5-2019

A Mobile Application for Crowdsourced Acquisition of Urban Street-View Pedestrian Facility Data

Andrew Fink

The University of Southern Mississippi

A Mobile Application for Crowdsourced Acquisition of Urban Street-View Pedestrian Facility Data

by

Andrew Fink

A Thesis
Submitted to the Honors College of
The University of Southern Mississippi
in Partial Fulfillment
of Honors Requirements

April 2019

Approved by

Chaoyang Zhang, Ph.D.,
Thesis Adviser
Professor of Computer Science

Andrew Sung, Ph.D.,
Director
School of Computing

Ellen Weinauer, Ph.D.,
Dean
Honors College

Abstract

In recent years, pedestrians have been dangerously overrepresented in traffic crashes, and the pedestrian fatality rate has steadily increased during the last decade. Additionally, studies have shown that the majority of pedestrian-involved traffic accidents occur in urban non-intersections, which suggests that a more well-connected pedestrian facility network in cities would lower the rate of pedestrian involvement in traffic accidents. One way to improve the pedestrian facility network coverage is to first have up-to-date, accurate, and thorough data regarding the presence of existing pedestrian facilities. However, state departments of transportation have stated that the current methods of acquiring this data are expensive and time consuming. In this project, we developed a mobile application prototype for crowdsourced acquisition of street-view images containing pedestrian facilities, or more specifically, crosswalks. The resulting application used modern full-stack development techniques and is a native Android application that allows the user to take pictures using their mobile devices and automatically upload those pictures, along with relevant metadata (such as location data), to a server where they are classified using a machine learning model that was trained to recognize the presence of crosswalks in images.

Keywords— crosswalk, crowdsourced, mobile application, pedestrian facility, software development, street-view, transfer learning, transportation

Acknowledgements

I would like to recognize and thank my thesis advisor, Dr. Joe Zhang, for his mentorship in this project during the last year. This project would not have been possible without his efforts and direction. I would also like to thank Bailey Luttrell for his guidance and his work on the machine learning portion of this project and Dr. Yuanyuan Zhang for allowing me to join her team for the NCHRP-209 project and for her insight on the constraints and goals of the software.

Additionally, I would like to thank the faculty of the School of Computing for their instruction during the past four years and the Honors College for their hard work in challenging me and in providing me the resources and guidance that I needed to be successful.

Lastly, I would like to pay special thanks to my wife and my parents for their love, encouragement, and support during my time at the University of Southern Mississippi.

Table of Contents

List of Figures

List of Definitions and Abbreviations

Below are some terms, acronyms, and abbreviations that are commonly used in computer science and that will be used in this paper.

- *Software Development Kit (SDK)*: a collection of software and/or libraries used for developing applications for a given system or device.
- *Application Programming Interface (API)*: a set of tools and methods for communication among different components of a system or between systems.
- *User Interface (UI)*: a graphical interface.
- *JavaScript Object Notation (JSON)*: a standard text notation for storing and transporting data in a human-readable and universal format.
- *Extensible Markup Language (XML)*: a general markup language that can be used for storing and transporting data, as well as being a foundation for building customized markup languages.
- *Representational State Transfer (REST)*: a software architectural style that defines a set of operations for developing and using web services.
- *RESTful*: describes a web service that conforms to the REST style.
- *Hypertext Transfer Protocol (HTTP)*: the standard protocol for data communication in the client-server computing model.
- *PHP: Hypertext Preprocessor (PHP)*: a scripting language that is commonly used for web development.

Chapter 1: Introduction

One growing concern of cities and state departments of transportation is pedestrian safety. In a 2015 survey [1] by the National Highway Traffic Safety Administration (NHTSA), it was shown that 4,735 of the 32,719 total fatalities from traffic crashes in 2013 were pedestrian fatalities, which makes up 14 percent of the total number of fatalities. This is the result of a steady increase from 11 percent in 2004. Additionally, 73 percent of these pedestrian fatalities occurred in urban areas, and 69 percent occurred at non-intersections. This highlights the importance of a well-connected pedestrian facility network in urban areas, as it would decrease pedestrian-involved traffic accidents [2]. However, to build a well-connected pedestrian facility network, there must first be accurate and thorough data of existing pedestrian facilities [3]. The problem is that there is not currently an efficient method of collecting these data [4]. Currently, state departments of transportation must manually survey cities to map out these facilities, which is both expensive and time-consuming; therefore, three out of fifty state DOTs have identified the collection of these data as a primary goal [5], [6]. The IDEA project, NCHRP-209, that we are working on aims to provide a more efficient and cost-effective solution that would allow state DOTs to map existing pedestrian facilities, which would help their efforts to build a more well-connected pedestrian facility network.

