ESTIMAGE: A MOBILE APPLICATION FOR IMAGE TO OBJECT COUNT ESTIMATION

by

Yonghong Chen

Thesis submitted in partial fulfillment of the

requirements for the Degree of

Bachelor of Computer Science with

Honours in Computer Science

Acadia University

March 2016

© Copyright by Yonghong Chen, 2016

This thesis by Yonghong Chen
is accepted in its present form by the
Jodrey School of Computer Science
as satisfying the thesis requirements for the degree of
Bachelor of Computer Science with Honours

Approved by the Thesis Supervisor

Dr. Daniel L. Silver	Date

Approved by the Head of the Department

Dr. Darcy Benoit	Date

Approved by the Honours Committee

Dr. Anna Redden	Date

I, Yonghong Chen, grant permission to the University Librarian at Acadia University to reproduce, loan, or distrubute copies of my thesis in microform, paper or electronic formats on a non-profit basis. I, however, retain the copyright in my thesis.

Signature of Author

Date

Acknowledgements

First of all, I would like to express the deepest appreciation to my supervisor, Dr. Daniel Silver, for the expertise, assistance, guidance, and patience throughout the process of writing this thesis. Without your help this paper would not have been possible.

Thanks to Dr. Jim Diamond, my second reader, and Dr. Darcy Benoit, director of the school, for carefully reading and reviewing my thesis. Thanks to all the other professors who have taught me these few years, and all the teachers I remember and miss from the days of school back home.

Thanks to my SCUT and Acadia friends over the years; you've made these four years unforgettable. To everyone whom I've known at SCUT and Acadia, you've been a great addition to my life.

And last but not least, thanks to old friends and family from home for all the encouragements and words of advice. I would never have made it through without all of you.

Contents

Glossary **xvii**

Abstract **xix**

1 Introduction **1**

1.1 Definition of the Problem 2

1.2 Project Objectives 2

1.3 Project Scope 2

1.4 Overview 3

2 Background **5**

2.1 Android Platform 5

2.1.1 What is Android 6

2.1.2 Why Android 6

2.1.3 Android Version 7

2.1.4 Android Frameworks 7

2.1.5 Android Emulator 8

2.2 Server 8

2.2.1 RESTful Design 8

2.2.2 Server application 9

2.2.3 Machine Learning 9

2.2.4 Random Regression Forest Algorithm 10

2.2.5 ilastik 10

2.2.6 Octave 12

3 Requirement Analysis **13**
3.1 Refinement of Problem . . . 13
3.2 Context Diagram . . . 14
3.3 Use Cases . . . 16
3.3.1 Photo Selection . . . 16
3.3.2 Image Record Management . . . 17
3.4 Requirement Specification . . . 18
3.4.1 Functional Requirements . . . 18
Photo Selection . . . 18
Image Record Management . . . 18
3.4.2 Non-functional Requirements . . . 19
System Platform . . . 19
Application User Interface . . . 19
System Performance — Time and Space . . . 20
Error Handling . . . 20

4 System Design **21**
4.1 Interactions Design . . . 21
4.1.1 User—Estimage Client Application Interaction . . . 22
Photo Selection . . . 22
Image Record Management . . . 22
4.1.2 Estimage Client—Server Application Interaction . . . 23
4.1.3 Estimage Server Application—Octave and ilastik Interaction . 25
4.2 Estimage Client Application . . . 26
4.2.1 Interface Flow . . . 26
4.2.2 Interface Design . . . 28
Home and Image Repository Interface . . . 28
Edit Record Interface . . . 29
Record Information Interface . . . 30
4.2.3 Internal Architecture . . . 31
4.3 Estimage Server Application . . . 33

4.4 Database Model 34

5 Implementation **37**

5.1 Problems and Solutions 37

5.1.1 Storing Image Records 37

Maintain records when application is exited 38

Maintain records when the screen is rotated 38

Conclusion 39

5.1.2 Reducing Active Memory Usage of Image Records 39

Solution 39

Conclusion 39

5.1.3 Tracking Estimation Progress of Image Records 39

Solution 40

Conclusion 41

5.1.4 Ensuring Only Valid Image Formats are Submitted 41

Solution 41

5.2 Open-Source Techniques 41

5.2.1 Native Camera and Gallery 42

5.2.2 Recycler View with Card View 43

5.2.3 Pager Sliding Tab Strip 43

5.2.4 Floating Action Button 43

5.2.5 Touch Image View 44

5.3 Interface Implementation 44

5.3.1 Home Interface 44

5.3.2 Image Repository Interface 44

5.3.3 Edit Record Interface 46

5.3.4 Image Record Information Interface 46

5.3.5 Image Display Interface 47

6 Testing and Evaluation **49**

6.1 Functional Testing 49

Photo Selection 50

Image Record Management 50
6.2 Non-functional Testing 52
System Platform 53
Application User Interface 53
System Performance — Time and Space 54
Error Handling 55
6.3 Estimation Evaluation 56
6.3.1 Coin Estimation 56
Objective 56
Materials and Methods 57
Results 57
Discussion 60
6.3.2 Log Estimation 60
Objective 60
Materials and Methods 60
Results 61
Discussion 64
6.3.3 Blueberry Estimation 65
Objective 65
Materials and Methods 65
Results 67
Discussion 70
6.3.4 Conclusion 70

7 Summary and Conclusion **71**
7.1 Future Work 72

A Test Log **73**

Bibliography **79**

List of Tables

A.1 10 Tests on Estimage Client Application Performance 73
A.2 10 Tests on Size of Data Created on User's Device 74
A.3 10 Tests on Estimation Time Performance 74

List of Figures

2.1 ilastik ML Model Training Sample 11
2.2 ilastik Menu Options 12

3.1 Context Diagram 14
3.2 Use Case Diagram 16

4.1 User—Estimage Client Application Interaction Diagram 23
4.2 Client—Server POST 24
4.3 Client—Server PUT 25
4.4 Estimage Server Application—ilastik Interaction Diagram 26
4.5 Estimage Client Application Interface Flow Diagram 27
4.6 Home and Image Repository Interface 29
4.7 Edit Record Interface 30
4.8 Record Information Interface 31
4.9 Estimage Client Application Internal Architecture Diagram 32
4.10 Database Model Diagram 35

5.1 Open-Source Techniques in Estimage Client Application 42
5.2 Design and Implementation of the Home Interface 45
5.3 Design and Implementation of the Image Repository Interface 45
5.4 Design and Implementation of the Edit Record Interface 46
5.5 Design and Implementation of the Record Information Interface . . . 47
5.6 Implementation of the Image Display Interface 48
5.7 Image Display Interface with Zoom-In 48

6.1 Sample Image of Coins for Training 57
6.2 Sample Image of Coins for Testing 58
6.3 Coin Estimation Results Diagram 59
6.4 Coin Estimation Error Rate Diagram 59
6.5 Image of Logs for Training 61
6.6 Original Images of Logs for Testing 62
6.7 Density Map Images of Logs for Testing 62
6.8 Log Estimation Result Diagram 63
6.9 Log Estimation Error Rate Diagram 63
6.10 Log Estimation Error Rate Diagram—Images with Similar Logs . . . 64
6.11 Sample Image of Blueberries for Training 66
6.12 Image of Blueberries with Consistent Background for Testing 66
6.13 Image of Blueberries with Mixed Background for Testing 67
6.14 Blueberry Estimation Result Diagram 68
6.15 Blueberry Estimation Error Rate Diagram 68
6.16 Blueberry Estimation Error Rate Diagram—Consistent Background . 69
6.17 Blueberry Estimation Error Rate Diagram—Mixed Background . . . 69

A.1 Logging Example 1 75
A.2 Logging Example 2 76
A.3 Logging Example 3 77

Glossary

Material Design: Material design is a popular interface design style that is recommended by Google [Goo15e]. It encourages good design with innovation and unified user experience across different platforms and device screen sizes.

Interface: In this thesis, interface is the user interface in an Android mobile application that is responsible for displaying contents to users.

Inflate: In Android application development, the action to create a user interface is called inflating an interface.

Desirable: A desirable requirement is an optional requirement that would be good for the system.

Essential: An essential requirement must be satisfied by the system to provide key functions to users.

Activity: An activity is a data structure for Android application development. It is able to inflate an interface for user interactions [Goo15a]. Most of the time only one activity is displayed on screen and runs as the main thread in order to handle all click or touch events. Multiple activities can be run in sequence to achieve one function.

Fragment: A fragment is usually viewed as a behavior or a small piece of an activity's interface that can be reused by other activities [Goo15d]. Several fragments can be pieced together to form a well-designed interface in an activity.

AsyncTask: In Android application development, the asyncTask function is used in an activity or a fragment to execute a task without interrupting the response of the current interface. To achieve this, the asyncTask function runs as a new asynchronous thread.

SQLite Database: The SQLite database is a popular light-weight open source database that is fully supported by the Android operating system. Users' data can be easily stored in and retrieved from the SQLite database without causing heavy burden on the memory or CPU of their hand-held device.

Image Record: Image record is an essential data structure in the Android application of the harvest-estimation system. It contains information about an image and its associated estimate. Instead of the original image, the image record keeps a thumbnail image version to reduce the memory requirement.

Estimate: An estimation on an image produces the estimate of the number of objects in the image and a density map image.

Density Map Image: The density map image is one of the results from the estimation on an image. It recognizes the density of the objects in an image by marking the non-object pixels black and the object pixels white.

Pending: Pending is a status of an image record indicating that the image record is ready to be uploaded for estimation.

Uploading: Uploading is a status of an image record indicating that the image record is being uploaded for estimation.

Abstract

A major problem in the agricultural field is accurately estimating the yield of produce. Typically, farmers must wait to measure their crops after harvest using manual mechanical equipment. There is value in having better methods of yield estimation based on data that can be captured with inexpensive technology in the field, such as a smartphone. We develop a smartphone application for the Android platform with access to a cloud-based machine learning (ML) service that can estimate the amount of crop on a bush or tree from an image. The development of an image-to-object-count estimation system called Estimage is presented. The Estimage system consists of an Android client application for user interaction, a PHP server application for request handling, an Octave program for image normalization, and an open-source ML software package called ilastik that applies a predictive model to an image. The functionality of the mobile client application is tested and the system satisfies all of the functional requirements and most of the non-functional requirements. The system is tested on the images of coins on a table, logs stacked in a pile, and blueberries on the bush. We show that the system is capable of accurately estimating the count of objects in the images that have objects of the same size, colour, and shape as that used to train the model. The system is good at distinguishing object pixels from the image background pixels provided the background is consistent with that used to train the model.

Chapter 1

Introduction

Industries are in need of high-tech methods to aid in their work flows so as to maintain efficient operation. In the agricultural field, farmers rely on manual methods and mechanical equipment to measure harvest, and this can only be done after the crop is picked. It would be beneficial to have a system that can estimate the crop yield prior to harvest so that, for example, farmers can better plan the use of human and material resources on the farm.

Machine learning (ML), a sub-area of artificial intelligence, focuses on the study and construction of algorithms that learn from and make predictions from data [KP98]. One method of machine learning that could help with harvest measurement is called object classification. However, object classification is suitable for identifying separate objects and less so for estimating the number of overlapping objects [FNKH12]. A better approach is to estimate the density of pixels associated with the object integrated across the image. A machine learning method known as a random regression forest can be used to do this integration. A random regression forest can provide an accurate result if the proper learning parameters are used and sufficient training examples of the object are provided. This thesis will illustrate the development of an image-to-object-count estimation system called Estimage, which includes an application for the Android platform that calls cloud-based machine learning methods. We will evaluate how well the application functions to provide object count estimates to users.

The remainder of this introductory chapter proceeds as follows: Section 1.1 defines the problem; Section 1.2 defines the project objectives; Section 1.3 presents the project scope; and finally, Section 1.4 outlines the thesis document.

1.1 Definition of the Problem

In agriculture, farmers have a difficult time accurately estimating the yield of their produce prior to harvest. They are unable to make a good estimate of the crop yields on the tree or on the bush until they harvest and measure all the crops with manual mechanical equipment.

We wish to develop a smartphone application with access to a cloud-based machine learning service that can estimate the amount of crop on a bush or tree, such as the number of pints of berries on a blueberry bush. With this application, farmers would be able to predict their harvest volume and to take actions in advance to increase profit, such as making better marketing plans or planning use of available material and human resources on the farm.

1.2 Project Objectives

The objective of this project is to design and develop the Estimage software system to help farmers evaluate crop yields in advance, and to assure the practicability of machine learning on a real life problem in the agricultural sector. The objective is achieved in two steps. First, a smartphone application is designed and implemented to allow farmers to upload images of crop fields to the server for estimation. Second, predictive models are built using existing machine learning software on the server to estimate the crop yields pictured in the uploaded images.

1.3 Project Scope

The project presented herein is restricted in several ways to make it practical for an honours research project. The smartphone application is developed for the Android

platform, which is an operating system for hand-held devices. The Android platform provides a convenient way for users to perform operations in the field and it has a larger user group than any other hand-held operating system, such as iOS or Windows Phone [McC13].

We have initially trained the Estimage system to estimate (1) the number of coins laying on a table, (2) the number of logs that are stacked in a pile, and (3) the number of berries on a bush. Theoretically, the system can be used to estimate the count of any object in an image, such as fish, apples, cans, and hats.

The goal of this project is not to create the best machine learning method for developing the estimation. Instead, predictive models are trained using existing machine learning and image processing software. The focus is on the development and testing of the Android application that must capture, format, and request execution of the ML models on the server and then receive and display the predicted results on the mobile platform. On-going research is intended to build upon the preliminary ML methods that are used in this thesis.

1.4 Overview

The remainder of this thesis is laid out as follows. Chapter 2 introduces background information on both Android platform and machine learning. Chapter 3 presents the requirement analysis stage of the project. Chapter 4 presents the project's design stage and shows the interactions between the user, the Android application, the server and the machine learning model. Chapter 5 presents the implementation stage showing the major problems and software solutions. Chapter 6 presents the testing stage and evaluates the system on its performance in estimating the number of coins, logs, or blueberries from images. Chapter 7 concludes the thesis, followed with a summary and suggested improvements for the future.

Chapter 2

Background

This chapter presents the relevant background knowledge on the problem at hand and the methods and technologies that are utilized to solve the problem. Information in the following sections will lay the groundwork for how we analyse, design and implement an image-to-object-count estimation system. The system, called Estimage, has the ability to estimate crop yields depicted in images of the crops in the field.

The basics of the Android smartphone platform, including its general usage and development framework, are described in Section 2.1. In Section 2.2, technologies used on the server side are reviewed.

2.1 Android Platform

Dating back to 2003, the Android operating system was originally developed for digital cameras [Wel13]. Later, the company that created Android diverted its efforts to a mobile operating system and was acquired by Google as a move into the mobile phone market [Elg05]. Over the last 10 years, Android has become the most popular mobile operating system in the world and is currently developed by Google.

The following subsections describe the internal framework and architecture of Android, and why we chose Android as the platform for development.

2.1.1 What is Android

Android is a mobile operating system based on the Linux kernel. Based on the Java Virtual Machine, the Dalvik Virtual Machine is specially designed, created, and optimized for Android so as to utilize some core features of Linux [Poi15]. Android provides a complete set of core development libraries written in the Java programming language. With these libraries, developers are able to build Android applications using the Java programming language. In the Estimage system, the Android application is developed in Java using Android Studio, which is an integrated development environment (IDE) developed by Google.

2.1.2 Why Android

Android has several significant benefits:

Wide Use: Android is the most popular mobile operating system, having 82.8% of the worldwide smartphone market share in 2015 [Cor15].

Convenience: Android provides a convenient platform for mobile computing on hand-held devices.

Low Cost: Compared to other mobile operating systems, Android provides a low-cost alternative in the smartphone market.

Frequent Updates: From 2008 until now, more than 22 versions of the Android operating system have been developed, on average three versions per year [Goo15c]. Frequent updates increase the functionality and stability of the Android system and help guarantee its future potential.

Low Cost Development Entry: There is no cost for the Android application development kit. Everyone is free to download the development environment and access open-source libraries for Android application development without cost or registration.

2.1.3 Android Version

It is important to decide which version of Android we want our Estimage system to support. Many versions of the Android operating system have been developed, such as 4.0 (Ice Cream Sandwich), 4.4 (KitKat), 5.0 (Lollipop), and 6.0 (Marshmallow). Accordingly, for developers, there are many Application Programming Interface (API) levels for development, from level 1 to level 23 [Goo15c]. In the Estimage system, the Android application supports all Android versions above 4.0 Ice Cream Sandwich, which accounts for 90% of the total Android user population.

2.1.4 Android Frameworks

An Android framework is a set of guidelines that Android application developers can follow to develop a robust Android application. Several well-developed Android frameworks are available, such as the Model-View-ViewModel (MVVM) design pattern and the Model-View-Controller (MVC) design pattern. In the MVC design pattern, the Model represents the original data, the View represents the user interface layout, and the Controller determines which Model is used and which View is displayed on screen in response to any action called from the View [Soh11]. The MVC design pattern allows the Controller to mix all the actions called from different Views together and this pattern greatly increases the complexity of the application as the foreground layouts and the background data structures become more complicated.

The MVVM design pattern uses the ViewModel, which is an intermediate Model that binds the View to a corresponding Model by storing a presentation format of the Model data. In other words, each View has a ViewModel to handle its actions and any change in a bound Model will notify the ViewModel to update the View. For example, a list View is bound with the Model data of each item in its list in a ViewModel.

Compared to MVC, the MVVM design pattern has two major benefits. The MVVM design pattern creates a ViewModel for each View to handle the View's interactions with the Models, which separates the interactions of each View for different purposes and increases the application's maintainability. Second, the ViewModel

groups the interactions of a View together and allows each View to be tested individually. For the above reasons, the MVVM design pattern is used in the Android application of the Estimage system.

2.1.5 Android Emulator

For Android application developers, one of the main tools for testing is the Android emulator. The Android emulator serves as a virtual mobile device that can test Android applications on computers without the need for a real hand-held device. Developers are able to quickly test interfaces and functionalities of their applications and make adjustments before deploying the applications to physical devices for further testing.

2.2 Server

In this project, the server side programs for estimating objects in an image are running on a popular operating system called Ubuntu Linux. A server application was written in PHP to communicate with the Android client application over HTTP. The server application is able to call an Octave program to invoke ilastik to run a machine learning predictive model on an image to generate an estimate of object count. The estimation results are saved in a MySQL database. The following subsections introduce these server-side techniques that are utilized in the Estimage system.

2.2.1 RESTful Design

There are two common protocols called Transmission Control Protocol (TCP) and Hypertext Transfer Protocol (HTTP). While TCP establishes and maintains a channel session for data transmission, HTTP follows a connectionless request/response standard that is useful for client-server systems [Dif15]. In this project, HTTP is chosen as the bridge for communication between the Android client application and the server application in the Estimage system. However, HTTP cannot maintain a session or session state. To accomplish this, the Estimage system adopts a software

architecture style called Representational State Transfer (REST), which communicates over HTTP. In a RESTful system, four types of HTTP requests are commonly used: "GET", "POST", "PUT", and "DELETE".

The POST operation is a request from the client to add new data to the database on the server. The GET operation requests the server to retrieve a set of specific data from its database and send it back to the client. A PUT request is utilized when the client wants to update a piece of data on the server database. The DELETE operation tells the server to delete a certain piece of data.

2.2.2 Server application

A server application is a program that runs on a server computing system that is capable of handling requests from clients. A server application can be written in different programming languages such as C++, PHP, or Ruby on Rails. While most of these programming languages take weeks for developers to learn, PHP is easy to learn and developers can create a functioning server application within several days. There are many mature PHP frameworks for development and the performance of a PHP server application is sufficient to handle requests for most business applications. PHP nicely supports communication over HTTP and MySQL database access by default, which is one of the most popular server-side databases. For the above reasons, PHP is used in the Estimage system to develop a HTTP server application.

2.2.3 Machine Learning

As early as the 1950s, computer scientists studying artificial intelligence have been interested in enabling computers to mimic a human's way of learning from examples [RN03]. However, the field of machine learning was not valued until the 1980s, when methods of developing more complex non-linear models were developed [Lan11]. **Since then, machine learning has flourished and is now one of the most promising fields of artificial intelligence.**

Machine learning applications tackle the problem of how to develop predictive models directly from training examples. For example, marketing models are developed

based on historical sales data. One of the most popular applications of machine learning is for recommending suitable products and services to customers who are surfing the web; examples of this are provided by Amazon and Yahoo.

2.2.4 Random Regression Forest Algorithm

One category of algorithms defined in machine learning is called supervised learning, which trains an algorithm to learn to produce approximate outputs for specific inputs [MRT12]. The "random regression forest algorithm" belongs to this category. A random regression forest is an ensemble learning method that works by building multiple decision trees at training time and outputting the mean prediction (regression) of the trees [Ho95]. A decision tree is a predictive model which maps input variables related to an item to the estimate or prediction of the item's target value [Qui86]. In the tree structure, leaves serve as class tags and branches show unions of features that bring on those class tags. In random regression forest, decision trees are built with random selection of features.

The number of trees in a random forest and the maximum depth of each single tree are two key factors that affect the smoothness of the prediction and performance in the algorithm. In other words, few and shallow trees can create a model that is under-fitted to the data, while many and deep trees may create a model that is over-fitted to the data [FNKH12]. In the Estimage system, machine learning predictive models are built with 10 decision trees and 50 maximum depth.

2.2.5 ilastik

ilastik is an open-source image analysis tool that generates machine learning predictive models based on training images and parameters [SSKH11]. Various workflows are provided to automate the processes of image classification, segmentation and analysis.

One of the provided workflows, called density counting, satisfies our need to count overlapped objects of similar appearance in images such as blueberries or coins [FNKH12]. The process of density counting is as follows: (1) The objects

to be estimated and their surroundings are manually labelled in a set of training images using a marking tool. (2) ilastik uses image processing techniques to transform the marked pixels of labelled objects into image features. (3) The random regression forest algorithm introduced in Section 2.2.4 is applied to train a predictive model based on the image features. (4) The model is used to predict the fraction of object per pixel. (5) ilastik calculates the estimate of the number of objects in an image by adding all of the fractions of objects per pixel in the image.

