DUBLIN CITY UNIVERSITY

SCHOOL OF ELECTRONIC ENGINEERING &

SCHOOL OF MECHANICAL & MANUFACTURING ENGINEERING

Mobile Health:

Android Application for Diet Management using Machine Learning, Image Classification & IoT Device

Luke Scales

April 2018

BACHELOR OF ENGINEERING

IN

MECHATRONIC ENGINEERING

Supervised by Dr. B. MacDonald

Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

I have read and understood the DCU Academic Integrity and Plagiarism at https://www4.dcu.ie/sites/default/files/policy/1%20-%20integrity_and_plagiarism_ovpaa_v3.pdf and IEEE referencing guidelines found at https://loop.dcu.ie/mod/url/view.php?id=448779.

Name: _____ Date: _____

Acknowledgements

I would like to thank my supervisor Dr. Bryan MacDonald for his support in this project, and for undertaking its supervision. Without him it would not have been possible to perform this project and for that I am grateful. I would also like to thank Bryan for introducing me to Android development, as a project he gave me in the past brought about a set of skills I cherish and would not have developed otherwise. I would also like to thank my family for their support throughout my studies.

Abstract

This project outlines an approach for improving upon existing applications in the meal-logging and calorie-counting category. The approach adopts a user-centric design process, while implementing state-of-the-art, mobile optimised image classification and Machine Learning technologies. This approach is further improved by implementing a Bluetooth kitchen scale, which measurably improved the user-product interaction. This results in an application which is more accessible and user-friendly than existing applications.

This work describes the development process of the Android application, while also reviewing the relevant literature and ethical considerations of the work. The work is evaluated using measurements adapted from user experience (UX) design principles through usability testing. Some analysis on the retraining process involved with the TensorFlow Machine Learning library using the MobileNet image classification model is also explored in the work, but the results of this are inconclusive.

Table of Contents

List of Figures

List of Tables

Chapter 1 - Introduction

This project presents an Android application which was researched and designed to improve upon the user experience of current diet management applications, and increase their accessibility. This is achieved by implementing an image classifier which is based on convolutional neural networks, as well as an integrated Bluetooth kitchen scale. The result is an improvement in the efficiency of the user-product interaction for typical users of these applications, with indications of enhanced accessibility for others. Section 1.1 describes the area in which this project resides, while Section 1.2 defines the problem being addressed and then briefs the reader on the proposed solution to this problem.

1.1 Overview of the Area

In recent years, the "mHealth" or "Mobile Health" category for mobile applications has seen considerable growth. In 2017 the mHealth market had a reported worth of $23 Billion [1], a contrast to the $6.7 Billion that the market was worth in 2012 [2]. This market is predicted, by many sources, to at least double in value by 2020 [1] [2] [3]. Concurrently, obesity has become a "global epidemic", as stated by the World Health Organisation (WHO), who refer to it as "one of today's most blatantly visible – yet most neglected – public health problems" [4]. Aside from direct obesity, a large proportion of the world's population is overweight, with another WHO study finding that "In 2016, 39% of women and 39% of men aged 18 and over were overweight" worldwide [5]. Figure 1 shows a visual representation of the global obesity and overweight problem, adapted from the WHO website [5]. Being classified as overweight or obese has significant health implications, with innumerable studies finding correlations between excess body mass and diseases such as diabetes, cardiovascular diseases, and some forms of cancer [6].

Mobile applications and technologies have the capability to address obesity and many other health problems, as outlined in other WHO studies which were summarised in 2011 [7]. In the seven years since the WHO reported this, technology and mobile devices have progressed significantly, which this author believes should make these findings more salient and applicable now than at the time they were published. Notably, the meteoric rise of Machine Learning over this period, as well the ever-growing support for its implementation on mobile devices, represents significant potential for innovation in this field. Machine Learning has upended the computer vision industry in particular, with the classification capabilities of this field far surpassing all other approaches to computer vision [8] [9] [10]. Concurrently, the adoption rate and number of applications of Internet-of-Things (IoT) devices has increased, with reports of the overall IoT market being worth $157 billion in 2016 and predictions of a $457 billion value by 2020 [11]. With these developments, capabilities for disruption and innovation follow. The writer aims to explore the potential for Machine Learning and computer vision systems, as well as IoT devices, to improve upon existing technology in the mHealth industry.

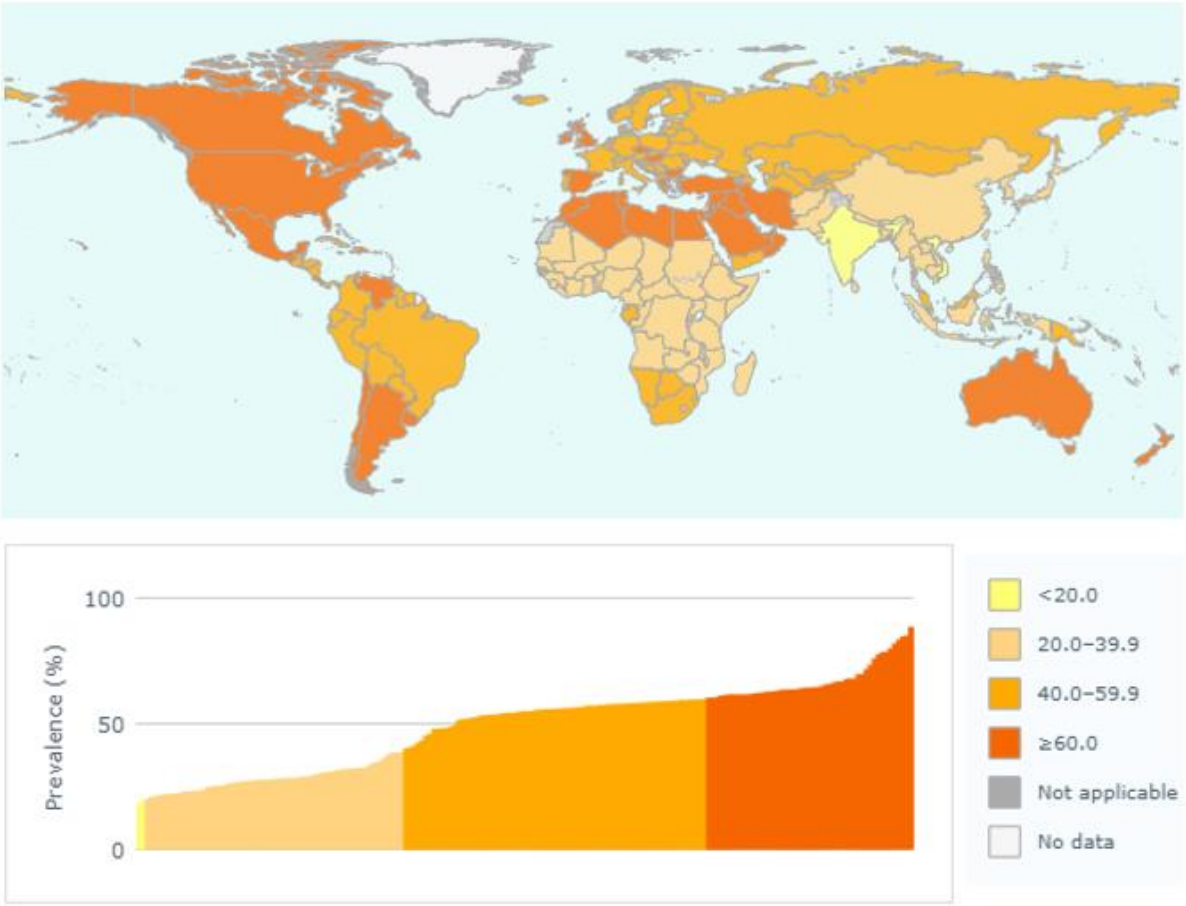

Figure 1: Global Obesity and Overweight Statistics (2016), Adapted from WHO [5]