The system that was proposed is a system for automated acquisition of pedestrian facility data from satellite and street-view images through the use of machine learning algorithms. As a subset of that project, we are building an additional tool for crowdsourced collection of street-view image data. This tool is a mobile application

that allows the user to take a picture of a pedestrian facility at street level. The picture and its related metadata are then sent to a server and stored in a database so that it can be queried and analyzed. Then, a model trained with machine learning classifies the image as either containing or not containing a crosswalk.

Chapter 2: Literature Review

Mobile application development is a complex process from start to finish, and machine learning is even more complex; therefore, building mobile applications that utilize machine learning algorithms is a daunting task. In addition, there are many factors to consider that would greatly affect the development process. This review of literature will present and discuss existing research regarding mobile application development and the integration of machine learning algorithms in mobile application development, as well as some of the documentation available to mobile application developers to aid them in the development process.

2.1 Multi-Platform Development

One of the more controversial topics in mobile application development is the problem of multi-platform development. In a study done by the International Data Corporation (IDC) in 2017 [7], it was determined that more than 99 percent of all smartphones shipped to vendors worldwide in 2016 were either Android or iOS phones, with 81.4 percent being Android devices and 18.2 percent being iOS devices for the final quarter of 2016 (during which time 429.8 million units were sold). This suggests the importance of developing mobile applications that can be used on both Android and iOS devices, at the very least. However, because of the fundamental differences in the native development libraries and supported languages of different platforms, multi-platform development is quite complicated and can be very time-intensive and expensive. There are several approaches to multi-platform development, each with its own advantages and disadvantages.

In an article by Heitkötter, Hanschke, and Majchrzak [8], the authors explain the problems of developing applications for multiple platforms and give the solution of cross-platform development tools. They compare the following different approaches to multiple-platform development: mobile Web applications, Titanium Mobile, PhoneGap, and native application development. Developing separate native applications generally allows the developer to make the most of the hardware and features of the targeted device by using its platform's SDK. However, developing native applications for multiple platforms is very time-consuming. This is the reason for the evolution of cross-platform development tools. These tools allow the developer to write an application using a single platform or development environment and then either run the application on multiple platforms or export the application and generate different platform-specific applications from it. The authors note that the latter type of cross-platform tools is still new and that there are no commercial tools of that category available yet. In the article, the authors compare the different approaches on seven criteria from the infrastructure perspective: license and costs, supported platforms, access to platform-specific features, long-term feasibility, look and feel, application speed, and distribution. They also compare seven criteria from the development perspective: development environment, Graphical User Interface (GUI) design, ease of development, maintainability, scalability, opportunities for further development, and speed and cost of development. The authors concluded that PhoneGap is the preferred alternative to native development if the developer is willing to sacrifice the resemblance of the UI to that of a native application.

Of course, not all authors are willing to name a conclusive “best option”. An article by A.I. Wasserman [9] describes some of the issues regarding mobile application development. The author conducted a survey of the opinions and practices of most app developers and made the following four conclusions: 1) most developers made smaller applications, with fewer than three developers responsible for developing each application, 2) there was a strong difference of opinion and practice between the development of native applications and the development of cross-platform mobile web applications, 3) most developers did, in fact, adhere to the established recommended practices, and 4) developers rarely gathered organized metrics of their development process and efforts. The article gives an overview of the popular development environments used for different platforms and why they are so useful. Finally, the article identifies some of the new areas of research and development in mobile application development, assesses the idea of recommended programming practices, and explains the importance of finding good techniques for effectively managing increasingly complex development projects. Wasserman does not give a conclusive best approach from his own experience but instead presents conclusions of other developers. Similarly, an article by M. Emiliano et al. [10] examines the different development technologies available to mobile application developers and compares them to determine which technologies are recommended to use for given situations. The article studies three development approaches: native development, web development, and hybrid development. Native development refers to developing applications for a platform in its native SDK. Web development refers to developing a pure web application that can be executed on multiple platforms.

Hybrid development refers to using a mix of the native and web development approaches. The authors look at several case studies and interview many technical experts in the field of application development to obtain the data, and they use this data to write a guide for deciding which of these technologies to use (which they present at the end of the article).

2.2 Android Development

While native development is very time-intensive and expensive, all the aforementioned authors agree that it is a great option if the developer is insistent on giving the application a native look and feel on all platforms. For native application development, there are several tools available to the developer, especially in Android and iOS development. The Android developer site [15] contains the documentation for Android development and Android Studio, which is the official development environment for Android development. Android Studio allows the developer to develop Android applications easily with the use of a feature called the visual layout editor, which enables the editing of the layout through a visual interface. This speeds up the process of application design by allowing the developer to edit the layout of the user interface through a simple drag-and-drop feature for inserting, moving, and resizing new UI elements, which is an improvement on manually writing the layout XML, a process that can be complicated and time-consuming. Android Studio also features an Android Emulator tool, which allows the developer to build and run apps on various emulated Android devices, including tablets, phones, and even Android Wear OS devices. This is a welcome and easy alternative to running the application on a real device over USB, which can be rather slow.

