentTask extends AsyncTask<Void,Void,Void> {
```

Fragments

Fragments represent a portion of an activity. Fragments can be used to display a number of different screens, depending on the scenario, within the context of one activity.

In this application, a progress bar fragment was implemented. This fragment displays a spinning white ring, while background processing is being carried out. The progress bar fragment takes advantage of the *onPreExecute()* and *onPostExecute()* methods of an *AsyncTask*, discussed in the previous section. The progress bar fragment is called in almost every activity of the program. Any activity that uses an AsyncTask, to carry out some computationally expensive processing, displays the progress bar fragment during this time and hides it after completion. The motivation behind this design is discussed further in the following chapter.

The tab layout used in the administrator profile activity uses a separate fragment for each tab. Each fragment retrieves company data for either open cases, activated cases or live campaigns. It inflates this array of objects into a list and displays this list in the corresponding tab.

Requests

Database queries that simply require the retrieval of information can be carried out using a built-in java class called a *StringRequest*, however, if it is necessary to embed parameters, a customised Request class must be created. These Request classes are observable throughout the class diagrams in the previous section. An instance variable of these request classes is the string value of the of the PHP file stored online, which contains instructions on how the database should be queried.

Serializable Classes

A number of classes in the system were created because certain complex datatypes cannot be passed from one activity to another via an intent. An intent can only pass a finite number of datatypes as parameters. In order to pass some information, it must be encapsulated in a *serializable* class. A *serializable* class means that it can be stored as a sequence of bits (Developer.android.com, 2018).

Array Adapters

Implementing a list of items in Android requires an array adapter to be created. The data which will comprise the list should be retrieved from the database and stored as an array. This array is passed into an array adapter, which inflates it into a list.

Important Methods

onCreate()

The primary component of an activity is the *onCreate()* method, which is called as soon as the activity is initiated. The first operation in every *onCreate()* method is the initialisation of the user interface layout which is associated with that particular activity. The activity may also contain a number of other methods and nested private classes which may be called from the within the *onCreate()* method.

```
protected void onCreate(Bundle savedInstanceState) {
```

onBackPressed()

Working with fragments adds a degree of complexity to the backwards navigation of the application. For example, if the back button is pressed by a user, it is not desirable to return to the progress bar fragment which was previously displayed. For this reason, the *onBackPressed()* method is used, to handle backwards navigation, in activities where fragments are used.

```
public void onBackPressed() {
```

getResizedBitmap()

Whenever images are handled in the application, such as prior to verification or during company registration, they are first compressed to a smaller size using the *getResizedBitmap()* method. The motivation behind this is discussed in the following chapter.

```
public Bitmap getResizedBitmap(Bitmap image, int maxSize) {
```

dispatchTakePictureIntent()

This method is used to retrieve an image from either the phone camera or the gallery depending on the context. The result of the image retrieval is handled in the *onActivityResult()* method.

```
public void dispatchTakePictureIntent() {
```

Manifest and Gradle Files

The Android Manifest is an XML file which outlines some properties of the program, such as which activity should be displayed when the application is first launched. The Gradle files are where supporting libraries can be compiled.

Chapter 6 - Non-Functional Requirements

The non-functional requirements outlined during requirements elicitation phase were:

1. To create smooth transitions between activities.
2. To provide the user with constant feedback.
3. To create an easily learnable graphical user interface.

This chapter will discuss the methods used to achieve these requirements. User interface designs are implemented in android using Extensible Markup Language (XML). XML files define the layout of the different elements visible in a particular view.

6.1 Smooth Transitions

Before the application processes images, it compresses these files to reduce processing time. This image compression has multiple benefits. It reduces the amount of memory needed to store images in the database, and algorithms in the verification phase run faster with smaller image sizes. This reduces the wait time for users of the application. According to Google (2016), 53% of users abandon sites that take longer than 3 seconds to load. Small margins are significant in terms of human computer interaction.

Progress bars are used, within the application, to inform the user that processing is being carried out in the background. Whenever an operation such as a database call or a complex calculation is carried out, a progress bar, such as the one in Figure 6.1, is displayed on the phone screen. Without this mechanism, the app would have periods where nothing is being displayed or updated on the user interface. In this scenario it may, therefore, seem to the user as if the application is frozen and the likelihood of abandonment is high.

6.2 User Feedback

Progress bars are one method of providing the user with feedback about what is happening in the application. Another method is confirmation messages. Two examples of confirmation messages are shown in Figure 6.2. Figure 6.2 (a) shows a small dialog at the bottom of the screen confirming a minor task, and Figure 6.2 (b) shows a full screen confirmation message for the completion of a more significant task. User feedback is essential for helping users to discover capabilities and understand the results of actions (Developer.apple.com, 2018).