Calorie-counting and diet management applications are undoubtedly a popular component of the mHealth category, with the application quality review company Applause reported to have analysed over 3.3 million reviews of calorie-counting applications [12]. This was performed as part of an analysis of the entirety of calorie-counting applications available at that time (2015), and the number of reviews that users have left is a testament to the prevalence of these applications. The core features of these applications allow users to record and monitor their caloric intake and the nutritional content of their food. The typical method in which these applications are used is where the user, while they are preparing a meal, weighs each ingredient individually and records its weight. This weight could be recorded separately for later use, but the applications are designed to allow these weight values to be entered immediately. The user can search for the food item they are currently using in their preparation process, retrieve the nutritional content values for it from a database, and then input the weight value to the application. With this completed for each ingredient of the meal, the application can calculate the total calories and the nutritional composition of the meal.

While these applications work and are used by millions each day [13], there is room for improvement. The concept of implementing a Machine Learning, image classification system with diet applications is not novel, nor is the use of IoT devices. IoT devices have been successfully integrated, well-received, and been met with modest adoption rates due to the success of wearable technologies such as Fitbit, Apple Watch, and Samsung Gear. This is juxtaposed with the implementation of image classification, where the likes of Samsung, Google, and Instagram have each mused with the concept, however they have been unsuccessfully executed. Nevertheless, the triad of Machine Learning, computer vision, and IoT technologies has the potential to improve upon the current solutions in the field of diet management applications. The innovations these technologies have jointly enabled show the broad range of possibilities and engineering problems they can solve together: from the autonomous cars being developed in the automotive industry, to the cashier-less shops in the retail industry. With the proper research these technologies could improve upon diet management applications. With improvements, diet management applications could become accessible to a broader range of users, while improving the user experience for those who already use them.

1.2 Problem Definition and Approach

This work aims to develop a novel approach to mobile calorie-counting and diet management applications, and one that enhances the user experience and the accessibility of such applications. The core objective is to remove text input fields where possible, which will reduce the complications and potential frustration associated with typing and spelling. By removing text-based features the application will also become more accessible to people who are dyslexic, illiterate, or those who are minorly visually impaired, to the degree where small text becomes illegible. In addition to this, the removal of the necessity to log foods manually could improve accessibility for those who have dexterity issues with their hands which limit their use of mobile devices.

The writer postulates that by implementing Machine Learning-based image classification, the efficiency of the meal-logging process will be increased, as well as user satisfaction. By allowing users to utilise their smartphone's camera to detect their food, the complexity of using text to search for food could be reduced, as well as the time required for this part of the meal-logging process. This work intends to explore the effect of this approach to meal-logging, whilst also investigating the effect of using a Bluetooth kitchen scale to replace the manual input of weight values. The aim is to measurably reduce the time needed to log a meal and improve the user experience, while also performing an analysis on the retraining process of the image classifier.

Chapter 2 – Literature Review

This chapter reviews the literature relevant to this project. Many areas of research intersect with this project, but to cover them all would be unwieldy. The writer could reasonably focus on the wealth of literature covering the science of health and nutrition, Android application development, or other areas of computer science. It was decided, however, to devote this chapter to the two most significant and interesting areas of relevance to this project: the technology which enabled the image classification system used in the application, and the history thereof (Section 2.1); and the principles of user-centric user experience design (Section 2.2).

2.1 Machine Learning and Image Classification

Whereas typical computer science applications involve algorithms with explicit and conditional commands, Machine Learning refers to the branch of computer science where algorithms learn from data and devise their own decision-making processes. For brevity, Langley [14] describes Machine Learning as the scientific field "concerned with the processes by which intelligent systems improve their processes over time". Langley addresses the similarities that Machine Learning shares with artificial intelligence and cognitive science, whilst also highlighting the broader scope that Machine Learning encapsulates. The broad scope of the topic is due to the presence or potential for learning in all industries, where the focus of the Machine Learning community is not on the application but upon the learning processes.

The learning process of Machine Learning systems often happens iteratively, where the algorithm is given some initial data such as rules or data classes/labels. From these initial conditions, the algorithm forms a basic understanding of the relationship between the data types, and the algorithm then uses experience to test and refine these initial assumptions. The experience here refers to taking labelled input data, removing the labels, and attempting to classify samples of the data. The labels are then reapplied, and the results of the classification process are compared to the true labels, which researchers such as Passonneau et al. [15] refer to as the "ground truth". If the algorithm misclassifies an input, the assumptions are adjusted to improve accuracy. For example, an algorithm playing a board game would be told the rules of the game, from which it would develop a basic understanding of how to play it. Each move made and observed would be the input data that the algorithm would try to classify or predict the outcome of, and from each move its understanding of the game strengthens and its predictions become more accurate.

The term Machine Learning is often cited [16] [17] as being coined by Arthur L. Samuel, in his work "Some Studies in Machine Learning Using the Game of Checkers", published in the IBM Journal in July of 1959. This attribution of the term to Samuel cannot be confirmed by the writer, but Samuel's work is the earliest text the writer has found with the term used. In his study, Samuel explores the application of Machine Learning and verifies its principles by teaching an algorithm the rules of a game and letting it learn how best to play [18]. This work influenced many others, as game-play has been an oft-tackled problem in the field. The most recent development in terms of game-playing machine learning algorithms is Alphabet Inc.'s

AlphaGo, which beat the world's best player of the board game Go at the "Future of Go Summit" in 2017 [19]. The significance of this achievement is due to the complexity of Go, which is an ancient Chinese abstract strategy game [20]. Where Chess has been estimated as having 10^{43} to 10^{50} possible positions within the game, the work of Tromp and Farneback [21] estimated that Go has a lower limit of 20^{170} possible positions. To compute this, Tromp and Farneback report that it took over 250,000 CPU-hours of computations and 30 petabytes of disk space. This highlights how impressive the feat that AlphaGo performed is, proving how effective it is at *learning* how to play the game, as it proves how impossible it would be for AlphaGo to win using conditional or explicit commands.

The advances in Machine Learning which made AlphaGo possible are a direct result of gradual progress in an area of scientific study. Before the turn of the millennium, Provost and Kohavi [22] refer to the science involved in Machine Learning as being a science of engineering, in that it is inherently a science of problem solving. It involves hypothesis, observation, analysis, and then practical application; however, they also highlight the lack of real-world applications of Machine Learning at that time. In the 20 years since this publication, Machine Learning has progressed from a stage of infancy, with the field mostly existing within scientific study, to a stage of adolescence where countless instances are applied in real-world scenarios each day. Examples of these applications can be found on the millions of mobile devices where Google, as outlined in the work of Abadi et al. [23], have implemented Machine Learning with Google Search, Maps, Photos, Translate and YouTube services, since as early as 2011.

