



# **ASHESI UNIVERSITY COLLEGE**

**PROJECT EFUA:**

**A USER TRAINED, MOBILE-FIRST IMAGE CLASSIFICATION AI**

**APPLIED PROJECT**

B.Sc. Computer Science

**Ariel Arman Woode**

**2019**

**ASHESI UNIVERSITY COLLEGE**

**PROJECT EFUA:**

**A USER TRAINED, MOBILE-FIRST IMAGE CLASSIFICATION AI**

**APPLIED PROJECT**

Applied Project submitted to the Department of Computer Science, Ashesi

University College in partial fulfilment of the requirements for the award of

Bachelor of Science degree in Computer Science.

**Ariel Arman Woode**

**2019**

## **DECLARATION**

I hereby declare that this Applied Project is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature:

.....

Candidate's Name:

.....

Date:

.....

I hereby declare that preparation and presentation of this Applied Project were supervised in accordance with the guidelines on supervision of Applied Project laid down by Ashesi University.

Supervisor's Signature:

.....

Supervisor's Name:

.....

Date:

.....

## **Acknowledgements**

To my parents for their dedicated belief in me.

To my friends without whom I certainly would have forgotten most deadlines.

To my supervisors (previous and current) for their guidance.

And to my best friend Maame Efua Boham after whom this project was named.

## **Abstract**

Machine learning and the development of Artificial Intelligence (AI) has grown nearly exponentially over the past few years. However, growing fears over the nature of AI and the use of user data by large companies has put an air of distrust over the ML community. This makes it hard to collect more data with better user representation is needed to train more useful and accurate models and creates a unique problem space that my project seeks to tackle. It raises the question; how might researchers improve public understanding of AI while creating more representative datasets? To this question I propose the solution, Project Efua; a user taught mobile AI application meant to bridge the gap between the developer community and the wider public.

## CONTENTS

|   |                                     |
|---|-------------------------------------|
| CHAPTER 1 : INTRODUCTION .....                    | 1                                   |
| 1.1 INTRODUCTION .....                            | 1                                   |
| 1.2 PROBLEM STATEMENT .....                       | 1                                   |
| 1.3 SYSTEM DISCRIPTION .....                      | 2                                   |
| 1.4 BENEFIT OF SYSTEM .....                       | 3                                   |
| 1.5 MOTIVATION .....                              | 4                                   |
| 1.6 RELATED WORK AND EXSISTING SOLUTIONS.....     | 6                                   |
| CHAPTER 2: REQUIREMENTS SPECIFICATION .....       | 10                                  |
| 2.1 PROJECT SCOPE .....                           | 10                                  |
| 2.2 TECHNICAL DESCRIPTION .....                   | 11                                  |
| 2.3 PROJECT OBJECTIVES .....                      | 12                                  |
| 2.4 PRODUCT PERSPECTIVE.....                      | 12                                  |
| 2.4.1 System Features .....                       | 12                                  |
| 2.4.2 System Users .....                          | 13                                  |
| 2.4.3 System Use Cases .....                      | 14                                  |
| 2.4.4 Use case and scenarios.....                 | <b>Error! Bookmark not defined.</b> |
| 2.4.5 Operating Environment.....                  | 16                                  |
| 2.4.6 Design and implementation constraints ..... | 17                                  |
| 2.5 SYSTEM FEATURES .....                         | 17                                  |

|   |                                     |
|---|-------------------------------------|
| 2.5.1 Requirements .....                    | 17                                  |
| 2.5.2 Additional Features .....             | 23                                  |
| 2.6 OTHER NON-FUNCTIONAL REQUIREMENTS ..... | 23                                  |
| 2.6.1 Usability: .....                      | 23                                  |
| 2.6.2 Reliability: .....                    | 24                                  |
| 2.6.3 Transparency: .....                   | 25                                  |
| CHAPTER 3: ARCHITECTURE AND DESIGN .....    | 26                                  |
| 3.1 DESIGN SPECIFICATION .....              | 26                                  |
| 3.1.1 Architecture Description .....        | 26                                  |
| 3.1.2 Web Service .....                     | 26                                  |
| 3.1.3 Model View Controller (MVC) .....     | 28                                  |
| 3.1.4 Layered View .....                    | 29                                  |
| 3.1.5 Architecture View .....               | <b>Error! Bookmark not defined.</b> |
| 3.1.6 Mobile app Activity Diagram .....     | 32                                  |
| CHAPTER 4: IMPLEMENTATION .....             | 33                                  |
| 4.1 IMPLEMENTATION .....                    | 33                                  |
| Languages .....                             | 33                                  |
| 4.1.1 Python .....                          | 33                                  |
| 4.1.2 JavaScript .....                      | 33                                  |
| 4.1.3 Java .....                            | 34                                  |

|   |                                     |
|---|-------------------------------------|
| 4.2 PLATFORM.....                           | <b>Error! Bookmark not defined.</b> |
| 4.3 Mobile Application.....                 | 35                                  |
| 4.3.1 Android Studio.....                   | 35                                  |
| 4.3.2 Tensorflow Lite.....                  | 35                                  |
| 4.4 Web Application.....                    | 36                                  |
| 4.4.1 Vue.js.....                           | 36                                  |
| 4.5 Neural Network .....                    | 36                                  |
| 4.5.1 Keras.....                            | 36                                  |
| 4.6 Web service and Database .....          | 36                                  |
| 4.6.1 Node.js.....                          | 36                                  |
| 4.6.2 Express.....                          | 37                                  |
| 4.6.3 Mongoose .....                        | 37                                  |
| 4.7 TECHNOLOGIES .....                      | 37                                  |
| 4.7.1 Mobile application .....              | 37                                  |
| 4.7.2 Android.hardware.Camera.....          | 38                                  |
| 4.7.3 Com.android.volley .....              | 39                                  |
| 4.7.4 Org.tensorflow.lite.Interpreter ..... | 39                                  |
| 4.7.5 Web service .....                     | 41                                  |
| 4.7.6 Python-shell.....                     | 41                                  |
| 4.7.7 Mongoose .....                        | 42                                  |



|   |    |
|---|----|
| 4.7.8 Python environment.....                   | 42 |
| 4.7.9 Pymongo .....                             | 43 |
| 4.7.10 TensorFlow 1.13.1 .....                  | 43 |
| CHAPTER 5: TESTING AND RESULTS .....            | 48 |
| 5.1 DEVELOPMENT TESTING .....                   | 48 |
| 5.1.1 Unit testing.....                         | 48 |
| 5.1.2 Component testing .....                   | 50 |
| 5.1.3 System testing .....                      | 51 |
| 5.2 USER TESTING .....                          | 54 |
| 5.2.1 Testing results .....                     | 55 |
| CHAPTER 6: CONCLUSION AND RECOMMENDATIONS ..... | 59 |
| 6.1 CONCLUSION .....                            | 59 |
| 6.2 LIMITATIONS.....                            | 61 |
| 6.2.1 Mobile application .....                  | 61 |
| 6.2.2 Webservice .....                          | 62 |
| 6.2.3 Python environment.....                   | 62 |
| 6.2.4 Further limitations.....                  | 63 |
| 6.3 RECOMMENDATIONS AND FUTURE WORK .....       | 64 |
| 6.3.1 Recommendations.....                      | 64 |
| 6.3.2 Future work.....                          | 65 |

|                  |    |
|------------------|----|
| REFERENCES ..... | 67 |
|------------------|----|

# **Chapter 1: Introduction**

## **1.1 Introduction**

Machine learning and the development of Artificial Intelligence (AI) has grown nearly exponentially over the past few years. Its potential often surpasses human capabilities, beating us at our hardest games [10] and interpreting medical data for diseases such as cancer [8] in patients with superhuman accuracy levels. However, growing fears over the nature of AI and the use of user data by large companies [4] has put an air of distrust over the ML community. And with examples such as the Facebook Cambridge Analytica scandal [5], it's not hard to see why this distrust of data collection exists. However, the fact remains that more data with better user representation [7] is needed to train more useful and accurate models. This creates a unique problem space that my project seeks to tackle. It raises the question, how might we as researchers apply the advances made in AI to improve public understanding of AI, while creating more representative datasets needed to better serve the public with AI? To this question I propose the solution, Project Efua; a user taught mobile AI application meant to bridge the gap between the developer community and the wider public.

## **1.2 Problem Statement**

Data representation is one of the biggest challenges plaguing the machine learning community. Although there is far more data available to researchers now than there has ever been, the problem remains that data is often not representative of the entire human population.

Take the ImageNet [14] data set for example. This dataset is one of the largest openly accessible image libraries with over 14 million different images available to researchers around the world. However, even with its massive size, the dataset still fails to represent most of the world's population with about 45% of the images coming from the USA alone [11], a country whose population makes up less than 5% of the global population [12]. While this over representation is cause for some concern, it does show that there is a huge deficit in data from other parts of the world the developing world, Africa in particular, thus highlighting a keen opportunity for data creation and collection.

However, there is still the looming distrust consumers have of big data and the nature of AI that stands in the way of collecting super diverse data on a large scale. This means that the machine learning and AI community needs a solution that bridges the gap between the public and its efforts to grow larger and more representative data sets. This is where the subject of this paper comes in; Project Efua.

### **1.3 System Description**

Project Efua is a mobile first image classification application. It collects user labelled data to use for training on a larger online Neural network. This network is used to update the on-device model that is then tested by the existing user base on real world images used as test data.

The system is divided into two main parts. The mobile application, which serves as the client-side research tool, allows users to collect and label new data via a dedicated camera in

the “Explore” feature. It also allows users to test newly trained models in the “Guess What?” feature, where users can quiz the model on real world data, answering correct or wrong to its predictions or “guesses” and labelling images that it got completely wrong.

The second part is the webservice that runs and maintains the system. Data from the “Explore” feature is sent to the service, where database management and model training are handled. Once the train logic for the system is met, training is triggered, thus updating the model's abilities. Newly trained models are made available to users on application start up, thus allowing them to test the improvements made on new real-world data via the “Guess What?” feature.

The system uses ideas of human-in-the-loop active learning and crowd sourced public contribution in its implementation. By so doing, it involves end users in the process of “teaching” an AI to classify or “see” new images as more users add to the systems database. This will be done in the hopes that users will gain an appreciation for the AI and the way it works, since they will become stakeholders in the development of the model, thus bridging the gap between technicians and the wider public.

#### **1.4 Benefit Of System**

Project Efua seeks to humanise AI by allowing users of this mobile first application to “teach” their own instance of an AI to recognise images by exposing it to new examples of images through its fun games and activities right on their phone. In doing this, users are brought

into the ML community and helping data scientists tap into the hugely diverse, crowd sourced database that is the human experience as seen through the cameras of our smart phones.

Its other purpose is to serve as a research tool for machine learning engineers. In this capacity, Project Efua will work to collect data from a diverse user base, giving researchers access to examples they otherwise would not be able to come by easily. It will also serve as a platform for researchers to test new user first AI applications.

A final expected benefit of this system is the nature of the data it collects and how it could affect the relevance of the model. Given that data is added intentionally by users, it is expected that the data added will be highly relevant to the context of the users use case. It is therefore a major hypothesis of this project that, given the users source test images from the same context, the model can be expected to perform better in the users given context than one trained on externally sourced data. As such, this project, through live user testing, will attempt to see how well the same model performs when exposed to real world data and standard data from external sources.

## **1.5 Motivation**

This project, while expected to have global impact, is mainly aimed at boosting the adoption of AI in Africa and the wider developing world. As such, its design and implementation will be made to intentionally reflect its unique African origin as this is the space in which this project seeks to make the most impact.