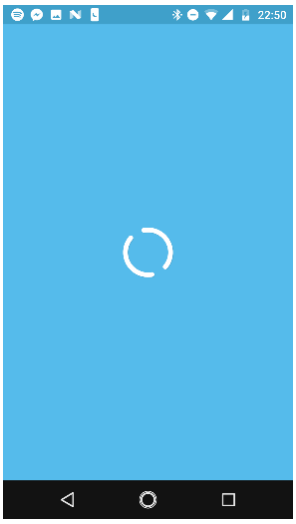
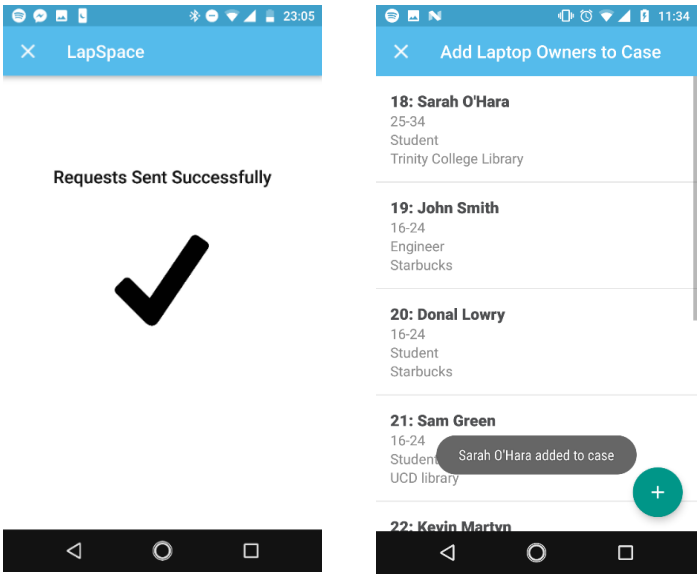


Figure 6.1 *Progress bar*



(a) ***(b)***

Figure 6.2 *Confirmation messages*

6.3 Learnable Graphical User Interface

Android maintains and encourages the use of a design specification called *Material Design*. *Material Design* acts as a visual language which helps applications to be more learnable for new users (Material.io, 2017). If most android applications were designed using this same specification, a user may subconsciously recognise certain interface feature despite never having used the application before.

Material Design relates to such factors as the dimensions of layouts and consistency of the colours used throughout the interface. This application has adhered to *Material Design* in order to increase its learnability.

Chapter 7 – Conclusions

This chapter will reflect on and critique the work carried out during the project.

7.1 Reflection

This section will address the results with respect to the initial aims, outlined during requirements specification, as well as any problems encountered.

7.1.1 Verification

The first aim of the project was to implement a method of verifying that a laptop-owner was advertising their logo as intended. The primary approach to this was to use computer vision techniques to assess the presence or absence of the logo, in an image uploaded by a user. During the implementation of the verification programs, a problem was encountered in configuring a necessary element of the Open Source Computer Vision library with Microsoft Visual Studio. This element was required for the feature tracking method of image matching. After two weeks of failed attempts, the problem was eventually overcome.

The final performance results of this aspect of the project are shown in Figure 3.27. The precision score, of 0.95, indicates that the program has an error rate of 5% in relation to positive classifications. A recall score, of 0.8 however, shows that 20% of positive test inputs do not get recognised. These figures reflect one of the initial aims of the project, which was to place slightly more emphasis on precision over recall. The reasoning behind this aim was that an unrecognised positive classification error, was favourable to an incorrect positive classification. Taking both positive and negative test inputs into account, the final accuracy score was 0.83, indicating that the program had an error rate of 17%. These figures could certainly be improved by methods discussed in the next section.

7.1.2 Database

The second aim of the project was to create a database that would store the information required by the system and host it in an online server so that it would be accessible from any device. Given that the author had limited experience with databases prior to this project, this aim presented difficulty. Following four weeks of learning PHP and MySQL a functional database was implemented. The result of this work achieved the initial project aim, in that information gathered from the android application was successfully stored and retrieved from the database. Necessary future work relating to the database, will be discussed in the next section.

7.1.2 Application Functionality and Usability

The third aim of the project was to create a functional application interface with high usability and learnability. Of the 18 initial functional requirements shown in Figure 2.1, 16 were fully or partially implemented. Future work, regarding this, would involve implementing unfinished functional requirements. This will be discussed in more detail in the next section.

The usability and learnability aims of the project were addressed using progress bars to create smooth transitions, confirmation messages to provide user feedback and the *Material Design* visual language to increase learnability. The effectiveness of these methods was measured by a usability test.

The usability test involved 10 participants. The test would involve a short session with each participant using the application while being watched by a tester, who would take notes and encourage the participant to think aloud. The audio from this test was recorded to be analysed later. Following the test, the participant was asked to complete a short survey about their experience with the application. This questionnaire measured the application's adherence to Nielson and Molich's (1990) Heuristic Guidelines for interface usability. The questionnaire can be observed in Figure 7.1. The participant reaction from the usability test was positive and valuable insights were gained from analysing the result. Unfortunately, ethical approval for the usability test was achieved very close to the project deadline. Changes to the interface based on the survey must, therefore, be included in future work.

7.2 Future Work

This section will address how the project could be improved and developed further.