With the emulator's ability to simulate almost all features available on a real device, it has become a popular and useful tool. Most importantly, Android studio features a powerful code editor. The code editor includes support for Java, Kotlin, and C/C++. It has Lint tools for finding and fixing problems such as version incompatibility, deprecated methods and libraries, unoptimized syntax, unused variables, and poor performance. It also includes useful shortcuts for importing libraries automatically, inline debugging, code completion, and code reformatting. The Android Studio IDE is a powerful tool for Android development in many ways, and this source provides information about using it.

2.3 iOS Development

For iOS development, there are similar tools available to the developer. Apple, Inc.'s Xcode IDE web page [14] contains the documentation for the Xcode integrated development environment, which is the official environment for iOS application development. The documentation includes descriptions of all of the features of Xcode 9, the newest release of Xcode. The author of the documentation emphasized the speed and aesthetic of Xcode, especially that of its source code editor, which boasts convenient gestures for quick selection and editing of code segments and built-in powerful code refactoring. The IDE also features version control with full Git integration, complete with a Git source control navigator. The documentation specifies a new way of connecting iOS devices for running and debugging applications: network connection. Xcode projects can be installed, run, and debugged via a wireless network. For those without iOS devices for debugging, Xcode provides a device simulator app, which allows the developer to run and debug

applications on a virtual iOS device on his or her computer. The documentation includes other resources as well, such as an API reference guide, sample code, and relevant articles related to Xcode application development. Apple's Xcode is, unfortunately, only available on MacOS; therefore, iOS development can only be done on MacOS.

2.4 Machine Learning in Mobile Development

For a developer to integrate machine learning algorithms into a mobile application, he or she would first need to understand the concept of machine learning, or as in this case, deep learning. An article by J. Schmidhuber [11] gave an introduction to deep learning in neural networks, addressed its various problems and themes, and described the different categories of neural networks and their histories. In this article, the author also explains the fundamental credit assignment problem, which defines the general idea of deep learning. Specifically, it states that certain components of a learning system are to be credited for its success, and changes to specific components can improve or diminish the performance of the system. The article also describes the different types of neural networks, such as feedforward neural networks and recurrent neural networks. Another topic discussed is the difference between supervised learning, unsupervised learning, and reinforcement learning. The author concludes the article by giving an assessment of the future of deep learning in neural networks and some problems that would need to be solved to facilitate its advancement.

One of the most popular machine learning libraries and systems available to developers is the TensorFlow machine learning system. A paper written by M. Abadi

and the rest of the Google Brain team at Google [12] describes the TensorFlow system, its applications and performance, and the data model that it uses. The Google Brain team developed TensorFlow based on what they learned in their experience using TensorFlow's predecessor, DistBelief. TensorFlow improves upon DistBelief by making it more approachable and by broadening its applicability to a wider range of ideas, making it an available tool for a broader variety of researchers and developers. TensorFlow can be used to train models and run them on several platforms, from mobile devices to large datacenters and HPC clusters. This article describes the design principles of TensorFlow, its execution model, some case studies, and the implementation of the TensorFlow library. The core library for TensorFlow is written in C++, and it supports several client languages, with an emphasis on C++ and Python. The authors also give an evaluation of the performance of TensorFlow in various use cases.

When a developer can successfully integrate machine learning into mobile application development, the resulting mobile application could have great potential. For example, an article by A. Zainab [13] explains the work the author did to develop an Android application that uses a TensorFlow machine learning model to detect objects in a camera's field of vision in real time. The model used in the application is based on Scalable Object Detection to track twenty different classifications in the camera preview. The application is also able to support both multiple-object detection and moving-object detection. The author proposes that the technology would be useful for traffic detection, surveillance, facial recognition, robotics, and other areas.

There are clearly many factors to consider in mobile application development. Which approach to multi-platform development the developer chooses is incredibly important and affects all aspects of the development process. For native application development, there are several useful tools available to the developer, and knowledge of these tools is crucial to efficient development of powerful applications. One powerful tool that can, and should, be used in mobile application development is machine learning. Of the systems available, the arguably most popular system is TensorFlow, which can be used to train and use models in applications (such as applications for image processing). This technology can be used for many types of applications, and its potential is nearly limitless. It is important for a developer to be aware and knowledgeable of all these tools to efficiently develop powerful applications.

Chapter 3: Methodology

3.1 Development Approach

After consideration of the available options for developing a mobile application, we decided to develop a native Android application, with the intent to later develop a native iOS application in the future. The primary reason for this decision was simply that implementing a native custom camera preview that uses the device's hardware, as is required for this project, would be unnecessarily complex in a cross-platform mobile development solution, whereas it is somewhat simple in native development [8]. Android was chosen as the initial platform, rather than iOS, because of the restrictions Apple has placed on the use of its development environment for iOS; i.e., the Xcode IDE can only be used on MacOS systems [14].