The developing world is lagging in the adoption of AI and this is poised to have real economic effects on these countries. Bughin, Jeongmin, Manyika, Chui and Joshi describe the growing digital divide in the world and how slow AI adoption may only make this worse for developing countries [2]. Africa especially is taking the hardest hit due to its poor adoption of AI and related technologies. This is not to say that there is no hope for the continent as Africa has some of the world's fastest growing smartphone and web penetration in the world.

According to IBM, 90% of all the worlds stored data was created within the last two years [6]. This is largely due to the surge in content creation by growing adoption of web technologies such as social media, online repositories and IoT (Internet of things) devices. A major contributing factor to this surge is a similar rise in the use of smartphones around the world. These devices present a ready window into the lives of the average user, collecting several gigabytes of data a day. In the developing world, smartphones often serve as the first point of contact to the internet for brand new adopters.

Another key truth is that the vast majority of smartphone users have already interacted with artificial intelligence in some form or the other. This is either through assistants such as google assistant and Siri, or even just simple Google searches powered by AI algorithms such as RankBrain [3]. What this means is that the average smartphone user is already familiar with the nature of AI as it is currently being implemented and this familiarity is key in winning the hearts and minds of the public.

This project seeks to build on these insights to work on a locally developed AI application on the African continent. Using technologies such as TensorFlow, TensorFlow Lite and android OS, this project hopes to make these goals a reality.

### **1.6 Related Work And Existing Solutions**

The technical problem this project seeks to solve is the issue of data collection with a focus on the human element of the system. This type of learning is most like active machine learning. This is where machine learning algorithms employ unique techniques to select specific unlabeled data points for labeling, based on how useful it believes the data will be to further unsupervised learning. The direct problem this seeks to tackle is that of a situation where unlabeled data is easy to come by but takes massive effort to label. Ranganathan, Venkateswara, Chakraborty and Panchanathan discuss a novel approach to this problem by introducing a novel deep active learning algorithm which is designed to exploit deep learning's ability to learn a discriminating set of features for a given task. [9] Their approach led to consistently higher results on accuracy for large unlabeled datasets. This is significant to this paper as it provides an ideal reference point for model development in a future iteration of the system where users will be prompted to provide labels for data the model calculates will be beneficial to the learning process.

This project is also like another technique employed in machine learning and that is human-in-the-loop learning. This method involves humans in “both the training and testing stages of building an algorithm, which creates a continuous feedback loop that allows the algorithm to produce better results each time.”[13] and is a combination between active and



supervised learning. This is an extremely useful technique for boosting the performance of a machine learning algorithm by having the humans validate low confidence predictions in a feedback loop. Gengo Ai [13] is a company that seeks to provide such solutions for machine learning, particularly NLP and machine translation problems. This idea is what is used for the “Guess What?” feature, where users validate the model’s predictions on real world data, which will then be used to reinforce the model's abilities in the future.

These implementations can be classified under crowd sourced machine learning. Abhigna, Nitasha and Shilpa outline the benefits of crowd sourcing machine learning. In their paper, they state, “Crowdsourcing platforms are rapidly replacing the traditional modes of experimenting in areas like sociology and psychology”[1]. This is an important consideration for this project as, beyond its technical goals, it seeks to improve the perception of AI and machine learning in the public eye. The paper further outlines three main contributions that can be made here. Namely, data production, debugging and assessment of models and developmental analysis of machine learning. Under each of these, it outlines the issues they face but also steps that have been taken to mitigate them and the overall benefits of all of them. The paper further goes on to state that, “Winning over the confidence and appreciation of the crowd workers is very important for researchers”. This falls in line with the social goal of this project. However, where the paper looks at crowd workers as employees of a company, this project seeks to broaden it out to more general users and members of the public, to boost their interest and introduce them to the workings of AI and machine learning through a publicly accessible project.

These implementations however rely heavily on data that was collected using traditional means, where there is little in the way of initial user input. This is where my project seeks to make a meaningful contribution. Using the ideas and principles of human-in-the-loop active learning, this project seeks to introduce the human element at every stage of the machine learning process.

The key contribution here is in the style of data collection and use. Rather than relying on externally collected data for users to label, this system takes a novel approach to the data production phase by relying on direct user input that is labeled at the time of creation. Abhigna et al outlined the need for greater incentive for crowd workers during the labeling process in their paper. Project Efua works around this by reducing the cognitive load on users since data is labeled at the time of input. This means that users will only be expected to add images that they are interested in, thus further reducing the need for external incentive. In this sense, Project Efua seeks to introduce and test a new approach to model training here called the Active Teaching approach.

Debugging and assessment is also done by the end user through the “Guess What?” feature, where users are given the updated model in near real time to use in order to appreciate its growth. The contribution made here is that users will have a greater sense of what the model can and cannot “see”, thus influencing the data they collect in subsequent “Explore” sessions. This is important as it means that with use, the model will get better at classifying images that users want it to classify correctly, thus improving its performance within the users’ context or use case.

Gaining insight into how users use the system as well is also another contribution under developmental analysis of machine learning. The idea of context, here defined as the space in which the system is used, and use case will be information that can be inferred from the system over time. This is an especially important contribution as the system will be intended for public use. This means that over time, the interaction between humans and machine learning can be monitored, allowing developers to track what users are interested in and test hypothesis on human computer interactions concerning machine learning. In this version of the, the system can be said to be heavily biased towards user interest, thus allowing us to study how user bias can affect the performance of a model and iteratively test new methods of controlling for user bias.

## **Chapter 2: Requirements Specification**

### **2.1 Project Scope**

Project Efua is a multi-system web service spanning over a user oriented mobile android application, a research, control and maintenance oriented Node.js web application and a python virtual environment set up for running the machine learning model. For the purposes of this academic study, Project Efua will be developed to Version one (V1) stage which is primarily designed to test the validity of the system structure and serve as a developer prototype that can later be split into the full public and research-oriented client versions. This project assumes development is geared towards the research version of the application and thus will have the following components;

- An android data collection and model testing mobile application tool.
- A system management node.js web application.
- A research-oriented model observation node.js web application tool.
- A mongoDB database.
- An Image classification Convolutional Neural Network model.
- A python virtual environment for running the model and its supporting files.

The system in its current form is strictly designed for test and prototyping purposes to evaluate the active teaching method of data collection and model development. As such, this system:

- Is not a user interaction optimized.
- Does not have a user experience component.

- Does not have user specific profiles.
- Does not allow for dataset requests or downloads.
- Does not have a production ready data validation system.
- Is not security optimized.

The system will also only be deployed on the Ashesi University campus for initial user testing and so will be highly localized to this context.

## **2.2 Technical Description**

From a technical standpoint, Project Efua is thus a mobile first image classification application that collects user labeled data to use for training on a larger online Neural network and tests updated models through an interactive guessing game.

Project Efua works by using a client android application to take pictures of new user examples. These pictures are sent to a mongoDB via a web application API. The web application manages the photo collection, model training trigger, updating and observation. It uses a python virtual environment to run model related processes such as picking new data from the database and sorting it for training. The model uses a TensorFlow backend to run a Convolutional Neural Network trained on the user added data. This model is sent back to the database to be stored for sending to the mobile application upon update request. The mobile application uses this model in an interactive guessing game where the user takes a picture of a new item and answers either correct or wrong to the models top 5 predictions of the photo's

label. This photo and the data are then added to the database and used to retrain the model in its next training cycle.

## **2.3 Project Objectives**

This project will seek to accomplish the following objectives.

- Build a research mobile application to teach an AI image classification by exposing it to real world examples, tagged by users of this application.
- Use server-side technology to collect, train and update the applications classification abilities.
- Build a proof of concept system based on image classification to show the active teaching technique is feasible.
- Use the application for open, consensual data collection in a local setting.

## **2.4 Product Perspective**

### **2.4.1 SYSTEM FEATURES**

This project focuses on four main features to be developed and tested.

- Data collection via the “Explore” function.
- System management via a web application.
- Model training and observation.

- Model testing via the “Guess What?” game.

These features form the core of Project Efua’s functionalities. As such, they are the main priority of this project to ensure that the system is feasible and can be used for more nuanced research.

#### 2.4.2 SYSTEM USERS

The system is intended for two main user groups; public users and technical users.

##### 2.4.2.1 PUBLIC USERS

These are people who are not directly involved in machine learning or AI research but have a curious interest in it. Within the scope of this project, these users are considered the early adopters who will test out the system functionalities and give user feedback into its feasibility. These users will primarily use the mobile application for data collection and model field testing.

##### 2.4.2.2 TECHNICAL USERS

These are people who are not actively involved in machine learning or AI research and have an interest in researching the development of an actively user taught AI. Within the scope of this project, these are students and lecturers at Ashesi University who are interested in the performance of the model and familiar with the underlying technologies. These users will primarily use the web application for model observation and the mobile application if they choose to enroll in the field-testing phase of this project.

### 2.4.3 SYSTEM USE CASES

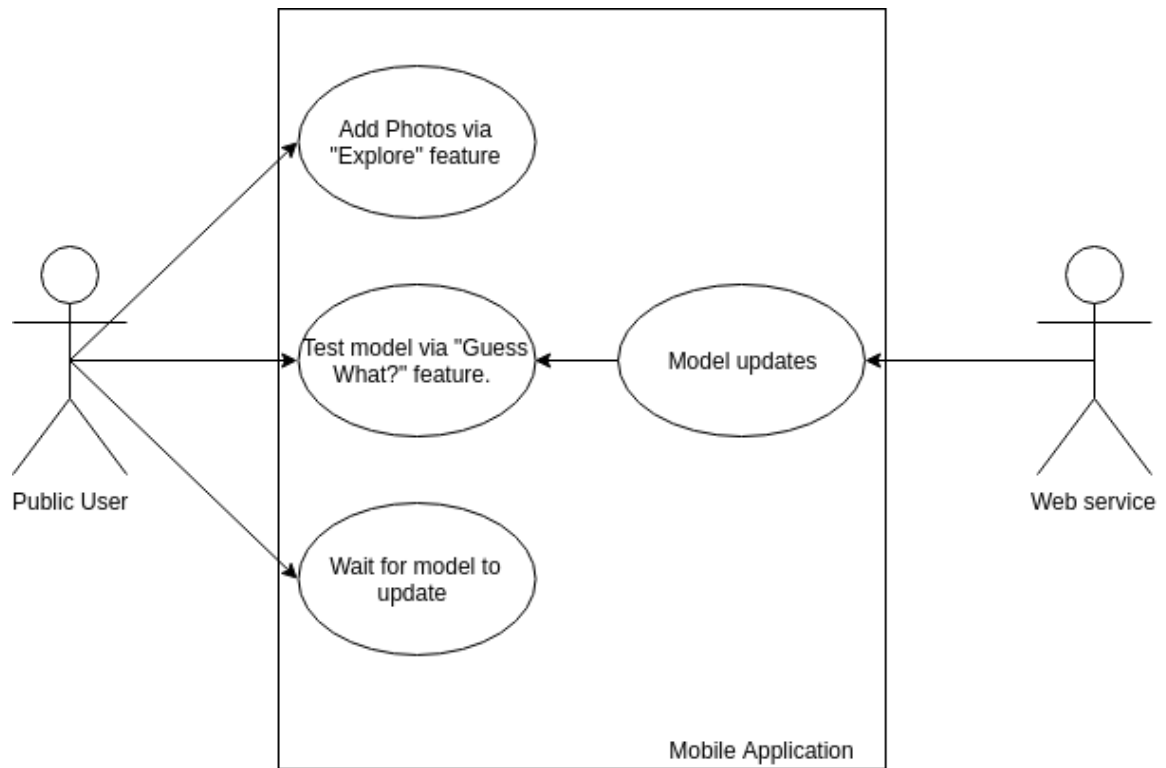
#### 2.4.3.1 PUBLIC USERS

##### Use case scenario:

A curious individual wishes to explore the capabilities of AI and contribute to an open source data collection project. This user will download the app on their Android device and proceed to interact with its features. The user, wishing to “teach” the AI to recognize new objects, engages the “Explore” feature of the application. This opens the dedicated camera view and allows the user to send labeled images to the online server. Wishing to test the accuracy of the model in an interactive way, the user engages the “Guess What?” feature. This opens another dedicated camera view. The user takes a picture of an intended subject, “asking” the AI to guess what is in the image. This system returns the top 5 predictions, querying the user for a correct or wrong response to its “guess”. The user answers either correct or wrong to the responses. The user's response, image label and the subject image are sent back to the server and added to the image database for reinforcement training. After a successful retraining of the model, an updated version is sent out to the user base, allowing the user to test the new model and see how well it does.