In this decade alone, the capabilities and applicability of Machine Learning has progressed significantly. This progression is highlighted by the reduction in computational power required to implement the algorithms: from laboratory-based applications requiring the processing power of large clusters of computers across a network with thousands of CPUs [23], to the current instances such as this project, which can be compiled on a laptop and ran on a mobile device. In the latest volume of the journal in which Provost and Kohavi called for real-world applications of Machine Learning, Brazdil and Giraud-Carrier [24] show appreciation for the advancements made in recent years. They express how the application of Machine Learning to classification and regression problems has "expanded outside the boundaries of research into the realm of applied research, industry, commerce, and government". These classification and regression problems that Brazdil and Giraud-Carrier mention are present in innumerable industries and settings, however a common classification application of Machine Learning is in the field of image classification. Image classification is the process through which an

algorithm examines an image and labels it according to predetermined classes. Image classification techniques have a broad range of applications, such as the pedestrian detection system developed by Tomé et al. [25] for autonomous vehicles, or the many systems designed for facial detection such as the work of Taigman et al. [26], Lu and Tang [27] or Sun, Wang and Tang [28].

The use of Machine Learning for image classification has successfully disrupted the computer vision community, as seen in the facial detection systems mentioned above, which Lu and Tang prove can surpass a human's ability to detect faces [27]. Each of these systems employ what are known as *convolutional neural networks* or *deep convolutional neural networks*, also known as *ConvNets*. A convolutional neural network is an approach to Machine Learning which Matsugo et al. [29] refers to as being inspired by nature and biology. The neural network consists of computational *layers*, with each layer contributing to the overall recognition or classification task incrementally. This approach is not restricted to image classification and has been used for problems such as speech recognition, as seen in the work of Deng and Li [30], Abdel-Haimid et al. [31] and others. In image classification problems, this concept replicates biological vision systems by using layers which can be likened to the neurons and receptors of an animal's visual system. LeCun and Bengio, in their 1995 paper on the application of convolutional networks to image classification problems [32], credit the advent of this approach to the work of the neurophysiologists D. H. Hubel and T. N. Wiesel. Hubel and Wiesel performed experiments which analysed the visual cortex and interaction of neurons within the brains of cats and monkeys [33]. Their findings inspired many within the field of computer vision, with Fukushima's work on the application of convolutional networks to image recognition [34] [35] [36] [37] consistently giving credit to Hubel and Wiesel.

From the feature extraction techniques developed at the turn of the millennium to the innovative, convolutional neural network approaches popularised in the past decade, the developments which led to the image classification capabilities used in this application involved the work of many researchers and numerous collaborations. Lowe [38] published a method for image classification whereby features which are invariant to scale or rotation are extracted from an image and then are compared to a large database of features. When comparing features, a fast nearest-neighbour algorithm is employed to find similarities, followed by a Hough transform to isolate the features belonging to individual objects within the image. The result of this process is then verified with a least-squares analysis and can effectively detect objects or patterns within a cluttered or busy image. Lowe's method, known

as the Scale Invariant Feature Transform (SIFT), has been widely applied in the field of computer vision [39]. SIFT was then utilised in what Sanchez et al [40] refer to as "the most popular image representation for classification": The Bag-of-Visual-Words (BoV) approach.

The BoV approach developed from the Machine Learning Bag-of-Words model: a method used to train Machine Learning classifiers on data sets containing natural language information. Wang et al. [41] explain how this model is used to analyse the frequency of the occurrence of words within a text to capture the information of a document, and how this technique progressed to be used in computer vision. Proposed by Csurka et al. [42] in 2004, the authors show how the Bag-of-Words approach can be applied to images in what they term a *bag of keypoints*. The method employs Lowe's SIFT for feature detection and uses the results of the SIFT to develop a histogram and analyse the frequency of features. The frequency distribution of features could then be compared to distributions learned off-line to find possible classification matches. This method has since become a standard machine learning approach to image classification, with numerous studies developing improvements, optimisations, and other applications of this method [43] [44] [45] [46]. Almost a decade after the work of Csurka et al. was published a novel approach arose, disrupting the status quo within the computer vision community. This new approach led to rapid developments in the field, developments which have made the image classification feature of the writer's application possible. This approach, however, was not widely accepted initially [47].

In 2012, Yann LeCun, current Director of AI Research at Facebook and prolific researcher of convolutional neural networks [48], submitted a paper to the review panel of the 2012 Conference on Computer Vision and Pattern Recognition (CVPR). The method proposed in the paper involved training a neural network to classify images rather than using the other approaches such as the extracting of engineered features and BoV technique, as was typical at the time. The CVPR rejected the paper, offering "very negative reviews" [47], despite the solution showing a significant improvement on the state of the art at that time. The paper was subsequently submitted to the International Conference on Machine Learning of 2012 where it was successfully published [49]. In the following months, AlexNet, the convolutional neural network approach to image classification developed by Krizhevsky et al. [50], achieved record-breaking results in the ImageNet Challenge of 2012: an image classification challenge which has been held annually since 2010 [10]. LeCun credits this achievement with changing the attitude of the computer vision community to convolutional neural networks [47].

The ImageNet Large Scale Visual Recognition Challenge is considered as the gold standard of image classification, using hundreds of classification categories and millions of images, while attracting competitors from around the world [10]. AlexNet was the first convolutional based neural network to be employed at the challenge, and it won by a significant margin with an error rate of 15.3%, whereas the second-place competitor produced an error-rate of 26.2% [50]. In the updated version of their paper which was released last year, Krizhevsky et al. address the rejection of LeCun's paper in 2012, as well as the shift in perspective which occurred within the computer vision industry following their success [8]. They express how the computer vision community believed that classification could only be performed using a vision system which was "hand-designed" and would never be achieved by simply supplying a neural network with sample data and labels. The authors posit that, once the computational power and required data is available, general-purpose machine learning is more applicable to large-scale problems than a programmer with domain expertise, which is something the computer vision community failed to recognise at that time.

AlexNet, known as SuperVision by Krizhevsky et al. [8], inspired a series of innovations in the application of convolutional neural networks to image classification. In the ImageNet Challenge of 2014 the record set by AlexNet was beaten by another convolutional neural net: Inception. This neural network, developed by Szegedy et al. [9], is reported by the authors to produce significantly more accurate results than AlexNet, while using 12 times fewer parameters. This Google led research progressed to further developments, with another team of Google researchers focusing on producing smaller, more efficient neural networks for mobile vision. Nearly a year ago, Howard et al. produced an effective, efficient and "lightweight" approach to convolutional neural networks, enabling its application on mobile devices: MobileNet [51]. Since the publication of Howard et al., the MobileNet model has become readily available with TensorFlow, the Machine Learning library being used in the writer's work. The highly accurate Inception v3 model is also available for deployment through TensorFlow but, due to its comparatively substantial size, will not be considered for this application. Instead, the MobileNet model will be explored as the image classification model for this project.