7.2.1 Verification

In relation to the verification aspect of the project, future work should be tested on a larger scale in order to improve the accuracy of performance metrics. Computer vision projects are commonly tested with upwards of 300 test inputs (berkeley.edu, 2018).

An issue may arise where it is possible for a user to "trick" the program into verifying their image despite the user not advertising the logo as intended. A user could, for example, take a picture of a logo on a screen. This issue could be addressed in future work by altering the verification process as follows. Upon joining a campaign and receiving their sticker, the laptop-owner would be required to submit a photograph of their laptop, with the logo sticker attached to it. This image would only be able to be submitted via the app which would only allow access to the phone camera and not the gallery. The image would be viewed by a human administrator who would ensure that the logo is correctly displayed on their laptop. This image would then be used as the template with which future verification would be done. This change is simply an operational change to the process, and the computer vision techniques used would be the same as those developed in this project.

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
	1	2	3	4	5
I found the system easy to use.					
I found the system easy to learn (i.e. perhaps not easy to use at first but the learning curve was fast).					
I was able to complete the tasks necessary quickly.					
I was aware of what I was supposed to be doing at all times.					
The system gave error messages that clearly showed me how to fix problems.					
The information provided in the system was clear and used understandable language.					
The interface/style of the system was pleasant.					
Whenever I made a mistake in the system, I could recover easily and quickly.					
The organisation of information on the screen was clear.					
Overall, I am satisfied with the system.					

Figure 7.1 *Usability test questionnaire*

Additionally, the aforementioned issue may be solved by using computer vision techniques to detect the presence of a screen in an input image. As can be observed in Figure 7.2, the pixelated nature of a screen can add a degree of complexity to an image. Figure 7.2 (a) is a photograph of a logo on a laptop. Figure 7.2 (b) is a photograph of a laptop screen displaying the image in Figure 7.2 (a). Even to

the naked eye these images are considerably different, and this fact is also reflected in the number of features that are detected in each image. The number of features detected Figure 7.2 (a) differs from Figure 7.2 (b) by 103. With this knowledge, the verification process should be altered further so that, not only *matching* features are considered, but also the *number* of features. The number of features detected in the template image should be roughly the same as the number of features detected in the input image.

These alterations to the verification process should prevent users from "tricking" the system. Another future addition to the verification program would be to cross check the geolocation of the input image with the location data entered by the laptop-owner at registration. This ensures that the laptop-owner is using their laptop in the general area where they claim to use it most. This is significant because laptop-owners are chosen primarily based on where they use their laptop.

(a) **(b)**

Figure 7.2 *Illustration of the complexity of a picture of a screen*

7.2.2 Database

Future work relating to the database involves both security and scalability. Given that it is necessary for the system to store sensitive information about users, such as their addresses, data protection measures must be implemented.

Using prepared statements, instead of direct SQL queries, prevents SQL injection. SQL injection involves data being maliciously injected into an SQL query, but prepared statements only accept preestablished parameters and can only perform very specific functions (Upguard.com, 2018). An example of a simple direct SQL query is as follows:

```
$sql = "UPDATE company SET has_live_campaign = 1, WHERE id = $id";
```

The same query, presented as a prepared statement is as follows:

```
$stmt = mysqli_prepare($con, "UPDATE company SET has_live_campaign =
1, WHERE id = ?");

mysqli_stmt_bind_param($stmt, "i", $id);

mysqli_stmt_execute($stmt);
```

A record of all database events should be kept as well as automatic notifications of significant events, such as a new administrator being added to the system, or repeated failed attempts to login. The passwords of all administrators should be complex and frequently updated.

Scalability is another element of future work relating to the database. In order to store a higher volume of data, the database would need to be migrated to a paid hosting service which offers more memory than the free service currently used.

7.2.3 Application Functionality and Usability

Additional future work would involve the implementation of the unfinished functional requirements. Before deployment, the application must support a payment system which allows companies to pay advertising fees. Another incomplete requirement is the ability of users to edit the settings on their profile.

Another aspect of future work would involve making changes to the interface based on the insights gained from the usability test. Additionally, more usability testing may be carried out, following the completion of further iterations of the application.

7.2.4 Monetisation

Another element of future work for the project would be to form a monetisation structure. The service may be monetised in a way that laptop-owners partake in a full advertising campaign and receive payment at the end of this period. An alternative system would be for a program to be installed on a user's laptop which monitors their geolocation when turned on. The laptop-owner could then be paid per hour spent using their laptop in areas that have previously been deemed to be prime advertising locations.

7.3 Concluding Note

This report has outlined the design and implementation of an android application which enables users to rent out the back of their laptop as adverstising space. The completion of this project has allowed the author to develop skills in a range of different disciplines in computer science. The project has

resulted in a satisfactory first increment of the application with scope for significant future development and improvement in later increments.

REFERENCES

Acharya, K., Venkatesh Babu, R. and Vadhiyar, S. (2014). A real-time implementation of SIFT using GPU. *Journal of Real-Time Image Processing*, 14(2), pp.267-277.