Figure 2.1 depicts how a machine learning predictive model is trained using ilastik.

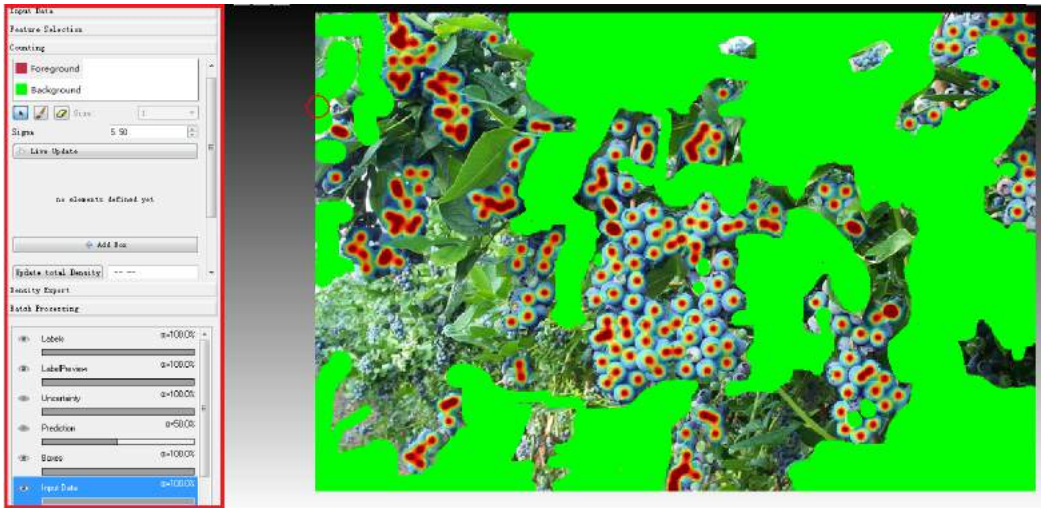


Figure 2.1: Using ilastik to train a machine learning model on an image

The left hand side of Figure 2.1 is a functional menu detailed in Figure 2.2. The menu lists all the functions provided by the density counting workflow of ilastik. (1) Images can be imported to ilastik in the Input Data option. (2) The Feature Selection option provides features that the random regression forest algorithm can use for constructing predictive models. (3) The Counting option allows the user to mark selected pixels as being of the object class to count. The Update Total Density button starts running the random regression forest algorithm on an image to produce an estimate. (4) The Density Export and Batch Processing options are used for generating a density map image for each image to be estimated in a batch process. The density map image depicts the density of objects in an image by marking the

non-object pixels black and the object pixels white. (5) The visibility options below are used to show or hide the colour marks that were made before.

Using ilastik, we are able to train ML predictive models for different objects and store them on a server for access by a client application.

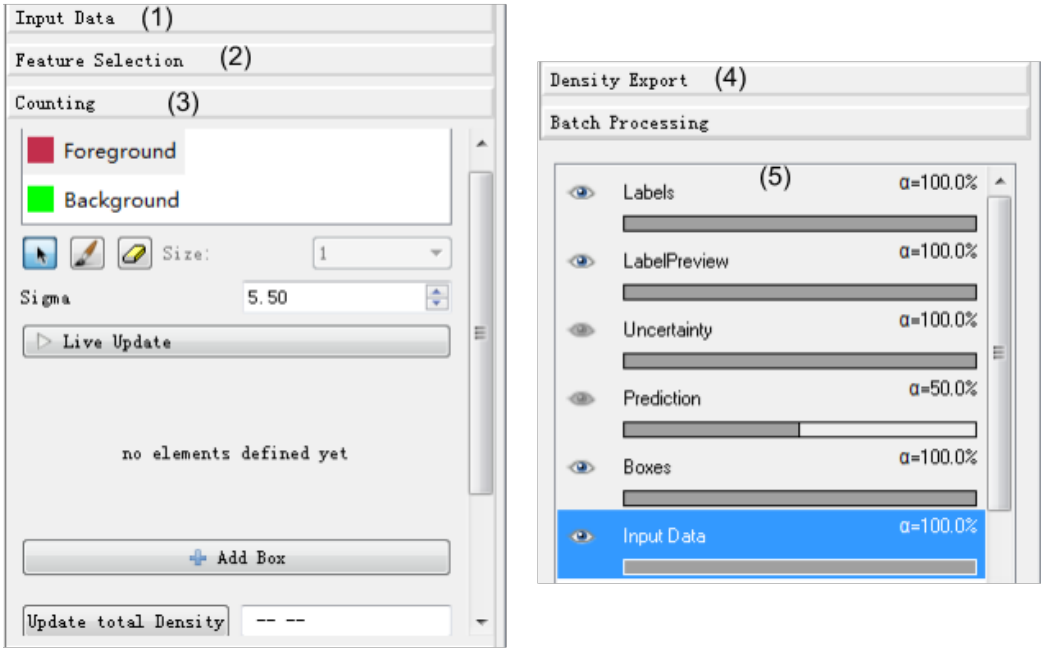


Figure 2.2: Object mark-up options and other menu options in ilastik

2.2.6 Octave

Octave is an open-source version of the popular MATLAB programming environment. MATLAB was created to be a programming language to help people do numerical computations with computers on large sets of data [EBrHW15]. It has become a common programming language for developing and executing machine learning algorithms.

In the Estimage system, an Octave program runs on the server and is responsible for reformatting the selected image and calling ilastik to apply a chosen ML predictive model to estimate the number of objects in the image.

Chapter 3

Requirement Analysis

This chapter describes the requirements analysis stage of the system development process, which serves as a beginning to develop the Estimage system. The problem is refined and presented in Section 3.1. Section 3.2 proposes a system architecture and specifies its internal components. The key use cases are presented in Section 3.3 to better describe scenarios in which users would interact with the system. We present both functional and non-functional requirements that the system must or should meet in Section 3.4.

3.1 Refinement of Problem

The primary goal of this project is to create the Estimage system through which farmers are able to take a photo of a crop in the field and estimate its yield. The Estimage system has two components to be developed: an Android application and a server application. The Android application should provide an intuitive and user-friendly user interface and allow farmers to easily upload images of crop in the field to the server for estimation. The server application should be able to estimate the crop yield from the uploaded images using machine learning predictive models.

3.2 Context Diagram

This section presents how external actors (users) interact with the Estimage system. The following is a list of entities found within and external to the system described in the context diagram in Figure 3.1.

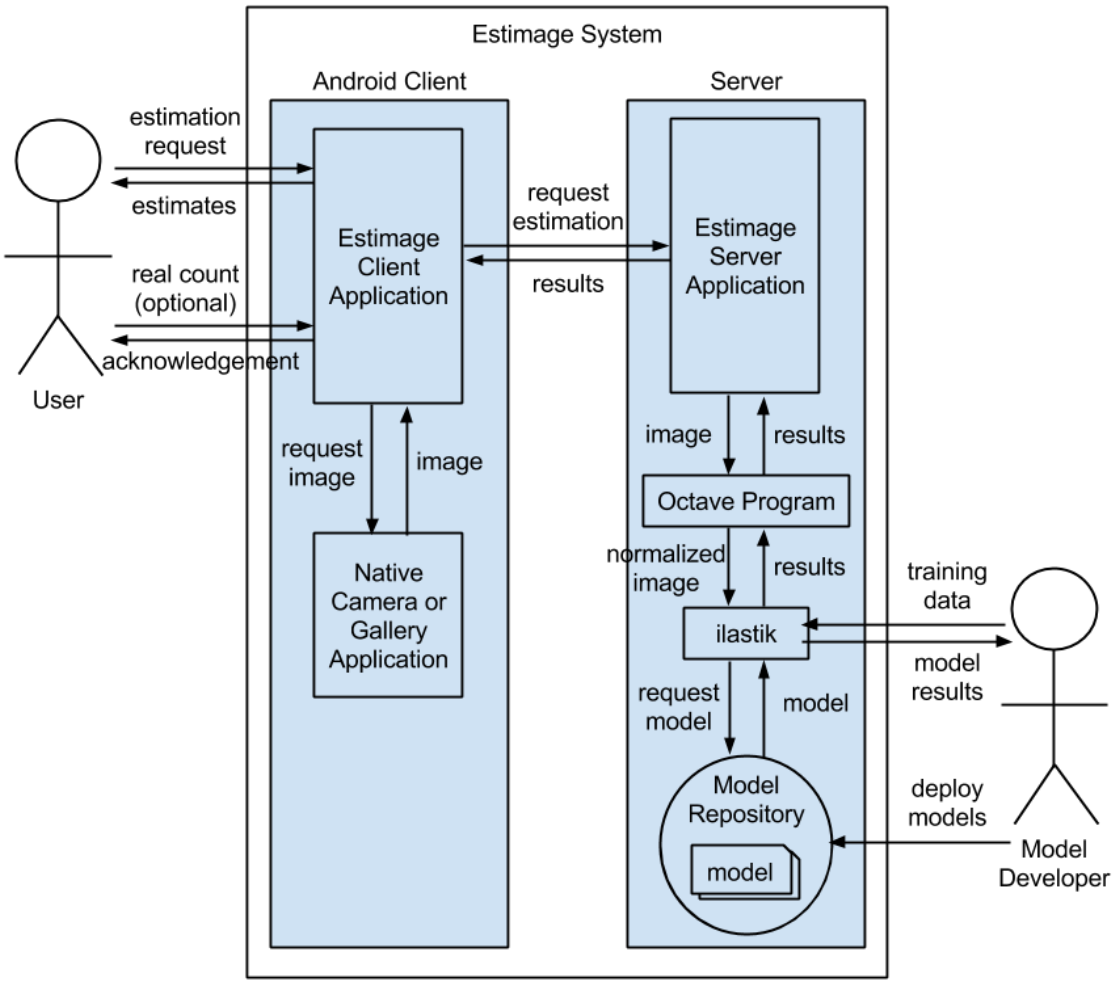


Figure 3.1: Context Diagram

User: Users are the farmers in the field who can use the system to estimate their harvest. They interact with the Estimage client application found inside the Estimage system.

Estimage Client Application: As the core component in the system, the Estimage client application provides an interface for the Estimage system to communicate with the users. It helps users take a photo with a camera application or select an image with a gallery application. It is also responsible for posting requests to the Estimage server application and displaying returned results.

Estimage Server Application: The Estimage server application handles requests from the Estimage client application, and calls the Octave program to invoke ilastik to run an estimation on an image. The Estimage server application then receives the estimation results from the Octave program and returns them back to the Estimage client application.

Octave Program: The Octave program is responsible for normalizing the images from the Estimage server application and calling ilastik to perform the estimation task on the normalized image. The estimation results from ilastik are returned to the Estimage server application.

ilastik: ilastik is an open-source machine learning software package that creates machine learning predictive models based on training images provided by the model developer. It is also responsible for running an estimation using a model from the model repository.

Model Developer: Model developers are people who are responsible for preparing and selecting training images, developing ML predictive models using ilastik, and deploying the models in the model repository for use on the server.

Model Repository: The model repository is a place to store the ML predictive models for different types of objects to be estimated. For the same object type, several models are also prepared for different sizes of objects in order to achieve the maximum accuracy. The model repository responds to requests from ilastik.

3.3 Use Cases

A use case is a series of interactions between an actor and a system to achieve an objective. Each use case describes a scenario in which the system completes a task and identifies a key requirement of the system. The following subsections classify the use cases shown in Figure 3.2 into two categories: photo selection and image record management. An image record is a data structure that contains detailed information about an image but does not contain the image itself.

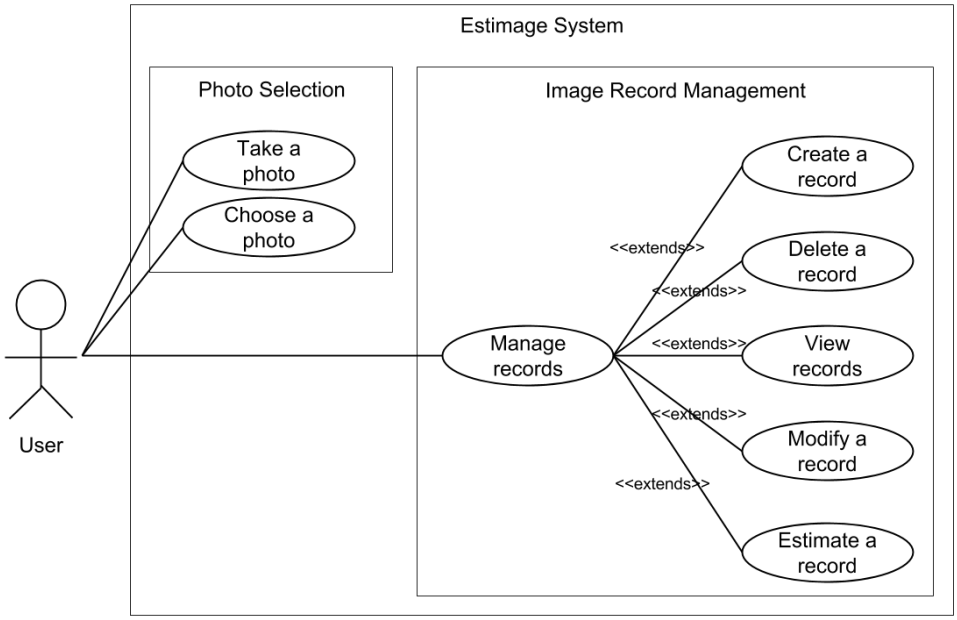


Figure 3.2: Use Case Digram

3.3.1 Photo Selection

These use cases describe how a user can choose any photo on their hand-held device or take a photo with the Estimage system. Users can ask the system for photo selection. The system then opens the device folders and provides a list of photos for users to select for estimation. Users can also instruct the system to take a photo. The system then opens the mobile device's camera application for users to take a new photo or to view a taken photo.

3.3.2 Image Record Management

The image record management category has use cases that describe how users manage their image records in the Estimage system. Five scenarios are specified as follows:

(1) Users can create an image record. After a user selects an image to be estimated, the system asks the user for a list of optional information about the image. Once the user provides the image information, the system creates an image record with the provided information and presents the new image record to the user.

(2) Users can modify an image record. A user can select an existing image record, change the information for the image record, and then save the changes. The system should then display the revised image record to the user.

(3) Users can delete an image record. A user can select an image record for deletion. The system checks that the user wishes to delete the image record. The system deletes the image record and indicates to the user that the image record has been successfully removed.

(4) Users can review their image records. A user can view the general information of the image records by instructing the system to display the image records in a list. The system then provides a list of image records for the user to scroll up and down. A user can also view the detailed information of an image record by selecting the image record. The system then retrieves the selected image record data and presents the image record's detailed information to the user, such as creation date and chosen object type, on a separate screen. If the image record has been uploaded for estimation, the user can see the estimate of the number of objects in the image and visually compare the density map image with the original image.

(5) Users can estimate the number of objects in an image attached to any image record with the system. When a user chooses an image record with an image to be estimated, the system starts uploading the image for estimation. During the estimation progress, the system updates the progress status bar indicating whether the image record is being uploaded or being estimated. After the estimation process finishes, the system displays the estimation results to the user.

3.4 Requirement Specification

The following subsections classify requirements into two categories: functional requirements and non-functional requirements.

3.4.1 Functional Requirements

Functional requirements describe tasks that the Estimage system is to perform. They are listed and classified based on use cases outlined in Section 3.3.

Photo Selection

1. [Desirable] Users can take a photo (or image).
2. [Essential] Users can select a photo (or image) from their Android albums.

Image Record Management

1. [Essential] Users can attach information to a selected image to create an image record.
2. [Essential] Users can view their image records in a list.
3. [Essential] Users can reorder the image records in the list based on information such as the creation date and the magnitude of the estimate.
4. [Essential] Users can view any of their image records in detail.
5. [Essential] Users can zoom in or out on the image of an image record.
6. [Essential] Users can modify an image record.
7. [Essential] Users can delete an image record.
8. [Essential] Users can send their image records and images for estimation — less than 25% error rate is required for this initial prototype.

9. [Essential] Users can compare an original image with its density map image after estimation.

10. [Essential] Users can save image records on their mobile device.

11. [Essential] Users can save estimation results of an image record, including an estimate and a density map image, on their mobile device.

12. [Desirable] The Estimage server application can save an image record in its database.

13. [Desirable] The Estimage server application can save estimation results of an image record including an estimate and a density map image in its database.

3.4.2 Non-functional Requirements

Non-functional requirements are features that the Estimage system should possess to support daily operations. They are concerned with platform configuration, user interface design, and system performance. The following list describes all the possible non-functional requirements.

System Platform

1. [Essential] The Estimage client application should be able to run on mobile devices with Android 4.0 Ice Cream Sandwich and above.

2. [Essential] The Estimage client application should have access to the internal and external storage on the Android device if any.

Application User Interface

1. [Essential] For better user experience, the Estimage client application should adopt the material design style, which is a popular interface design style recommended by Google.

2. [Essential] When displaying image records in a list, the thumbnails of images should be larger than the standard thumbnail size of 100×100.
3. [Desirable] The minimum font size of the Estimage client application should be 14 scale-independent pixels, which will be adjusted for both the screen density and user's preference.
4. [Desirable] The navigation of the Estimage client application should be intuitive.

System Performance — Time and Space

1. [Essential] Users should be always able to continuously interact with the Estimage client application without being blocked.
2. [Essential] Little delay should be noticed when switching user interfaces in the Estimage client application.
3. [Desirable] Each button click should respond in less than 0.2 seconds.
4. [Essential] The Estimage client application should spend less than ten seconds to display the whole list of image records (up to 50) on the screen.
5. [Desirable] Each image record should occupy no more than two megabytes in the SQLite database on the Android device.
6. [Essential] The Estimage client application should take no more than three seconds to upload an image record to the server with a network speed of one megabyte per second.
7. [Desirable] The Estimage server application should take less than five seconds to complete the estimation process of an image.

Error Handling

1. [Desirable] When the network is unavailable or the Estimage server application is down, the Estimage client application should stop all estimation tasks and indicate the error to users.

Chapter 4

System Design

In the system development process, the system design stage describes the interactions between the Estimage system entities mentioned in the previous chapter. In addition, the detailed internal design of each entity is presented.

Section 4.1 describes the interactions between users and the system with detailed communication among the Estimage client application, the Estimage server application, and the machine learning components within the system. The overall design of the Estimage client application, including interface design and internal architecture design, is presented in Section 4.2. Section 4.3 shows the image record estimation process in the Estimage server application in detail. Finally, Section 4.4 presents the database schema design for both the Estimage client application and the Estimage server application.

4.1 Interactions Design

Based on the context diagram depicted in Figure 3.1, the goal of the design stage is to develop how each system entity interacts with each other at the micro level.

In order to express the interaction steps, sequence diagrams are used that contain the following components: a big rectangle represents a system entity, and the straight line under it is the execution time line; each small rectangle on the time line indicates a time slot, and arrows between time lines indicate the interactions between two entities.

4.1.1 User—Estimage Client Application Interaction

The interactions between the user and the Estimage client application describe in detail what users can do with the system through the Estimage client application. In order to make the design meet users' requirements, the detailed sequence diagrams are derived based on the use cases drafted in Section 3.3.

In Figure 4.1, there are three main entities involved, which are the user, the Estimage client application, and the Android local database called SQLite. In general, when the user gives an instruction, the Estimage client application must take an action and return with a response. The Estimage client application would also have options to access the remote server or its SQLite database depending on actions to be taken. According to the use cases summarized in Section 3.3, user instructions can be classified into the following categories.

Photo Selection

The Estimage client application allows users to take a photo or select an existing photo with the embedded native camera application and gallery application. Once a photo is taken or selected, the camera or the gallery application returns the photo to the client application and the client application displays it to the user for creating an image record.

Image Record Management

After an image is selected, users can fill in a list of related information and instruct the Estimage client application to create an image record. Then, the created image record is directly saved in the SQLite database and the client application will display it to the user if the saving action is successful. The image record deletion or modification processes are similar to the image record creation process. Users can also request the previous image records. The client application will ask its SQLite database to retrieve all the existing image records and display the image records in a list to users.

When a user asks the client application to estimate the objects in an image of an image record, the client application sends the image record to the Estimage server

application and receives the estimation results. If the estimation is successful, the returned results should contain the estimate of the number of objects in the image and a density map image. These are saved in the Android SQLite database. Finally the client application will display the complete estimation results to the user.