##### Use case diagram:



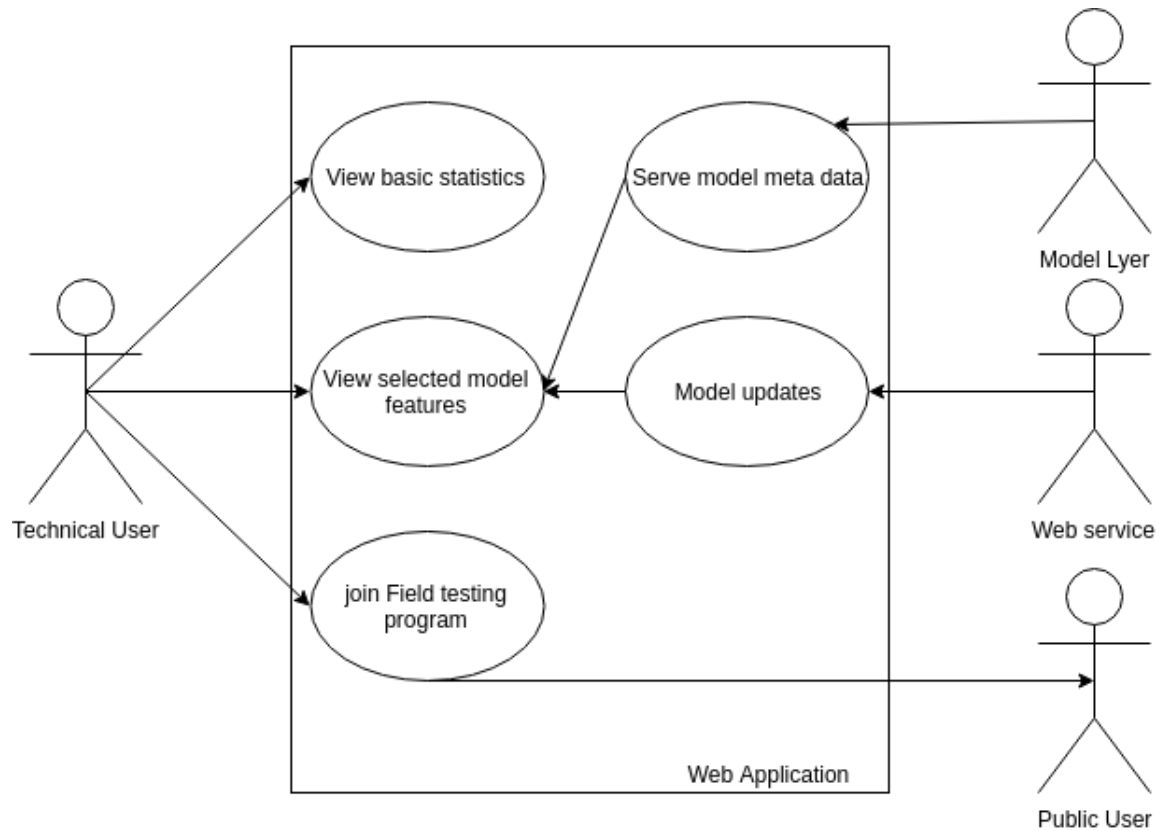


#### 2.4.3.2 TECHNICAL USERS

##### Use case scenario:

An interested researcher wishes to test the capabilities of the AI and contribute to its open source development and data collection. The user will access the web application to view statistical details on the model's performance such as number of active users, validation accuracy over time, number of new labels added each time, time to next scheduled training session, etc. The user can then choose to select a model and view its details using tensorboard. The user can also choose to join the field-testing program and download the application, at which point they use it as a public user.

Use case diagram:



#### 2.4.4 OPERATING ENVIRONMENT

The system will be built off of the Node.js Server environment in JavaScript for the web service. The database will be managed with mongoDB and Vue.js, JavaScript and CSS will be used for the web application view. Python virtual environment running a TensorFlow backend for the CNN and model training. The system will also rely on tensorboard for model observation. Android SDK will be used to build the mobile application to be run on Android OS enabled devices.

#### 2.4.5 DESIGN AND IMPLEMENTATION CONSTRAINTS

Given that the system will be run on local servers for the scope of this project, model training will be limited by the availability of resources on these servers. This will therefore affect training time and the design choices for training cycles. This will also mean that TensorFlow cannot be hardware optimized or allowed to run on GPU's if resources are not available. Space is also a major constraint to consider since data consumption may be several hundred gigabytes if the system use is allowed to grow exponentially.

### 2.5 System Features

#### 2.5.1 REQUIREMENTS

##### 2.5.1.1 DATA COLLECTION VIA THE “EXPLORE” FUNCTION

This feature allows users to photograph new subjects and label them for use in later training. This is the main point of data input for the system.

#### *Request/ response sequence*

1. The feature is accessed via a button on the app home page labeled “Explore”.
2. The explore page is launched, allowing users to interact with the in built camera.
3. The user enters a new label, points the camera at a desired subject and click on the capture button.
4. This triggers the capture sequence and allows the application to take a picture.
5. This picture is stored on the device in a local folder using the label and date as its name.

6. The picture is also sent over the network in a string post request to the web service using a dedicated API along with its name and label.
7. The camera view is rest and readied for the next capture sequence.
8. A “Photo Saved!” response is sent by the API if saving was successful, or an error message if not.

#### *User requirements*

- The user must enter a label before image capture.
- The user must ensure network access.
- The user needs the system to give feedback on task completion.

#### *System requirements*

- The system must send data in a reliable manner.
- The system must provide user feedback after every request.

#### *Input*

- This feature takes as input images taken by the user.
- This feature takes as input the labels of images taken by the user.

#### *Output*

- Response notifications on task completion or error.
- A photo sent to the database via web service API.
- A photo stored on device in proprietary folder.

#### 2.5.1.2 SYSTEM MANAGEMENT VIA A WEB APPLICATION:

This feature is a collection of features with a common theme of managing the communications, data transfers and reliability of the system. It mainly concerns the use of API's on the webservice and how they interact with other parts of the system.

##### *Request/ response sequence*

1. The system uses REST API's to handle data input and storage.
2. The system uses REST API's to handle data requests by the web application.
3. The system uses REST API's to handle model training and updating requests.

##### *User requirements*

- The user needs the system to reliably maintain client functionalities.
- The user needs the system to send request responses in a timely manner.
- The user needs the system to be easily accessible within the local context.

##### *System requirements*

- The system needs API functions to not be computationally intensive to improve responsiveness.
- The system needs the webservice to simplify connections between higher and lower levels of the system architecture for the user.

##### *Input*

- This feature takes as input the requests from various clients on the webservice.

#### *Output*

- Expected application responses.

#### 2.5.1.3 MODEL TRAINING AND OBSERVATION

This feature is a low-level feature of the system that works off the python virtual environment bundles with the system. This handles the creation of new model using a TensorFlow backend and is supported by other python programs using the same environment.

#### *Request/ response sequence*

1. The webservice makes a request to the python training layer once the model training logic has been met by the surrounding system.
2. A database call is made by a python database handler utility program.
3. This program sorts the data into relevant folders for use during training.
4. Once complete, the training application is run using the newly added data.
5. Upon completion, the application returns relevant model files to the webservice for application updates and model observations.

#### *User requirements*

- The user needs the model training to occur at predictable intervals.
- The user needs the system to make model meta data available for observations.
- The user needs the system to show progress of model performance.

### *System requirements*

- The system requires the training layer to not interfere with higher level functionalities.

### *Input*

- This feature takes as input the new images from the database.

### *Output*

- A trained image classification model.
- Model metadata.

#### 2.5.1.4 MODEL TESTING VIA THE “GUESS WHAT?” GAME.

This feature is the on-device serving of the trained model. It allows users to test the model on real life data through a dedicated camera.

### *Request/ response sequence*

1. The feature is accessed via a button on the app home page labeled “Guess What?”.
2. The guess what page is launched, allowing users to interact with the in built camera.
3. The user takes a picture of the subject they wish to test the model on.
4. The application returns the top 5 predictions in order, waiting for the user's response to each.
5. The user clicks either the correct or the wrong buttons in response to the model's predictions.

6. If the correct label is not found in the application prompts the user to input the correct label of the image.
7. The users answer is then sent back to the database via the webservice along with the test image and label predictions for reinforcement learning.

#### *User requirements*

- The user needs the application to perform rapid predictions so as not to slow down the testing processes.
- The user needs the application to reliably guess the labels of images with an above random accuracy.
- The user needs the system to give feedback on task completion.

#### *System requirements*

- The system must send data in a reliable manner.
- The system must provide user feedback after every request.

#### *Input*

- This feature takes as input images taken by the user.
- This feature takes as input the labels of images taken by the user and their correct or wrong answer values.

#### *Output*

- Response notifications on task completion or error.



- A photo sent to the database via web service API.
- A photo stored on device in proprietary folder.

#### 2.5.2 ADDITIONAL FEATURES

### **2.6 Other Non-Functional Requirements**

#### 2.6.1 USABILITY:

Given that the system is designed for use by humans, it is imperative that users understand how to use this system and its abilities.

##### 2.6.1.1 SIMPLE INTUITIVE INTERFACE

The system must have a simple, easy to use interface that minimizes the complexity and number of actions required to accomplish a task.

##### 2.6.1.2 COMPELLING GAME MODES

The system must have compelling game modes in order to draw users in to wanting to train and test the model, therefore driving the research.

##### 2.6.1.3 IMPROVING MODELS

The system must show evidence of an improving model. This is to ensure that users see a tangible difference in how well the system is performing the more they use it.

#### 2.6.1.4 HIGHLY DIVERSE, STANDARDIZED DATA SETS

The system must collect diverse data from its user group in a standard format to make it useful for later researchers.

#### 2.6.2 RELIABILITY:

The system must behave in a way that minimizes error and in the event of an error, keeps system down time to an absolute minimum. This is to ensure that users can rely on the system.

##### 2.6.2.1 LIGHT WEIGHT APPLICATION (RESOURCES NON-INTENSIVE)

The mobile application must be small and resource non-intensive to make sure it can be run on the largest number of devices. It must also do this to reduce the number of errors users may get from using the application.

##### 2.6.2.2 RAPID CLASSIFICATIONS

The model run on the mobile application must be small enough to allow for rapid classification, minimizing the amount of time the user must wait for the model to make a prediction.

##### 2.6.2.3 TIMELY UPDATES

The system must push out updates to the user as quickly as possible to allow users the chance to sense the model grow the more they use the application.