For classification of the images in this prototype, we focused only on crosswalks, with the intention of later adding other pedestrian facilities. The implementation of the classification logic could be done in one of two ways for this project. The first was to perform classification in the client (i.e., in the Android application); the second was to perform classification on the server. We chose to perform the classification on the server for two reasons: 1) by moving classification to the server, the logic is abstracted from the mobile application, and therefore, any updates to the machine learning model would not require an update to the mobile application, and 2) the model may take up a lot of storage and memory on the device performing classification if it were done client-side, which would not be ideal. By moving

classification to the server, it speeds up the processing time on the client and allows classification to be performed without the server's dependence on the client [13].

3.2 Development Environment

3.2.1 Android Application

The Android application is written in Java, which is the most popular and most well-supported language for Android development. The primary reason that we chose to use Java was because of its convenience for use with the Android SDK [15]. The application's targeted SDK is API level 27 (Android Oreo, v8.1.0), and the application's minimum SDK is API level 21 (Android Lollipop, v5.0).

3.2.2 Server

The back-end of the system consists of an Apache2 HTTP server that is running on Ubuntu 16.04. The server hosts the database and the REST API endpoints for interfacing the database. The reason we chose Apache2 for our server was because Apache2 is a powerful and flexible web server software, and it is very easily scalable and configurable. The back-end was primarily written in vanilla PHP, and the scripts for using the machine learning model are written in Python.

3.2.3 Database

The DBMS we chose for the database is MySQL, which was an obvious choice for several reasons. A relational database was ideal for this project, as opposed to a nonrelational database, because we knew that as the project grew in complexity, it would be useful to be able to easily model relationships between tables. Also,

MySQL is a great option for small to medium scale projects, and this project, as a whole, would not need to scale past medium-scale. Lastly, as we are using PHP and Apache2 on a Linux machine, it made perfect sense to use MySQL because those four technologies work exceptionally well together and are colloquially known as a "LAMP" stack (for Linux-Apache-MySQL-PHP).

3.3 Procedure

3.3.1 Apache2 Server

To install and configure the Apache2 HTTP server, we simply installed Apache2 through Ubuntu's default package manager and then opened port 80 in the firewall to allow access to the server from within the USM network. At this point, the server was configured to serve resources within the /var/www/html directory by default. The basic authentication that is provided by default to Apache2 was then added to the server to prevent access without authentication.

3.3.2 MySQL Database

To install the MySQL database, we installed mysql-server through Ubuntu's package manager then configured it through the use of the security script provided by MySQL. At this point, MySQL was installed and ready for use. The next step was to create the Transportation database and add the necessary table for storing the data related to uploaded images. The necessary fields for the image_data table are: id (which is the primary key), location_lat for storing the latitude, location_long for storing the longitude, image_path for storing the filename of the uploaded image,

upload_time for storing the time at which the image was uploaded, and classification for storing the result of classifying the image, as shown in Fig. 3-1.

Figure 3-1 MySQL image data table schema

```
mysql> USE Transportation;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> DESCRIBE image_data;
+----------------+------------------+------+-----+---------+----------------+
| Field          | Type             | Null | Key | Default | Extra          |
+----------------+------------------+------+-----+---------+----------------+
| id             | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| location_lat   | float            | YES  |     | NULL    |                |
| location_long  | float            | YES  |     | NULL    |                |
| image_path     | varchar(260)     | NO   |     | NULL    |                |
| upload_time    | datetime         | NO   | MUL | NULL    |                |
| classification | varchar(30)      | YES  |     | NULL    |                |
+----------------+------------------+------+-----+---------+----------------+
6 rows in set (0.00 sec)

mysql>
```

3.3.3 API Development

The API for the system is a set of REST API endpoints for uploading image data to the server and for retrieving data from the database, and it uses JSON for client-server data interchange, as it is universally supported and easily parsed by object-oriented programming languages [16]. The API contains endpoints for receiving POST requests for submitting resources to the server (e.g., image data from the Android app) and for receiving GET requests for requesting resources from the server (e.g., image data from the database). The API resides in the /var/www/html/api directory, and it contains a folder called “geodata”, which is the endpoint for the image data being used for this project. It is a good idea to use API versioning when developing a REST API [17], so the first version of the geodata API resides in /var/www/html/api/geodata/v1. There are other endpoints in the API for other functions not related to the scope of this project. Furthermore, we wanted to be

able to update these different endpoints separately, so we moved the versioning inside of each endpoint (…/api/geodata/v1) rather than having all the endpoints contain each version (…/api/v1/geodata). Inside the v1 folder, there is an index.php file that contains the logic for handling requests to the geodata endpoint. In this file, we utilized PHP's built-in functions for receiving HTTP requests and sending responses.

The first step in developing the API endpoint is to implement the logic for handling HTTP requests. Using PHP's $_SERVER variable, which is an associative array that contains information about the server and any requests sent to it, one can get information like the request method (e.g., POST, GET, PUT) and the remote address (the IP address of the request's origin device). This variable was used to get the relevant information and handle each request appropriately.