Altman, D. and Bland, J. (1994). Statistics Notes: Diagnostic tests 1: sensitivity and specificity. *BMJ*, 308(6943), pp.1552-1552.

Android. (2018). *Introduction to Activities | Android Developers*. [online] Available at: https://developer.android.com/guide/components/activities/intro-activities.

Booch, G., Rumbaugh, J. and Jacobson, I. (2005). *The unified modeling language user guide*. 2nd ed. Upper Saddle River, NJ: Addison-Wesley, p.496.

Christel, M. and Kang, K. (1992). Issues in Requirements Elicitation. *Software Engineering Institute*.

Codd, E. (1974). Recent Investigations into Relational Data Base Systems. *IBM Research Report*.

Dawson-Howe, K. (2014). *A Practical Introduction to Computer Vision with OpenCV*. 1st ed. Chichester, West Sussex: John Wiley & Sons, pp.44-47.

Developer.apple.com. (2018). *Providing User Feedback - User Interaction - Human Interface Guidelines for macOS Apps*. [online] Available at: https://developer.apple.com/macos/human-interface-guidelines/user-interaction/providing-user-feedback/.

Docs.oracle.com. (2018). *Class (Java Platform SE 7)*. [online] Available at: https://docs.oracle.com/javase/7/docs/api/java/lang/Class.html.

Everts, T. (2016). *53% of mobile users abandon sites that take longer than 3 seconds to load | SOASTA*. [online] SOASTA. Available at: https://www.soasta.com/blog/google-mobile-web-performance-study/.