## MODEL CONFIDENCE ASSURANCE

The system must show that it is improving over time and allow users to easily see its progress whenever they wish.

### 2.6.3 TRANSPARENCY:

Given that this system is intended for intentional data collection, it is imperative that all parties involved are aware of the data being collected and how it is used.

#### 2.6.3.1 PRIVACY CONSCIOUS

The system must do well to make all data collected anonymous.

#### 2.6.3.2 READY USER BASE

The system must guarantee a ready user base for future research.

## **Chapter 3: Architecture And Design**

### **3.1 Design Specification**

The focus of this system is the mobile client and how it can be used in field testing. As such, the system is designed to support the application to the best of its abilities, given that data collection and model testing happen through it. This was the major design condition that was considered for the creation of the architecture.

#### **3.1.1 ARCHITECTURE DESCRIPTION**

##### **3.1.2 WEB SERVICE**

The system uses a web service architecture to manage components of the system. The justification for this were twofold. The first reason was that TensorFlow lite has yet to add features that allow for easy on device training. This is important as it means that devices cannot run the train and update sequence on their devices which would be the only other way to handle this user taught AI sequence. This comes as an advantage however as it means client devices don't have to spend resources and time training locally, which, for a mobile device, would take too long to complete to make the system worth its expenditure. It also means that the application can be kept light weight, allowing it to be used on multiple devices, regardless of hardware limitations that would otherwise hinder on device training.

The second reason is that a web service structure enforces uniformity. Uniformity is big advantage of this system as it stands. It means that client applications can all add to the same database which will be used to train the same model. It also means that as more people use the

application, the model used for prediction will improve relative to the data added by all users, therefore improving performance for all other users, and growing the model's overall performance far faster than if individually deployed and trained for each user. This architecture also makes it easier to keep track of the model's performance via the web application since its only one model that gets trained and retrained.

#### *3.1.2.1 WEB SERVICE ARCHITECTURE*

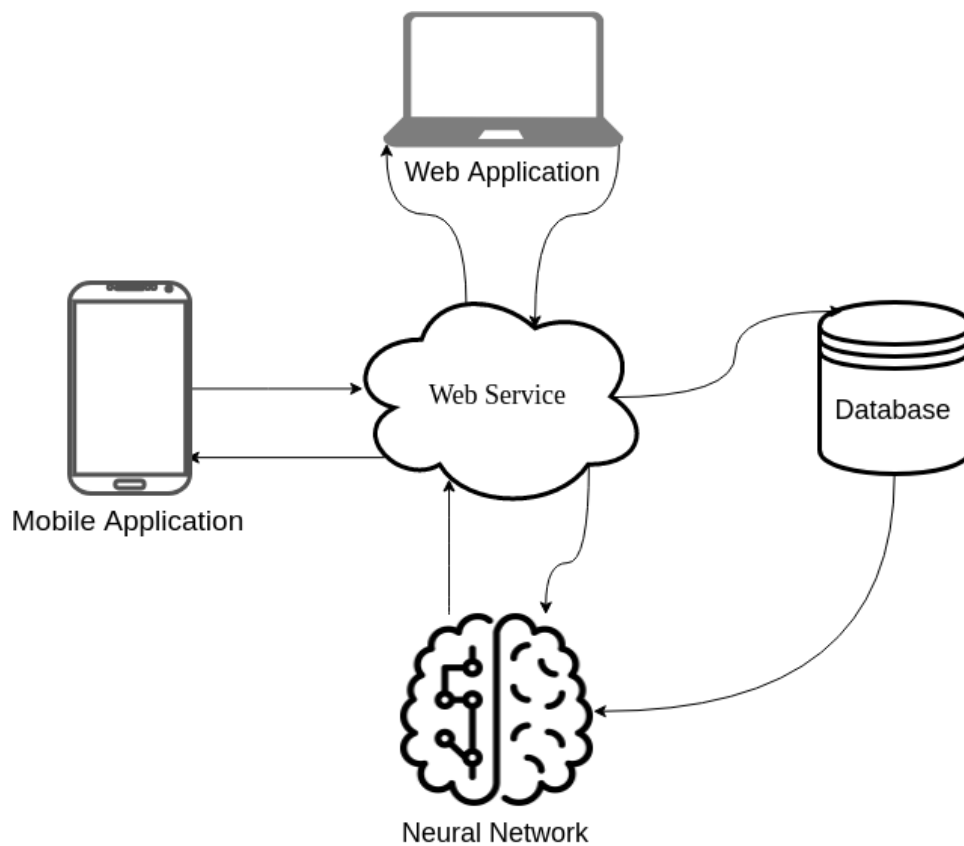


Figure 3.1

### Web Service Architecture Components

| Function           | Mobile Application | Web Application | Web Service  | Database | Neural network             |
|--------------------|--------------------|-----------------|--------------|----------|----------------------------|
| Environment        | Android SDK        | Vue.js          | Node.js      | MongoDB  | Python & TensorFlow 1.13.1 |
| Model handling     | TFLite             | Tensorboard     | Python-Shell | -        | Keras & TFLiteConverter    |
| Data communication | Volley & Retrofit2 | Vue-router      | Mongoose     | -        | Pymongo                    |

Table 3.1

#### 3.1.3 MODEL VIEW CONTROLLER (MVC)

For the observation web application, an MVC architecture was chosen. This was to allow for separate development and improvement of each component of the system. Given that this system is being deployed as a research platform, it is expected that frequent changes will be made to its functionalities over time. An MVC architecture makes it easier to work on the web application's components separately without affecting the functionalities of its other components. Also, since it will be built on top of the web service, the structure reduces the likelihood of critical system failure should something go wrong during further development of the webservice.

### 3.1.4 LAYERED VIEW

For simplicity sake, the system can also be viewed as having a layered architecture. This is because of the way each component communicates with the other, abstracts functionalities and transfers data. The four main layers are:

#### 3.1.4.1 THE PRESENTATION/CLIENT LAYER

This layer comprises of the mobile and web applications. This layer is the layer that interfaces with the users of the system, allowing them to add new data, test the model and observe its performance. Little to no heavy computations are done in this layer. This is a design choice made to simplify the users experience and reduce load on client devices, therefore speeding up system interactions.

#### 3.1.4.2 THE WEBSERVICE LAYER

This layer is the main communications layer of the system, handling all aspects of the system logic. This layer handles the calls to the two lower layers and manages the logic involved in their use. However, this layer only handles system level logic and does not do any major computations in order not to over work the server (a consideration made as a result of one of Node.js' weaknesses to be discussed in a later section). In this sense, it two major functions are serving data between the presentation layer and the database and triggering the train cycle on the lower levels.

#### 3.1.4.3 THE DATABASE LAYER

This layer manages all data storage and management. This is considered a separate layer as it can be independently accessed and has direct connections to the model train layer that are not managed by the webservice layer. This was a design decision made primarily to reduce the load on the webservice and simplify the complexity of training. As such, the system has a separate database handler working on the lower layer to ensure that data is pulled and sorted in the best way possible for the model trainings use. It also increases the level of abstraction and makes the system scalable since in a complete system, the model training and database will be hosted together on larger servers.

#### 3.1.4.4 THE MODEL TRAINING LAYER

This layer handles the heavy lifting and computations involved in training the model. Running off a python 3 virtual environment with a tensorflow back end, it runs separately from the rest of the system, only communicating with the database for data consumption and the webservice for model serving. Even though it is triggered by a web service API, this layer does not run any of its functions on the webservice in order to reduce the computational over head on the server.