A function called handlePostRequest was defined in another file and is an implementation for handling a multipart/form-data POST request from the client. It takes the origin's IP address (for generating unique filenames for the images), a latitude coordinate value, a longitude coordinate value, and the file path of the uploaded image. When the function is called, the image is renamed and saved to the server, and the file path and coordinates are added to a record in the database. The function then returns a response object, which is sent back to the client in an HTTP response. After the image is added to the database, and the response is sent, the image will be classified using the machine learning model to determine whether a crosswalk is present in the image. Once this is determined, the record in the database is updated with the classification label of the image.

3.3.4 Mobile App UI

For the Android application, the user interface was intentionally very simple, as the actual requirements of the user interface were not yet clearly defined. The primary features it needed are a main page for displaying a button for launching the camera preview, the camera preview page itself, and a page for displaying the results. To minimize the number of pages present in the application, the main page and the results page were combined into one page. The typical way to display these pages in Android is by using something called an activity. An activity is the primary container for interfaces in an Android app, and it contains a view for containing UI elements and a set of lifecycle methods for defining actions and events that are connected to the UI elements [15]. An activity would be the simplest way to implement a page in an app, and Android apps are required to have at least one activity anyway. Therefore, since the Android app needs to have two pages (the main page and the camera preview), the app has two activities (named MainActivity and CameraPreviewActivity, respectively).

The MainActivity contains a button for launching the CameraPreviewActivity and a set of toggle buttons for toggling between the image result view, which allows the user to view the image that was taken, and the upload result view, which allows the user to see the status of the upload and coordinates of the location at which the image was taken.

The CameraPreviewActivity contains a view for displaying the live camera feed and a button for taking a picture. There were two different ways to implement the camera preview. The Android SDK provides a library for accessing the device's

default camera preview by allowing the app to launch an intent to the camera app on the device. Once a picture is taken from the camera app, the image is sent back to the Android app and can be used. However, while this approach is very simple and convenient for most uses, it certainly has its drawbacks. The primary drawback is its inability to be customized. Because we have to be able to display camera overlays in future stages of this project, the camera preview must be able to support this. Additionally, a greater control of camera resolutions and aspect ratios is required. Neither of these is possible through the use of the default camera preview, so we decided to implement our own camera preview.

The way this was done was through the use of the Android SDK's Camera2 hardware package, which allows the app to directly access the camera hardware, not just the default camera app. Through this, the app is able to display the live feed from the camera in a view within the activity. This makes it possible to add custom buttons to the preview, add overlays, and programmatically change the camera's resolution and aspect ratio. The resulting CameraPreviewActivity allows for simple changes and customizations in the future [15].

Once the picture is taken, the image is automatically processed and sent to the server, and the user is directed back to the home page, where he or she can view the picture and the image upload status or take another picture.

3.3.5 Initial Client-side Image Preprocessing

When the picture is first taken, it is in a rather raw and poorly formatted state, so it must undergo some preprocessing on the client before being loaded into the home page view and sent to the server. For example, there is an obscure bug in which

pictures taken on certain Android device models are saved to the device in landscape orientation, regardless of the actual orientation of the device. Since this only occurs on certain Android devices and not all, device orientation must be checked against the image orientation to determine whether this error occurred, and if so, the image is rotated accordingly.

3.3.6 Location and Image Upload Logic

The last step in the processing of the image is obtaining the device's location and sending the image and metadata to the server. For obtaining the device's location, we used the Android Location API provided by Google Play Services [15]. Through this library, the device is able to request location updates from the device's GPS or from the network (whichever is more accurate at the time).

The next step was to send the image data to the server. This was done by building a custom HttpPostRequest class through the use of the Apache HTTPClient library [18]. By using the HTTPClient library, we implemented the HttpPosRequest that builds a multipart/form-data POST request and allows for adding authentication and other headers to the request and multiple types of data to the request body, and it makes it simple to send the request and wait for the response. Next, we implemented the logic for building the request through this class and sending the request to the server. This was done in a class called UploadTaskHTTP that extends the Android SDK AsyncTask class [15], which defines a task that is executed on a background thread so as not to use the UI thread, which would consequently freeze the UI of the application.