Forbes.com. (2014). *Forbes Welcome*. [online] Available at: https://www.forbes.com/sites/kimberlywhitler/2014/07/17/why-word-of-mouth-marketing-is-the-most-important-social-media/#1b02174554a8.

Hessinger, S. (2018). *What is the Cheapest Way to Advertise? - Small Business Trends*. [online] Small Business Trends. Available at: https://smallbiztrends.com/2018/04/cheapest-way-advertise.html.

Najman, L. and Talbot, H. (2010). *Mathematical Morphology: From Theory to Applications*. London: John Wiley & Sons.

Nielson, J. and Molich, R. (1990). Heuristic evaluation of user interfaces. *CHI '90 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp.249-256.

Nuseibeh, B. and Easterbrook, S. (2018). Requirements Engineering: A Roadmap. *ICSE '00 Proceedings of the Conference on The Future of Software Engineering*, pp.35-46.

Opencv.org. (2018). *About - OpenCV library*. [online] Available at: https://opencv.org/about.html.

Pulli, K., Baksheev, A., Kornyakov, K. and Eruhimov, V. (2012). Real-time computer vision with OpenCV. *Communications of the ACM*, 55(6), p.61.

Rosten, E. and Drummond, T. (2006). Machine learning for high-speed corner detection. *ECCV 2006*, pp.430-443.

The Irish Times. (2016). *LogoGrab: a small company with big ideas makes the most of Irish support*. [online] Available at: https://www.irishtimes.com/business/logograb-a-small-company-with-big-ideas-makes-the-most-of-irish-support-1.2067030.

Upguard.com. (2018). *11 Steps to Secure SQL*. [online] Available at: https://www.upguard.com/blog/11-steps-to-secure-sql.