#### 3.1.4.5 LAYER ARCHITECTURE



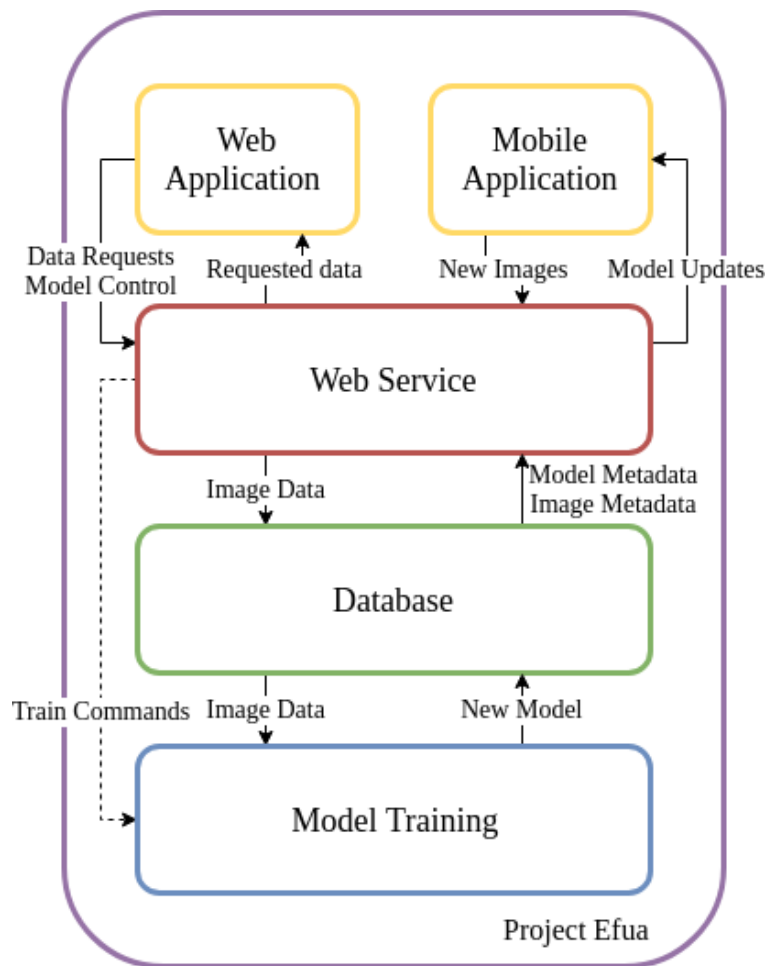


Figure 3.2

### 3.1.5 MOBILE APP ACTIVITY DIAGRAM

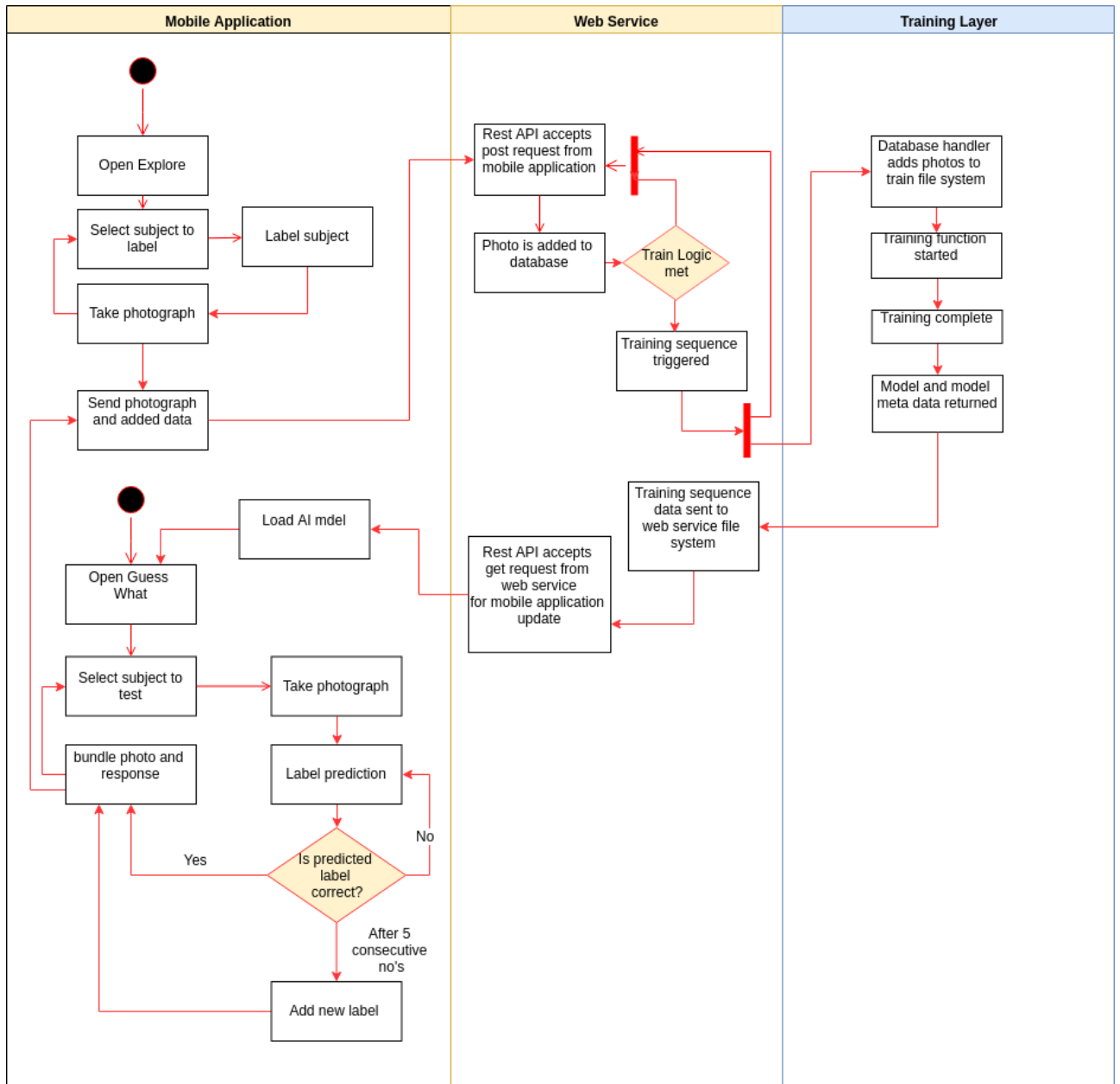


Figure 3.3

## Chapter 4: Implementation

### 4.1 Implementation

#### LANGUAGES

##### 4.1.1 PYTHON

Python was chosen as the base language for developing the Convolutional Neural Network. This was for its vast library of supporting packages for machine learning, most notably, TensorFlow and Keras. These two packages form the base of the model creation. Python was also chosen for its ease of use and strong ability for handling intense processing.

##### 4.1.2 JAVASCRIPT

JavaScript was chosen as the base language for the webservice. The decision for this came from the strength of the Node.js runtime environment especially when it came to server-side applications. The other candidate language selected for this purpose was PHP. However, JavaScript won out for a number of reasons. The main reasons were that

- Node.js, being its own runtime, doesn't rely on other servers to run, thus making it more light weight than PHP since it also has fewer dependencies.
- Node.js was also considered as running faster than PHP since it runs on a single process with a non-blocking I/O (Input/Output) events, a useful consideration for this project given its heavy use of I/O.

- Node.js also makes better use of CPU and memory resources since it doesn't use multi-threaded networking. This means that the webservice can run on more constrained servers, making it more practical for initial testing.
- Node.js also has better integration with noSQL databases such as mongoDB and made front end development simpler since a JavaScript framework would integrate better with it than with PHP.
- Node.js being written in JavaScript and served through NPM, also means that there is a vast library of packages to assist with development.

The biggest reason, however, was that Node.js is asynchronous where PHP is synchronous, making it better for Realtime data serving than PHP. This reduces the load on the server when hit with multiple requests, since it also runs on a single process. This is an important functionality for this project's requirements as it means that the server can handle far more clients at the same time without blocking the process, making it far more scalable than PHP. It also means that clients don't have to be held up by server-side logic while large processes are running, such as model training.

All these reasons made JavaScript a more suitable language for this project than PHP for server-side scripting.

#### 4.1.3 JAVA

Java was chosen as the language for the mobile application. This was because android OS was the most accessible mobile platform to develop for. This made it ideal for prototyping

the mobile application. It's also compatible with tensorflow's mobile package tensorflow lite. All of these reasons contributed to java being selected as the development language for the mobile application, along with the language's familiarity.

## **4.2 Mobile Application**

The platform chosen for the mobile application was android OS. This was because of its ease of access and how familiar java was. Android OS was also compatible with TensorFlow Lite making it the ideal platform to prototype with. Android OS also has the largest market share of mobile devices, meaning that in any given local, one is more likely to find Android OS users than any other mobile platform. This means that when it comes to testing out a prototype based on high usage, developing for android OS will yield the best results during testing. To develop the mobile application, the following tools were used:

### **4.3.1 ANDROID STUDIO**

Android studio is the native IDE for android development meaning it has the most optimised tools for development.

### **4.3.2 TENSORFLOW LITE**

TensorFlow lite is an Android package developed by Google under TensorFlow to bring machine learning model usage to android. As such, it was the simplest to use. It also meant easier communication with the online model which is written in Keras using a TensorFlow backend.

### **4.3 Web Application**

#### **4.4.1 VUE.JS**

Vue.js is a front-end web application framework written in JavaScript, HTML and CSS. It was selected for its simplicity and easy integration with a JavaScript backend.

### **4.4 Neural Network**

#### **4.5.1 KERAS**

Keras was selected as the leading package for model development for its relative ease of use when it comes to developing machine learning models, especially neural networks. Keras abstracts several steps in model development making it easier to prototype model architectures. Being based off TensorFlow however, it is still compatible with TensorFlow API's including tensorboard and TOCO which is used to turn TensorFlow models into TensorFlow lite models.

### **4.5 Web Service And Database**

#### **4.6.1 NODE.JS**

Node.js, as mentioned earlier, was chosen for its more ideal development environment and the way it handles data requests and serving. Its asynchronous nature made it ideal for this project and lightweight deployment fell in line with the requirements of this project. However, being a JavaScript runtime, it is ill advised that Node.js be used for CPU intensive applications. This major drawback means that a lot of the heavy processing must be done off the service, which influenced the design choices of this project's architecture, as was suggested in an earlier section. Besides this drawback though, Node.js is still ideal as the web service platform.

#### 4.6.2 EXPRESS

Express is a Node.js web service framework that simplifies the development of these services. Express was chosen for its simplicity and support. As one of the most popular frameworks, development in express is simple and easy to trouble shoot. Express also handles many boiler plate functions such as setting up routes, the project structure and dependencies, while still allowing the developer to control the structure of the project. In this sense, it is much lighter than PHP In Laravel, which strictly enforces the MVC model, but still more forgiving and better supported than pythons Flask which requires one to manage a lot of the maintenance and development by hand (hence why it wasn't in the top running for server side scripting). Express is easy to use but light weight enough to allow for unique system configurations.

#### 4.6.3 MONGOOSE

The chosen database was mongoDB. Mongoose therefore was the best ORM choice for this project.

### **4.6 Technologies**

Throughout development, many different technologies were used to support the systems creation. However, for each of the systems layers, a few notable technologies were instrumental to the systems development.

#### 4.7.1 MOBILE APPLICATION

In the mobile applications development, three technologies aided the most in the app's development.

#### 4.7.2 ANDROID.HARDWARE.CAMERA

The Android hardware camera package was the main package used to implement the ShowCamera.java class which handled the camera view of the application. This package allows in app access to the device's physical camera, allowing for a customized camera view and image capture handling. An inbuilt camera was proffered for this application (over an intent call as is common practice for most camera use cases) for two main reasons.

- The TensorFlow lite interpreter takes in a bitmap object which it then converts to a byte buffer for use in the interpreters run function which does the actual classification. As such, it was more efficient to send in the raw image bitmap generated on image capture than to launch an external camera application, get the Image URI then convert it back into a bitmap for use in the interpolator.
- An inbuilt camera makes the application more immersive and involved than one that is launched whenever a user needs it. It always keeps the user in the application and minimizes the number of screen changes that need to happen whenever a user uses the application. It also allows for custom interactions with the camera to be made such as labeling on the fly in the explore activity or answering correct or wrong to image classifications in the guess what activity.

These reasons created the need for an in-built camera module. However, the decision to go with a custom built one as opposed to a ready built package to allow for a greater degree of development freedom and for the wider support available to the native camera package than other ready built packages. Though the behavior of the camera is rudimentary at this level, it



offers far more in the way of use case customization in further developments of the system than other ready built packages.

Customisation is important for this application since data standardisation is a major requirement of the system in its future development. As such, being able to programmatically limit or even lock the aspect ratio, image orientation, focus level and other image features, is a valuable option afforded by a custom built camera since these are options that can be setup when creating the camera surface using the `camera.getParameters()` function. For this implementation, the parameters set were the max camera size available to the camera and auto focus to keep the principle object in focus.

#### 4.7.3 COM.ANDROID.VOLLEY

Communication with the web service is a major requirement of this application. As such, the android Volley library was a key component of the applications functionalities as it was the main handler of the systems network communications. Volley works by creating a request queue that adds new requests as they are made to itself. These request objects handle the raw parsing of data, leaving the request queue to handle all network transactions.

A volley manger was built for the application to modularise the use of the request queue. String requests are used to send and receive data over the request queue.

#### 4.7.4 ORG.TENSORFLOW.LITE.INTERPRETER

The TensorFlow lite interpreter in the main package used to run classifications using the downloaded model. The interpreter used In the application is a java wrapper for the lower

level C++ API that runs the model from a “.tflite” file, the file format of the converted model. This package forms the core of the applications prediction functionality.

The interpreter relies on the FlatBuffers file format which is a robust cross platform serialisation library, allowing models to be used by the C++ API and via the Java wrapper class used in the application. As such, data is converted into byte buffers for use by the model.

The surrounding TfLiteImageClassifier.java class follows Shekhar’s TensorFlow lite example implementation, which borrows heavily from TensorFlow’s own example implementation of on device image classification. This is considered the standard implementation as it contains the supporting functions for model use such as the bitmap to byte buffer conversion, the load model function and get sorted results function. The main differences between the TensorFlow example and Shekar’s is;

- The use of a Classifier interface, which defines the Recognition class, a class that represents the form each recognition is supposed to take.
- A static constructor function that simplifies model use in the given activity.
- A more modular approach to the classes uses since it allows hyper parameters like image input size and model path to be specified in said class.
- It has the facility for quantized models to be used by specifying a Boolean value to use a quantized model. This is an important consideration for future works as it will require little effort to test and use quantized models which run faster than byte models (being used here).

This implementation was also chosen for its use case similarity to this project. The example being followed worked using a conventional camera and so was more ideal as a starting point than the TensorFlow example which ran classifications on a live preview screen, meaning that heavy modification would be required to make the standard implementation more suitable for this use case.

Modifications were however still required, namely in the model path requirement for the application. In order to allow the application to update the model over time, the model needs to be written to internal storage. This meant that the class had to be modified to remove its dependency on the assets folder that is read at APK build time and thus, is not write accessible. This was implemented in the model load and static create functions to allow for an activity context to be passed, allowing for the model path to be called.

#### 4.7.5 WEB SERVICE

The web service facilitated the majority of the communication in the system. As such, even though its routing package server-side technologies were extremely useful, the two main modules that assisted the most in cross system communication were:

#### 4.7.6 PYTHON-SHELL

Python-shell was an instrumental module to the functionality of the web service. Python-shell is a package used to communicate with a specified python environment, allowing the Node.js web service to run python scripts from within the service. This is what allows the service to issue commands to the model training layer by calling the script and setting the

python and script path. It records all the print outputs of the python console to its data callback as an array, thus allowing one to query the output using javascript array slice method to retrieve specific data from the output.