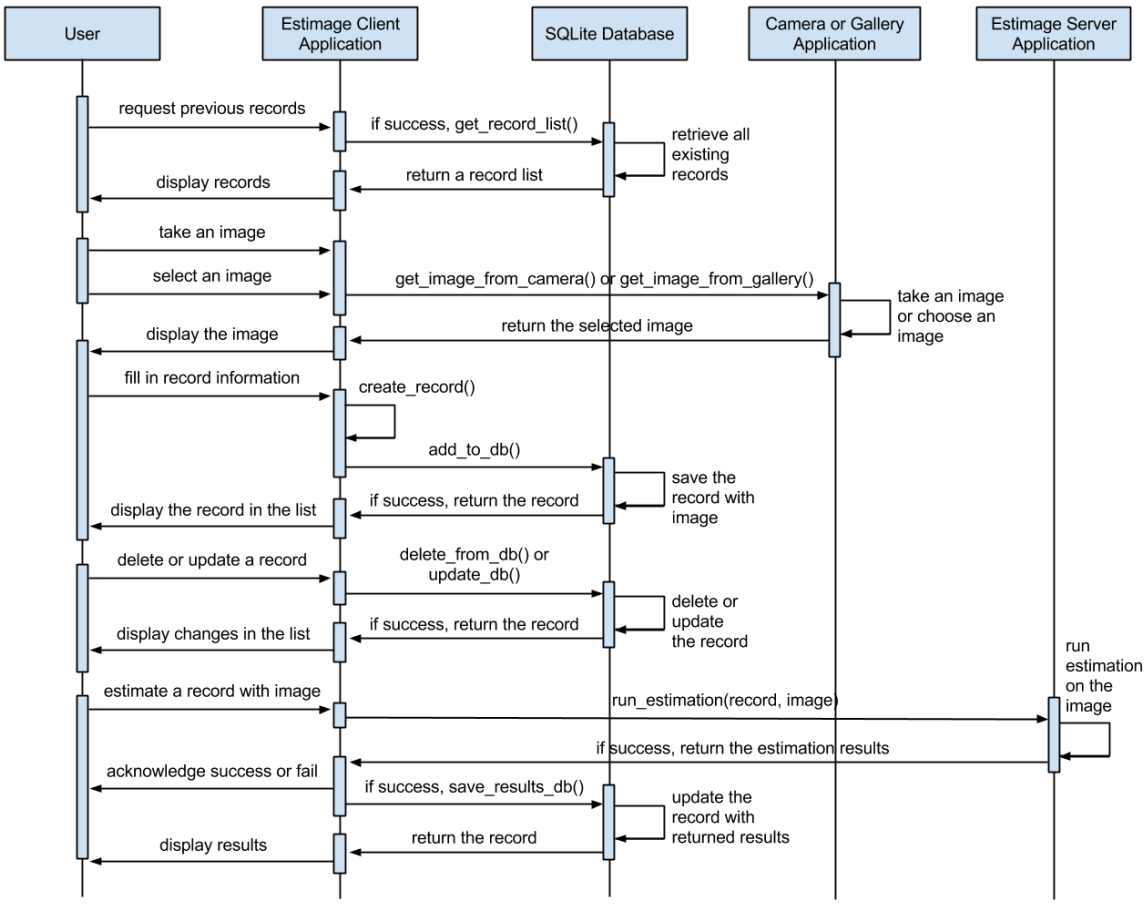


Figure 4.1: Interactions between the user and the Estimage client application

4.1.2 Estimage Client—Server Application Interaction

The detailed interactions between the Estimage client application and the Estimage server application can also be derived from Figure 3.1. According to the RESTful

system introduced in Chapter 2, two types of requests can be used in the Estimage system: POST and PUT. Each request type derives a sequence diagram to explain how the client application makes requests to the server application in different cases.

The first request method to be introduced is the POST request. In Figure 4.2, the client sends a POST request when a user wants to estimate the number of objects in the image of an image record. If the image record is not saved on the server, the image record and its image are sent to the server and the Estimage server application would add them to its database. If the image record was previously saved on the server, only the image record is sent to the server and the server application retrieves the corresponding image from its database. After running the ML model on the image, the server application saves the estimation results in its database and returns them to the Estimage client application. Additionally, a synchronization flag is attached to each image record to indicate whether it has been saved on the server.

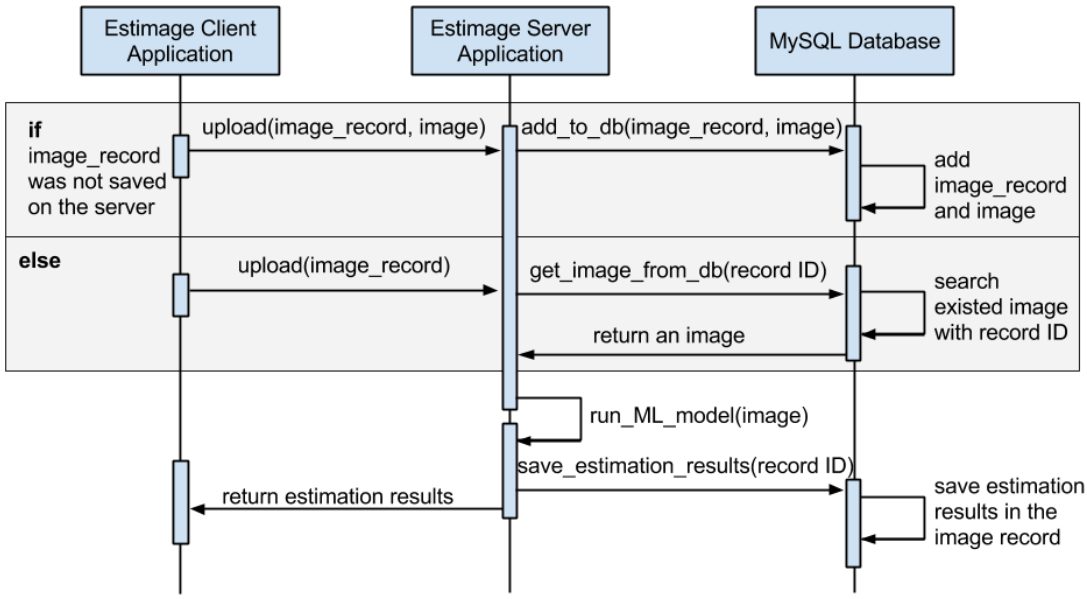


Figure 4.2: A POST request from the client application to the server application

Another request method we present here is the PUT method. In Figure 4.3, users' image records can be saved on the server as back-up. This operation skips

image records that already have copies on the server.

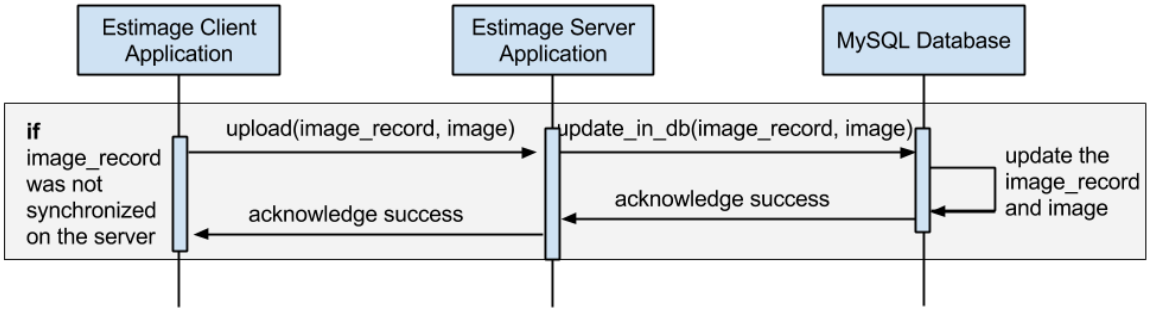


Figure 4.3: A PUT request from the client application to the server application

4.1.3 Estimage Server Application—Octave and ilastik Interaction

The last component interaction shown in Figure 3.1 is the interaction between the Estimage server application, Octave, and ilastik. A detailed design of the interaction is given by the sequence diagram of Figure 4.4.

Starting with inputs from the Estimage client application, the Estimage server application receives an image record with an image to be estimated. The Estimage server application calls an Octave program to normalize the image as the inputs for the ML predictive model. The Octave program then invokes the ilastik software to perform an estimation on the normalized image. The name of the selected estimation model is specified in the image record by the client application and is passed through the server application and the Octave program to ilastik. Using the model name, ilastik must retrieve the correct predictive model from the model repository and apply the model to the normalized image. ilastik produces a density map image and an estimate of the number of objects in the image. These are returned back to the client application through the Octave program and the server application. If the desired model is not found in the model repository, the server application will issue an error message to the client application.

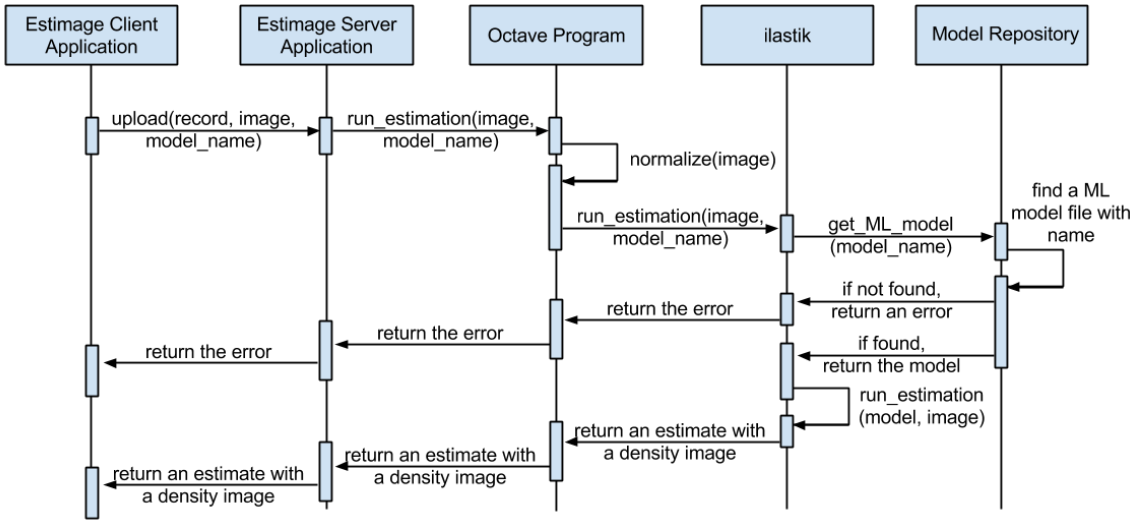


Figure 4.4: Interactions between the Estimage server application and ilastik

4.2 Estimage Client Application

The following subsections describe the detailed design of the user interfaces as well as the internal architecture of the Estimage client application.

4.2.1 Interface Flow

Figure 4.5 depicts the interface flow of the Estimage client application. The interface flow is used to sketch out how user interfaces switch from each other based on user actions. In this way, the relationship between interfaces is determined in advance for later consideration of architectural design. Each interface switching is described as below.

When the Estimage client application starts, the Estimage interface should first appear with two sub-interfaces embedded in it: the home interface and the image repository interface.

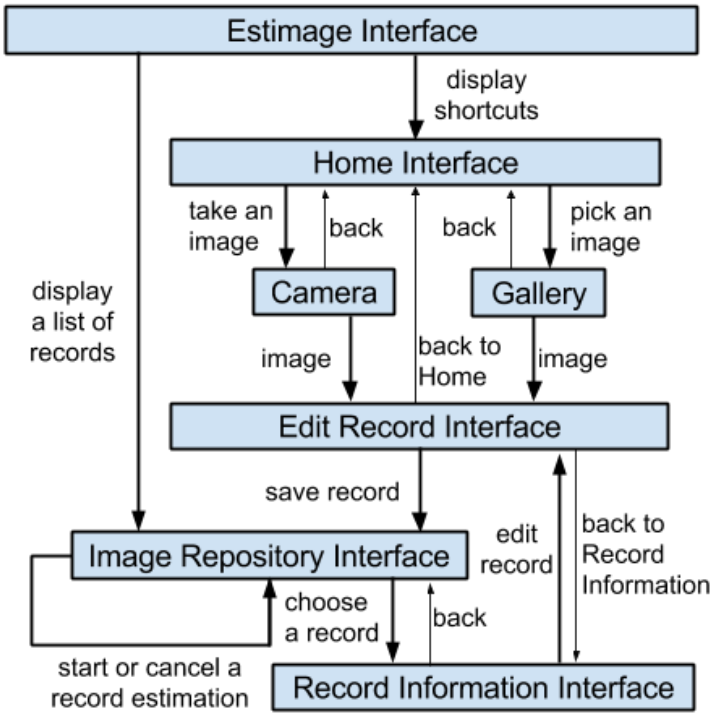


Figure 4.5: Interface flow of the Estimage client application

In the home interface, shortcuts are displayed for users to get started quickly. Users can navigate to the native camera application to take new images or the native gallery application to pick a previous image. Both of these actions lead users to the edit record interface.

In the edit record interface, the chosen image is displayed and users can create an image record with the image's detailed information, such as the title, the location, and the object type. Once this image record is saved, the client application will bring users back to the image repository interface inside the Estimage interface.

In the image repository interface, the recent image records are displayed in a list with the latest one on the top. With this interface users can manage their previous image records, such as adding, removing, editing or browsing. Users are also able to choose and review any image record shown in the list, thus bringing up the record information interface.

The record information interface displays not only an image record's detailed information, but also the image record's image with a density map image. In this way, we can easily compare the image and the density map image and understand how well the ML predictive model identifies the objects. Users can either go back to the image repository interface or navigate to the edit record interface to make changes to the selected image record.

In addition, a task of image record estimation is activated in the image repository interface when users want to estimate an image record in the list.

4.2.2 Interface Design

This subsection details the design of each user interface that is mentioned in Section 4.2.1. Each user interface has a header indicating which interface the user is in.

Home and Image Repository Interface

As mentioned in Section 4.2.1, the Estimage interface should appear once the Estimage client application is started. The Estimage interface should provide users with the access to all key functionalities of the Estimage system. In Figure 4.6, a tab design is applied to the Estimage interface so that users can switch between the home interface and the image repository interface easily. In this way, the Estimage interface achieves its two objectives.

(1) The home interface provides users with shortcuts to create image records conveniently. Two buttons are placed in the center to allow users to take a photo or select a photo.

(2) The image repository interface offers users a place to browse their image records in a list. Each image record in the list has a thumbnail of its image on the left, a title with big font above, a static status text below, and an upload button on the right. While the thumbnail helps users to distinguish the image records from each other, the status text tells users whether an image record is pending, being uploaded, or being estimated. Furthermore, users can use the upload button to start or cancel

an estimation at any moment. In addition, a floating action button is displayed at the bottom-right corner to provide users with a convenient way to perform operations such as “compose an image record”, “upload all image records”, “cancel all uploading processes”, and “delete all image records”. The last thing to be mentioned is the switch at the top of the image repository interface, which allows users to reorder the list of image records either by the creation date or the magnitude of the estimate.

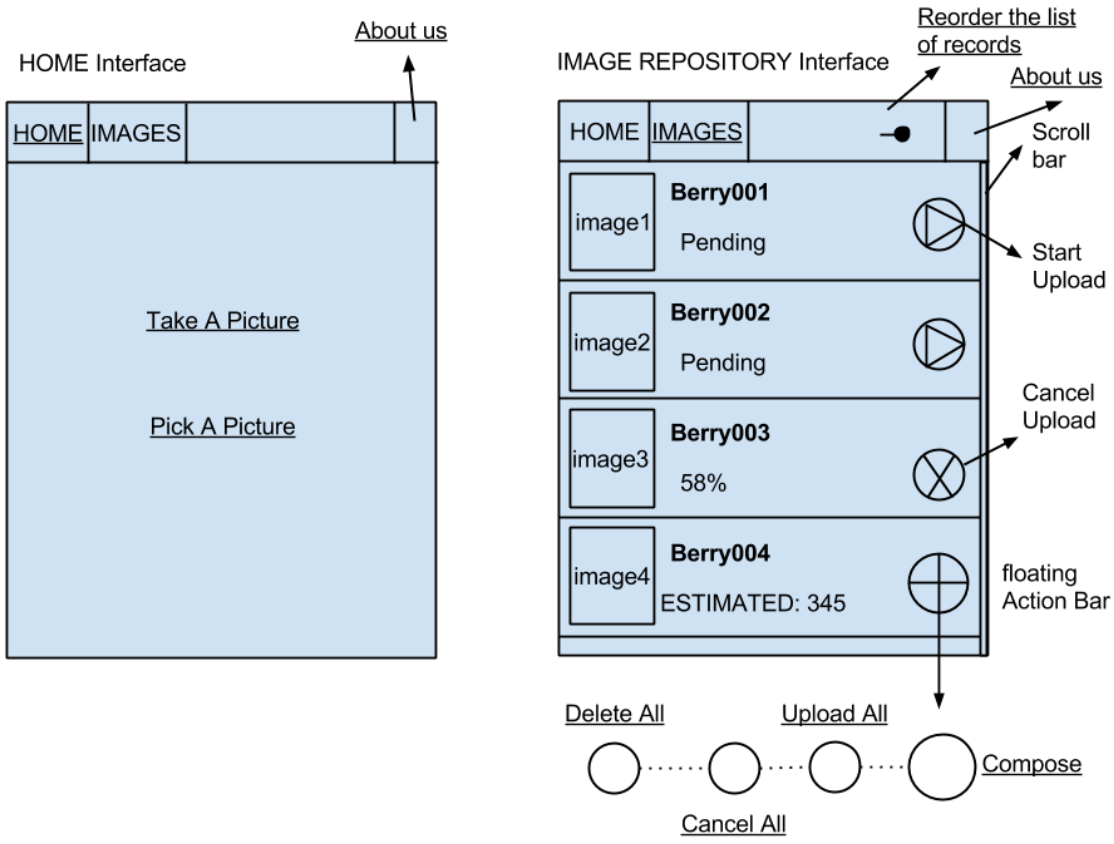


Figure 4.6: Depiction of the Home and Image Repository Interface

Edit Record Interface

Figure 4.7 depicts the edit record interface. After users take a photo or select a photo, they will be directed to the edit record interface with a selected image from either

the camera interface or the gallery interface.

The edit record interface displays the selected image that is big enough for users to recognize. A list of related text fields are presented for users to fill in, such as the file name, the object type, the location, and the optional actual count. The save button is located at the top-right corner for users to access easily.

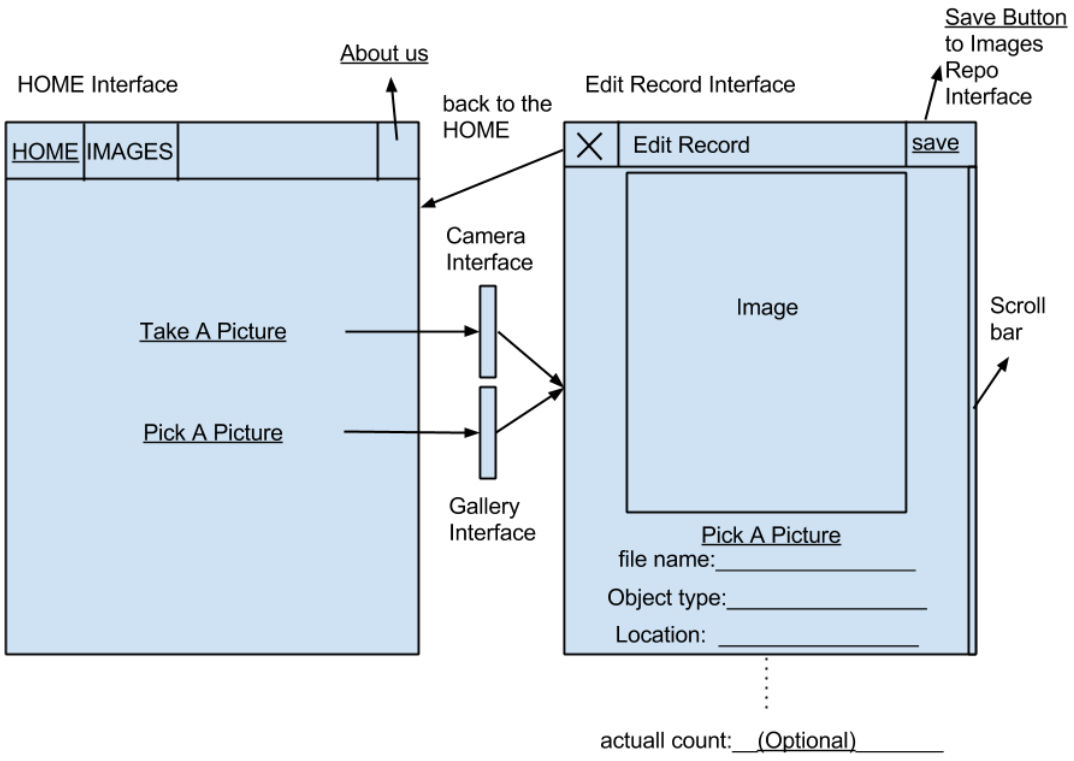


Figure 4.7: Depiction of the Edit Record Interface

Record Information Interface

Figure 4.8 depicts the record information interface. When users click on any image record shown in the image repository interface, they are directed to the record information interface, which displays the detailed information of the chosen image record. Compared to the edit record interface in Figure 4.7, the record information interface turns all the filled text fields into static texts and displays them neatly on screen.

Directly below the image, there is a slider that controls the opacity of the density map image from the estimation results. Users can conveniently compare the original image with the corresponding density map image. Additionally, a floating edit button is displayed at the bottom-right corner to direct users to the edit record interface.

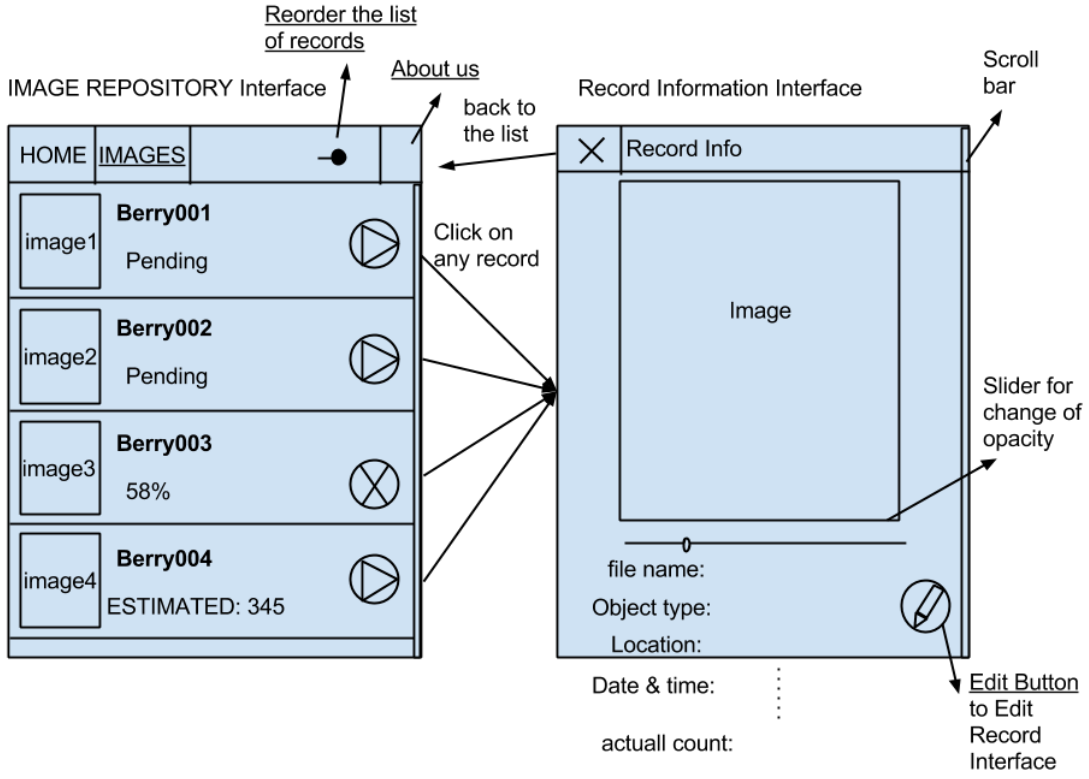


Figure 4.8: Depiction of the Record Information Interface

4.2.3 Internal Architecture

The internal architecture of the Estimage client application is the combination of data structures that support user interface presentation and the logic associated with each user interface. Based on the interface flow diagram shown in Figure 4.5, the internal architecture is designed as depicted in Figure 4.9.