2.2 User Experience Design

The user experience (UX) of a product, according to Garrett [52], is how it functions in real-world scenarios, in the hands of real users. The objective of user experience design is to marry the form and function of the product or service, while considering the context of its use and the perspective of the user. The result of this design process can be the deciding factor in the success of a product or service, as Garrett gives many examples where customer loyalty and customer conversion rates revolve around the user experience. Garrett's book, which was written to address user experience and user interface design for web applications, quickly became a standard reference in the field. In the years following its publication, with the advent of the mobile application and with the development of a broader interest in UX design, Garrett's work was applied to new and emerging fields. As such, Garrett released an updated edition of his book addressing these new areas of UX and UI design, and it has since been cited over 1900 times [53]. In this book, Garrett asserts that the process of designing an effective user experience depends upon asking the right questions and putting the user at the centre of the design process. This "user-centred design" is what Garrett considers as the most efficient method of user experience design.

The efficacy of a user-focused design process has also been confirmed in many other studies, such as the work of Harris et al [54], Smith et al [55], and Bacha [56], with user experience design often being emphasised as a fundamental focus of a successful product or business. Lin and Cheng [57] state that the design of a product's user experience is crucial to its success within a competitive market. They highlight how an underachieving UX design effort can affect the business' customer retention and revenues, but this is commonly presented within the topic and appears in many of the above-mentioned works. The goal of UX design is to reduce complications within the user-product interaction, ensuring that the customer can use the product with ease. The aim is to produce optimal usability of the product or service, which in turn will increase the user's satisfaction and benefit the business. The International Organization for Standardization (ISO) standard for usability (**ISO 9241-11:2018)** states that, when designing for usability, the product and service should allow "users to achieve goals effectively, efficiently and with satisfaction, taking account of the context of use" [58]. As such, the user's perspective should be considered when developing a prototype or initial version of an application, and the efficiency and effectiveness of the user experience should be measured along with customer satisfaction levels when testing.

To understand the user's perspective and develop an effective UX for a mobile application, the designer must gain an understanding of how the audience will interact with the data that is visually presented to them. As the work of Applen and Stephens highlights [59], people often develop biases which can affect their understanding of visual data. These biases can be culturally inherited or gained through training, as the authors show that people of different professions can interpret information in contrasting manners. The importance of understanding the audience's perspective is further emphasised by the work of Jones [60], which shows that the interpretation of information can change depending on the context, and to improve usability these varying interpretations must be understood. Jones' work also shows that the emotional effect that visual information can have on an audience, which Jones refers to as the "affective value". This can have a significant impact on how information is perceived.

The effect that cultural differences can have on a person's understanding and interpretation of data, and its subsequent effect on user experience, is well documented. Choi et al [61] claim that language only accounts for only 10% of cultural differences, and to successfully deploy localised versions of an application the remaining 90% must be understood. Marcus and Gould [62] found that people have culturally embedded expectations and understandings of visual cues within a web or mobile application. For instance, colours can be interpreted very differently in different cultures, with the colour red being associated with anger or warning in some cultures, yet it represents luck and prosperity in others. In certain scenarios, as Saint-Amant details [63], cultural differences can arise between images that are recognised as representing a specific purpose. He explains how the appearance of a mailbox could change between different cultures, and to effectively send mail in a new culture we should understand the local interpretation of a mailbox. In the application developed in the writer's work, confusion could arise in the use of icons for searching or saving features, therefore the use of such symbols should be researched for each prospective market. This work highlights the importance of creating localised versions of an application when entering new markets, and thoroughly researching the culture beforehand.

Acknowledging this research on user experience design, the findings outlined here have influenced the writer's work. The design of the application has been modified in response to user feedback, and the usability metrics mentioned above have been central to the evaluation of the work. The future work of this product will be an entirely user-centric process also. With a prototype successfully developed, the writer now has the capability to involve the user more and generate more feedback. As will be discussed in the "Future Work" section of this report,

the user experience design principles shown here will be focal in the progression of the work and the deployment of this application in other markets. One additional acknowledgement of UX design should also be mentioned, however.

Ineffective UX design can also lead to misinformation, which presents risks in certain scenarios. For this application, misinformation could lead to health risks for users. For example, this could present a serious threat for diabetic users if they rely on the application to monitor their sugar consumption. In Jones' work [60], he indicates how the representation of visual information can lead to apophenia, a term used in psychology to represent the perception of connections in data or meaning when none exist. This can occur when presenting large data sets or when presenting visual information with minimal labelling. In these cases, the observer must make assumptions based on their understanding of similarly presented visual data, which they have previously seen. That said, Jones also asserts the importance of presenting information in manners that are familiar to the user, such as pie or bar charts, as this enhances the ease at which the information can be transmitted.

The work of Hildon et al. [64], which analysed multiple studies on the visual depiction of health-related information, found that the optimal technique is to employ bar charts. Bar charts were found to be both best received by the audience and, in some instances, to produce the best results in terms of transmitting information. Hildon et al. found that bar charts produced the best results among diabetic users when portraying information related to health risks. Therefore, with diabetics being a user type of this application, the use of bar charts will be explored in future versions of this application as means of displaying information.

Chapter 3 - Technical Background

This chapter provides a basic level of knowledge of the underlying technologies used in this work. This information is provided to aid in the understanding of the remainder of the report. The Android platform and development environment are introduced in Section 3.1, with some description of the most critical aspects of the Android platform as well as an overview of the programming language used. Section 3.2 introduces the reader to the TensorFlow Machine Learning library with an overview of its history and operation. Finally, Section 3.3 details the USDA Food Composition database and its Application Programming Interface (API). This is the RESTful database from which the nutritional information of each food item is requested. As the writer requests this information in the JSON format for this application, an explanation of the JSON format and the methods used to parse it are provided in this section also.

3.1 Android

This section covers the Android terminology used throughout the work. It is designed to give the reader an understanding of the principles and aspects of Android development which are used in this project. Included in this section is the recent development of the Android framework, Android Room, which was announced at the Google I/O Conference of 2017 [65]. Android Room is a new library for data persistence and was used for the data storage capabilities of this project.

3.1.1 Android Studio

Android Studio is the official integrated development environment (IDE) for Android [66] and is used throughout this work. It is based on the Java IDE "IntelliJ IDEA", developed by Jetbrains [67]. Android Studio is fully optimised for Android development, with an "Intelligent code editor" which predicts the intention of the developer, completing code and checking for proper declarations and practices [68]. It replaced the Eclipse Android Development Tools with its first stable build, version 1.0, being released in December 2014 [69]. The current stable build is version 3.1 which was released on the 26th of March 2018 [70]. It is available on Windows, Linux and macOS, and both the Windows and Linux versions were used in the development of this application.

The layout of the applications files within Android Studio is known as the "project structure". The project structure of this application is shown at the beginning of Chapter 4, and this contains the entirety of the work developed throughout this project. The overview shown in Chapter 4 displays a series of nested folders containing the applications files. These folders are called "packages" in Android Studio and will be referred to as such throughout this work.

3.1.2 Programming Language

Java has been the official language of Android since its inception [69]. C++ is also supported in Android, however the modern, concise programming language named Kotlin, developed by Android Studio's co-developers JetBrains, became an official language of Android on the 17th of May 2017 [71]. Java will be used throughout this project so a brief synopsis of how it is used will be presented here.