To this effect, `data.slice(-2)` is called in the train API to get the last two string outputs which are the model name saved and a “train complete” statement. This is used by another callback function to trigger the release of the mobile model update procedure that allows the mobile application to make a successful model update API call.

#### 4.7.7 MONGOOSE

MongoDB was the database of choice and was used extensively to store the images and model train data. It served as the intermediate data layer of the entire system, storing all useful data points. Mongoose was used to add new images on behalf of the mobile application and retrieve new models for update for the application. It was also used to retrieve model data from the DB and send it to the mobile application. In this way, it facilitates data persistence and abstraction by not requiring data be stored in the services file system.

#### 4.7.8 PYTHON ENVIRONMENT

The python environment is a package specific distribution of python that hosts all the dependencies needed to run the functions associated with the model training. However, in this context, it is considered as the model train layer of the system where all the python-based functionality runs. As such, two major packages were instrumental to its proper execution.

#### 4.7.9 PYMONGO

As was mentioned above, mongoDB was selected as the DB of choice and so to communicate with it, Pymongo was chosen for its ease of use and wide support within the python community. This was used in a DB\_handler.py module where the image\_download and model\_upload upload functions are implemented.

The image\_downlaod function works by going through each photo in the Photos collection and first checking if the given label is recorded in the data file system of the python environment. If not, the function creates a new file, adds the image to the file and writes the label to lables.txt file for reference sake. However, if the label exists, the file directory with the corresponding label is opened and the image is written to the file using its given name from the DB. This makes all the images available for training purposes when called and is called at the beginning of the train\_handler.py script which handles all training logic.

The model\_upload function works by collecting basic training meta data such as number of trained images, trained labels, image count by label, time of train and the current train iteration. It then adds the data to the “models” collection along with the trained model. This function is called at the end of the train\_handler.py script.

#### 4.7.10 TENSORFLOW 1.13.1

TensorFlow provided the main backend for the systems training. However, the 1.13.1 release of TensorFlow was specifically useful as it provided two major benefits over previous releases of the package. These were its improvements to keras, the high-level model building

wrapper, and the addition of `TfLiteConvertor` class to the package. The specific improvements are as follows.

- **Keras:** In previous iterations of TensorFlow, keras was used mainly as a separate wrapper that worked with a TensorFlow backend. While this was not an issue, 1.13.1 vastly improved the functionality of keras by incorporating it directly into the TensorFlow library. This means that models built using this version of keras, I.e. `tensorflow.keras` as opposed to native keras, are optimised to run on the TensorFlow backend and thus run faster. The other benefit this has is that TensorFlow officially supports this version of keras and will be using it as its high-level wrapper in the upcoming 2.0 versions, meaning models created using this API will be supported in the long term with future improvements to TensorFlow. This helps future proof the system and further simplifies development overhead in future implementations of this project.
- **TfLiteConvertor:** The inclusion of this class to the 1.13.1 variant is one that vastly simplifies the development cycle of the system and helps make it more robust. The `TfLiteConvertor` class handles the conversion of models from their current form, either keras or raw TensorFlow models, into a TfLite flatBuffer file to be used on mobile devices. In previous iterations of TensorFlow, specifically the 1.11.0 version that was available at the start of this project, model conversions had to be done through a secondary TOCO package which needed to be built and run using Bazel, making it more difficult to work with, and thus, more prone to error. `TfLiteConvertor` completely

removes the reliance on Bazel and simplifies the conversion procedure by presenting useful function for running the conversion.

These improvements greatly simplified the development process, and given that they are slated for TensorFlow 2.0, system is guaranteed improvements with future releases.

#### 4.7.10.1 KERAS

As was mentioned above, the keras package used was the tensorflow.keras version, thus it benefits from the under the hood optimisations made in TensorFlow. Beyond this though, keras was used to build the model used for training. It was chosen over raw TensorFlow for its ease of use and relatively short code base. Although raw TensorFlow has more customisability and functionalities than keras does, most of these functionalities are unnecessary for this projects scope. Highly prototype-able model building was favored over highly functional model building, thus making it the better option for this project. Keras's sequential model class was used to build the model by adding new layers to the sequential model object.

The model was built using the following structure:

| Layer (type)                                    | Output Shape         | Param # |
|---|----------------------|---------|
| conv2d_6 (Conv2D)                               | (None, 124, 124, 32) | 896     |
| batch_normalization_v1_7 (Batch Normalization)  | (None, 124, 124, 32) | 128     |
| max_pooling2d_3 (MaxPooling2D)                  | (None, 62, 62, 32)   | 0       |
| dropout_4 (Dropout)                             | (None, 62, 62, 32)   | 0       |
| conv2d_7 (Conv2D)                               | (None, 60, 60, 64)   | 18496   |
| batch_normalization_v1_8 (Batch Normalization)  | (None, 60, 60, 64)   | 256     |
| conv2d_8 (Conv2D)                               | (None, 58, 58, 64)   | 36928   |
| batch_normalization_v1_9 (Batch Normalization)  | (None, 58, 58, 64)   | 256     |
| conv2d_9 (Conv2D)                               | (None, 56, 56, 64)   | 36928   |
| batch_normalization_v1_10 (Batch Normalization) | (None, 56, 56, 64)   | 256     |
| max_pooling2d_4 (MaxPooling2D)                  | (None, 28, 28, 64)   | 0       |
| dropout_5 (Dropout)                             | (None, 28, 28, 64)   | 0       |
| conv2d_10 (Conv2D)                              | (None, 13, 13, 128)  | 73856   |
| batch_normalization_v1_11 (Batch Normalization) | (None, 13, 13, 128)  | 512     |
| conv2d_11 (Conv2D)                              | (None, 11, 11, 128)  | 147584  |
| batch_normalization_v1_12 (Batch Normalization) | (None, 11, 11, 128)  | 512     |
| max_pooling2d_5 (MaxPooling2D)                  | (None, 5, 5, 128)    | 0       |
| dropout_6 (Dropout)                             | (None, 5, 5, 128)    | 0       |
| flatten_1 (Flatten)                             | (None, 3200)         | 0       |
| dense_2 (Dense)                                 | (None, 1024)         | 3277824 |
| activation_8 (Activation)                       | (None, 1024)         | 0       |
| batch_normalization_v1_13 (Batch Normalization) | (None, 1024)         | 4096    |
| dropout_7 (Dropout)                             | (None, 1024)         | 0       |
| dense_3 (Dense)                                 | (None, 14)           | 14350   |
| activation_9 (Activation)                       | (None, 14)           | 0       |
| Total params: 3,612,878                         |                      |         |
| Trainable params: 3,609,870                     |                      |         |
| Non-trainable params: 3,008                     |                      |         |

Figure 4.1



It uses an Image Data Generator in its input layer to create new augmented images from input images to boost the dataset and further improve its performance when given small datasets. This model was developed from unit testing and inspired by Rosebrok's tutorial [\[site\]](#)

#### 4.7.10.2 TFLITECONVERTOR

TfLiteConvertor was the class used to programmatically run conversions of the trained model. Given that the outputted model was a keras .h5 file, the `from_keras_model_file()` function was used to get the converted model for use. This conversion is run at the end of the `train_handler.py` script.

## Chapter 5: Testing And Results

### 5.1 Development Testing

#### 5.1.1 UNIT TESTING

During unit testing, individual components of the application were tested to ensure functionality. Key sections tested here were model validation accuracy, webservice communication with the application and web service communication with the python environment.

##### 5.1.1.1 VALIDATION ACCURACY

During this testing, the model was trained on 14 classes in the caltech 101 dataset[15]. These classes were chosen for their relative relevance to the local context of the systems expected deployment.

Number of files in Training-set: 1029

Number of files in Validation-set: 257

....

Training Epoch 28/30 --- Training Accuracy: 94.0%, Validation  
Accuracy: 64.0%, Validation Loss: 1.354

Training Epoch 29/30 --- Training Accuracy: 94.0%, Validation  
Accuracy: 64.0%, Validation Loss: 1.365

```
Training Epoch 30/30 --- Training Accuracy: 94.0%, Validation  
Accuracy: 64.0%, Validation Loss: 1.386
```

Training from script complete!

From the above text copied from the console output, one can see how well the model performed on this dataset when given 30 epochs to train on. The loss function is calculated using cross entropy

#### 5.1.1.2 APPLICATION COMMUNICATION

A major part of the systems functionalities is its ability to communicate with the mobile application in order to facilitate photo sending and model updating. To test this feature, the “test server” button was added to the app. This button triggers a function to send a string request to the server, which responds with “A- OK!” if there is a connection. This was used during development to ensure that the app was connected before testing any other components.

#### 5.1.1.3 PYTHON ENVIRONMENT COMMUNICATION

Communication with the python environment went through a number of iterations to reach a stable state. The main features that needed testing here were training trigger and database handling. To this effect, 3 functions were developed to test this using two different packages, child-process and python-shell, both of which could be used to communicate with the python virtual environment.

```
exports.db_pull = function (req, res) {
```

```

var options = {
    // pythonPath: "path", TODO: use virtual env path here
    args: ["label"]
};

PythonShell.run('/home/sai-pher/work/test_projects/CNN_test/mongo_test.py', options,
function (err, data) {
    if (err) res.send(err);
    res.send(data.toString())
});
};

```

This function uses the python-shell package to specify the python path and arguments for the function. It runs the mongo\_test.py file which returns a list of the labels in the database to the server. This show that communication between all layers of the system via the service is working.

### 5.1.2 COMPONENT TESTING

This testing puts together multiple units in the system to test. Major components tested were the photo sending sequence and retrieval. A mock gallery was created to show photos added to the database via the application. This tested both the send function on the application and the API on the service responsible for saving pictures in the database.

This component was shown to work as new photos were added to the database after capture and shown in the gallery. The python train.py function was also tested and shown to

work, showing that that the train trigger component also worked. In the mobile application as well, the entire Explore component is shown to work as intended, since it has been able to meet the system requirements set out for this feature.

Another component tested was the “Guess What?” feature. This was done by manually by moving a self-trained model to the Android app and seeing how it worked in the app. To this effect, the “Guess What?” feature was shown to work by displaying the predictions of captured images. This showed that the camera activity in the feature was successfully able to send images to the TFLite module and send back predictions.

### 5.1.3 SYSTEM TESTING

This testing put together system components for testing together as an entire system. For this, a larger test was used to validate the system. Brand new images were taken with the “Explore” feature.