Www2.eecs.berkeley.edu. (2018). *The Berkeley Segmentation Dataset and Benchmark*. [online] Available at: https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/.

Appendix 1 - Verification Threshold Optimisation

1.1 Precision-Recall Data

Histogram Comparison Program

Threshold	Precision	Recall
0.7	0.5	0.25
0.71	0.5	0.25
0.72	0.5	0.25
0.73	0.5	0.25
0.74	0.5	0.25
0.75	0.5	0.25
0.76	0.5	0.25
0.77	0.5	0.25
0.78	0.5	0.25
0.79	0.666667	0.25
0.8	0.666667	0.25
0.81	0.666667	0.25
0.82	0.666667	0.25
0.83	0.666667	0.25
0.84	0.666667	0.25
0.85	1	0.25
0.86	1	0.25
0.87	1	0.25
0.88	1	0.25
0.89	1	0.25
0.9	1	0.25
0.91	1	0.25
0.92	1	0.125
0.93	1	0.125
0.94	1	0.125
0.95	1	0.125
0.96	1	0.125
0.97	1	0.125
0.98	1	0.125
0.99	1	0.125

Feature Matching Program

Threshold	Precision	Recall
1	0.571428571	1
2	0.727272727	1
3	0.727272727	1
4	0.888888889	1
5	0.875	0.875
6	0.875	0.875
7	0.857142857	0.75
8	0.857142857	0.75
9	0.857142857	0.75
10	0.857142857	0.75
11	0.857142857	0.75
12	0.857142857	0.75
13	0.857142857	0.75
14	0.833333333	0.625
15	0.833333333	0.625
16	0.8	0.5

1.2 F_β Data

Feature Matching Program

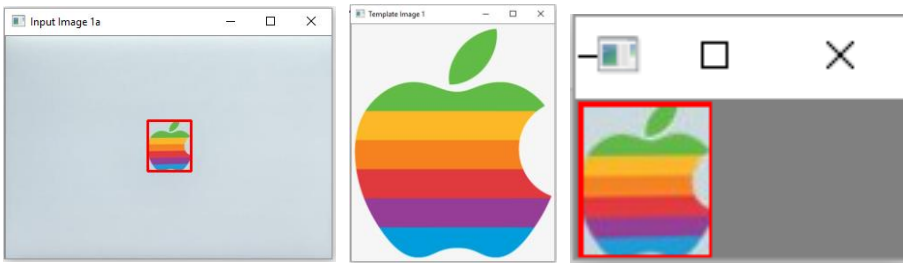
			fβ	
Threshold	Precision	Recall	*f(0.9)*	*f(1)*
4	0.888888889	1	0.935400517	0.941176471
5	0.888888889	1	0.935400517	0.941176471
6	1	0.875	0.939910979	0.933333333
7	1	0.875	0.939910979	0.933333333
8	1	0.875	0.939910979	0.933333333
9	1	0.75	0.870192308	0.857142857