Java is an object-oriented language that operates via classes, objects, and methods. Within Java classes, the developer can utilise standard Java "methods" (i.e. **`toString()`**, **`printIn()`**), and define or "declare" their own methods. Methods are series of statements and actions that work as one operation to produce some result. These methods can be called within the Java class and, depending on the method's modifier (detailed below), may be called from within other classes. If a method is declared as **`void`**, this indicates a class which does not require a return statement. If, however, a method is declared with a specific Java object type, this method must be called with the same object type assigned to it. In brief, a void method performs a task without returning a specific result to the place where it was called in the code, whereas methods with return statements must return the assigned object type:

```
//Declaring methods
//This void method does not return anything
void doSomeThing(){
…
}

//This method must return a String object
String returnAString(){
String thisString = "some text"
} return thisString

//Calling methods
…
doSomething()
String myString = returnAString()
```

3.1.3 Activities, Fragments, Layouts & Intents

In the Android framework, the interface that the user interacts with is controlled by a Java class known as an Activity. Almost all activities on Android call for user interaction, and it is the Activity class that is responsible for creating and adapting the UI windows on the Android device as needed [72]. Activities can produce and manage smaller UI elements known as Fragments. Data can be passed through activities using Intents, which are used to launch other activities. The Activity creates the user interface window through the *void* command **`setContentView(View)`**, where the View here is often a layout inflated from an .xml layout file through specifying the layout resource ID:

```
void setContentView (int layoutResID)
```

3.1.4 Android Manifest

The Android Manifest (AndroidManifest.xml) is an XML file which contains a log of all activities within the application. When an activity calls an Intent to launch another activity, the app will crash if the activity is not registered in the Android Manifest. Amongst the log of activities in the manifest file the developer indicates which activity should be launched when the app is initiated. This is provided through an intent filter.

3.1.5 Data Persistence – Android Room

Data persistence (the storing of data) has been performed in Android with SQLite since the first iteration of the operating system [73]. SQLite is an open-source library that implements a compact, "self-contained" SQL database [74]. Developers create SQL databases by declaring the database and defining each table and the data to be stored in each of its columns with commands of the form:

```
CREATE DATABASE databasename;
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
); [75]
```

At the Google I/O Conference in 2017, Google announced that the new Architecture Components Library was being released for Android, bringing with it new elements such as a class for handling Lifecycle events (described below) and the new Room Persistence Library [65]. Room provides a simpler method for applying SQLite databases by, among other improvements, reducing the amount of code required to implement data persistence, optimising network connectivity and data syncing, and improving the performance of data retrieval. The author had previously developed applications using raw SQL code and had no experience using Room prior to this work, so some research and self-learning was required to implement the new, recommended library. Due to this the "Android Persistence Codelab" tutorial provided online by Google Developers [76] was relied upon during implementation.

Room is implemented by creating Java classes for the data that must be stored and tagging the classes with **`@Entity`**. This annotation indicates to the application that this class represents fields to be included in the database. For each class marked **`@Entity`**, a table is created in the database with a column defined for each field within the class. Within each entity of the database, one of the fields must be declared as the primary key of the table, even if the table has only one field. This is done by annotating a field with @PrimaryKey, and can set to automatically generate and increment numbers for ID numbers by setting autogenerate = true. The following code is an example of an entity to store a user's first name and last name while automatically creating an ID number for them:

```
@Entity
class User {

    @PrimaryKey(autogenerate = true)
    public int id;


    public String firstName;
    public String lastName;
```

Once data has been stored in the database, the information is retrieved by using a Data Access Object (or Dao). These are Java Interface classes annotated with @Dao and consist of methods with SQL statements used to access and edit the information stored in the application database. To achieve this, methods are declared within the Dao that are annotated with @Query, @Insert, or @Delete. The following is an example of a method that would query the database and return a list of objects of the User class defined above:

```
@Query("SELECT * FROM user WHERE user_name LIKE :name AND last_name " +
"LIKE :last")

public abstract List<User> findUsersByNameAndLastName(String firstName, …
String lastName);
```

3.2 TensorFlow

TensorFlow is an open-source library which allows users to train and employ complex machine learning models using a standard computer. As stated by the TensorFlow team in their "Preliminary White Paper" on the 9th of November 2015: "TensorFlow… is an interface for expressing machine learning algorithms, and an implementation for executing such algorithms" [23]. On the same day, as TensorFlow was publicly launched, Sundar Pichai, the CEO of Google, wrote a blog post on the official Google blog stating that TensorFlow is a "faster, smarter, and… highly scalable machine learning system—it can run on a single smartphone or across thousands of computers in datacenters" [77]. Pichai mentions the power and speed of TensorFlow, stating that it can train neural networks five times faster that Google's first-generation machine learning system, DistBelief.

TensorFlow emerged from DistBelief, the Google Brain team's early work on deep learning. DistBelief was built in 2011 as a framework which utilised tens of thousands of CPU cores to train large models of machine learning and AI neural networks [23]. It was Google's "first-generation scalable distributed training and interference system" [78]. The year following its internal launch at Google, researchers were employing DistBelief across networks of thousands of computers, training the largest deep network of its time [78]. DistBelief was closed-source and very experimental but was utilised across Google and Alphabet companies, being implemented with features and services such as Google Search, Maps, Translate and YouTube, among others [23].Over years of iterations, DistBelief was refined and optimised by these researchers, with many novel techniques and advancements in the field of deep learning being discovered. By 2009, the Google Brain Team had achieved significant reductions in the error rate of the neural networks, while also producing a simpler, faster, and more robust codebase. This was publicly launched in 2015 as TensorFlow.

Since its experimental launch in 2015, version 1.0.0 was released in February 2017, with the latest stable build (version 1.6.0) being released on the 28th of February 2018. It has become a well-received and well-adopted machine learning library, with applications in both open-source projects and private ventures [79], as well as a continuously-growing and active community [80]. The codebase is readily available on GitHub under the Apache 2.0 license, allowing anyone to contribute to it [81] [77]. To date it has over 30,000 commits from over 1,300 contributors, across 52 releases. As development progressed mobile support was gradually implemented, with support for both Android and iOS.

On the 14th of November 2017, the new, mobile-optimised TensorFlow Lite was released, expanding the reach of the platform and unlocking the potential for TensorFlow to solve more use cases [82] [83]. This new version of TensorFlow is a "lightweight solution" and allows the machine learning interference of TensorFlow to be applied on devices with "low latency and a small binary size", i.e. mobile devices and embedded hardware such as the Raspberry Pi [82].

TensorFlow is a Python based library which operates via data flow graphs. These graphs are representative of a dataflow computation, consisting of nodes and connections in a manner displayed in the example of Figure 2. Each node on the data flow graph represents a mathematical operation whereas the edges represent the "tensors" between each node. These tensors are multidimensional data arrays that flow from node to node as input/output values. The underlying element type of the tensors is decided during the construction of the data flow graph.