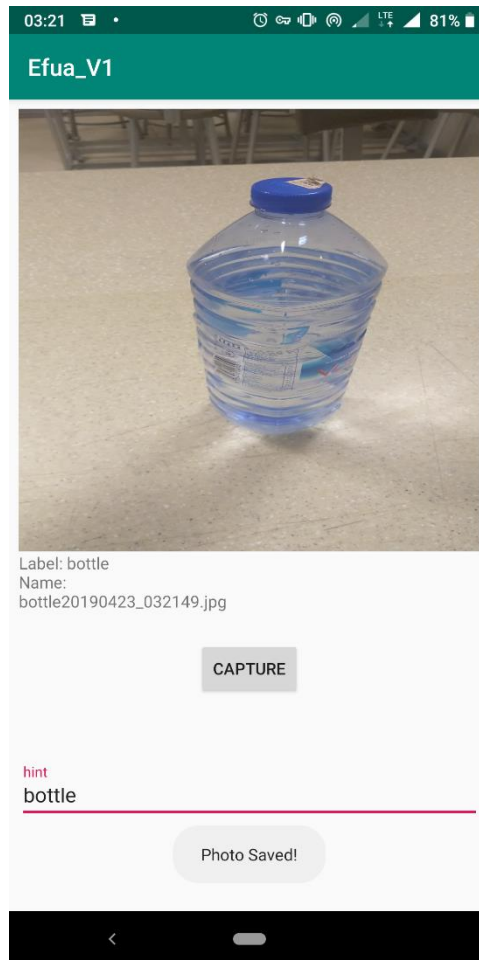


Figure 5.1

Labeled images were uploaded to the database. See image bottle20190423\_032149.png (record one) for reference

| 📁 photos |                          |              |                             |
|----------|--------------------------|--------------|-----------------------------|
|          | _id ObjectId             | label String | name String                 |
| 1        | 5cbe84ca87d7c33b899681cc | "bottle"     | "bottle20190423_032149.jpg" |
| 2        | 5cbe84d287d7c33b899681cd | "bottle"     | "bottle20190423_032156.jpg" |
| 3        | 5cbe84d387d7c33b899681ce | "bottle"     | "bottle20190423_032158.jpg" |
| 4        | 5cbe84d687d7c33b899681cf | "bottle"     | "bottle20190423_032201.jpg" |
| 5        | 5cbe84d987d7c33b899681d0 | "bottle"     | "bottle20190423_032204.jpg" |
| 6        | 5cbe84e387d7c33b899681d1 | "bottle"     | "bottle20190423_032213.jpg" |
| 7        | 5cbe84e887d7c33b899681d2 | "bottle"     | "bottle20190423_032218.jpg" |

Figure 5.2

The train sequence was triggered after fifty images were added with the post request, in accordance with the train logic.

```
[nodemon] app crashed - waiting for file changes before starting...
[nodemon] restarting due to changes...
[nodemon] starting `node ./bin/www`
(node:19252) DeprecationWarning: current URL string parser is deprecated
Use MongoClient.connect.
POST /photo/create 200 481.877 ms - 12
training...
```

Figure 5.3

The trained model was then uploaded to the database for use on the mobile device.

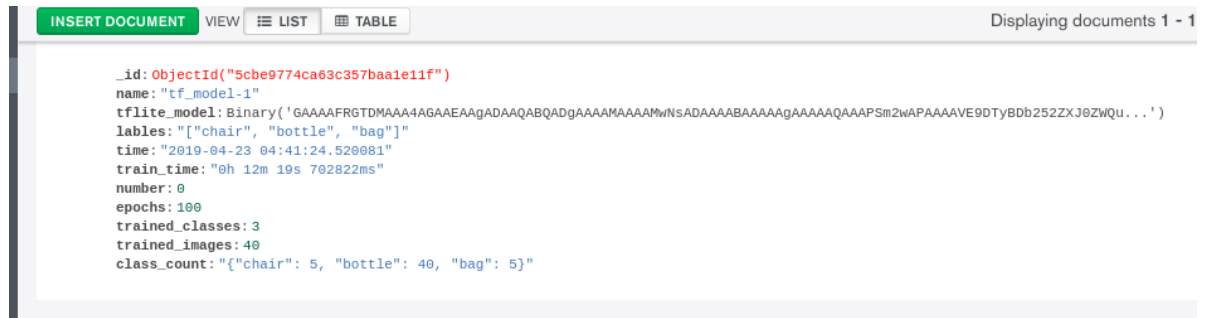


Figure 5.4

The application was then updated with the new model and tested in the “Guess What?” feature. This showed that the entire train sequence worked as expected and was ready for user testing.

## 5.2 User Testing

In user testing, the system is deployed and given to users to see how they interact with the system. For the scope of this project, the test environment is the Ashesi University campus. The testing phase will be carried out with the following plan.

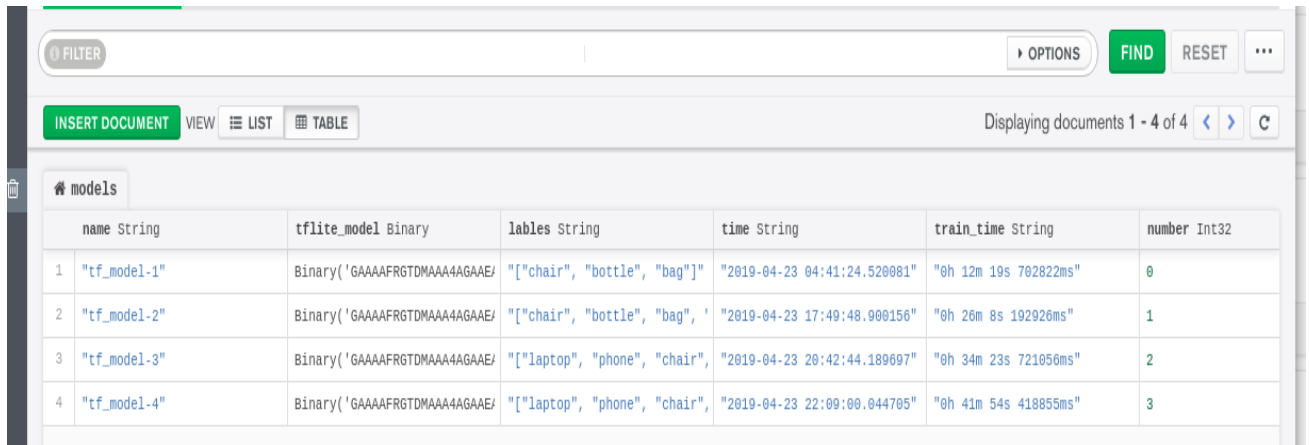
- The system will be deployed on a developer computer to simulate a server.
- A call for users will be sent out.
- All would be users will be briefed on the scope of this project and how to use the application in its test stage.
- Users will be given the mobile application to use within the Ashesi context.
- The model will be trained on after every 50 images.
- The applications will be updated once training is complete.



- Users will be asked to test the model frequently and add specific data as they see fit.
- The model will be tested against a baseline model trained on generic data to see how well the user taught model performs compared to it.

### 5.2.1 TESTING RESULTS

The system was successfully user tested within a controlled environment. Five individual users were asked to add images within a specific area (the Ashesi Engineering block) that had the following labels: bottle, chair, bag, plant, laptop and phone. Two hundred images were added at the end of testing and models generated for each fifty-image block.



|   | name String  | tflite_model Binary           | lables String                 | time String                  | train_time String     | number Int32 |
|---|--------------|-------------------------------|-------------------------------|------------------------------|-----------------------|--------------|
| 1 | "tf_model-1" | Binary('GAAAAFRGTDMAAA4AGAAE/ | "["chair", "bottle", "bag"]"  | "2019-04-23 04:41:24.520081" | "0h 12m 19s 702822ms" | 0            |
| 2 | "tf_model-2" | Binary('GAAAAFRGTDMAAA4AGAAE/ | "["chair", "bottle", "bag", " | "2019-04-23 17:49:48.900156" | "0h 26m 8s 192926ms"  | 1            |
| 3 | "tf_model-3" | Binary('GAAAAFRGTDMAAA4AGAAE/ | "["laptop", "phone", "chair", | "2019-04-23 20:42:44.189097" | "0h 34m 23s 721056ms" | 2            |
| 4 | "tf_model-4" | Binary('GAAAAFRGTDMAAA4AGAAE/ | "["laptop", "phone", "chair", | "2019-04-23 22:09:00.044705" | "0h 41m 54s 418855ms" | 3            |

Figure 5.5

The image break down at the end of testing was as follows: {"laptop": 31, "phone": 19, "chair": 35, "bottle": 41, "bag": 22, "plant": 52}. This was a small dataset and as such, the individual models did not perform well enough to be useful for the “Guess What?” feature, with most predictions yielding “chair”.

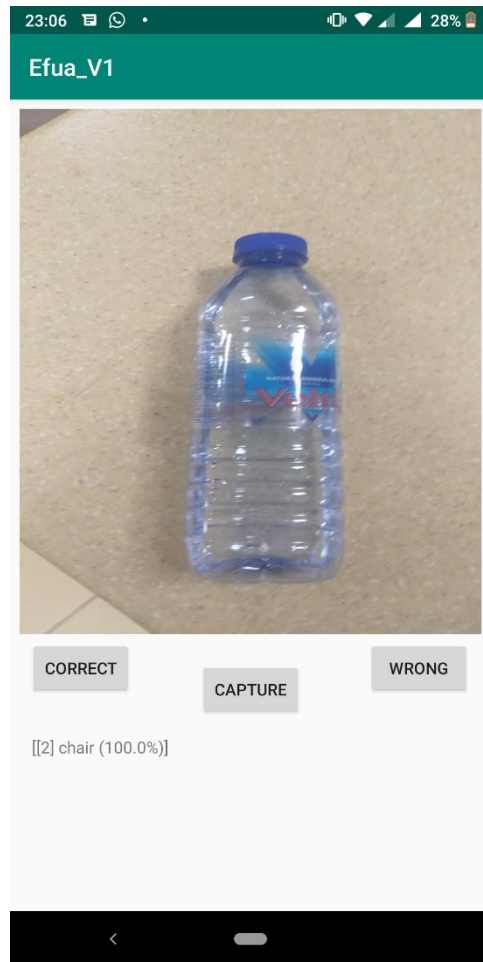


Figure 5.6

The testing and validation loss especially showed that the model was not behaving as one would hope, with validation loss showing no improvement after twenty epochs on the two

hundred image model.

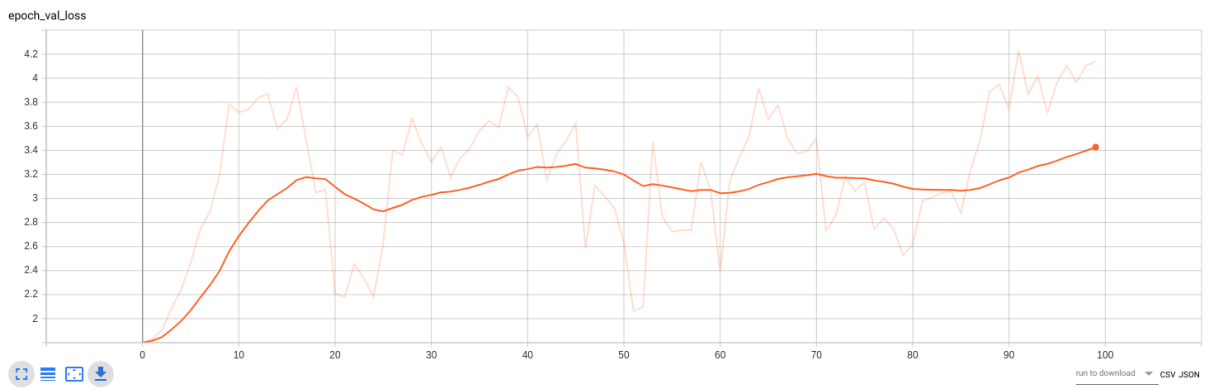


Figure 5.7

However, this behavior is to be expected as more data will be required for any real improvement to show. A positive result of the user testing however was that two hundred images was reached within a remarkably short time even with only 5 participating users. Each user group was able to add fifty images within the space of 10 minutes of image capture. With each user capture above twenty images during the session. The testing phase was thus a success in that it showed users could reliably add images to the system and that sufficient data amounts can be reached within a short period of time if the user base is increased.