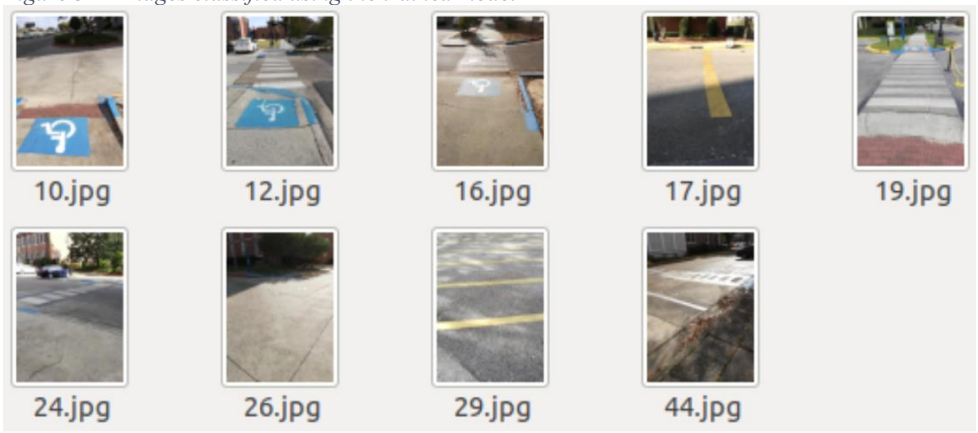
All these tasks are executed automatically in the background, so the user is free to take more pictures during this time. Once the server's response is received, the upload status in the UI is updated with the success/failure message, and the coordinates are also displayed.

3.3.7 Prototype Image Classification Model

After the image is uploaded and indexed by the server, it is passed into a convolutional neural network that we trained to detect crosswalks in images captured at street level. This crosswalk detection model was based on a pre-trained MobileNetV2 [19] model that is available in the Keras API [20] and was originally trained on the ImageNet Large Scale Visual Recognition Challenge dataset [21] (an object detection dataset containing 1.2 million images belonging to 1000 classes). We utilized transfer learning in order to produce a model applicable to our classification task. The process of transfer learning involves taking a pre-trained model designed to recognize features in a specific domain and using it as the foundation to train a new model that can learn more specific features of another dataset in a new domain [22]. To achieve this, we removed the last layer of the network and replaced it with a new softmax output layer (with two nodes) for performing binary prediction (presence or absence of a crosswalk). After defining the model architecture, we trained the model on a subset of a publicly available street-level image dataset which contains labelled crosswalks [23]. All images used with the model were resized to 224x224 pixels using the resize function of python's cv2 module. In order to integrate our prototype classification model into the system and demonstrate some of the future capabilities of the complete system, we manually

gathered and classified a small dataset of several street images using the application. These images, as seen in Fig. 3-2, were captured in normal lighting conditions during the day on the Hattiesburg campus of the University of Southern Mississippi.

Figure 3-2 Images classified using the trained model



filename	Confidence = [neg pos]
'streetview_crosswalk_samplelmages/campusCrosswalk/3_26_19/10.jpg'	[0.00323599 0.99676394]
'streetview_crosswalk_samplelmages/campusCrosswalk/3_26_19/29.jpg'	[0.935935 0.0640649]
'streetview_crosswalk_samplelmages/campusCrosswalk/3_26_19/24.jpg'	[0.10402941 0.8959706]
'streetview_crosswalk_samplelmages/campusCrosswalk/3_26_19/19.jpg'	[0.00924422 0.99075574]
'streetview_crosswalk_samplelmages/campusCrosswalk/3_26_19/17.jpg'	[0.46300498 0.53699505]
'streetview_crosswalk_samplelmages/campusCrosswalk/3_26_19/3.jpg'	[0.8355405 0.16445959]
'streetview_crosswalk_samplelmages/campusCrosswalk/3_26_19/12.jpg'	[0.0055032 0.99449676]
'streetview_crosswalk_samplelmages/campusCrosswalk/3_26_19/26.jpg'	[0.4914022 0.5085978]
'streetview_crosswalk_samplelmages/campusCrosswalk/3_26_19/44.jpg'	[0.27094656 0.7290534]

Chapter 4: Results

On the Android client, the user is able to take a picture using the user interface, and the resulting image automatically undergoes some simple preprocessing, including rotating and cropping the image if necessary. Once the preprocessing is complete, the image and the device's location data are sent to the server via a multipart/form-data POST request to one of the project's API endpoints. When the server receives the request, it saves the image data and creates a record in a MySQL database with the file path of the image, the upload time, and the GPS coordinates corresponding to the location where the image was taken. On the server, the image is then classified using the trained model, and the label ("crosswalk" or "no_crosswalk") is stored in the image's database record. These database records can then be queried later, or perhaps undergo other data analysis. Screenshots of this process are seen in Fig. 4-1 and 4-2.

Figure 4-1 Screenshots of using the app to take a picture and upload it to the server