10	1	0.75	0.870192308	0.857142857
11	1	0.75	0.870192308	0.857142857
12	1	0.75	0.870192308	0.857142857
13	1	0.625	0.788327526	0.769230769

Appendix 2 - Verification Program Tests

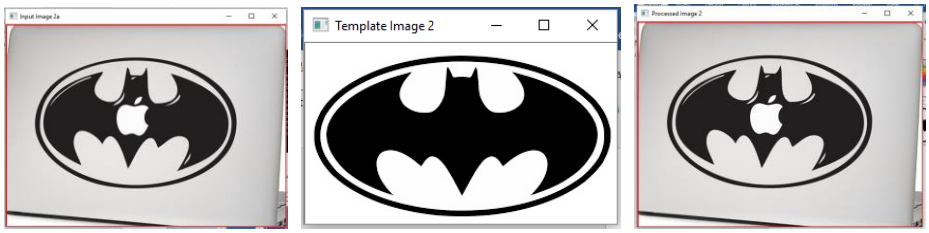
2.1 First Round of Testing

Histogram Comparison Program

[Input image – Template Image – Processed Image – Match Result]

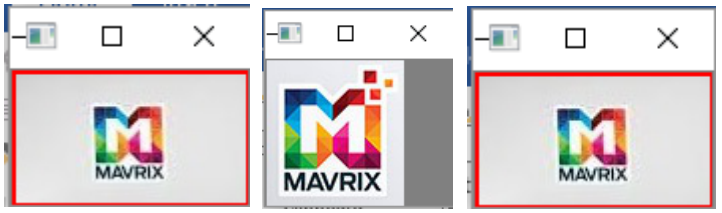


= Match (TP)

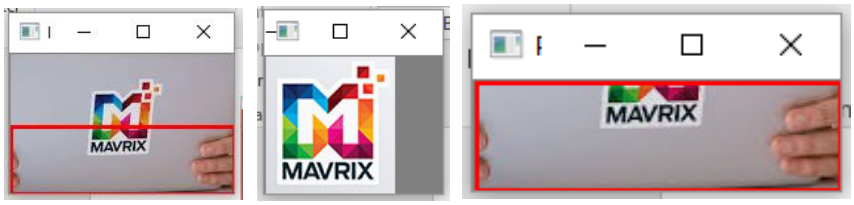


= No Match (FN)

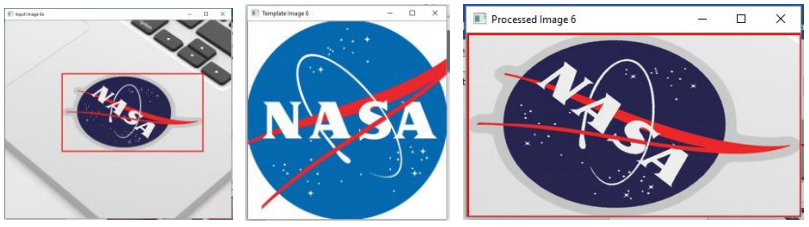
= No Match (FN)



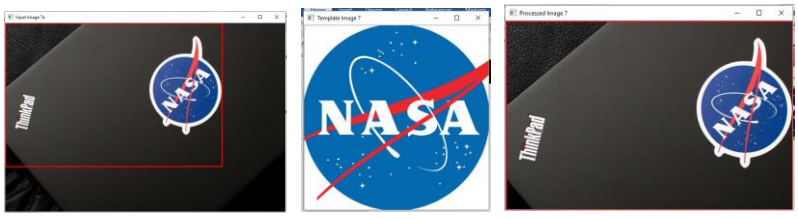
= Match (TP)



= No Match (FN)



= No Match (FN)

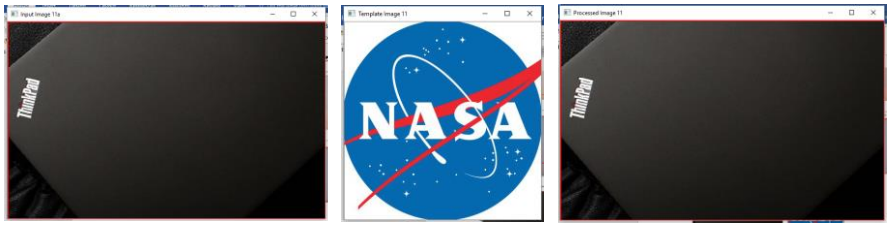


= No Match (FN)

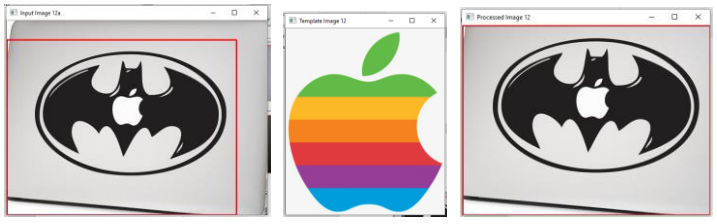
= No Match (FN)

= No Match (TN)

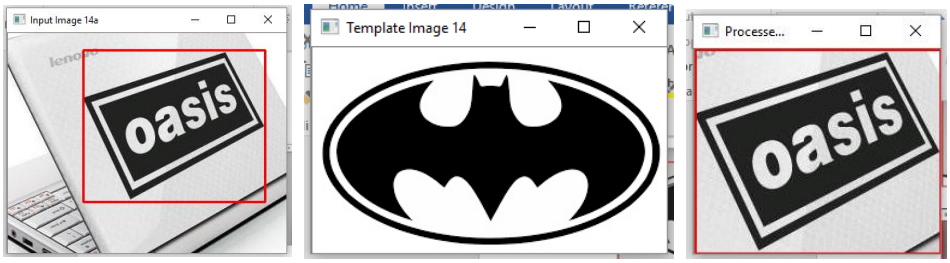
No Match (TN)



= No Match (TN)


= No Match (TN)

= No Match (TN)



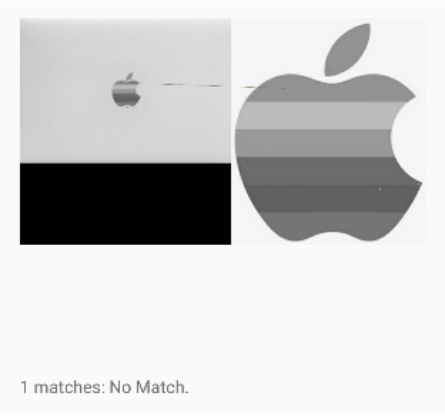
= No Match (TN)

= No Match (TN)

2.1 First Round of Testing

Feature Matching Program

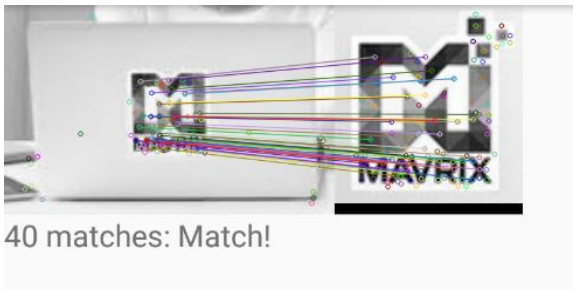
[Match Image – Match Result]



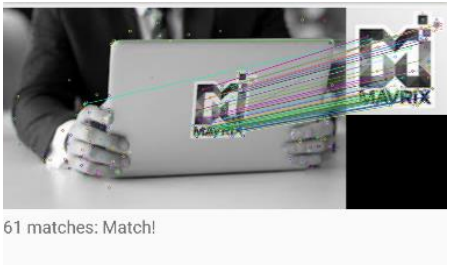
= No Match (FN)

= Match (TP)

= Match (TP)



= Match (TP)



= Match (TP)

= Match (TP)

= Match (TP)

= Match (TP)

= Match (FP)

= No Match (TN)

= No Match (TN)

= Match (FP)

= No Match (TN)

= Match (FP)

= No Match (TN)

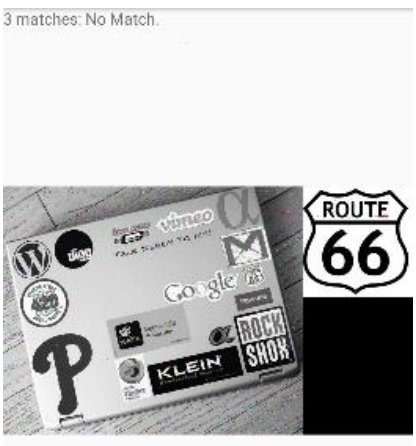
2.2 Further Testing of Feature Matching Program

[Match Image – Match Result]

= Match (TP)