An interesting behavior that was observed during testing was between the third and fourth training sessions. The system was updated to show users how many of each label had been added to the system after each successful image addition. Users in this training block were asked to take pictures of any of the labels. However, when a user saw how many plants were trained on in session 3 (14 plants) the user decided to gather only images on plants, increasing

the plant count to 52. When asked why they did this, the user stated that they wanted to model to make more plant predictions, hence their interest in gathering plant data.

This falls in line with the argument made for incentive in the introduction, where users found it easy to add usable labeled images of things they wanted the system to do better on. This behavior will be an asset to the system once it begins to reach a critical data point where it can adequately begin to reduce the validation loss as expected.

## **Chapter 6: Conclusion And Recommendations**

### **6.1 Conclusion**

To assess the systems achievements, we look at the objectives this project set out to achieve.

- Build a research mobile application to teach an AI (Assistive Intelligence) image classification by exposing it to real world examples, tagged by users of this application.
- Use server-side technology to collect, train and update the applications classification abilities.
- Build a proof of concept system based on image classification to show the active teaching technique is feasible.
- Use the application for open, consensual data collection in a local setting.

From these objectives, we can deduce the following.

The system, built as a proof of concept, achieved its main goals by successfully showing that active teaching is a viable option for training new AI and Machine learning algorithms. the system adequately allows users to label new images and test the model on similar images. The system adequately manages data across the various facets of the system, successfully moving it around to where it is needed and when. The system proves that a model can benefit from user added data quickly and performs reasonably well within the domain of its use.

Beyond its technical achievements though the system shows that with more development, a good case can be made for a system that relies on both public involvements as much as on the technicians building the AI and machine learning algorithms of the future. Within a relatively short period and with only a few initial test units, the system was able to generate a significant number of images within a lab testing scenario, showing that the system will perform even better with greater numbers.

The system also shows promise for its wider goal, which is to create a home grown, African AI project that seeks to tackle the issue of bias plaguing the ML community. Having been tested within the Ashesi community, the system shows similar results to those seen when tested on generic available data. And although the testing did not feature a “people” class for data privacy reasons, it is easy to see how a traditional AI’s bias towards primarily Caucasian individuals being classified correctly would be flipped on its head in an African context. This thus shows that in its current form, the system can perform well within its used domain since the bias it will generate will match what the domain requires of it.

This presents its own interesting set of challenges, namely being that the system may begin to reflect societal biases more prominently than ever before. However, this too can be an opportunity to bring the public closer to the development of AI and an avenue for greater discussion into how much of bias is the fault of AI versus that of the society that trains it?

This isn’t to say that the systems sensitivity to bias is all bad. The systems bias can also be used to balance out current biases, especially when the new data coming in is from previously

underrepresented groups, thus making a model generate more realistic predictions than those tested on available data. However, further testing will be required to show just how sensitive the system can be to bias.

But perhaps the systems greatest possible benefit is how it opens the possibility for other machine learning projects to access new useful data, especially for projects where lack of data is a problem like low resource language machine translation. As a system intended to be a platform for collaborative, crowd sourced AI development, it is possible to see a future where this system is used to gather useable data for low resource languages from the people who use them the most. This presents an opportunity for a new wave of African developers to adopt the old adage, “it takes a village to raise a child” for a new world. One where AI is taught by a global village of technicians and users that hope to see it grow into a responsible part of our society.

## **6.2 Limitations**

Though the system functions as expected, there are still many areas where improvements can be made across the stack.

### **6.2.1 MOBILE APPLICATION**

- The application is a prototype model. As such, its views are functional only and do not represent a production ready application.
- The Explore feature cannot be used properly offline, as it requires an internet connection to send data to the server, and has no offline caching implemented.

### 6.2.2 WEBSERVICE

- The webservice only supports the functional requirements of the system and thus, does not provide a good user experience.
- The train logic currently runs on a counter that resets after every 50 pictures. This is not ideal for a full system as it means the training will be triggered too often. Beyond changing the limit however, it is still not ideal to keep it locked to a fixed number as a fixed number will not guarantee significant improvements to the model, relative to the resources training will take up. A better solution may be to make training happen when the new image counter is perhaps equal to 20% of the full data set and server resource consumption is below an acceptable threshold.

### 6.2.3 PYTHON ENVIRONMENT

- The `image_download()` function currently has an  $O(n)$  runtime where  $n$  is the total number of images in the database at the time it is called. This is problematic as it means that it adds to the overhead of the `train_handler.py` script which has an  $O(n^2)$  run time. A more ideal solution will be to download only the most recent images from the database at train time. This won't reduce train time, but it will reduce the computational overhead of the script. The recourse efficiency of this will however be tied to the train activation logic and needs to be considered in future implementations.
- The `train_handler.py` script currently runs at an  $O(n^2)$  run time over multiple training runs. This is because the training function takes longer each time as more data gets added since it trains on the entire available dataset each time. This will be a problem as



the dataset grows and training gets triggered more often depending on the train trigger logic. This can be fixed by using transfer learning to only train on new data and can be implemented in the future in order to greatly reduce the resource consumption of training.

#### 6.2.4 FURTHER LIMITATIONS

Beyond the individual components of the system, there are other limitations that this project has.

- The system currently does not have a way to validate data coming in is accurately labeled.
- The system does not have a way to stop unwanted data, such as people's faces or sensitive information, from being added.
- The system does not currently collect user related data. Though this may be good for privacy, it limits the potential of the system to learn from user data collection behavior.
- In its current form, it is technically possible to over fit a model to a given context by saturating the database with images from this context. For example, if all users take pictures of the same tree, the system will perform extremely well on that tree but terribly on other plant images. There is currently no technique to control for this eventuality.
- Given the current train logic, it is possible to overlap multiple training sessions, putting massive load on the server hardware and slowing down train and update times as the

data grows. This is a major issue that will have to be tackled to ensure the full scalability of the system within a resource constrained environment.

## **6.3 Recommendations And Future Work**

### **6.3.1 RECOMMENDATIONS**

In order to improve the system, the following recommendations can be followed up on handle the limitations of the system.

- The single most useful thing that can be done to improve the system in its current state would be to improve the training logic. This will minimize the number of trains, making each cycle as beneficial to the model as possible without overloading the server. The factors to consider for this are;
  - Spacing: The logic should space out each training cycle as much as possible while keeping updates within a reasonable, predictable time from.
  - Data availability: Train triggering should still be tied to the availability of new data. However, using data ratios may be more useful than a fixed limit, since as the model improves, it is unlikely that a fixed amount of data will affect its incremental improvement.
  - Off peak times: Time should also be a factor for training. Specifically, using a scheduled time to run training when the server is at its lowest load. This will be to ensure that resources will be available and that users can reliably expect updates.

- Overlapping: The logic should also work to minimize training overlaps whenever the API is hit to reduce the number of redundant trains. This can either be done with processes locking or a check system on the webservice that makes sure a cycle is done before it triggers the next cycle.

### 6.3.2 FUTURE WORK

Beyond the improvements that can be made to the system by tackling the limitations outlined above, working towards making the app more user friendly and production ready is something that can be tackled in future work. Namely,

- Optimizing the user interaction of the application.
- Paying close attention to the user experience of the system.
- Adding user specific profiles to make models more personal.
- Allow for dataset requests or downloads from other researchers, providing access to a standard new dataset.
- Adding a production ready data validation system.
- Making the app security optimized.

Something else that can be tackled in future work would be to use transfer and reinforcement learning as part of the systems core suite of tools for improving a base model as more data is added and users use the guess what feature.

Another thing that can be tackled with future work would be to use this system to develop more useful AI and Machine Learning models such as those used for Low resource

language machine translation. This system can be used as a novel approach to gathering data since users will effectively be teaching the system how to speak their language.

## References

- [1] Abhigna B.s., Nitasha Soni, and Shilpa Dixit. 2018. Crowdsourcing – A Step Towards Advanced Machine Learning. *Procedia Computer Science* 132, (January 2018), 632–642.  
DOI:<https://doi.org/10.1016/j.procs.2018.05.062>
- [2] Jacques Bughin, Seong Jeongmin, James Manyika, Michael Chui, and Raoul Joshi. 2018. Notes From The Frontier: Modeling The Impact Of AI On The World Economy. *McKinsey&Company*. Retrieved October 11, 2018 from  
<https://www.mckinsey.com/featured-insights/artificial-intelligence/notes-from-the-ai-frontier-modeling-the-impact-of-ai-on-the-world-economy>
- [3] Jayson DeMers. 2015. What Is Google RankBrain And Why Does It Matter? *Forbes*. Retrieved October 11, 2018 from  
<https://www.forbes.com/sites/jaysondemers/2015/11/12/what-is-google-rankbrain-and-why-does-it-matter/#24dbf8de536b>
- [4] E. Gent. 2015. AI: Fears of “playing God” [Control amp; Automation Artificial Intelligence]. *Engineering Technology* 10, 2 (March 2015), 76–79.  
DOI:<https://doi.org/10.1049/et.2015.0210>
- [5] J. Isaak and M. J. Hanna. 2018. User Data Privacy: Facebook, Cambridge Analytica, and Privacy Protection. *Computer* 51, 8 (August 2018), 56–59.  
DOI:<https://doi.org/10.1109/MC.2018.3191268>
- [6] Ralph Jacobson. 2013. 2.5 quintillion bytes of data created every day. How does CPG & Retail manage it? - IBM Consumer Products Industry Blog. *IBM Consumer Products*

- Industry Blog*. Retrieved October 11, 2018 from <https://www.ibm.com/blogs/insights-on-business/consumer-products/2-5-quintillion-bytes-of-data-created-every-day-how-does-cpg-retail-manage-it/>
- [7] S. Leavy. 2018. Gender Bias in Artificial Intelligence: The Need for Diversity and Gender Theory in Machine Learning. In *2018 IEEE/ACM 1st International Workshop on Gender Equality in Software Engineering (GE)*, 14–16.
- [8] Chintan Parmar, Patrick Grossmann, Derek Rietveld, Michelle M. Rietbergen, Philippe Lambin, and Hugo J. W. L. Aerts. 2015. Radiomic Machine-Learning Classifiers for Prognostic Biomarkers of Head and Neck Cancer. *Front. Oncol.* 5, (2015). DOI:<https://doi.org/10.3389/fonc.2015.00272>
- [9] H. Ranganathan, H. Venkateswara, S. Chakraborty, and S. Panchanathan. 2017. Deep active learning for image classification. In *2017 IEEE International Conference on Image Processing (ICIP)*, 3934–3938. DOI:<https://doi.org/10.1109/ICIP.2017.8297020>
- [10] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. 2017. Mastering the game of Go without human knowledge. *Nature* 550, (2017). DOI:<https://doi.org/10.1038/nature24270>
- [11] James Zou and Londa Schiebinger. 2018. AI can be sexist and racist — it’s time to make it fair. *Nature* 559, 7714 (July 2018), 324. DOI:<https://doi.org/10.1038/d41586-018-05707-8>

- [12] 2018. Population by Regions in the World (2018) - Worldometers. *worldometer*. Retrieved October 11, 2018 from <http://www.worldometers.info/world-population/population-by-region/>
- [13] 2018. What is Human-in-the-Loop Machine Learning? *Gengo AI*. Retrieved April 23, 2019 from <https://gengo.ai/articles/what-is-human-in-the-loop-machine-learning/>
- [14] ImageNet. *image-net*. Retrieved October 11, 2018 from <http://image-net.org/>
- [15] Caltech101. Retrieved April 23, 2019 from [http://www.vision.caltech.edu/Image\\_Datasets/Caltech101/](http://www.vision.caltech.edu/Image_Datasets/Caltech101/)