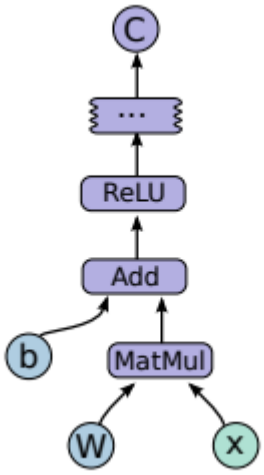

Figure 2: TensorFlow Computation Graph Example [23]

```
import tensorflow as tf

b = tf.Variable(tf.zeros([100]))                    # 100-d vector, init to zeroes
W = tf.Variable(tf.random_uniform([784,100],-1,1)) # 784x100 matrix w/rnd vals
x = tf.placeholder(name="x")                        # Placeholder for input
relu = tf.nn.relu(tf.matmul(W, x) + b)              # Relu(Wx+b)
C = [...]                                           # Cost computed as a function
                                                    # of Relu

s = tf.Session()
for step in xrange(0, 10):
  input = ...construct 100-D input array ...        # Create 100-d vector for input
  result = s.run(C, feed_dict={x: input})           # Fetch cost, feeding x=input
  print step, result
```

Figure 3: TensorFlow Code Fragment for Graph of Figure 2 [23]

3.3 USDA Food Compositions Database API & JSON

The United States Department of Agriculture's (USDA) Food Composition Database API was implemented in this application to facilitate the retrieval of nutritional information. This database was formed from the work of the USDA's Nutrient Data Laboratory in Beltsville, MD, and contains a wealth of nutritional knowledge [84]. The services used in this application are the "Food Search" and "Nutrient Report" facilities. The "Food Search" allows a text search query for a food item and returns any items in their database with a similarity to the search query. An example of this is if a user inputs the text "apples" as a search query, many items are returned such as "applesauce" and others. The "Nutrient Report" is dissimilar in that it has no leniency in its search queries: the user must enter an exact search term in the form of a unique identifier code, known by the system as an "NDB no." or "ndbno". This name represents "Nutrient DataBase no.", and every food in the database has their own unique code. This code is what links each food item to its nutritional information. The following is an example of the URLs used in each search type:

Food Search:

https://api.nal.usda.gov/ndb/search/?q=apples&ds=Standard_Reference&format=json&api_key=DEMO_KEY

Nutrient Search:

https://api.nal.usda.gov/ndb/reports/?ndbno=01009&type=f&format=json&api_key=DEMO_KEY

Both URLs shown above begin with what will be referred to as the base URL of the database: "https://api.nal.usda.gov/ndb/". Following the base URL, the different search queries diverge. The Food Search query is first identified by the database through the term "search/?", whereas a Nutrient Search, which generates a "Nutrient Report", is identified by the term "reports/?". Search queries follow these terms: for the Food Search it shows the query "q=", with the text of the food being searched for inputted as its parameter, such as "apples" in the example above; for the Nutrient Report, as mentioned above, a specific ndno must be entered following the query "ndbno=".

Following the search query for the Food Search is the element “ds=”, which represents “database source”. This can be set to either “Standard Reference” or “Branded Food Products”. The following search query for the Nutrient Search is “type=”. This query is used to indicate the type of report the user wants returned, ranging from “b” for basic, “f” for full, or “s” for stats. In this application the basic type will always be requested as the additional information provided in the full and statistical types are not needed and could result in unnecessary effects on performance.

The final two queries of each search type are “format=” and “api_key=”. The format query dictates the format that the response data is returned in, and the user has the option of either JSON or XML. Due to prior experience with parsing JSON data in Android, the author decided to request JSON from this database. The code used to parse the results is shown in the “Utilities” section of this work, and an example of the JSON response from each of the URL requests shown above can be seen in Figure 4 and Figure 5.

A JSON String usually contains multiple JSON objects and at least one JSON array. JSON objects (contained within braces: {}) are essentially single classes or instances of a class. A JSON array (contained within brackets: []) is an array which consists of uniformly structured JSON objects. For example, in Figure 4 below, the JSON array titled “item” is an array of JSON objects, each with the fields: “offset”; “group”; ”name”; “ndbno”; and “ds”. Through the techniques that will be shown in the Utilities section, individual JSON objects can be isolated from an area and the values stored in its fields can be retrieved. For example, the first food item in the “item” array shown below could be accessed, and its name “Croissants, apple” could be retrieved and used, or its ndbno number “18240”. Once the array is isolated the developer can iterate through the entire array retrieving the required data of each object in the array, as is done in this application.

```
{
    "list": {
        "q": "apples",
        "sr": "28",
        "ds": "Standard Reference",
        "start": 0,
        "end": 100,
        "total": 100,
        "group": "",
        "sort": "r",
        "item": [
            {
                "offset": 0,
                "group": "Baked Products",
                "name": "Croissants, apple",
                "ndbno": "18240",
                "ds": "SR"
            },
            {
                "offset": 1,
                "group": "Baked Products",
                "name": "Strudel, apple",
                "ndbno": "18354",
                "ds": "SR"
            },
            {
                "offset": 2,
                "group": "Baby Foods",
                "name": "Babyfood, apples, dices, toddler",
                "ndbno": "03115",
                "ds": "SR"
            },
            {
                "offset": 3,
                "group": "Baby Foods",
                "name": "Babyfood, juice, apple",
                "ndbno": "03166",
                "ds": "SR"
            },
```


Figure 4: JSON Example; USDA Food Search Response

```
{
    "report": {
        "sr": "28",
        "type": "Basic",
        "food": {
            "ndbno": "01009",
            "name": "Cheese, cheddar",
            "ds": "Standard Reference",
            "manu": "",
            "ru": "g",
            "nutrients": [
                {
                    "nutrient_id": "255",
                    "name": "Water",
                    "derivation": "NONE",
                    "group": "Proximates",
                    "unit": "g",
                    "value": "37.02",
                    "measures": [
                        {
                            "label": "cup, diced",
                            "eqv": 132.0,
                            "eunit": "g",
                            "qty": 1.0,
                            "value": "48.87"
                        },
                        {
                            "label": "cup, melted",
                            "eqv": 244.0,
                            "eunit": "g",
                            "qty": 1.0,
                            "value": "90.33"
                        },
                        {
                            "label": "cup, shredded",
                            "eqv": 113.0,
                            "eunit": "g",
                            "qty": 1.0,
                            "value": "41.83"
                        },
```


Figure 5: JSON Example; USDA Food Search Response

Chapter 4 - Application Design

This application was designed with many Java classes of varying purposes. The classes with similar purposes are grouped together in the applications project structure (Figure 6). The adapters, activities, and fragments of the application were all grouped in their own packages, while the classes and interfaces used for the applications database were stored in the package labelled “data”. Also stored together are the application’s custom “utilities”, a term which is described in the section of the same name below, as well as the data models used in the application. All Java files related to the image classifier are situated in the “classifier” package within the Java folder, while the TensorFlow build file, graph and labels are stored in the “assets” folder. The “res” folder contains all the resources of the application, such as the XML layout files for each UI element in the “layout” package, and all strings, colours and styles being found in “values”.