= Match (TP)

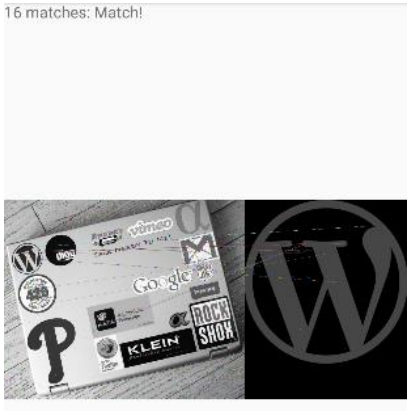
= Match (TP)



= Match (TP)

= Match (TP)

= Match (TP)

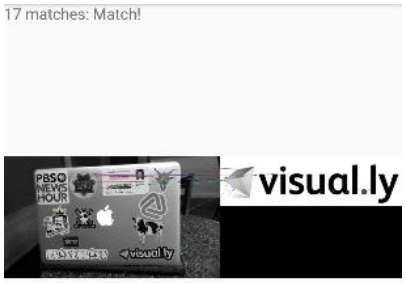


= Match (TP)

= Match (FP)

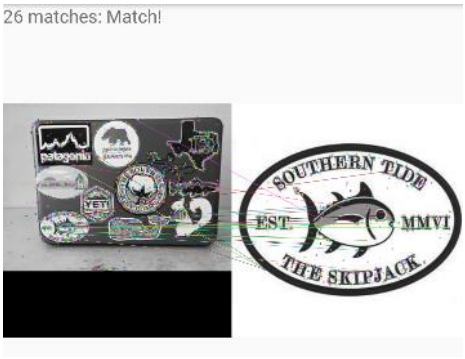
= No Match (FN)

= Match (TP)



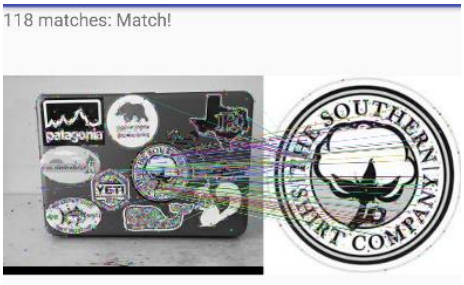
= Match (TP)

= No Match (TN)

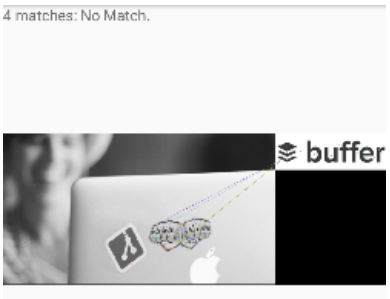


= Match (TP)

= Match (TP)



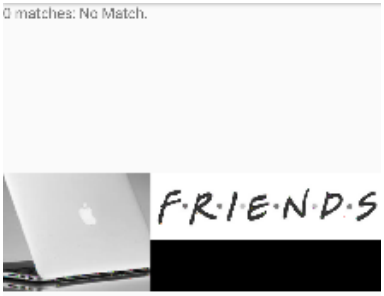
= Match (TP)



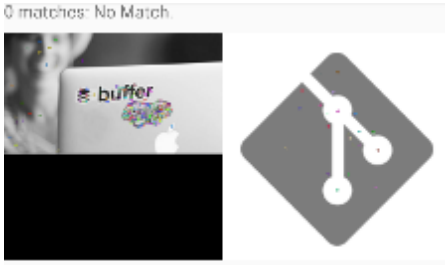
= No Match (TN)

= No Match (TN)

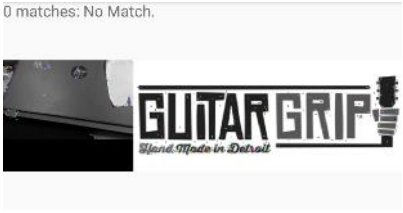
= No Match (TN)



= No Match (TN)

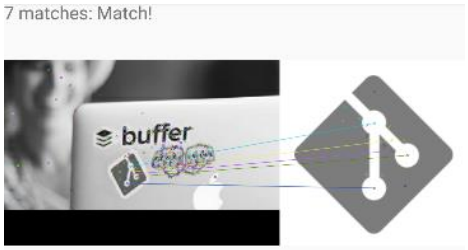


= No Match (TN)

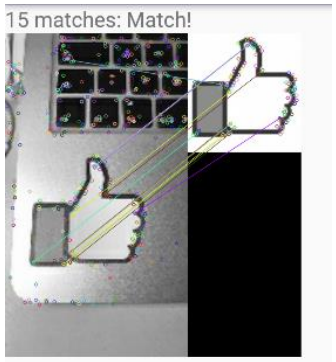


= No Match (TN)

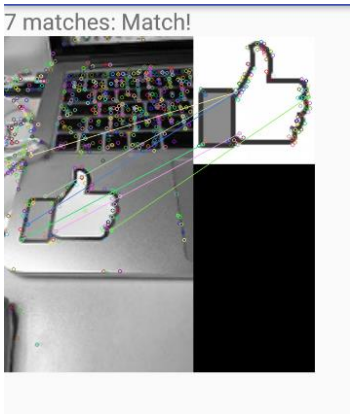
= Match (TP)



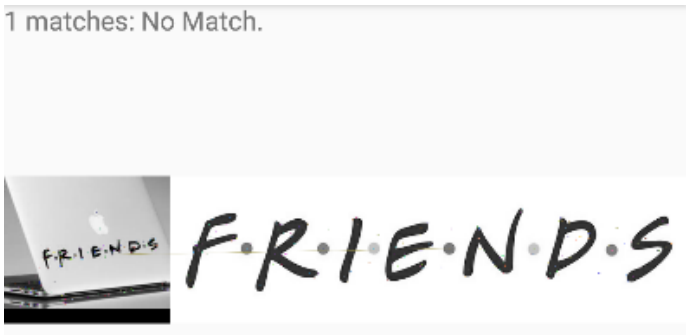
= Match (TP)



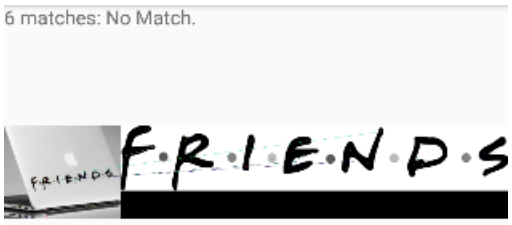
= Match (TP)



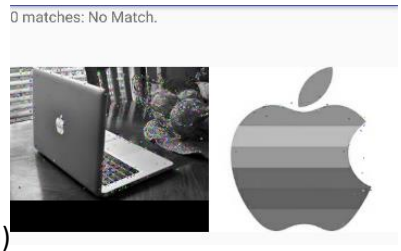
= Match (TP)



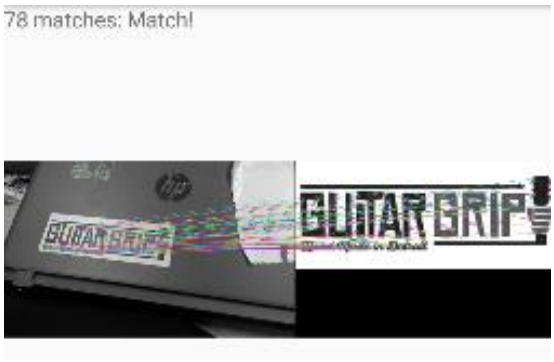
= No Match (FN)



= No Match (TN)

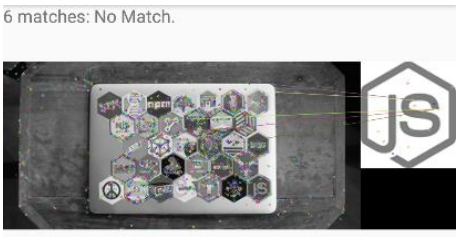


= No Match (FN)

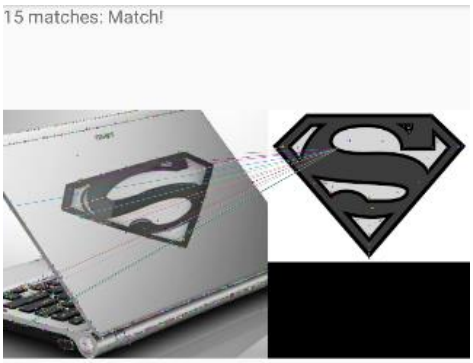


= Match (TP)

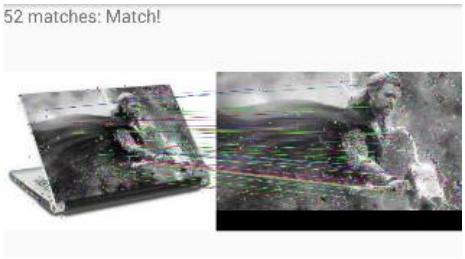
= No Match (FN)



= No Match (FN)

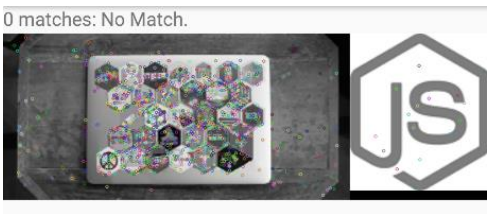


= Match (TP)

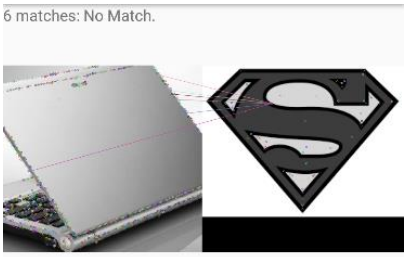


= Match (TP)

= Match (TP)



= No Match (TN)



= No Match (TN)

0 matches: No Match.

= No Match (TN)

Electronic Resources/Code