In Android development, an activity is a data structure that inflates (creates) a user interface and displays it to users to perform one function. A fragment is a

piece of an activity that performs a sub-function. An asyncTask is used to perform a time-consuming task without interrupting user interactions.

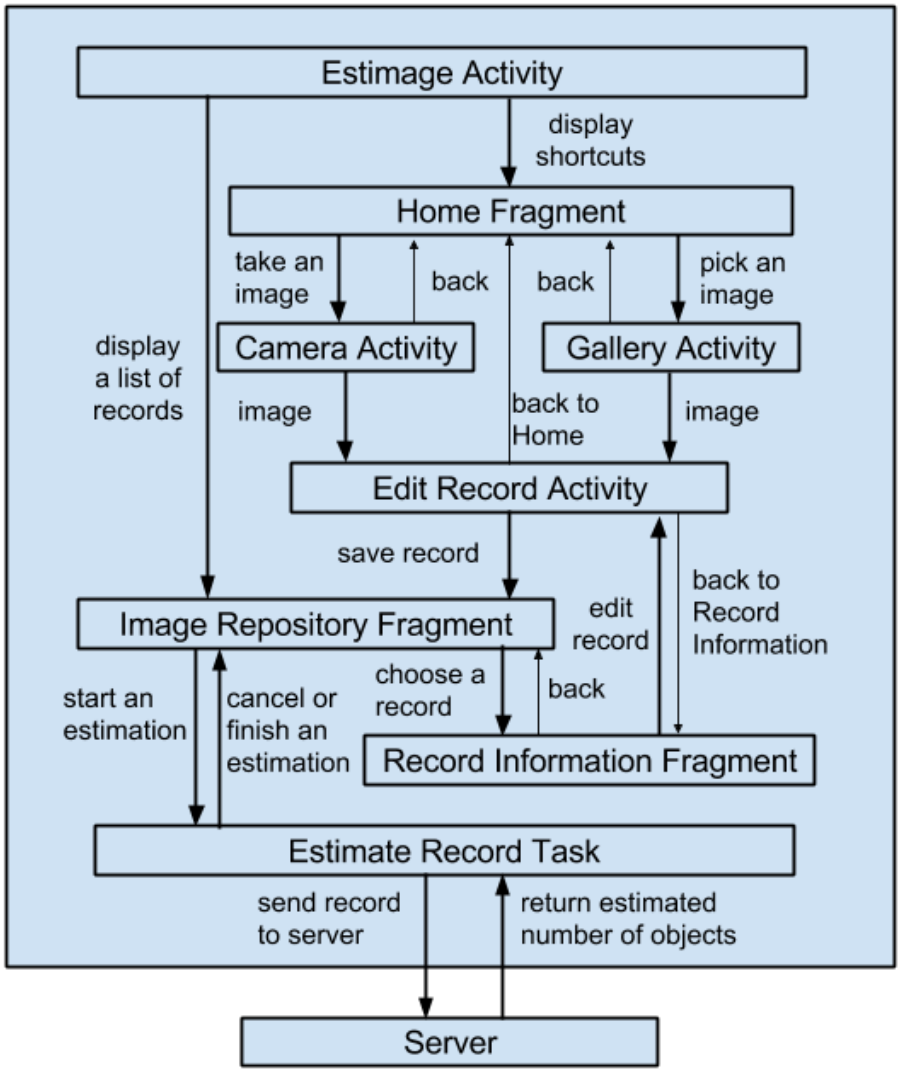


Figure 4.9: Internal architecture of the Estimage client application

First, the Estimage activity is started. The Estimage activity is responsible for displaying the Estimage interface embedded with the home interface and the image repository interface, which are inflated by the home fragment and the image repository

fragment respectively.

In the Estimage activity, the home fragment handles users' requests to take a photo or select an image. The native camera activity or gallery activity is then started to accomplish its task and returns an image back to the home fragment for later handling. Automatically, the home fragment stops the Estimage activity and starts the edit record activity to display the returned image in the inflated edit record interface.

In the edit record activity, input events are handled and input information are saved into an image record. After users instruct the edit record activity to save the image record, the Estimage activity is recovered and the image record is passed back to the Estimage activity through the home fragment. Once the Estimage activity receives the image record, it instantly updates the list of image records presented in the image repository fragment.

In the image repository fragment, touch events are handled when users scroll the list of image records or click any image record. When an image record is clicked, the record information fragment is created to temporarily display the image record's detailed information in the record information interface. Furthermore, the record information fragment can stop the Estimage activity and start the edit record activity for image record editing.

Additionally, the image repository fragment can handle estimation requests from users and create a separate thread to run a task called "estimate record task", which basically sends the chosen image record to the server and waits for the returned results.

4.3 Estimage Server Application

The Estimage server application is another core component in the system. Generally, the Estimage server application is responsible for image record estimation and temporary backup. The sequence diagram of Section 4.1.3 describes the data flow of the estimation process on the server. This section focuses on the detailed design of each step in the estimation process.

Because the Estimage server application, the Octave program, and the ilastik are not embedded in each other, they cannot directly communicate with each other. The default system command function in each program is an easy way to ask the Ubuntu Linux operating system to start running another program and a local file can be a medium for passing the data through programs. The estimation process consists of several steps. (1) The Estimage server application receives a request for image estimation. (2) When the image data is received, the server application writes the image data into an image file. (3) The server application then calls the Octave program and passes it the name of the image file and the name of the selected ML model. (4) Octave program reads the image file, normalizes the image data, and writes the normalized image data to a new image file. (5) Octave then invokes ilastik and passes it the file name of the normalized image and the selected model name. (6) ilastik reads the normalized image file, picks a ML model with the model name, and applies the model to the image data for estimation. (7) After the estimation is finished, ilastik outputs the estimation results to the Octave program, which include an estimate and a density map image from the normalized image. (8) Octave program writes the estimation results into two files and passes the file names to the server application. (9) Finally, the server application reads the resulting files and sends the estimation results back to the Estimage client application.

4.4 Database Model

This section presents the database model design for data storage on the Android client and the server. To allow users to manage their image records without the Internet, the Android client is designed as a main storage for image records. The server is used to cache the uploaded image records, images, and the estimations results to avoid running estimation on a previously estimated image. Therefore, both the SQLite database on the Android client and the MySQL database on the server share the same database model design.

Figure 4.10 shows the detailed design of the data model. In the Image Records table, four columns are specified: the image record ID, the image record data, the

resized image data, and the density map image data. The image record ID is a unique text string and serves as a primary key for the database to locate a specific image record. The image record data stores detailed information about an image. The resized image data saves a resized version of the original image for input to the ilastik model and the density map image data stores the density map image created by ilastik. In order to reduce the complexity of the Image Record table, an image record data structure, called Image Record Class, is created in the Estimage client application and can be transformed into a text string using a lightweight data-interchange format called JSON. The resized images and the density map images are compressed without loss and formatted into text strings with base 64 for storage.

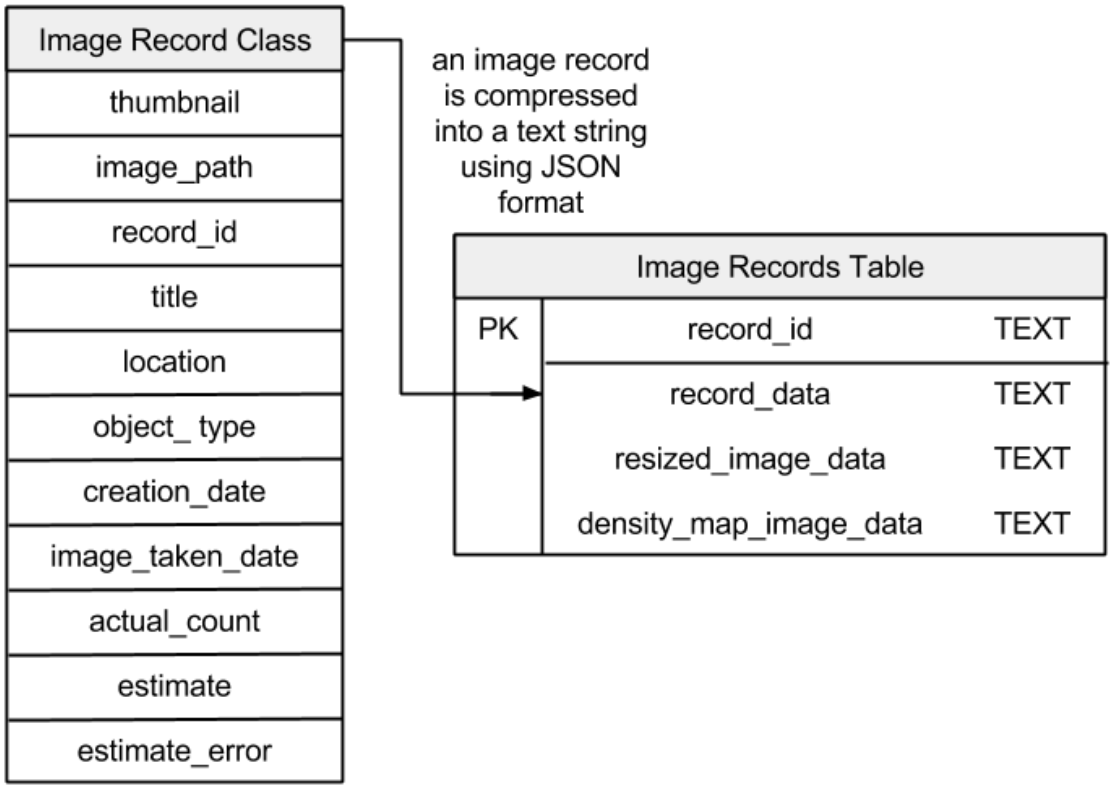


Figure 4.10: The data model design for the SQLite and MySQL databases on the Android client and server, respectively

Chapter 5

Implementation

The next stage in the system development process is the implementation stage. Rather than providing detailed execution codes, the implementation stage captures the challenging problems and the software solutions of the Estimage system.

In Section 5.1, encountered problems are presented with solutions explaining how they were solved. Section 5.2 introduces the open-source techniques that were adopted to build a robust and user-friendly Estimage client application. Finally, in Section 5.3 the user interfaces are presented to describe the Estimage client application's look and feel.

5.1 Problems and Solutions

Although the Estimage system was carefully designed, a number of practical problems arose in the implementation of the Estimage client application. While most of the problems were trivial and easy to solve, some of them were challenging and worthy of recording in this thesis.

5.1.1 Storing Image Records

Because of the internal operation mechanism of the Android system, image records that are created at run-time will be lost under two conditions: (1) when users rotate

the screen of the Estimage client application, and (2) when the client application is exited. Rotation of screen causes the Android system to destroy and recreate the current activity. The following explains how we maintain the list of image records in both situations.

Maintain records when application is exited

To maintain the list of image records when the client application is exited, we can save the image records in the external storage on the Android device. There are two options for data storage: a local data file or a local database. In our client application, a local SQLite database is used.

In the case of using a data file, any change in an image record results in the rewriting of the whole list of image records. Compared to the local data file, the SQLite database has several advantages. (1) When insertion, modification, or deletion of an image record happens, only one operation is needed to update the image record in the SQLite database and users do not feel any delay in the changes on screen. (2) The SQLite database can locate any image record with an image record ID easily. (3) Image records can be retrieved in any order.

However, this database solution has a disadvantage. The client application freezes when the SQLite database is retrieving the list of image records. To solve this problem, an asyncTask is used to asynchronously read each image record and display it at the same time when users are operating the client application.

Maintain records when the screen is rotated

The database solution is not suitable for maintaining the list of image records during screen rotation of the client application, becasue the whole list of image records must be retrieved from the SQLite database the next time the screen orientation is altered. Instead, a static reference to the image record list is used to keep the image records in memory through the whole life of the client application, and users have no need to wait for the process of reading image records again when they rotate the screen.

Conclusion

The SQLite database is used to maintain the list of image records in the external storage on the Android device when the client application is inactive. A static reference to the image record list is maintained in the memory when the client application is active.

5.1.2 Reducing Active Memory Usage of Image Records

Because media files like images usually occupy a large amount of space on hand-held devices, one of the problems in the Estimage client application is the heavy memory burden that is caused by images bound to each image record.

Solution

The active memory usage of the client application can be reduced in several ways. First, only the file path of an image instead of the image itself is saved in each image record. Second, a thumbnail of an image is created and saved in each image record. When displaying the image records in a list, the thumbnails are loaded into the active memory instead of the original images. Last but not least, the density map images received from the server after estimation are saved in the Android SQLite database directly and are not loaded into the active memory until they are displayed on screen.

Conclusion

The methods mentioned above use the external storage space to relieve the memory burden.

5.1.3 Tracking Estimation Progress of Image Records

In the Estimage client application, users must be able to start multiple requests for estimation of image records. Since the image records in the list can be reordered, it is necessary to guarantee that the estimation requests complete successfully and the estimation results are saved to the correct image record.

Solution

To allow users to continuously interact with the client application, all of the following methods use the asyncTask function to asynchronously maintain communication with the server application. Only one asyncTask is maintained and one extra thread is created for all of the estimation tasks to avoid the active memory burden on the Android device.

Queue method. Keep a queue data structure of the image records to be uploaded for estimation. The first image record in the queue is uploaded by the asyncTask for estimation. When an estimation task is finished, the estimated image record is dequeued and the next image record is ready.

This method has the advantage that because all the image records to be uploaded are lined up in a queue, it takes one operation to access the queue's first image record, which is the next image record for estimation. Unfortunately, this method requires searching through the queue in order to cancel the estimation task of an image record in the worst case. This method is time inefficient because users can cancel a queued estimation request at any time. Additionally, maintaining a queue of image records occupies extra active memory space when the client application is running. The more image records to be uploaded we have, the more duplicated image records we need to store in the queue.

Status tag method. Mark each image record with a status, such as "neutral", "pending", "uploading" and "estimated". Initially all the image records are in the neutral state. When a user instructs the client application to upload an image record, the image record is marked as pending. If the asyncTask is not running, the client application starts the asyncTask to upload this image record for estimation and the image record is marked as uploading. Once the estimation task finishes, the image record is marked as estimated. Then the client application searches the list of image records and starts the asyncTask to upload the next pending image record.

This approach requires only a little memory space (a four-byte integer tag for each image record) compared to the previous method, which requires a queue data structure for pending image records. Further, a pending image record can be cancelled instantly by setting its status to the neutral state. But in the worst case, it takes a

processing time of $O(n)$ complexity to get the next image record to be uploaded by searching the whole list from the beginning.

Conclusion

Considering the concern over the active program memory space, the status tag method is chosen to be implemented in the Estimage client application.

5.1.4 Ensuring Only Valid Image Formats are Submitted

Every colour pixel is represented by three values called the red, green, and blue channels. Some images have a fourth value called the alpha channel, which controls the transparency of an image. Because the ML predictive models from ilastik are trained using a set of three-channel images, they can only make predictions for images with three channels. Therefore, an image with the fourth alpha channel must be prevented from being sent to the server.

Solution

When a user creates a new image record, the Estimage client application first checks if the chosen image has the alpha channel. The client application only creates an image record with an image with the three basic channels and will inform the user to select another image if the alpha channel exists.

5.2 Open-Source Techniques

In the Android community, a huge number of open-source libraries have been developed and provide well-designed interface layouts for Android applications. Using these open-source libraries, developers can focus on the logic implementation in the background without spending too much time on building interfaces from scratch. This section introduces the open-source libraries that were adopted in the implementation of the Estimage client application. Figure 5.1 shows an example of how the open-source techniques were applied.

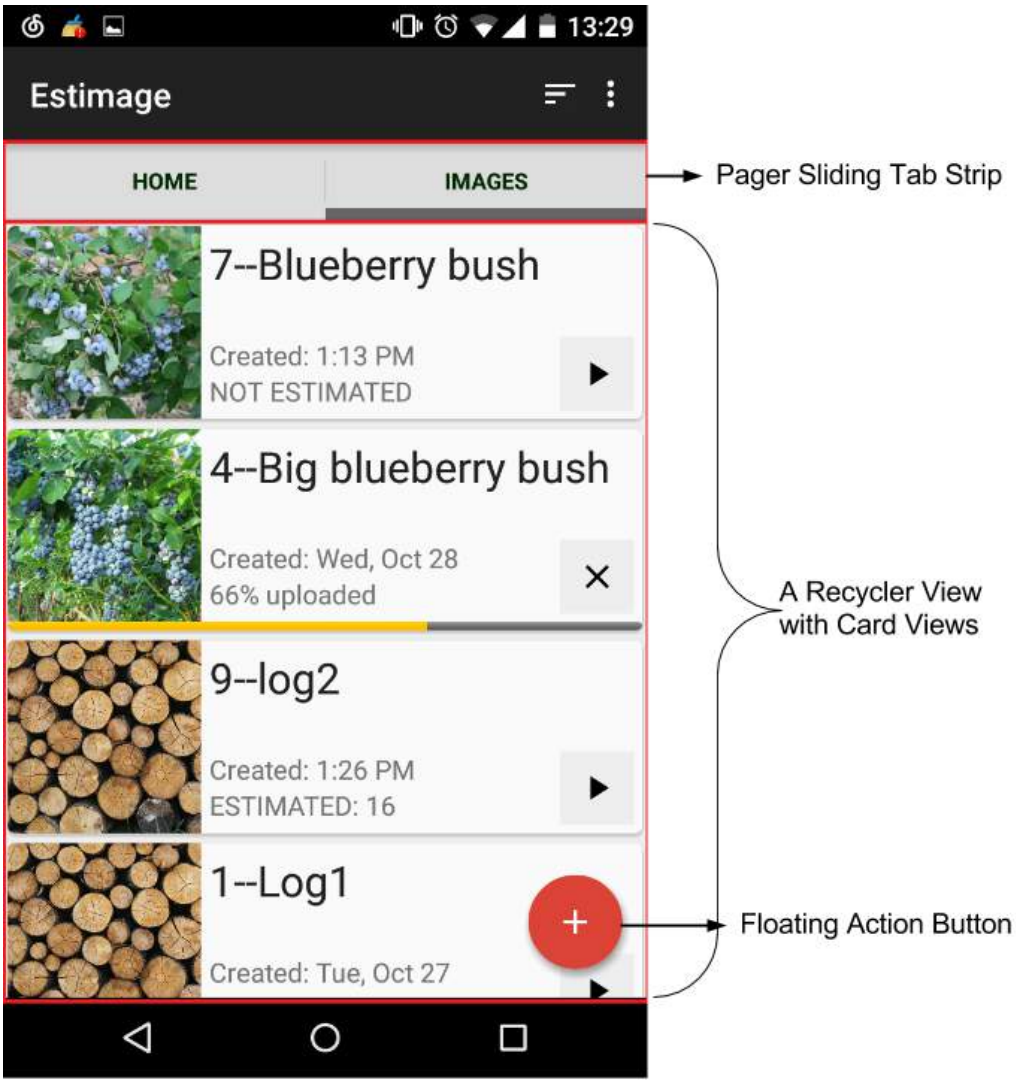


Figure 5.1: Example of open-source techniques applied in the client application

5.2.1 Native Camera and Gallery

For Android application developers, a native library called Intent is available to enable their applications to take photos or select photos by invoking an existing Android camera application or gallery application [Goo15b]. The Estimage client application uses this technique to allow users to capture or choose photos using their familiar Android camera and gallery application.

5.2.2 Recycler View with Card View

Developed by Google, "recycler view" is a layout that flexibly presents a set of data in a list or a grid [Goo15f]. It is called "recycler" because it recycles views that were previously displayed and reuses them later. This recycling feature improves performance since the views for each item in a list are not reconstructed when users scroll the list up and down. In the Estimage client application, a recycler view is used to display a set of image records. Another layout called "card view" is used to provide an attractive card-like appearance for each image record. The round corners and the shadow effect can be further customized.

5.2.3 Pager Sliding Tab Strip

The "pager sliding tab strip" is an open-source library that provides customized paging indicators for different pages on the same interface [Stu13]. Compared to using native-supported tabs, pager sliding tab strip is much more flexible since its position and appearance is easy to customize. Furthermore, the animation of swiping helps to improve user experience and the appearance of the indicators follows the material design style. In the Estimage client application, users can switch between the home interface and the image repository interface with the pager sliding tab strip.

5.2.4 Floating Action Button

The "floating action button" is another open-source library that provides a button layout following the material design style [Tar15]. The floating action button is displayed as floating on the top of the screen and can remain still regardless of the varying background. Since the floating action button is designed to only appear when needed, it is easy to add extra functions to the Estimage client application without affecting the layout design of an interface. With the floating action button, users can conveniently estimate multiple image records with one click in the client application.

5.2.5 Touch Image View

The "touch image view" is an open-source library that was developed on top of the native image view interface layout [Ort14]. The touch image view allows users to easily zoom in or out on an image with two-finger touching or double-clicks. In the Estimage client application, users can zoom in on the original image as well as the resulting density map image with the touch image view.