In this chapter, the elements directly related to the UI will be discussed along with some of the software design associated with them. The elements of the application which are pertinent to the understanding of the application will be discussed first, such as the data models used throughout the UI and the custom utilities used. This will be followed by a description of each UI element the user encounters, such as the activities and fragments of the application. As over 7500 lines of code were written for this project, not all the code will be detailed here. For details of other elements of the application, such as the many adapters and layout files which are involved in the UI, please refer to the source code of the project (Appendix A).

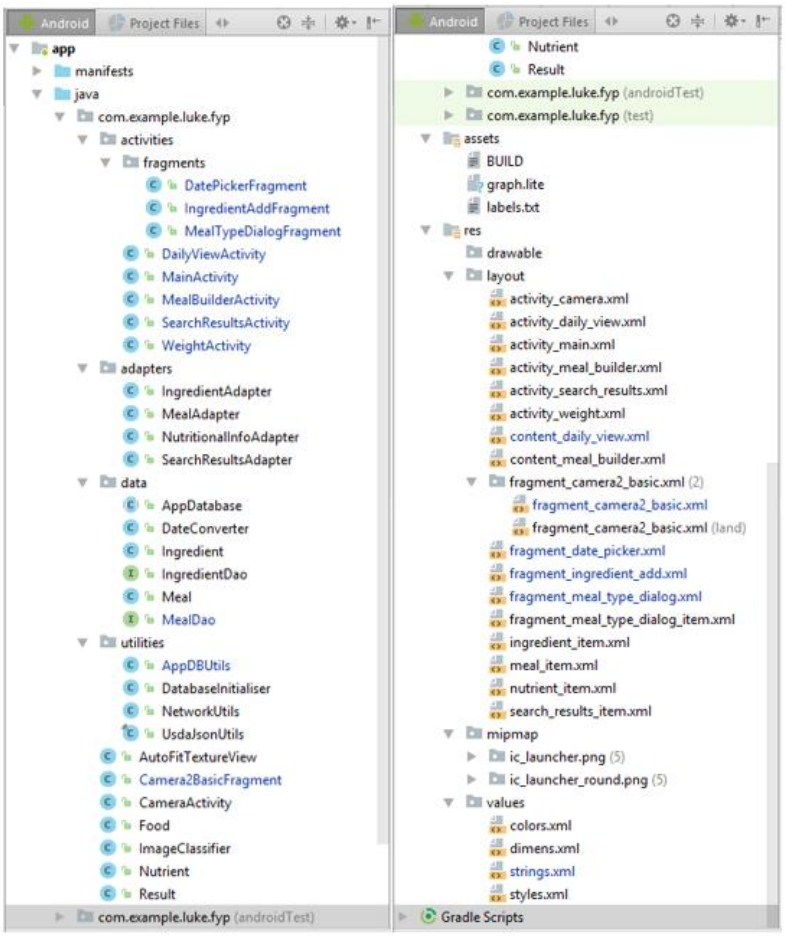

Figure 6: The Project Structure of the Application

4.1 Models

The models within an application are classes that are created to store data temporarily. This data is not data that is committed to the devices storage but is tied to the application Lifecycle, i.e. the data is destroyed when the application is closed [85]. It is data that is used somewhat instantaneously but could be later committed to memory through the use of other classes, as will be seen in Section 4.2 on the "data" package below. For the purposes of this application two models were created to utilise the data returned from the USDA API: **`Food`** and **`Nutrient`**.

Table 1: List of Objects & Variables; Food.java

Modifier	Variable/Object Type	Name
`private`	`String`	`group`
`private`	`String`	`name`
`private`	`String`	`ndbno`

4.1.1 Food

The **`Food`** class is used to store the results of the Food Search request made to the USDA Food Composition Database, described in Section 3.3. The class contains three String objects to store the **`name`**, **`ndbno`** and **`group`** of each food item. This is performed and managed by *getters* and *setters*. The name, as will be seen further in this work, is used to list each item of the response in the UI for the user to select from. When a selection is made, the ndbno is retrieved from the selected **`Food`** object, which is then used to form a URL to generate a Nutrient Report from the USDA database. The **`group`** string is not currently used but was included to implement a sorting feature in future builds. Some examples of the food groups that can be assigned can be seen in the JSON response in Figure 4, shown above in Section 3.3. This data model is used by the **`UsdaJsonUtils`** utilities file to store the parsed JSON response of the USDA Food Search. The model is then used again in the **`SearchResultsActivity`** and its data adapter **`SearchResultsAdapter`** to apply this data to the UI.

4.1.2 Nutrient

The **`Nutrient`** data model was created to store the nutritional information of the selected food. It stores the nutrients **`name`** (i.e. fat, protein etc.), **`unit`** (g, mg, µg, kcal, or IU for some vitamins), and **`value`**. The value here refers to the quantity of the nutrient that is present in 100g of the food, in terms of the unit provided. All these parameters are stored in **`String`** objects within the **`Nutrient`** class and are accessed and defined by *getters* and *setters*. As will be seen, this data model is first used in the **`UsdaJsonUtils`** file to store the processed results of the Nutrient Report. It is then applied in the **`WeightActivity`** and its corresponding data adapter, **`NutritionalInfoAdapter`**, to display the nutritional information of a food and alter the nutrient values as the weight of the ingredient increases or decreases.

Table 2: List of Objects & Variables; Nutrient.java

Modifier	Variable/Object Type	Name
`private`	`String`	`name`
`private`	`String`	`unit`
`private`	`String`	`value`

4.2 Data

The “data” package within the application contains all Java classes and interfaces required to store and retrieve data from the device’s memory. To access and store data on an Android device the following code must be added to the applications AndroidManifest.xml file:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

This code represents the “permissions” that the user must grant to the application to allow it to work with the device’s memory.

4.2.1 DateConverter

The DateConverter class used here is the same as that used in The Android Open Source Project and Google Developers codelab on Room Persistence implementation, licensed under Apache 2.0 open-source licensing. The code presented here is unedited from that provided. The purpose of the class is to convert Java **`Date`** objects to **`Long`** objects to act as timestamps when an ingredient or meal is being added to the database. The converter also does the opposite and converts the Long timestamp to a **`Date`** object when retrieving data. This is achieved with two methods: **`toDate`**, which takes a **`Long`** timestamp and returns a **`Date`** object; and **`toTimestamp`** to produce a **`Long`** object from a **`Date`** object:

```
@TypeConverter
public static Date toDate(Long timestamp) {
    return timestamp == null ? null : new Date(timestamp);
}

@TypeConverter
public static Long toTimestamp(Date date) {
    return date == null ? null : date.getTime();
}
```

4.2.2 Meal

This public Java class is used to store the data of each meal the user creates in the application database. It stores the title of the type of meal being created (i.e. breakfast, lunch, dinner, or snack) as a text string while also storing an integer value representing the meal type in a separate field, ranging from 1 to 4. The class also has fields to store the date and time the meal was created, which is passed as a **`Date`** object and converted to a **`Long`** timestamp with the **`DateConverter`** class, as well as **`Double`** objects to store the total calories, sugar, sodium, and macronutrients present in the meal.