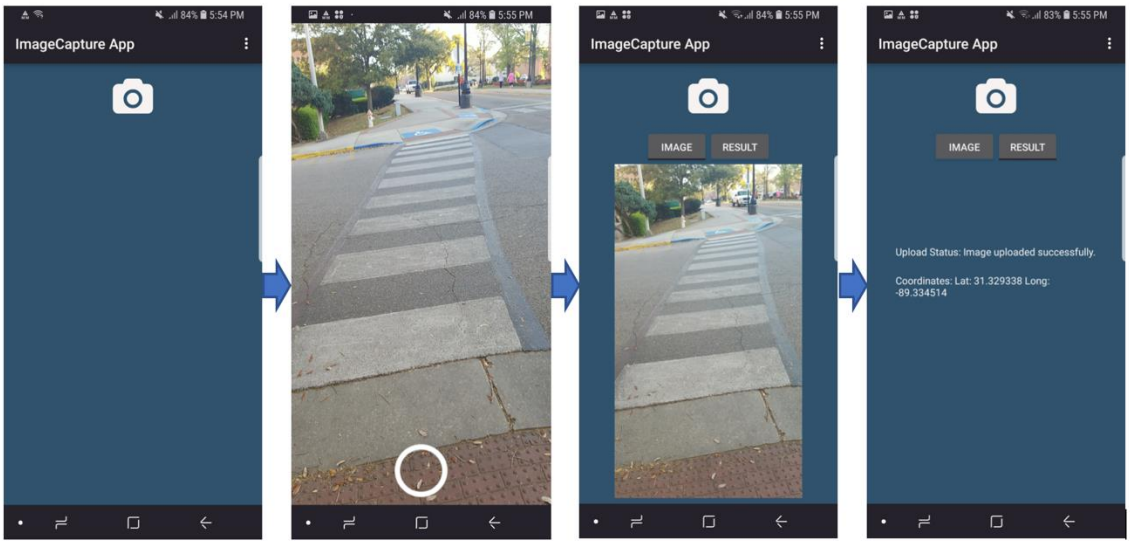


Figure 4-2 Record in the MySQL database for the uploaded image data

```
mysql> SELECT id, location_lat, location_long, image_path, upload_time, classification FROM image_data;
+----+--------------+---------------+-----------------------------------+---------------------+----------------+
| id | location_lat | location_long | image_path                        | upload_time         | classification |
+----+--------------+---------------+-----------------------------------+---------------------+----------------+
| 20 |      31.3301 |      -89.3346 | image_2019-01-22_123915_101613169.jpg | 2019-01-22 12:39:15 | crosswalk      |
| 40 |       31.328 |      -89.3327 | image_2019-03-27_125855_101613169.jpg | 2019-03-27 12:58:55 | crosswalk      |
| 41 |      31.3282 |      -89.3324 | image_2019-03-27_125910_101613169.jpg | 2019-03-27 12:59:10 | no_crosswalk   |
| 42 |      31.3293 |      -89.3345 | image_2019-03-27_175523_101687227.jpg | 2019-03-27 17:55:23 | crosswalk      |
+----+--------------+---------------+-----------------------------------+---------------------+----------------+
```

Chapter 5: Discussion

5.1 Problems and Considerations

The mobile application developed in this project aims to provide a means of crowdsourced acquisition of street-view images for mapping pedestrian facilities. Within the time frame allotted to development, we were able to develop a crude prototype for the primary features that should be present in the mobile client. However, there were a number of problems that arose during the lifecycle of the project. For example, we were originally developing two native mobile applications: one for Android, and one for iOS. This approach was initially chosen primarily because the features that the app is required to have, such as direct, raw access to the device's camera hardware, are easier to implement and more flexible in native development [8], [9]. However, developing separate native applications was very time-consuming. Therefore, because of the time constraints of the project, we chose to continue only with the development of the Android application for the prototype stage.

5.2 Future Revisions

Through the occurrence of these problems during the lifecycle of this project, we were able to gain some insight for the future phases of this project. While we initially thought that it would be better to develop two separate native mobile applications (Android and iOS) because of the requisite features in the applications, the development of this application led to the conclusion that it may be more efficient to use a cross-platform development solution, such as one of the web development

solutions (e.g., React Native) [10], and just use its ability to inject native code for Android and iOS into specific parts of the application while writing all other aspects using cross-platform development. While this approach would be more complex, it would more efficient in the long-term.

Another potential revision to consider for a later phase of the project is to migrate the back end from vanilla PHP to a back-end framework like Laravel [24]. In the development of the back end, the built-in features of PHP were used to handle requests, routing, and database access. While this was sufficient for the first phase of this project, it is not an easily scalable solution. Therefore, Laravel would be a more appropriate solution for scaling the back end, as it handles requests, routing, and database integration under the hood. This simplifies the project structure, as well as database migrations and API development [24].

5.3 Potential Use Cases

There are several revisions that could be made to this prototype in the future to simplify development or make the system more robust, but the basic goals of the project were mostly reached. We think that this application could have two important use cases for its parent project. Its primary use case would be to allow volunteers to help with acquisition of street-view images of pedestrian facilities in urban areas. This could be an additional source of data for mapping these facilities. Another potential use case would be to allow users to voluntarily check the location where a given pedestrian facility should be and validate the accuracy of this information by taking a picture of the facility or lack thereof. Either of these use cases for the

application would be useful to increasing accuracy and coverage of mapped pedestrian facility data in urban areas.