5.3 Interface Implementation

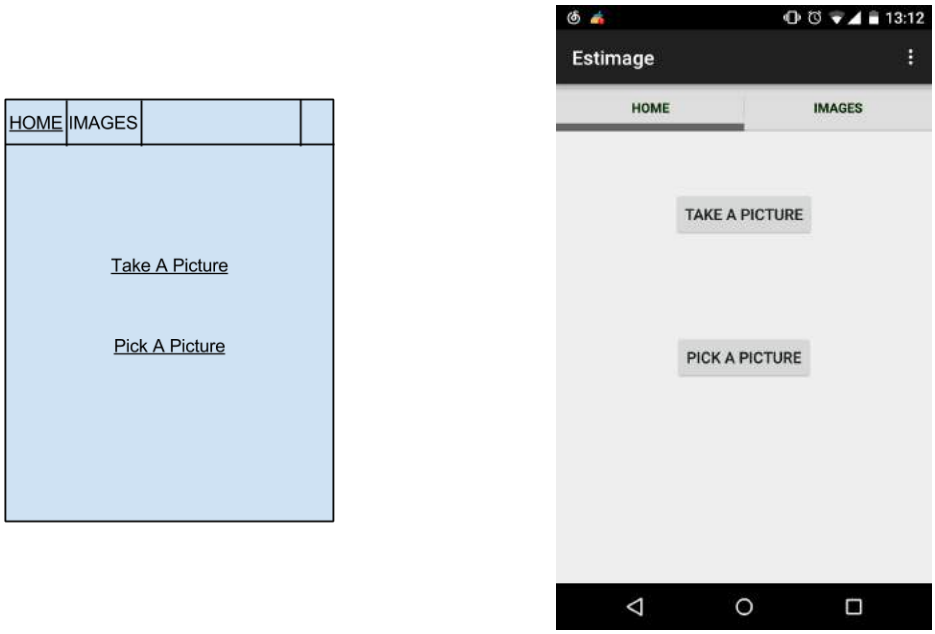
This section compares the implementation with the design of each user interface mentioned in Section 4.2.2. Specific improvements to these user interfaces are discussed.

5.3.1 Home Interface

The home interface provides users with shortcuts to create a new image record. There are limited differences between the design and the implementation of the home interface shown in Figure 5.2.

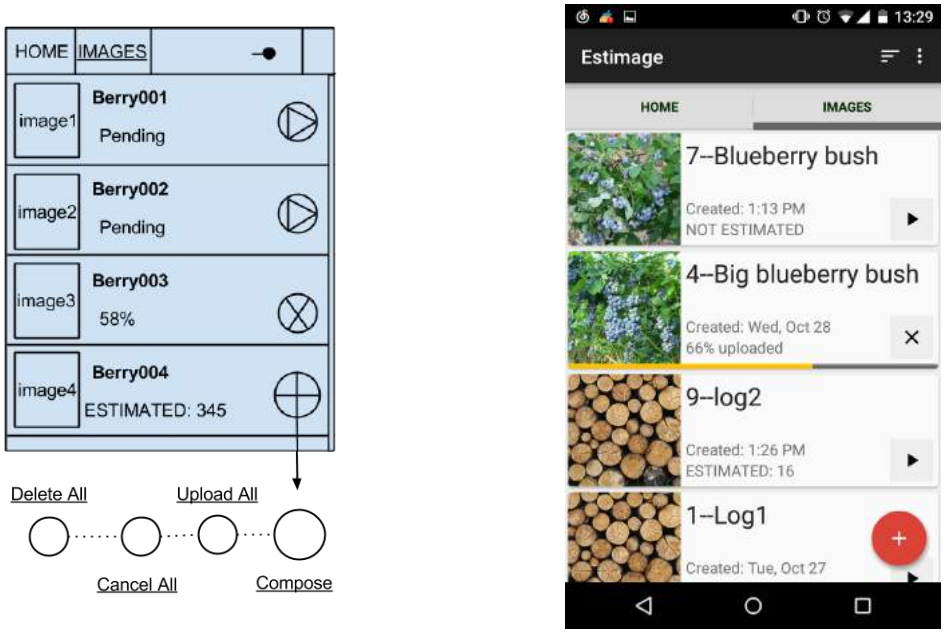
5.3.2 Image Repository Interface

The image repository interface is a place for displaying image records. In Figure 5.3, we can see several changes in the design of the image repository interface as compared to its implementation. First, the thumbnail of each image is presented in a bigger rectangular shape for better viewing. Second, the creation date of each image record is displayed. Third, a progress bar at the bottom shows the uploading progress of an image record to the user. The last change is the button with a sorting icon located at the top-right corner. Compared to using a switch, a sorting button provides users with more options to sort the list of image records.



(a) Design (b) Implementation

Figure 5.2: Design and Implementation of the Home Interface

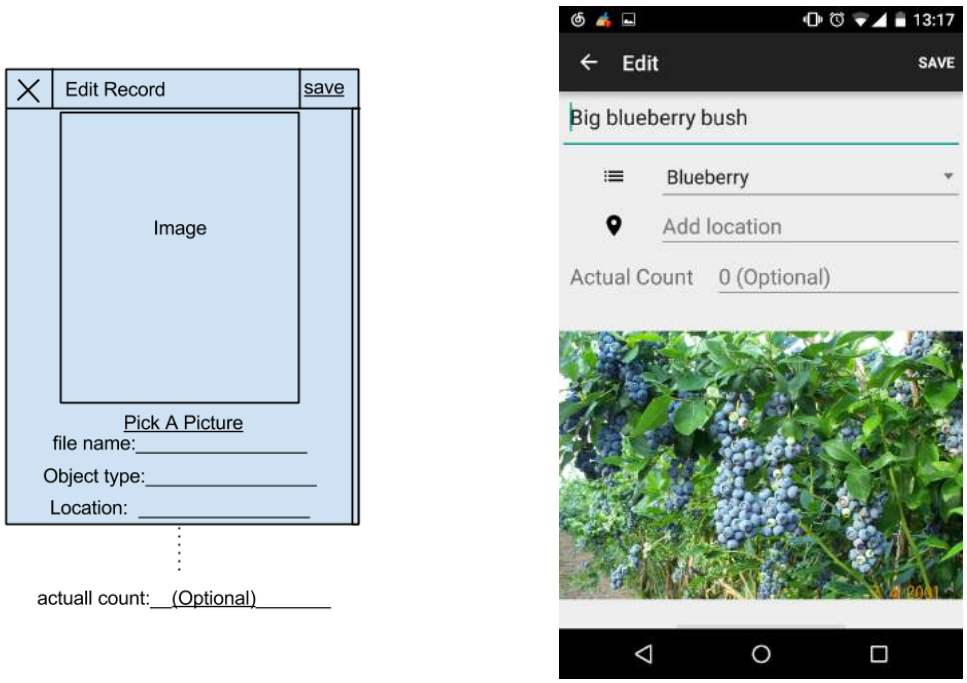


(a) Design (b) Implementation

Figure 5.3: Design and Implementation of the Image Repository Interface

5.3.3 Edit Record Interface

In Figure 5.4, there is one obvious difference between the design and the implementation of the edit record interface. The selected image is moved from the top to the bottom of the interface. In this way, the edit record interface makes users pay more attention to the textual information than the image. Other changes are that labels are replaced with icons and hints are provided in the text fields. For example, the location label is replaced with a location icon and the hint of "Add location" appears when the location text field is not filled.



(a) Design

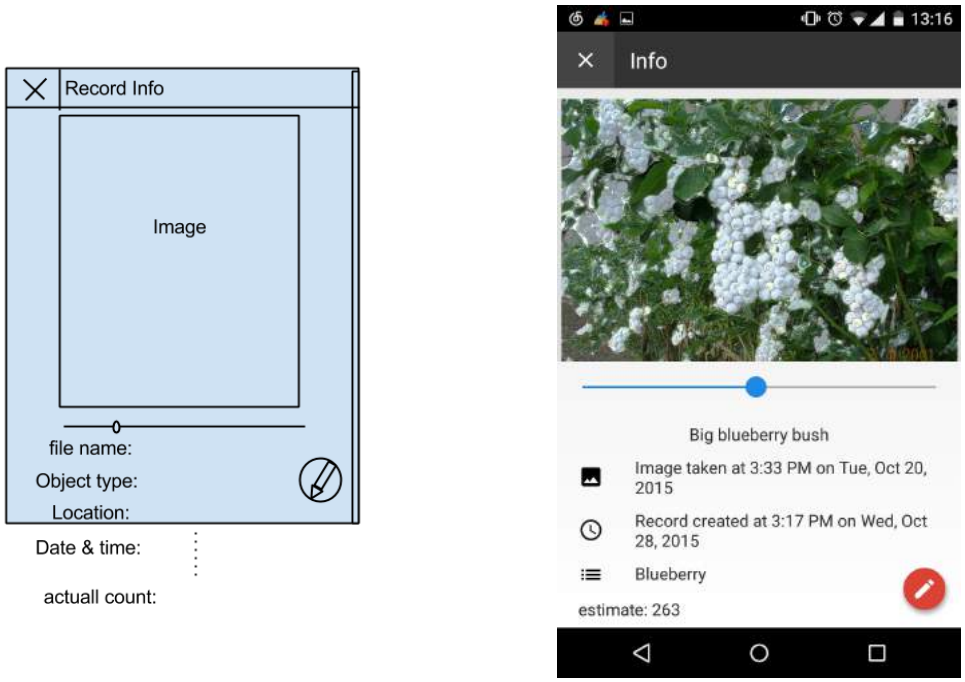
(b) Implementation

Figure 5.4: Design and Implementation of the Edit Record Interface

5.3.4 Image Record Information Interface

Figure 5.5 depicts the design and the implementation of the record information interface. Prior to estimation, only the selected image shows up. If the image has been estimated, a density map image appears and is laid over the selected image. A slider is provided below adjusting the density map image opacity. In this way, users can

not only view the information of an image record, but also get an idea of how the estimate is calculated by the ML predictive model by viewing the two overlapped images. Additionally, the labels in the design are changed to meaningful icons.

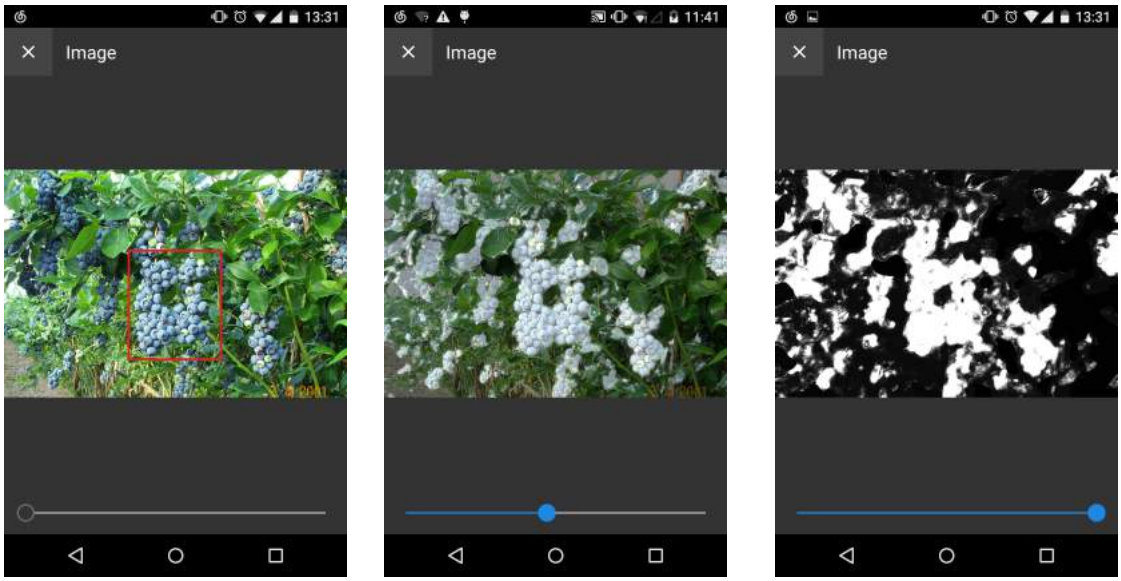


(a) Design (b) Implementation

Figure 5.5: Design and Implementation of the Record Information Interface

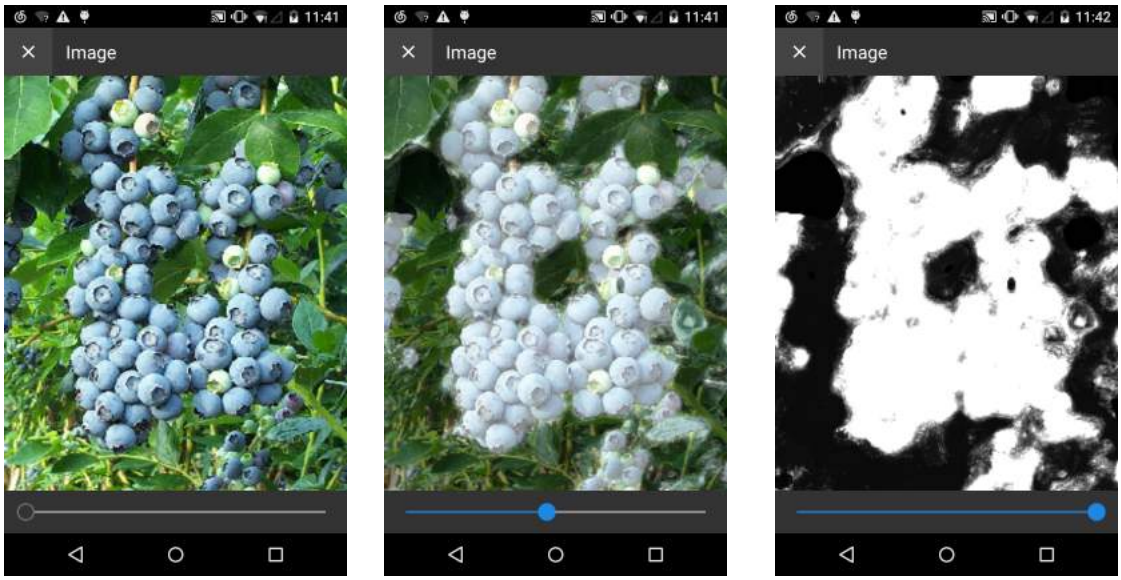
5.3.5 Image Display Interface

There is an additional interface that is not part of the original design. The interface is created to provide users with a way to zoom in on the original and density map images. In this interface, the touch image view technique mentioned in Section 5.2 is used (see Figure 5.6). The resulting zoomed images are shown in Figure 5.7. It is easy to control the opacity of the density map image by adjusting the slider below the images.



(a) Original Image (b) Mixed Image (c) Density Map Image

Figure 5.6: Implementation of the Image Display Interface



(a) Original Image (b) Mixed Image (c) Density Map Image

Figure 5.7: Image Display Interface with Zoom-In

In the next chapter, we give the results of testing this design and implementation.

Chapter 6

Testing and Evaluation

This chapter focuses on the validation and verification of the Estimage client application and its server-side components. Although secondary to the mobile client application and cloud server development, the machine learning model performance of the system is evaluated in terms of the accuracy of the estimation functionality.

Section 6.1 reviews the testing of the functional requirements. Non-functional requirements were tested and the performance of the system is evaluated in Section 6.2. Finally, Section 6.3 presents experiments on evaluating the estimation performance of the system.

6.1 Functional Testing

Since the Estimage system relies on the camera hardware of a hand-held device, all of its functions were tested on a physical Nexus 4 Android smartphone with Android versions 4.0, 4.4, 5.0, and 5.1. Each function was tested manually to ensure it worked as per the requirements.

The tested functional requirements are presented along with a status indicating whether they are satisfied or not. Based on the use cases mentioned in Section 3.3, functional requirements are classified into two categories: photo selection, and image record management.

Photo Selection

1. [Desirable] [**Satisfied**] Users can take a photo (or image).

   • The Estimage client application successfully opened the native camera application to allow the user to take a photo.

2. [Essential] [**Satisfied**] Users can select a photo (or image) from their Android albums.

   • The Estimage client application successfully opened the native gallery application allowing the user to select an image. A selected image with an alpha channel caused the client application to display a warning message, which tells the user to choose an image without the alpha channel.

Image Record Management

1. [Essential] [**Satisfied**] Users can attach information to a selected image to create an image record.

   • An image record was successfully created and displayed in the list after the title, the location, the object type and the actual count were input.

2. [Essential] [**Satisfied**] Users can view their image records in a list.

   • After multiple image records were created, the client application successfully displayed the image records in a list, which could be scrolled up and down.

3. [Essential] [**Satisfied**] Users can reorder the image records in the list based on information such as the creation date and the magnitude of the estimate.

   • The list of image records was successfully reordered by the creation date and the magnitude of the estimate. The image records could also be grouped into the group of estimated image records and the group of not-estimated image records.

4. [Essential] [**Satisfied**] Users can view any of their image records in detail.

- Each image record in the list was tested to be clickable and the detailed information of a selected image record was displayed without missing any information.

5. [Essential] [**Satisfied**] Users can zoom in or out on the image of an image record.

   - In the image display interface, it was tested that an image could be zoomed in or out with two-finger touching or double-clicks.

6. [Essential] [**Satisfied**] Users can modify an image record.

   - After changing the information of an image record in the edit record interface, the changes were successfully displayed. After restarting the client application, the changed image record was displayed with the updated information.

7. [Essential] [**Satisfied**] Users can delete an image record.

   - A long-click on an image record successfully removed it from the list. After restarting the client application, the deleted image record was not shown in the list.

8. [Essential] [**Satisfied**] Users can send their image records and images for estimation — less than 25% error rate is required for this initial prototype.

   - Image records with images of coins, logs, or blueberries were tested to be successfully estimated. The estimation results are presented in Section 6.3.

9. [Essential] [**Satisfied**] Users can compare an original image with its density map image after estimation.

   - In the image display interface, the resulting density map image is overlaid with the original image. The slider below them was tested to be adjustable to control the opacity of the density map image.

10. [Essential] [**Satisfied**] Users can save image records on their mobile device.

    - Image records that were previously created were successfully displayed in the image repository interface after restarting the client application.

11. [Essential] [**Satisfied**] Users can save estimation results of an image record, including an estimate and a density map image, on their mobile device.

    • The estimation results of each image record were successfully displayed in the record information interface after restarting the client application.

12. [Desirable] [**Satisfied**] The Estimage server application can save an image record in its database.

    • After an image record was estimated, the image record was found in the server database.

13. [Desirable] [**Satisfied**] The Estimage server application can save estimation results of an image record including an estimate and a density map image in its database.

    • An estimation on an estimated image record only took around three seconds to complete. This was because the image record's previous estimation results were returned instantly.

6.2 Non-functional Testing

This section presents testing results of the non-functional requirements. Non-functional requirements were tested on a Nexus 4 Android smartphone and an Ubuntu server. The Nexus 4 Android smartphone has a quad-core 1.5 GHz Krait CPU, a Adreno 320 GPU, 2 GB RAM, and 16 GB external storage. The system server has a 2.40 GHz Intel Xeon Processor E5-2620 v3 and 8 GB RAM, running the Ubuntu Linux operating system.

The non-functional requirements are classified into three categories for testing: system platform, application user interface, and system performance. The test results were recorded in Appendix A Table A.1, A.2, and A.3. Several sample screenshots of logs are presented in Figure A.1, A.2, and A.3.

System Platform

1. [Essential] [**Satisfied**] The Estimage client application should be able to run on mobile devices with Android 4.0 Ice Cream Sandwich and above.

   • The Estimage client application was installed and tested on the Android 4.0, 4.4, 5.0, and 5.1 operating systems and all functions mentioned in Section 6.1 worked well.

2. [Essential] [**Satisfied**] The Estimage client application should have access to the internal and external storage on the Android device if any.

   • In Section 6.1, image records could be saved in the SQLite database in the Android internal and external storage, and thus the client application must have access to the internal and external storage on the Android device.

Application User Interface

1. [Essential] [**Satisfied**] For better user experience, the Estimage client application should adopt the material design style, which is a popular interface design style recommended by Google.

   • In the Estimage client application, all the adopted open-source techniques mentioned in Section 5.2 follow the material design style.

2. [Essential] [**Satisfied**] When displaying image records in a list, the thumbnails of images should be larger than the standard thumbnail size of 100×100.

   • In the home repository interface, each thumbnail was displayed in a size of 230×230.

3. [Desirable] [**Satisfied**] The minimum font size of the Estimage client application should be 14 scale-independent pixels.

   • Most texts were displayed in 14 scale-independent pixels while the titles were 16 scale-independent pixels.

4. [Desirable] [**Satisfied**] The navigation of the Estimage client application should be intuitive.

   • Each user interface has a header indicating which interface the user is in and has a back button for the user to go back to the previous interface. These two features make the client application intuitive.

System Performance — Time and Space

1. [Essential] [**Satisfied**] Users should be always able to continuously interact with the Estimage client application without being blocked.

   • During the testing process of the functional requirements in Section 6.1, the main thread of the client application did not become busy during I/O and user interactions were not blocked.

2. [Essential] [**Satisfied**] Little delay should be noticed when switching user interfaces in the Estimage client application.

   • The maximum switching time between each pair of associated interfaces was tested to be less than 0.2 seconds, which is hardly noticed [see Appendix A Table A.1].

3. [Desirable] [**Satisfied**] Each button click should respond in less than 0.2 seconds.

   • The maximum response time of each button click was tested to be less than 0.01 seconds [see Appendix A Table A.1].

4. [Essential] [**Satisfied**] The Estimage client application should spend less than ten seconds to display the whole list of image records (up to 50) on the screen.

   • 50 records were loaded in and displayed on screen in less than eight seconds [see Appendix A Table A.1].

5. [Desirable] [**Satisfied**] Each image record should occupy no more than two megabytes in the SQLite database on the Android device.

• The space used by each of 10 image records was tested and, on average, each image record occupied around one megabyte of external storage on the Android device [see Appendix A Table A.2].

6. [Essential] [**Satisfied**] The Estimage client application should take no more than three seconds to upload an image record to the server with a network speed of one megabyte per second.

   • The maxium time it took to upload the image record and its image to the Estimage server application was less than two seconds [see Appendix A Table A.3].

7. [Desirable] [**Not Satisfied**] The Estimage server application should take less than five seconds to complete the estimation process of an image.

   • During the testing process of the estimation performance mentioned in Section 6.3, the time cost of 10 estimation tasks were recorded in Appendix A Table A.3. On average, each estimation task took around 13.4 seconds to complete. The maximum time of the 10 tests was less than 15 seconds. The average time cost of each phase in an estimation is as follows:
   — Android App Preprocessing - 0.12 seconds - 0.9%
   — Data Communication between Android to Server - 1.32 seconds - 9.8%
   — PHP Server Application - 1.22 seconds - 9.1%
   — Octave Program - 3.84 seconds - 28.5%
   — ilastik - 6.96 seconds - 51.7%

Error Handling

1. [Desirable] [**Satisfied**] When the network is unavailable or the Estimage server application is down, the Estimage client application should stop all estimation tasks and indicate the error to users.

   • To test this requirement, the network was turned off after multiple estimation tasks were started. Once the network was unavailable, all the tasks were cancelled and a connection error message was shown to indicate that the network

was down. When the Estimage server application was manually turned off, the same connection error message was shown.

6.3 Estimation Evaluation

The primary objective of this project is the development of the Android client application, not the ML models. However, it is important to provide an assessment of the full system using the predictive ML models developed on the server.

This section focuses on evaluating the image estimation functionality of the system. To test the estimation functionality, ML predictive models were created and trained for three types of objects: coins, logs, and blueberries. In this way, the estimation performance can be evaluated thoroughly by presenting results from a simple and ideal case to a complicated and realistic case that is the domain of harvest estimation.

The results from the following test cases are presented in three steps. (1) One of the test images is shown along with its density map image. (2) The actual count with an estimate of each estimation is presented. (3) The error rate of each estimation is calculated as follows:

$$error\ rate = \frac{|actual\ count - estimate|}{actual\ count} * 100\% \tag{6.1}$$

6.3.1 Coin Estimation

The first test case is coin estimation. Since coins of the same value on the same background have the same appearance, most of the variability of the objects to be estimated is eliminated, and the complexity of counting the objects in an image is reduced.

Objective

The objective of the coin estimation is to test the estimation performance of an ideal environment, where the appearances of the objects and the background colour are

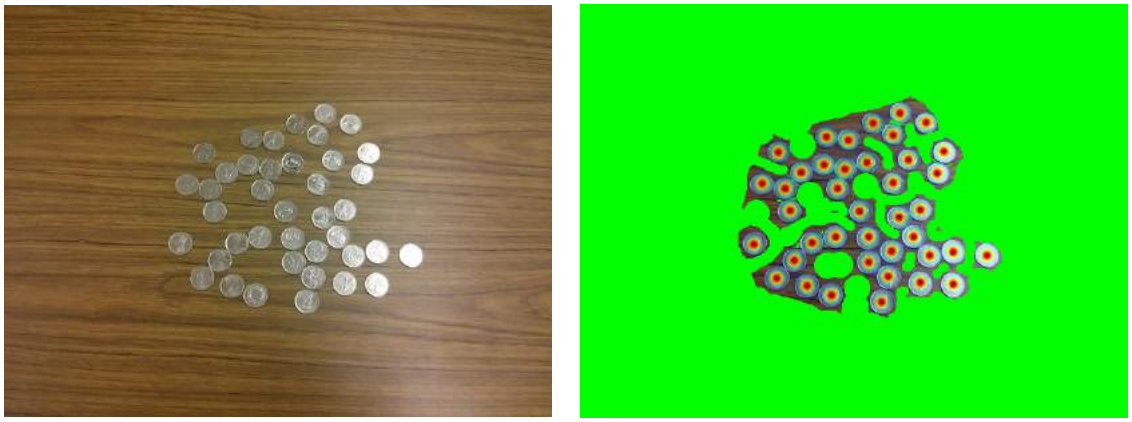
controlled.

Materials and Methods

Images of 25-cent Canadian coins on a wooden table were provided for training the ML models as well as testing the models' performance. The shiny appearance of the 25-cent coin made it easily identifiable on the brown wooden background.

One image for training is depicted in Figure 6.1. Using ilastik, all the coins were labelled as one class of object (shown in red) and examples of background pixels were marked as another class of object (shown in green). The ilastik system generated features from the marked pixels and built a random regression forest ML model using these pixels of coins and background.

To test the performance of the coin estimation, the Estimage client application was used to take 11 images of coins with the number of coins varying from 0 to 40. These images were taken at the same height above the table to maintain the size of coins in each test image. This way, we could see how the estimates varied as the number of coins increased.

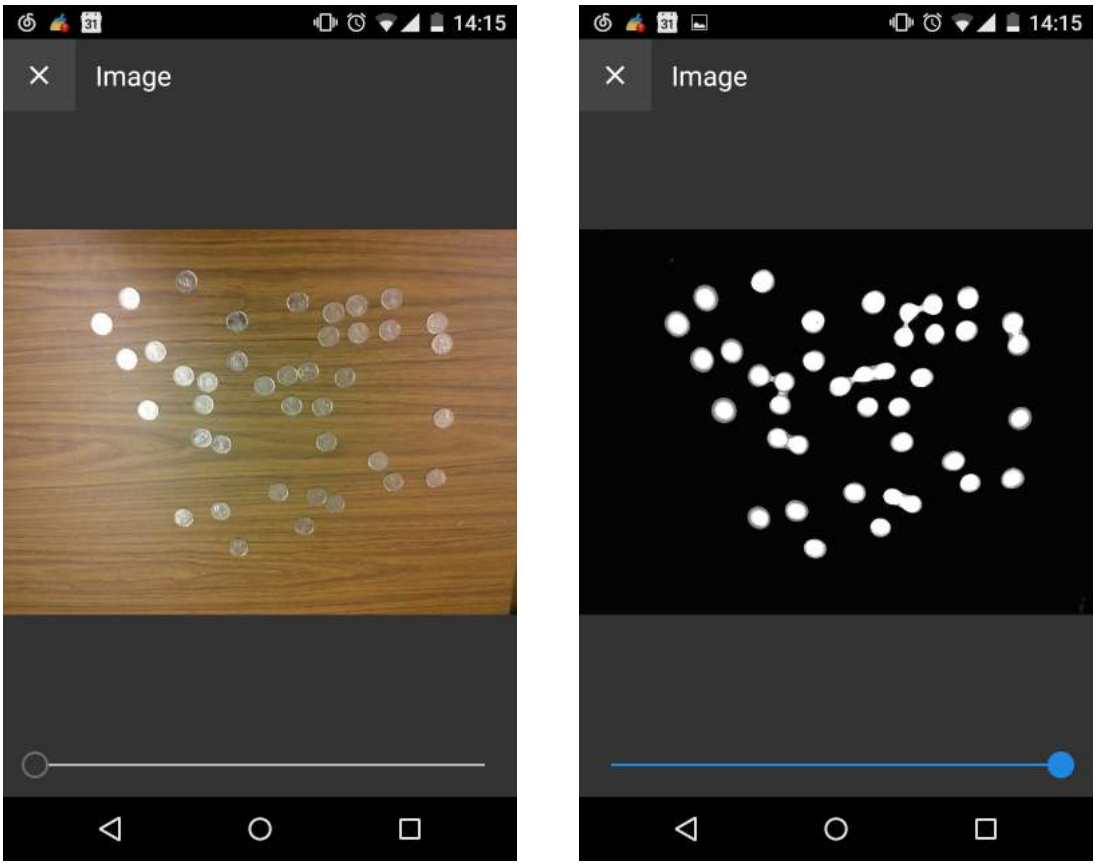


(a) Sample Image (b) Labelled Sample Image

Figure 6.1: Sample Image of Coins for Training

Results

In Figure 6.2, all the coins were properly identified by the system.



(a) Original Image (b) Density Map Image

Figure 6.2: Sample Image of Coins for Testing

Figure 6.3 depicts the estimates of 11 test images. When no coins were in the image, the estimate was six. As the actual number of coins increased from 0 to 24, the estimate converged to the actual number of coins. When the actual number of coins increased from 24 to 40, the error of the estimate gradually increased in each estimation.

Figure 6.4 depicts the error rate as a function of the actual number of coins in the image. Since the error rate of the result from the image with no coins could not be calculated, it was not presented. As the actual number of coins increased from 0 to 24, the error rate fell dramatically to 0.21%. When the number of coins increased from 24 to 40, the error rate gradually increased, such that at 40 coins the error rate was 2.38%.

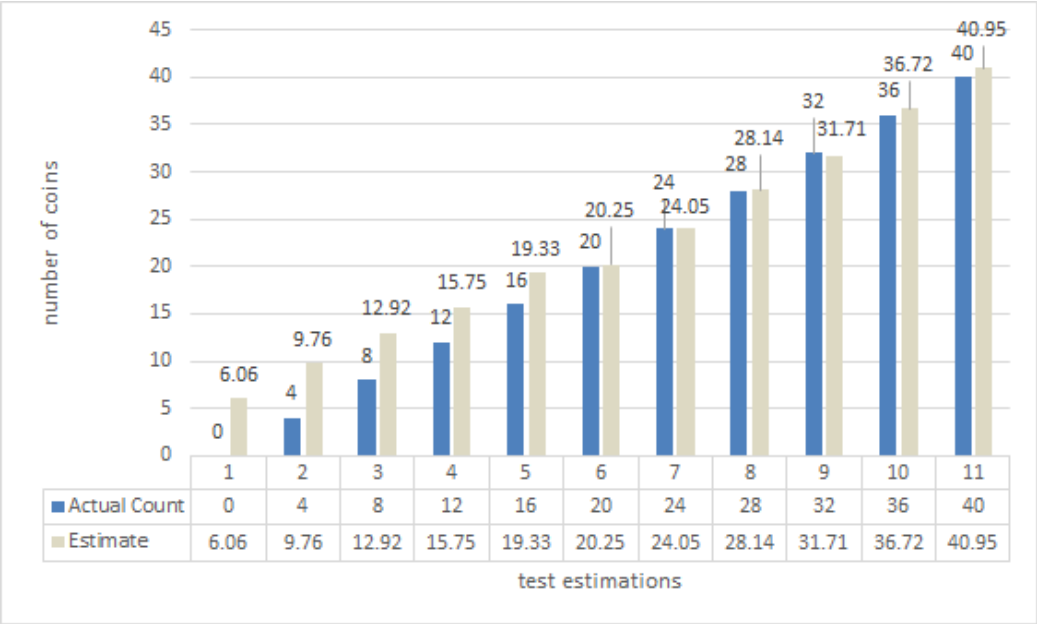


Figure 6.3: Comparison between the actual counts and the estimates of coins

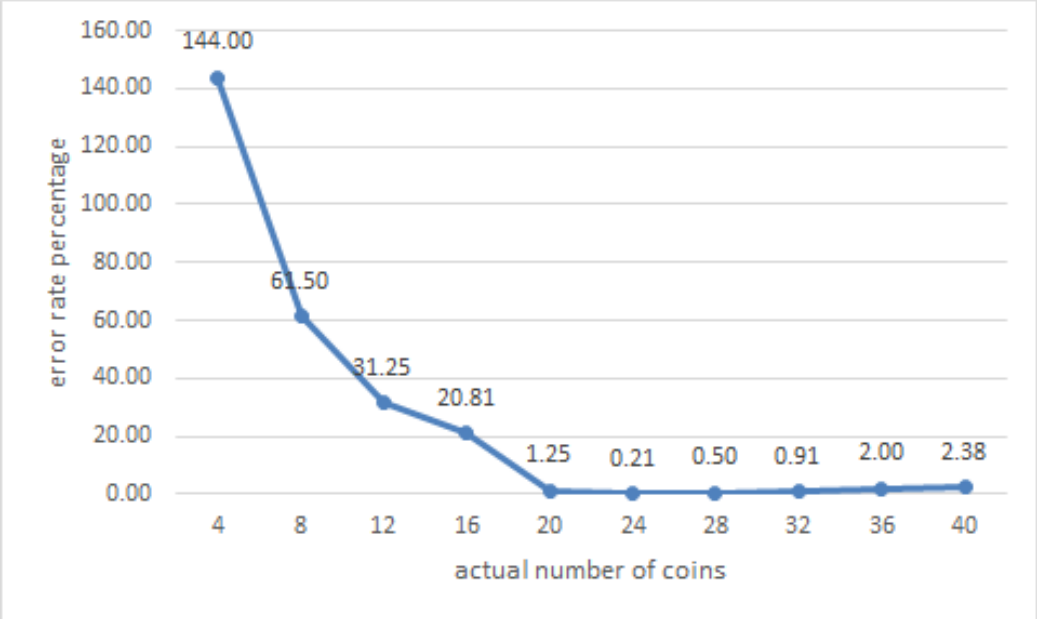


Figure 6.4: Graph of the error rate percentage as a function of the actual number of coins

Discussion

Based on the results, the estimation of coins is quite accurate when the actual number is at least 20. However, when the actual number of coins is 16 or less, the error rate increases significantly. This is because there is a minimum value for each pixel to be set when an image is being estimated by the ML predictive model. All values of each pixel in the image including the background are added up to create the estimate. This means that even when there are no objects in an image, the count will be greater than zero. This explains why the estimate would be six coins when there are no coins in the estimated image.

There are two conclusions from the coin estimation. (1) The system works pretty well on estimating the number of coins on a wooden table when the actual number is at least 20. (2) The ML estimation approach currently being used needs to be improved for small numbers of objects.

6.3.2 Log Estimation

The log estimation is the second test case. Compared to the coin estimation, the log estimation is more complex because different logs have different sizes. But the similar shape and colour of logs provide some constraints for log estimation. The log estimation is suitable to be a transitional test case between the ideal simple case, the coin estimation, and the more complex case, the blueberry estimation.

Objective

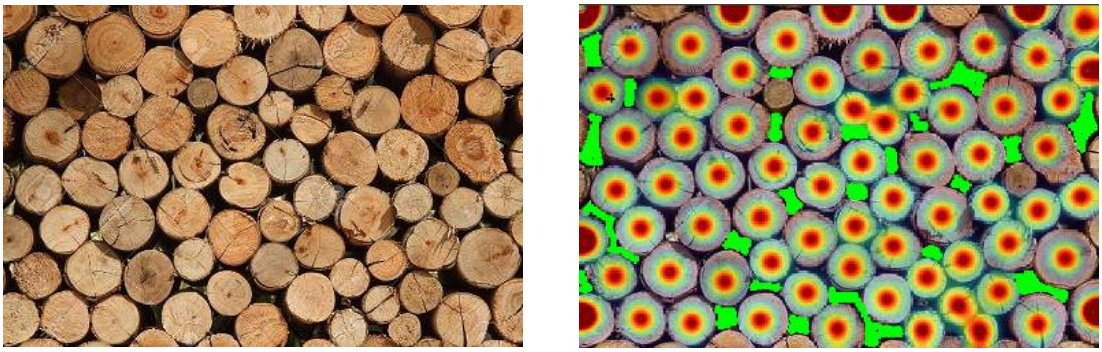
The goal of the log estimation is to evaluate the estimation performance in a less constrained environment where the size of the object varies and other factors remain stable, such as the background colour, the object colour, and the object shape.

Materials and Methods

Two images of stacked logs that are similar in size were selected from the internet for training the ML predictive models. The background of the chosen images was black since they were taken from one end of the pile of logs.

One of the images used in training is depicted in Figure 6.5. The logs in the images had a similar size, shape and colour. The logs were marked as one class and the background was marked as another class.

To test the estimation performance on logs with different sizes, seven images were provided for testing in Figure 6.6. Compared to the size of logs in the training images, the seven test images included one image with big logs, one image with small logs, and five images with logs of similar size.



(a) Sample Image (b) Labelled Sample Image

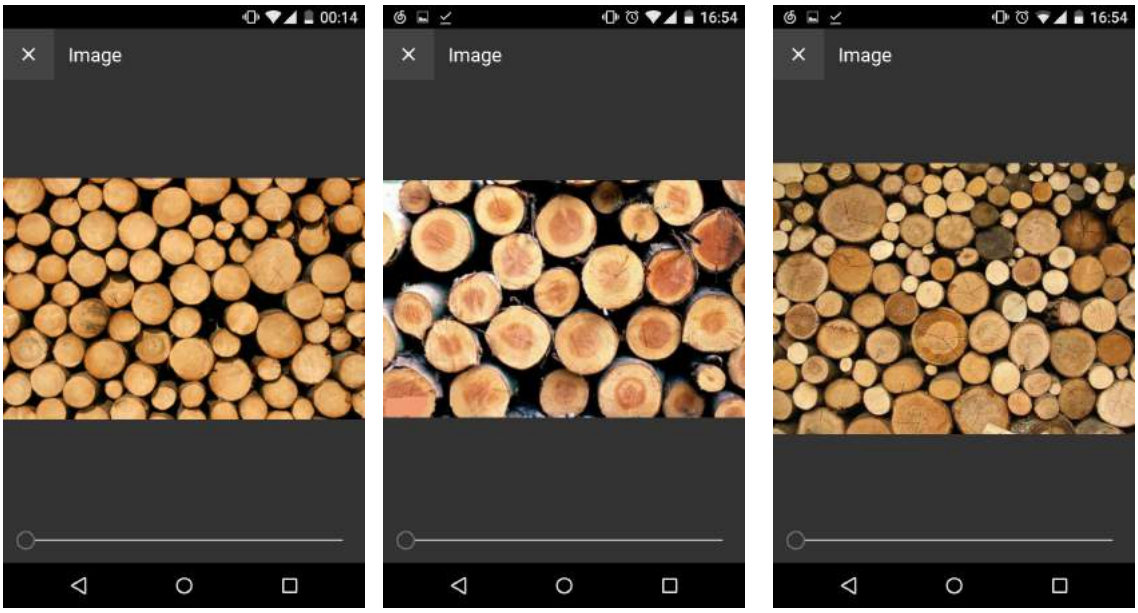
Figure 6.5: Image of Logs for Training

Results

Figure 6.6 and Figure 6.7 show three test images and their respective density map images produced by the ML model for comparison. These three test images include one with big logs, one with small logs, and one with logs of similar size.

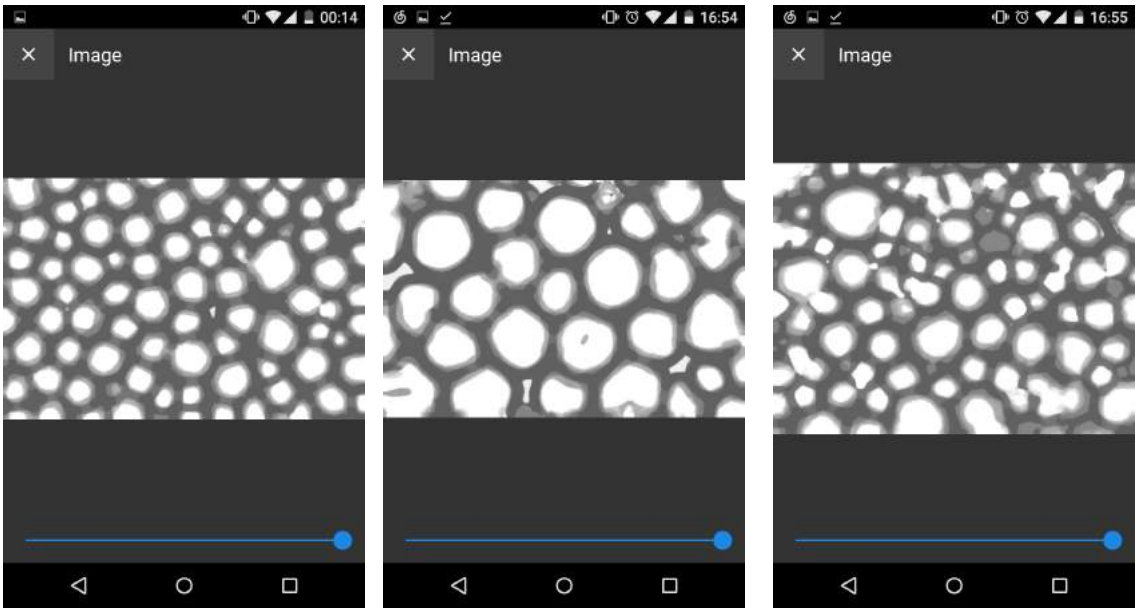
Figure 6.8 shows that the estimate of the number of logs was quite accurate in all five test images with logs of similar size to the training images. However, the estimate of the number of logs in images with big logs was around three times as much as the actual count. The estimate of the number of logs that are smaller in size was about half of its actual count.

The error rates are shown in Figure 6.9. The estimation of each image with logs of similar size had an error rate below 11%. The estimation of the image with either big logs or small logs had an error rate over 35%.