Chapter 6: Conclusion

In this project, we developed a mobile application for crowdsourced acquisition of street-view pedestrian facility data, which could be used to validate the accuracy of existing data or provide a source of supplemental data. In the implementation of the application, we used full-stack software development techniques, including the configuration of an HTTP server, the creation and administration of a database, the development of a REST API for interfacing the database on the server, and the design and implementation of a user interface in an Android application. This application allows users to take pictures at street level, after which the image and metadata are uploaded to a server, where it is added to a database and classified using machine learning to determine the presence of crosswalks in the image. Some future improvements and revisions would be to develop the application for iOS as well as Android, and to rewrite the back end to be more robust and scalable. With future development and enhancements, and paired with the automated system for the collection of pedestrian facility data as described in Chapter 1, we hope that these tools will give state departments of transportation a more efficient way of building well-connected pedestrian facility networks, which would decrease the likelihood of pedestrian-involved traffic accidents in urban areas by giving pedestrians safer, faster routes throughout these areas.

Chapter 7: Literature Cited

[1] National Center for Statistics and Analysis, “Pedestrians: 2013 data. (Traffic Safety Facts. Report No. DOT HS 812 124),” Traffic Safety Facts, Feb. 2015.

[2] Y. Zhang, J. Bigham, D. Ragland, and X. Chen, “Investigating the associations between road network structure and non-motorist accidents,” Journal of Transport Geography, vol. 42, pp. 34–47, 2015.

[3] T. A. Petritsch, C. B. Fellerhoff, J. P. Kubicki, and T. Scorsone, “Non-Motorized Facility Data Collection for Large Networks: Methods, Accuracy, and Application,” Transportation Research Board 94th Annual Meeting, 2015.

[4] F. R. Proulx, Y. Zhang, and O. Grembek, “Database for Active Transportation Infrastructure and Volume,” Transportation Research Record: Journal of the Transportation Research Board, vol. 2527, pp. 99–106, 2015.

[5] New Castle County, Delaware, and Cecil County, Maryland (WILMAPCO). “Creating a pedestrian facility inventory”.

[6] Mid-Ohio Regional Planning Commission (MORPC). “Mapping sidewalks in the Great Columbus Region”.

[7] International Data Corporation, "Worldwide Smartphone OS Market Share," 2017.

[8] H. Heitkötter, S. Hanschke, and T. A. Majchrzak, "Evaluating Cross-Platform Development Approaches for Mobile Applications," in International Conference on Web Information Systems and Technologies, Springer, Berlin, Heidelberg, 2012.

[9] A. I. Wasserman, "Software engineering issues for mobile application development," in FSE/SDP workshop on Future of software engineering research, ACM, 2010.

[10] M. Emiliano, et al., "Mobile apps development: A framework for technology decision making," in International Conference on Mobile Computing, Applications, and Services, Springer, Berlin, Heidelberg, 2012.

[11] J. Schmidhuber, "Deep learning in neural networks: An overview," Neural networks, vol. 61, pp. 85-117, 2015.

[12] M. Abadi, P. Barham, J. Chen, et al., "TensorFlow: A System for Large-Scale," in 12th USENIX Symposium on Operating Systems Design, Savanna, GA, USA, 2016.

[13] A. Zainab, "Realtime Object Detection on Android using Tensorflow," in Qatar Foundation Annual Research Conference, Qatar, 2018.

[14] Apple, Inc., "Xcode IDE," 2018. [Online]. Available: https://developer.apple.com/xcode/. [Accessed 25-May-2018].

[15] "Android Developer Site," [Online]. Available: https://developer.android.com/. [Accessed 27 April 2018].

[16] “Introducing JSON,” JSON. [Online]. [Accessed: 06-Mar-2019].

[17] S. Jauker, “10 Best Practices for Better RESTful API,” Thinking Mobile, 06-May-2014. [Online]. [Accessed: 02-Dec-2018].

[18] “HTTPClient Documentation,” HttpClient (Apache HttpClient 4.5.7 API), 24-Jan-2019. [Online]. [Accessed: 18-Mar-2019].

[19] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks,” ArXiv180104381 Cs, Jan. 2018.

[20] P. W. D. Charles, “Keras,” 2013. [Online]. Available: https://github.com/charlespwd/project-title. [Accessed: 20-Mar-2019].

[21] O. Russakovsky et al., “Imagenet large scale visual recognition challenge,” Int. J. Comput. Vis., vol. 115, no. 3, pp. 211–252, 2015.

[22] S. J. Pan and Q. Yang, “A survey on transfer learning,” IEEE Trans. Knowl. Data Eng., vol. 22, no. 10, pp. 1345–1359, 2010.

[23] R. F. Berriel, F. S. Rossi, A. F. de Souza, and T. Oliveira-Santos, “Automatic large-scale data acquisition via crowdsourcing for crosswalk classification: A deep learning approach,” Comput. Graph., vol. 68, pp. 32–42, 2017.

[24] T. Otwell, “Laravel - The PHP Framework for Web Artisans,” Laravel. [Online]. Available: https://laravel.com/. [Accessed: 20-Mar-2019].