(a) Similar logs (b) Big logs (c) Small logs

Figure 6.6: Original Images of Logs for Testing



(a) Similar logs (b) Big logs (c) Small logs

Figure 6.7: Density Map Images of Logs for Testing

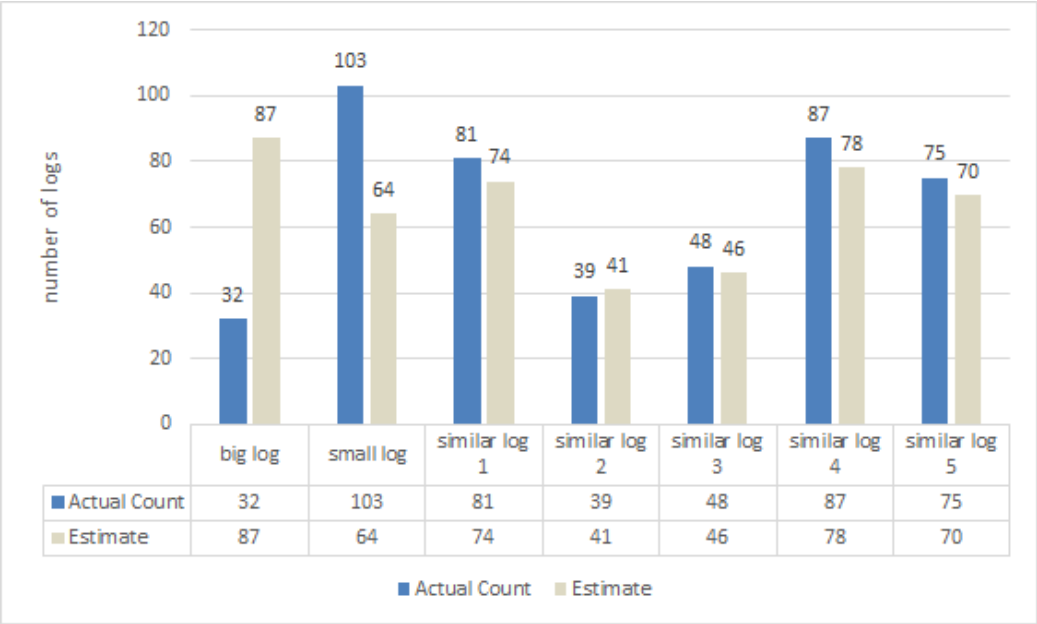


Figure 6.8: Comparison between the actual counts and the estimates of logs

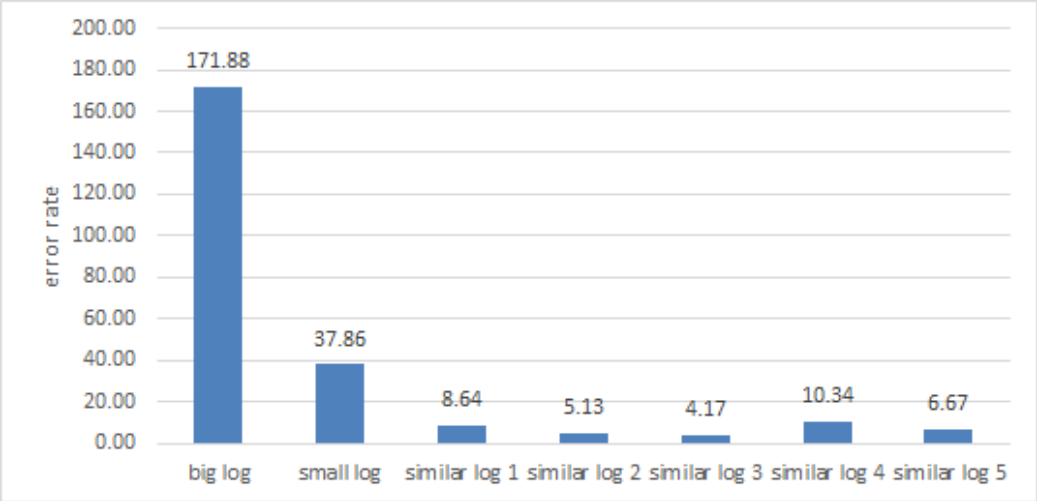


Figure 6.9: Depiction of the error rates in the log estimation

Figure 6.10 shows how error rates varied in the test images with logs of similar size. The mean of the error rates was 6.99% and the standard deviation was 2.53%.

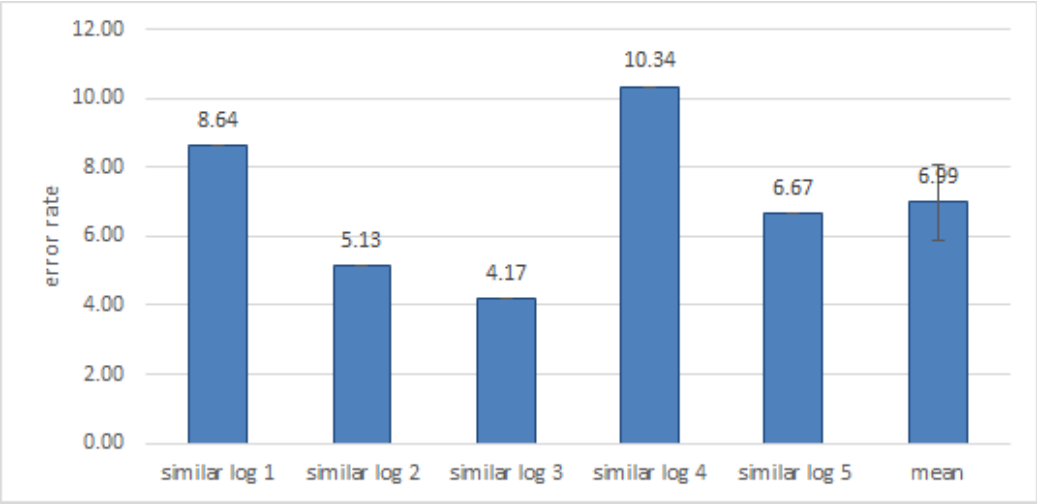


Figure 6.10: Error rates of test images with logs of similar size

Discussion

Judging from the results above, the actual number of big logs is overestimated while the actual number of small logs is underestimated. Since the ML model for log estimation predicts the number of logs in each pixel based on the size of logs in the training images, each big log has the same value as several normal logs and each normal log has the same value of several small logs. This causes an error in the estimation process of logs with different sizes. On average, the actual number of logs of similar size is estimated with much less errors. Since logs that are similar in size are not identical, some logs are slightly overestimated or underestimated, and therefore images with similar logs have error rates varying from 4.17% to 10.34%. In addition, a log with a dark colour in an image is assigned with a small value because its colour is similar to the background's colour rather than the normal logs' colour.

As a result, four conclusions are drawn. (1) The number of logs of similar size for training the ML model is estimated accurately. (2) The number of logs of very different size from the ones for training the ML model is estimated poorly. (3) A log with a different colour from the logs for training the ML model is ignored to a certain extent. (4) Improvements are needed in the ML approach being used to estimate the

count of objects with different sizes in an image.

6.3.3 Blueberry Estimation

The blueberry estimation is the most complex among the three test cases and it needs to take into account the different appearances of blueberries as well as varying surroundings. On a blueberry bush, there are mature blueberries with blue colour and immature ones with light green colour. Blueberries have surroundings such as green leaves, brown soil, and light sky. All of these factors can greatly affect the estimation result.

Objective

The goal of the blueberry estimation is to verify the performance of the estimation in a realistic environment, where a portion of the background has a colour similar to the objects to be estimated.

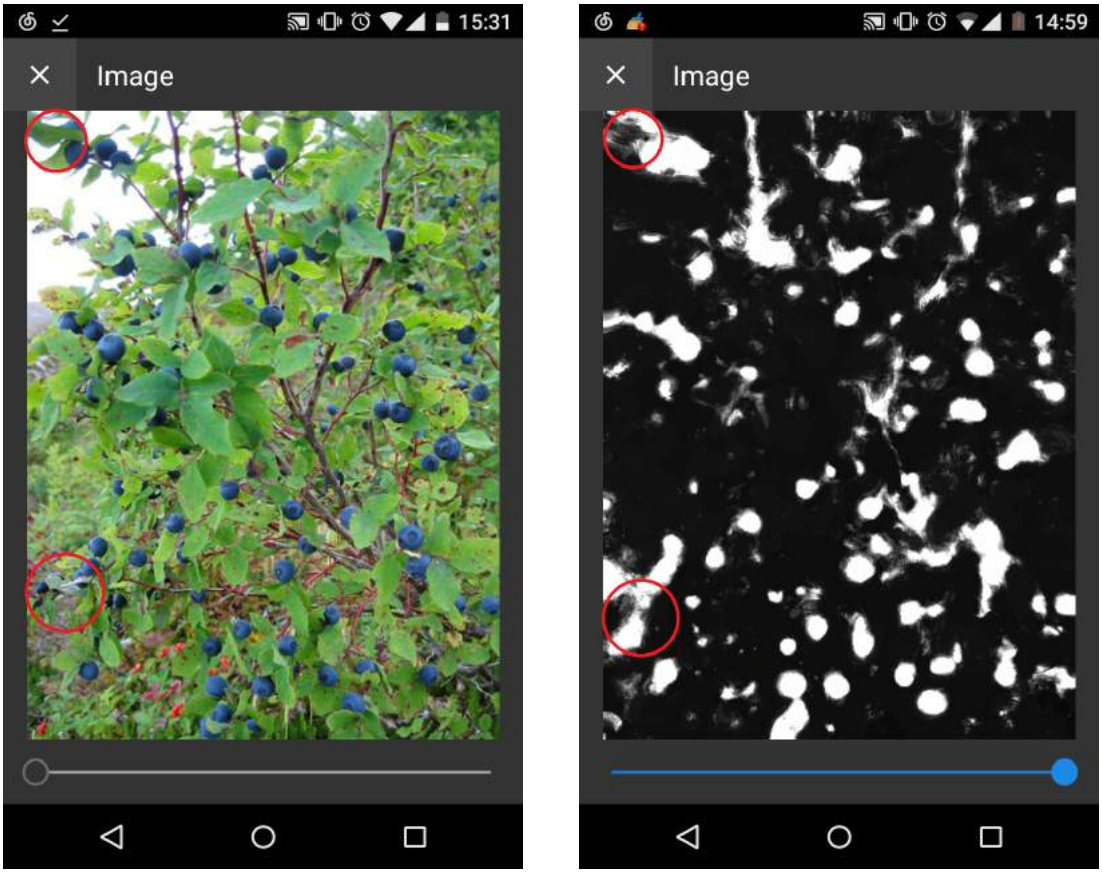
Materials and Methods

Three images of a bush with mature blueberries were selected from the internet for training the ML predictive model. In Figure 6.11 one of the training images pictured the mature blueberries with surroundings that could be easily distinguished from the blueberries. Most of the mature blueberries were labelled as one class while the small blueberries a short distance away were ignored. The immature blueberries and the background were marked as another class to indicate that they should be excluded.

In total eight images were selected from the internet for testing. The first four images had a background that was consistent with the images for training the ML model, while the other four images had a mixed background that was similar to the blueberries in colour. To reduce the influence of blueberry size and focus on the influence of varying background, these eight test images were resized so that the number of pixels each blueberry occupied is similar to the blueberry size for training the ML model. Figure 6.12 depicts one of the images with a consistent background and Figure 6.13 depicts one of the images with a mixed background.

(a) Original Image (b) Labelled Image

Figure 6.11: Sample Image of Blueberries for Training

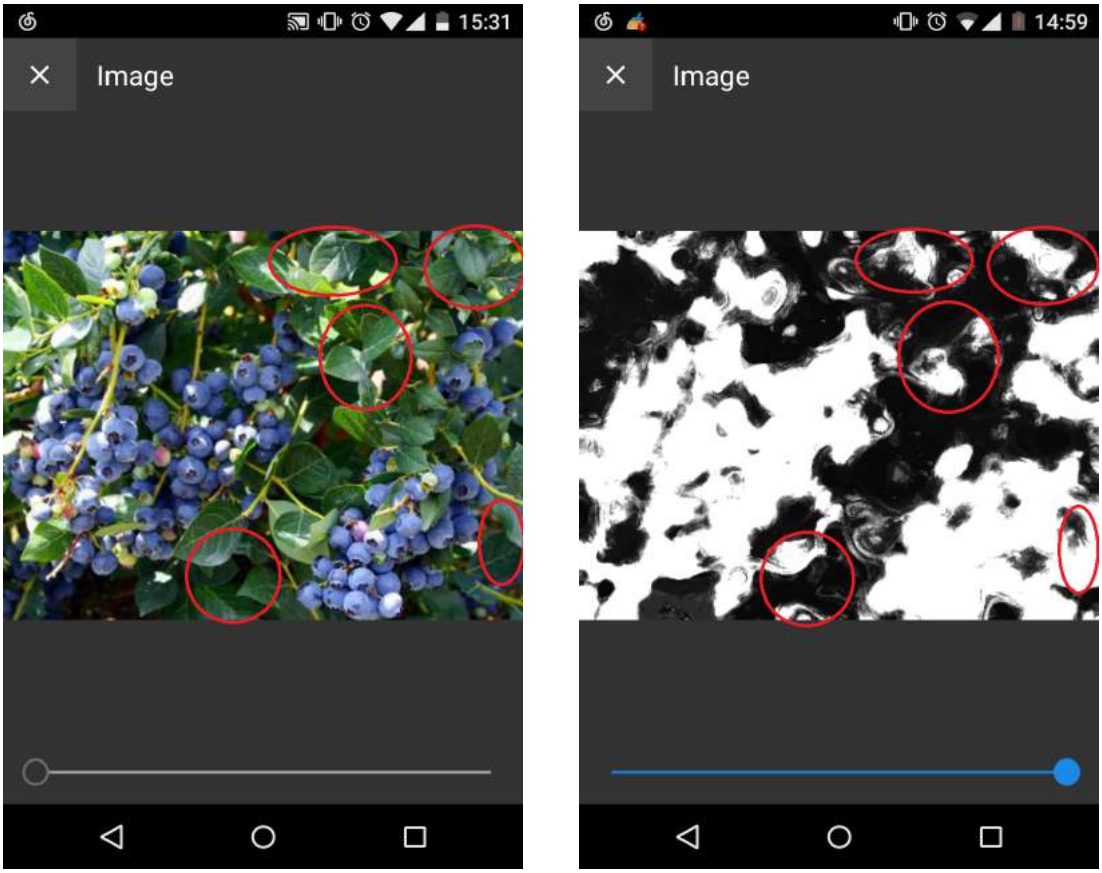


(a) Original Image (b) Density Map Image

Figure 6.12: Image of Blueberries with Consistent Background for Testing

Results

Figure 6.12 (a) shows the test result for one of the test images with a consistent background: all blueberries were recognized. While most of the background was recognized to be separate from the blueberries, a small portion of the background was mistakenly viewed as blueberries, some of which is circled in Figure 6.12 (b). Figure 6.13 shows the test results for one of the test images with a mixed background: all the blueberries were recognized. However, a large portion of the background at the top-right and bottom-right corner was misrecognized as blueberries.



(a) Original Image (b) Density Map Image

Figure 6.13: Image of Blueberries with Mixed Background for Testing

The Estimage system can be used to estimate any count or metric based on obejcts in the image. For a real application the model would likely estimate the number of

pints of berries; however, the number of berries is more practical for our current testing since it can be counted manually from the image. Figure 6.14 presents the actual counts and the estimates for the eight test images. The estimate of the number of blueberries in an image with a consistent background was close to the actual number of blueberries. However, in each image with a mixed background, the estimate of the number of blueberries was a lot more than the actual number of blueberries.

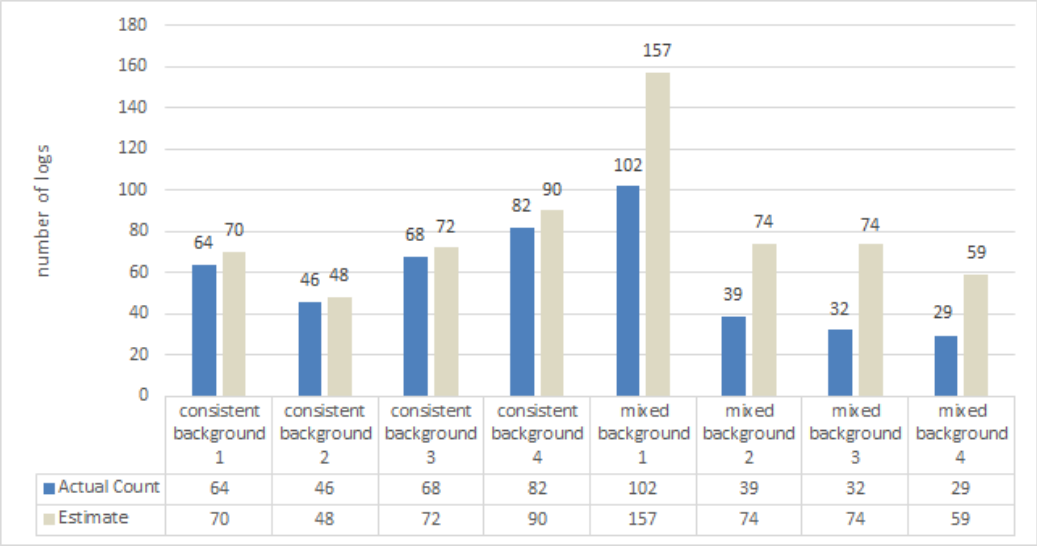


Figure 6.14: Comparison between the actual counts and the estimates of blueberries

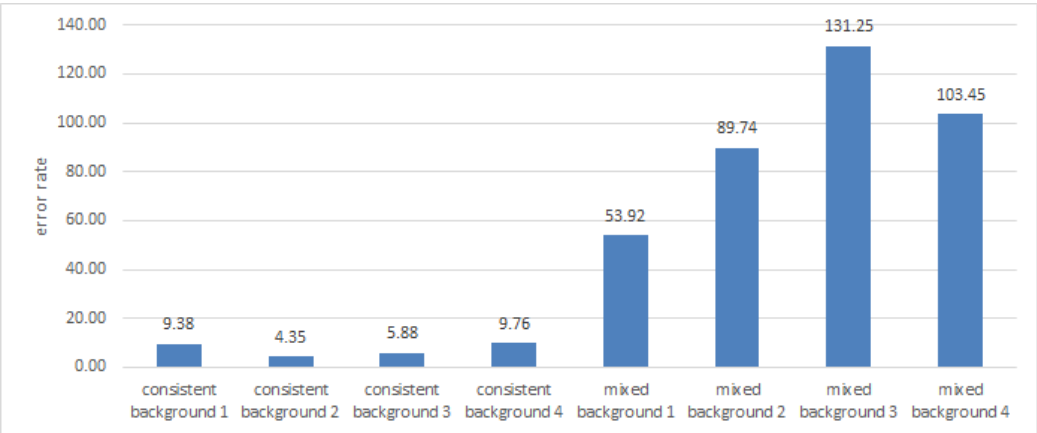


Figure 6.15: Depiction of the error rates in the blueberry estimation

Figure 6.15 depicts the error rates of the results from the estimations. The estimation of each images with a consistent background had an error rate below 10%, while the estimation of each image with a mixed background had an error rate above 50%.

Figure 6.16 presents the error rates of test images with a consistent background. The mean of the error rates is 7.34% and the standard deviation is 2.65%. Figure 6.17 presents the error rates of test images with a mixed background. The mean of the error rates is 94.59% and the standard deviation is 32.14%.

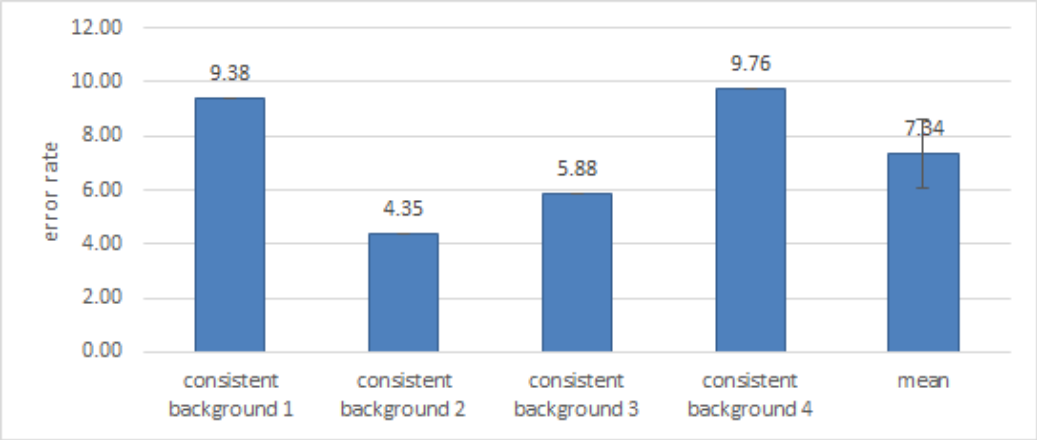


Figure 6.16: Error rates of test images with a consistent background

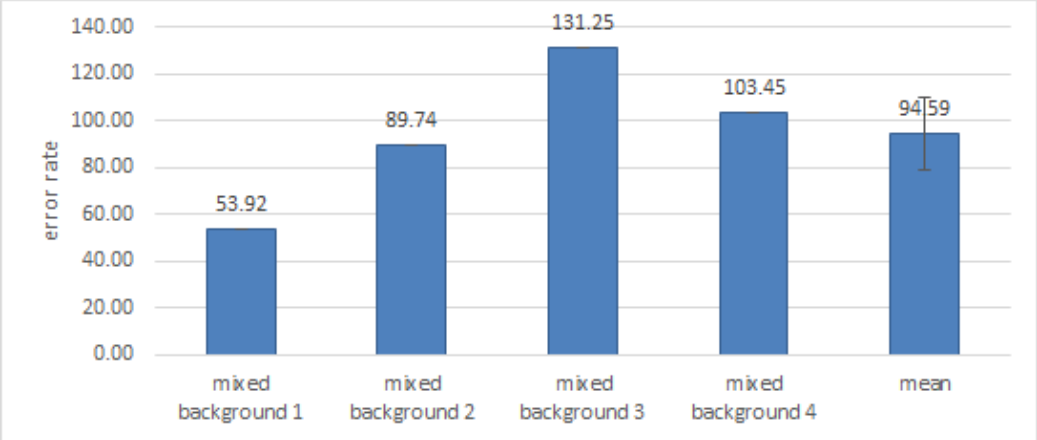


Figure 6.17: Error rates of test images with a mixed background

Discussion

The numbers of blueberries in all of the eight test images are overestimated because their backgrounds have parts recognized as blueberries. But a consistent background causes fewer errors in estimating the number of blueberries compared to a mixed background. Judging from the test results, the light sky, brown soil, and green leaves are all excluded in the first four test images with a consistent background because the ML model has been trained to ignore these background pixels. Therefore, the first four test images have accurate estimates. However, the other four test images have a mixed background that is similar to the blueberries in colour and the images tend to mislead the model to recognize some parts of their backgrounds as blueberries. This causes large errors in the estimates of the other four test images.

For the blueberry estimation, there are two conclusions. (1) The system can make an accurate estimate of blueberries provided the background is consistent to the one in the training images. (2) Improvements are needed in the ML approach being used to estimate the number of objects in an image with a mixed background.

6.3.4 Conclusion

Factors that affect the estimation accuracy include the appearance of objects to be estimated and the image background. When the objects in an image can be distinguished easily from the background and have a similar size, colour, and shape, the estimation of the system is accurate. When the background is consistent with the training images of the ML model, the estimation also performs well. In order to get an accurate result, several steps should be followed when taking a photo for estimation. (1) Select objects that have a similar appearance to the one used to train the ML predictive model. (2) Pick a background that is consistent with the training images. (3) Choose a distance and an angle so that the image pictures the objects with same size as the training images. (4) Cast enough light on the objects so that they are pictured in the same colour as the training images. (5) Take a photo in a way that the objects occupy at least half of the image.

Chapter 7

Summary and Conclusion

This thesis is concerned with the problem of providing farmers with a convenient software application for estimating crop yield in the field prior to harvest.

To solve the major problem, a software system called Estimage was created, which included an Android application for user interaction and a cloud-based machine learning service that could estimate the amount of crop pictured in an image. The Android application was analysed, designed, implemented and tested following the traditional system development process and was responsible for uploading images to the server for estimation. The Android application provided users with an intuitive and user-friendly interface to manage their image records and estimation tasks.

ML predictive models were built and trained using an existing machine learning software package called ilastik. A PHP server application was created to call an Octave program which in turn invokes ilastik to apply a machine learning predictive model to an image so as to estimate the number of objects in the image. With the Estimage system, users can conveniently take photos of the crops in the field and upload these photos to the server for estimation using their mobile device.

Three test cases were created to evaluate the estimation performance of the ML predictive models on different objects in order to find out the factors that affected the estimation accuracy. The first test case was an ideal case estimating the number of identical coins on a wooden table. The second test case was a transitional case estimating the number of logs stacked in a pile. The last test case was a realistic

case estimating the number of blueberries on a bush. The test results showed that the system could accurately estimate the number of objects in a suitable image. Suitable images depicted objects of the same shape, size, and colour as the ones used to train the ML model and thus they had a more accurate estimate. Further, the more consistent the background, the more accurate results an estimation had. These factors could be controlled by adjusting the distance or the angle to the objects when taking a photo of them.

7.1 Future Work

Although the Estimage system has satisfied the goal of this project, there are still many aspects to be improved as below.

(1) The system should have a user account management system that allows users to register a new account and login. The users' information can be saved for other research purposes. (2) The system should have backup copies of users' image records on the server so that users can retrieve their image records to any Android mobile device using their accounts. (3) The system should help users take a qualified image for better estimation of the number of objects. For example, the Android application could indicate how big an object should be in an image when users are taking a photo.

Currently, the prediction models are not very robust. There is need to process the image so as to better prepare the data prior to presentation to the machine learning model. For example, the images could be normalized in terms of object size.

There are other aspects of our system that could be explored. Theoretically, the system can be expanded to estimate the count of any object in an image, such as fish, apples, cans and hats. For any industry that has a problem of counting similar objects, the system provides a potential solution. For example, there is a need in biology to count the number of bacteria in a culture dish under a microscope. This counting process can be time-consuming and error-prone for human beings. The Estimage approach could be introduced as a solution to this task.

Appendix A

Test Log

Time (secs)	Displaying Record List	Interface Switching	Button Click
	7.779	0.117	0.005
	7.820	0.334	0.003
	7.742	0.267	0.006
	7.759	0.069	0.004
	7.704	0.291	0.003
	7.750	0.240	0.001
	7.633	0.042	0.001
	7.712	0.050	0.003
	7.725	0.279	0.001
	7.734	0.246	0.003
average	7.736	0.196	0.003
min	7.633	0.042	0.001
max	7.820	0.334	0.006

Table A.1: 10 Tests on Estimage Client Application Performance

Size (MB)	Image Record	Image	Density Map Image	Total
	0.100	0.654	0.016	0.770
	0.122	0.849	0.159	1.130
	0.109	0.765	0.113	0.987
	0.121	0.868	0.097	1.086
	0.104	0.719	0.023	0.846
	0.079	0.577	0.153	0.809
	0.119	0.675	0.162	0.956
	0.141	0.942	0.224	1.307
	0.154	1.067	0.075	1.296
	0.171	1.253	0.200	1.624
average	0.122	0.837	0.122	1.081
min	0.079	0.577	0.016	0.770
max	0.171	1.253	0.224	1.624

Table A.2: 10 Tests on Size of Data Created on User's Device

Time (secs)	Android App	Data Comm	PHP	Octave	ilastik	Total
	0.111	1.672	0.748	3.6164	6.7896	12.937
	0.135	1.349	1.338	4.2544	7.2176	14.294
	0.084	0.683	0.869	5.1037	8.1723	14.912
	0.096	1.136	1.43	3.644	6.704	13.010
	0.113	1.799	1.222	3.6023	6.6677	13.404
	0.132	1.371	1.536	3.6382	6.8218	13.499
	0.122	1.587	1.118	3.6256	6.7574	13.210
	0.115	1.108	0.821	3.6227	6.7083	12.375
	0.133	0.665	1.516	3.6519	6.6771	12.643
	0.127	1.837	1.568	3.6196	6.8434	13.995
average	0.1168	1.321	1.2166	3.83788	6.93592	13.428
min	0.084	0.665	0.748	3.6023	6.6677	12.375
max	0.135	1.837	1.568	5.1037	8.1723	14.912

Table A.3: 10 Tests on Estimation Time Performance

```
display 50 records

02-12 20:24:19.327 915-915/com.example.peter.berryestimator D/------: static recordList is created
02-12 20:24:19.328 915-915/com.example.peter.berryestimator D/------: on create view
02-12 20:24:19.329 915-955/com.example.peter.berryestimator D/------: querying imageRecords from database
02-12 20:24:19.403 915-915/com.example.peter.berryestimator D/--------: on resume
02-12 20:24:19.982 915-915/com.example.peter.berryestimator D/------: adding record 1
02-12 20:24:20.256 915-915/com.example.peter.berryestimator D/------: adding record 2
02-12 20:24:20.521 915-915/com.example.peter.berryestimator D/------: adding record 3
02-12 20:24:20.797 915-915/com.example.peter.berryestimator D/------: adding record 4
02-12 20:24:21.030 915-915/com.example.peter.berryestimator D/------: adding record 5
02-12 20:24:21.262 915-915/com.example.peter.berryestimator D/------: adding record 6
02-12 20:24:21.490 915-915/com.example.peter.berryestimator D/------: adding record 7
02-12 20:24:21.722 915-915/com.example.peter.berryestimator D/------: adding record 8
02-12 20:24:21.943 915-915/com.example.peter.berryestimator D/------: adding record 9
02-12 20:24:22.137 915-915/com.example.peter.berryestimator D/------: adding record 10
02-12 20:24:22.336 915-915/com.example.peter.berryestimator D/------: adding record 11
02-12 20:24:22.532 915-915/com.example.peter.berryestimator D/------: adding record 12
02-12 20:24:22.715 915-915/com.example.peter.berryestimator D/------: adding record 13
02-12 20:24:22.899 915-915/com.example.peter.berryestimator D/------: adding record 14
02-12 20:24:23.099 915-915/com.example.peter.berryestimator D/------: adding record 15
02-12 20:24:23.291 915-915/com.example.peter.berryestimator D/------: adding record 16
02-12 20:24:23.456 915-915/com.example.peter.berryestimator D/------: adding record 17
02-12 20:24:23.613 915-915/com.example.peter.berryestimator D/------: adding record 18
02-12 20:24:23.766 915-915/com.example.peter.berryestimator D/------: adding record 19
02-12 20:24:23.912 915-915/com.example.peter.berryestimator D/------: adding record 20
02-12 20:24:24.069 915-915/com.example.peter.berryestimator D/------: adding record 21
02-12 20:24:24.219 915-915/com.example.peter.berryestimator D/------: adding record 22
02-12 20:24:24.357 915-915/com.example.peter.berryestimator D/------: adding record 23
02-12 20:24:24.541 915-915/com.example.peter.berryestimator D/------: adding record 24
02-12 20:24:24.677 915-915/com.example.peter.berryestimator D/------: adding record 25
02-12 20:24:24.817 915-915/com.example.peter.berryestimator D/------: adding record 26
02-12 20:24:24.948 915-915/com.example.peter.berryestimator D/------: adding record 27
02-12 20:24:25.092 915-915/com.example.peter.berryestimator D/------: adding record 28
02-12 20:24:25.216 915-915/com.example.peter.berryestimator D/------: adding record 29
02-12 20:24:25.343 915-915/com.example.peter.berryestimator D/------: adding record 30
02-12 20:24:25.466 915-915/com.example.peter.berryestimator D/------: adding record 31
02-12 20:24:25.620 915-915/com.example.peter.berryestimator D/------: adding record 32
02-12 20:24:25.745 915-915/com.example.peter.berryestimator D/------: adding record 33
02-12 20:24:25.849 915-915/com.example.peter.berryestimator D/------: adding record 34
02-12 20:24:25.952 915-915/com.example.peter.berryestimator D/------: adding record 35
02-12 20:24:26.047 915-915/com.example.peter.berryestimator D/------: adding record 36
02-12 20:24:26.144 915-915/com.example.peter.berryestimator D/------: adding record 37
02-12 20:24:26.232 915-915/com.example.peter.berryestimator D/------: adding record 38
02-12 20:24:26.314 915-915/com.example.peter.berryestimator D/------: adding record 39
02-12 20:24:26.392 915-915/com.example.peter.berryestimator D/------: adding record 40
```

Figure A.1: Logging Example 1

```
02-12 20:24:26.581 915-915/com.example.peter.berryestimator D/------: adding record 42
02-12 20:24:26.663 915-915/com.example.peter.berryestimator D/------: adding record 43
02-12 20:24:26.730 915-915/com.example.peter.berryestimator D/------: adding record 44
02-12 20:24:26.800 915-915/com.example.peter.berryestimator D/------: adding record 45
02-12 20:24:26.860 915-915/com.example.peter.berryestimator D/------: adding record 46
02-12 20:24:26.918 915-915/com.example.peter.berryestimator D/------: adding record 47
02-12 20:24:26.979 915-915/com.example.peter.berryestimator D/------: adding record 48
02-12 20:24:27.019 915-915/com.example.peter.berryestimator D/------: adding record 49
02-12 20:24:27.057 915-915/com.example.peter.berryestimator D/------: adding record 50
02-12 20:24:27.062 915-955/com.example.peter.berryestimator D/-----: finish querying imageRecords from database
02-12 20:24:27.064 915-955/com.example.peter.berryestimator D/-----: load 50 image records time cost: 7734

interface switching & button clicks
02-12 20:26:47.829 915-915/com.example.peter.berryestimator D/--------: on pause
02-12 20:26:48.042 915-915/com.example.peter.berryestimator D/-----: create image record interface time cost: 117
02-12 20:27:14.826 915-915/com.example.peter.berryestimator D/--------: on resume
02-12 20:27:17.753 915-915/com.example.peter.berryestimator D/-----: button click time cost: 3
02-12 20:27:17.754 915-915/com.example.peter.berryestimator D/--------: on attach
02-12 20:27:18.088 915-915/com.example.peter.berryestimator D/-----: record information interface time cost: 334
02-12 20:27:30.927 915-915/com.example.peter.berryestimator D/--------: on pause
02-12 20:27:31.024 915-915/com.example.peter.berryestimator D/-----: create image record interface time cost: 69
02-12 20:27:37.869 915-915/com.example.peter.berryestimator D/--------: on resume
02-12 20:27:41.851 915-915/com.example.peter.berryestimator D/-----: image information interface time cost: 267
02-12 20:27:53.816 915-915/com.example.peter.berryestimator D/-----: dialog is dismissed
02-12 20:27:53.818 915-915/com.example.peter.berryestimator D/--------: on detach
02-12 20:27:55.590 915-915/com.example.peter.berryestimator D/-----: button click time cost: 1
02-12 20:27:55.592 915-915/com.example.peter.berryestimator D/--------: on attach
02-12 20:27:55.883 915-915/com.example.peter.berryestimator D/-----: record information interface time cost: 291
02-12 20:28:00.402 915-915/com.example.peter.berryestimator D/-----: image information interface time cost: 240
02-12 20:28:04.913 915-915/com.example.peter.berryestimator D/--------: on pause
02-12 20:28:04.976 915-915/com.example.peter.berryestimator D/-----: create image record interface time cost: 42
02-12 20:28:09.857 915-915/com.example.peter.berryestimator D/--------: on resume
02-12 20:28:11.936 915-915/com.example.peter.berryestimator D/-----: dialog is dismissed
02-12 20:28:11.938 915-915/com.example.peter.berryestimator D/--------: on detach
02-12 20:28:13.777 915-915/com.example.peter.berryestimator D/-----: button click time cost: 0
02-12 20:28:13.801 915-915/com.example.peter.berryestimator D/--------: on attach
02-12 20:28:14.080 915-915/com.example.peter.berryestimator D/-----: record information interface time cost: 279
02-12 20:28:20.267 915-915/com.example.peter.berryestimator D/--------: on pause
02-12 20:28:20.340 915-915/com.example.peter.berryestimator D/-----: create image record interface time cost: 50
02-12 20:28:26.746 915-915/com.example.peter.berryestimator D/--------: on resume
02-12 20:28:27.959 915-915/com.example.peter.berryestimator D/-----: image information interface time cost: 246

02-12 20:50:11.207 22440-25327/com.example.peter.berryestimator D/------: uploading record 103 and get estimate
02-12 20:50:11.423 22440-25327/com.example.peter.berryestimator D/record size: 154KB
02-12 20:50:11.423 22440-25327/com.example.peter.berryestimator D/image size: 1067KB
02-12 20:50:24.688 22440-22440/com.example.peter.berryestimator D/-----: madapter update record 103
02-12 20:50:25.361 22440-22440/com.example.peter.berryestimator D/density image size: 75KB
```

Figure A.2: Logging Example 2

```
estimation process
02-15 11:34:18.318 16732-17623/com.example.peter.berryestimator D/------: uploading record 96 and get estimate
02-15 11:34:18.453 16732-17623/com.example.peter.berryestimator D/------: retrieve image and image record time cost: 135
02-15 11:34:18.524 16732-17623/com.example.peter.berryestimator D/record size: 171KB
02-15 11:34:18.525 16732-17623/com.example.peter.berryestimator D/image size: 1253KB
02-15 11:34:18.586 16732-16747/com.example.peter.berryestimator I/art: Background partial concurrent mark sweep GC freed 612(29KB)
AllocSpace objects, 13(4MB) LOS objects, 28% free, 39MB/55MB, paused 5.981ms total 45.048ms
02-15 11:34:20.520 16732-17623/com.example.peter.berryestimator D/------: send image & image record data to server time cost: 1894
02-15 11:34:32.844 16732-17623/com.example.peter.berryestimator D/respond code: 200
02-15 11:34:32.857 16732-17623/com.example.peter.berryestimator I/System.out: sync status before:
02-15 11:34:32.857 16732-17623/com.example.peter.berryestimator I/System.out: sync status after: 1
02-15 11:34:32.857 16732-17623/com.example.peter.berryestimator I/System.out: connect to db successfully
02-15 11:34:32.857 16732-17623/com.example.peter.berryestimator I/System.out: Database exists or created successfully
02-15 11:34:32.857 16732-17623/com.example.peter.berryestimator I/System.out: table created successfully
02-15 11:34:32.857 16732-17623/com.example.peter.berryestimator I/System.out: record id = 96
02-15 11:34:32.857 16732-17623/com.example.peter.berryestimator I/System.out: record target type = Blueberry
02-15 11:34:32.857 16732-17623/com.example.peter.berryestimator I/System.out: image record not in db
02-15 11:34:32.857 16732-17623/com.example.peter.berryestimator I/System.out: ans = callilastik is running on modelBlueberry.ilp
02-15 11:34:32.858 16732-17623/com.example.peter.berryestimator I/System.out: INFO ilastik_main: Starting ilastik from
"/usr/local/ilastik-master-Linux".
02-15 11:34:32.858 16732-17623/com.example.peter.berryestimator I/System.out: Starting ilastik from "/usr/local/ilastik-master-
Linux".
02-15 11:34:32.858 16732-17623/com.example.peter.berryestimator I/System.out: INFO
ilastik.applets.thresholdTwoLevels.opThresholdTwoLevels: Using 'vigra' labeling implemetation
02-15 11:34:32.858 16732-17623/com.example.peter.berryestimator I/System.out: INFO ilastik.shell.projectManager: Opening Project:
/home/peterchen/harvest_estimator/Harvest_Estimator/Code/Blueberry.ilp
02-15 11:34:32.858 16732-17623/com.example.peter.berryestimator I/System.out: INFO ilastik.workflows.counting.countingWorkflow:
Exporting object density image 0 to /home/peterchen/harvest_estimator/Harvest_Estimator/scratch/ImageIn_Density.h5/exported_data
02-15 11:34:32.858 16732-17623/com.example.peter.berryestimator I/System.out: Result 0/1 Progress: 0.0 INFO
lazyflow.operators.ioOperators.ioOperators.OpH5WriterBigDataset: Data shape: (753, 500, 1)
02-15 11:34:32.858 16732-17623/com.example.peter.berryestimator I/System.out: 0.0 INFO lazyflow.utility.bigRequestStreamer:
Estimated RAM usage per pixel is 12 bytes * safety factor (2.0)
02-15 11:34:32.858 16732-17623/com.example.peter.berryestimator I/System.out: 0.0 100.0 100.0 100.0 100.0
02-15 11:34:32.858 16732-17623/com.example.peter.berryestimator I/System.out: ans = 0
02-15 11:34:32.858 16732-17623/com.example.peter.berryestimator I/System.out: ans = ilastik time cost is: 7.2176 seconds
02-15 11:34:32.858 16732-17623/com.example.peter.berryestimator I/System.out: ans = imagePrediction.m is running
02-15 11:34:32.858 16732-17623/com.example.peter.berryestimator I/System.out: time cost in Octave and ilastik is: 11472
milliseconds
02-15 11:34:32.858 16732-17623/com.example.peter.berryestimator I/System.out: succeed to get estimate, density and object image
02-15 11:34:32.858 16732-17623/com.example.peter.berryestimator I/System.out: New record created successfully
02-15 11:34:34.429 16732-17623/com.example.peter.berryestimator I/System.out: time cost in PHP, Octave and ilastik is: 12810
milliseconds
```

Figure A.3: Logging Example 3

Bibliography

[Cor15] International Data Corporation. IDC: Smartphone OS Market Share. Retrieved from `http://www.idc.com/prodserv/smartphone-os-market-share.jsp`, 2015. Accessed 23 Nov. 2015.

[Dif15] DifferenceBetween.net. Difference Between TCP and HTTP. Retrieved from `http://www.differencebetween.net/technology/internet/difference-between-tcp-and-http/`, 2015. Accessed 23 Nov. 2015.

[EBrHW15] John W. Eaton, David Bateman, Søren Hauberg, and Rik Wehbring. *The GNU Octave 4.0 Reference Manual: Free Your Numbers*. Samurai Media Ltd., 2015.

[Elg05] Ben Elgin. Google Buys Android for Its Mobile Arsenal. *Bloomberg Businessweek*, August 2005.

[FNKH12] L Fiaschi, R. Nair, U. Koethe, and F. A. Hamprecht. Learning to Count with Regression Forest and Structured Labels. In *Proceedings of the International Conference on Pattern Recognition (ICPR 2012)*, Tsukuba, Japan, November 2012.

[Goo15a] Google. Activity — Android Developers. Retrieved from `http://developer.android.com/reference/android/app/Activity.html`, 2015. Accessed 23 Nov. 2015.

[Goo15b] Google. Camera — Android Developers. Retrieved from

http://developer.android.com/intl/zh-cn/guide/topics/media/camera.html, 2015. Accessed 23 Nov. 2015.

[Goo15c] Google. Dashboards — Platform Versions — Android Developers. Retrieved from http://developer.android.com/intl/zh-cn/about/dashboards/index.html, 2015. Accessed 23 Nov. 2015.

[Goo15d] Google. Fragments — Android Developers. Retrieved from http://developer.android.com/guide/components/fragments.html, 2015. Accessed 23 Nov. 2015.

[Goo15e] Google. Introduction — Material design — Google design guidelines. Retrieved from https://www.google.com/design/spec/material-design/introduction.html, 2015. Accessed 23 Nov. 2015.

[Goo15f] Google. RecyclerView — Android Developers. Retrieved from https://developer.android.com/intl/zh-cn/reference/android/support/v7/widget/RecyclerView.html, 2015. Accessed 23 Nov. 2015.

[Ho95] Tin Kam Ho. Random Decision Forests. In *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, pages 278–282, Montreal, QC, August 1995.

[KP98] Ron Kohavi and Foster Provost. Glossary of Terms. *Machine Learning*, 30:271—274, 1998.

[Lan11] Pat Langley. The changing science of machine learning. *Machine Learning*, 82(3):275–279, February 2011.

[McC13] Harry McCracken. Who's Winning, iOS or Android? All the Numbers, All in One Place. Retrieved from http://techland.time.com/2013/04/16/ios-vs-android/, April 2013.

[MRT12] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. MIT Press, 2012.

[Ort14] Mike Ortiz. TouchImageView — GitHub. Retrieved from `https://github.com/MikeOrtiz/TouchImageView`, 2014. Accessed 23 Nov. 2015.

[Poi15] Tutorials Point. Android — Architecture. Retrieved from `http://www.tutorialspoint.com/android/android_architecture.htm`, 2015. Accessed 23 Nov. 2015.

[Qui86] John Ross Quinlan. Induction of Decision Trees. *Machine Learning*, 1:81–106, 1986.

[RN03] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2003.

[Soh11] Avtar Sohi. Understanding Basics of UI Design Pattern MVC, MVP and MVVM. Retrieved from `http://www.codeproject.com/Articles/228214/Understanding-Basics-of-UI-Design-Pattern-MVC-MVP`, July 2011.

[SSKH11] C. Sommer, C. Strähle, U. Köthe, and F. A. Hamprecht. ilastik: Interactive Learning and Segmentation Toolkit. In *Proceedings of the Eighth IEEE International Symposium on Biomedical Imaging (ISBI)*, pages 230–233, 2011.

[Stu13] Andreas Stuetz. Android PagerSlidingTabStrip — GitHub. Retrieved from `https://github.com/astuetz/PagerSlidingTabStrip`, 2013. Accessed 23 Nov. 2015.

[Tar15] Dmytro Tarianyk. FloatingActionButton — GitHub. Retrieved from `https://github.com/Clans/FloatingActionButton`, 2015. Accessed 23 Nov. 2015.

[Wel13] Chris Welch. Before it took over smartphones, Android was originally destined for cameras. Retrieved from `http://www.theverge.com/2013/4/16/4230468/`

android-originally-designed-for-cameras-before-smartphones, April 2013.