As the **Meal** class is a Room database entity it is annotated with **@Entity** (described above), and the use of the **DateConverter** class is annotated as a **@TypeConverter**:

```
@Entity
@TypeConverters(DateConverter.class)
public class Meal implements Comparable<Meal>{
```

The **Meal** class implements the Java **Comparable<T>** interface which allows a list of objects to be sorted or ordered by what Oracle refers to as the class's "natural ordering" [86]. It is used in this application to sort the user's meals on each day in the order breakfast > lunch > dinner > snacks. This is to produce a consistent appearance of each day regardless of the order the meals were added to database. The implementation of this with the UI will be seen in the section on the **DailyViewActivity**. As the interface is applied to the **mealType** integer the following code must be included in the **Meal** class:

```
public int compareTo(@NonNull Meal m) {
    return this.mealType - m.mealType;
}
```

As well as the above, the **Meal** class automatically generates and stores a unique ID for every meal created. This is an important feature as it is used to retrieve specific meals and, as will be seen in the following section, is what is used to connect the ingredients to the meal they belong to:

```
@PrimaryKey(autoGenerate = true)
public long id;
```

Table 3: List of Objects & Variables; Meal.java

Modifier	Variable/Object Type	Name
public	long	id
public	int	mealType
public	String	mealTitle
public	Date	mealTime
public	Double	totalCalories
public	Double	totalFat
public	Double	totalSats
public	Double	totalCarbs
public	Double	totalSugars
public	Double	totalProtein
public	Double	totalSodium

The variable used in the Meal class are shown in Table 3, and it can be noted that the Double object is used to store that total values of the meal. The importance of using the double type is expressed in the following section, however it can be highlighted here that the Double class object was used rather than a double primitive variable type. This decision was made as the Double class is provided with methods to convert double values to strings, which will be used for the UI of the application and discussed in the section on the **`WeightActivity`**.

4.2.3 Ingredient

The **`Ingredient`** class, similarly to the **`Meal`** class, generates and stores its own IDs, and stores double values for calories, sugars, sodium, and macronutrients. The double values here, however, store indexes of the amount of each macronutrient per 100g of the ingredient, whereas the **`Meal`** class stores the totals for the entire meal. The totals for each meal are found by calculating the total for each ingredient of the meal, and then adding these together. The total values for each ingredient is found dynamically by multiplying the indexes of each ingredient's nutrient with the double value representing the ingredient's weight. For example, a user might log a meal with a single ingredient: 50g of bread, which has a calorie index of 120 calories per 100g. In this example, the total calories of the meal would be stored as 60kcal, yet the calorie index of 120 would be stored in the **`Ingredient`** class under the **`Double`** object **`calories`**. This is stored along with the index values for the other nutrients shown in Table 4.

The weight of the ingredient in grams (value of 50 in the example used above) is stored with the **`Double`** object **`weight`**, which the user can change or edit at a later stage. This method of storing the weight and the caloric indexes, rather than the final values of the ingredient's nutrients, allows the ingredient's weight to be altered without issues or loss of accuracy. If the user changes the weight, like if the user decided to have another slice of bread in the example above, the total values of the meal are recalculated dynamically without loss of accuracy. The implementation of this will be seen in the **`WeightActivity`**.

With accuracy and reliability in mind, it was decided to use **`Double`** objects for the weight and macronutrient indexes as this would facilitate decimal places. Without the use of **`Double`** objects or double variable types, it would be required to round the values to the nearest figure following the multiplication operations between the ingredient's weight and its nutrient index values. While this solution may not have a significant effect on macronutrients or calories, it has a drastic effect on sodium and other nutrients that can be retrieved from the USDA database. For example, the Food Safety Authority of Ireland states that the recommended daily

allowance of sodium is 1.6g [87]. If this was rounded up to 2g to be stored as an integer it would be 40% over the recommended amount, an amount of sodium which has serious implications for cardiovascular health [87]. Conversely, if a user was to log four meals with 0.45g of sodium each, this could potentially be rounded down to 0.0g for each meal, resulting in the user unwittingly exceeding the recommended daily allowance of sodium, endangering their health in the process.

One of the most important fields of the Ingredient class is the long variable meal_id. This is used to brand each ingredient with its corresponding meal, linking the two data models of meals and ingredients, and is used any time a meal's ingredients are requested. It can also be noted that the Ingredient class does not need a TypeConverter, as it is not necessary for each ingredient to be timestamped since the meal is designed to be timestamped. The Ingredient class also has a field for storing the ndbno (unique identifying number for USDA database, described above) of the food, which can be used to retrieve further nutritional information such as vitamins and minerals.

Table 4: List of Objects & Variables; Ingredient.java

Modifier	Variable/Object Type	Name
public	long	id
public	long	meal_id
public	String	name
public	Double	weight
public	String	ndbno
public	Double	calories
public	Double	fat
public	Double	saturates
public	Double	carbs
public	Double	sugar
public	Double	protein
public	Double	sodium

4.2.4 MealDao

The MealDao.java file is the Data Access Object interface used to access and edit the database information relating to meals. All non-void methods of the interface return either **`Meal`** class objects or data of a **`Meal`**, except for the **`findMealIngredientsByDayandType`** method which returns a list of the **`Ingredient`** class. method which returns a list of the **`Ingredient`** class. Queries which use **`Date`** objects can be annotated with

@TypeConverters(DateConverter.class) to instruct they require the **Date** objects to be converted to **Long** objects, e.g.:

```
@TypeConverters(DateConverter.class)
@Query("SELECT * From Meal " +
        "WHERE Meal.mealTime > :dayStart " +
        "AND Meal.mealTime < :dayEnd")
List<Meal> findAllMealsByDay(Date dayStart, Date dayEnd);
```

Instead of the above, where each method that requires the type converter is annotated, it was decided to annotate the entire Dao. This provides the functionality of the TypeConverter to the methods that need it while reducing unnecessary code:

```
@Dao
@TypeConverters(DateConverter.class)
public interface MealDao {
…
```

Table 5 lists all the methods present in this Data Access Object, along with a description of their purpose. One of the most notable is the method **insertMeal**. This method inserts a meal into the database while returning the ID of the newly created meal. As will be seen in the activities below, this is a very useful feature as the returned ID can be used as the ingredient's **meal_id** parameter, without the need for an additional query after insertion. The inclusion of **(onConflict = OnConflictStrategy.REPLACE)** with this method allows meals to be updated and overwritten also.

Table 5: List of Methods; MealDao.java

Method Name	**Return Type**	Description
`findAllMealsByDay( Date dayStart, Date dayEnd)`	`List<Meal>`	Selects any **`Meal`** between the times of **`dayStart`** and **`dayEnd`**.
`retrieveMealType(long mealId)`	**`int`**	Selects the meal type from the **`Meal`** that matches the meal ID passed.
`retrieveMealType(long mealId)`	`Date`	Selects the meal **`Date`** object from the meal that matches the meal ID passed.
`findMealIngredientsBy DayandType(int mealType, Date dayStart, Date dayEnd)`	`List<Ingredient>`	Uses the SQLite "INNER JOIN" to retrieve the list of ingredients from the meal that matches the meal type and date passed.
`findMealIdByDayandTyp e(int mealType, Date dayStart, Date dayEnd)`	**`long`**	Returns the ID of the meal that matches the type and day passed
`insertMeal(Meal meal)`	**`long`**	Inserts a single meal into the database. If a conflict arises (i.e. meal with the same ID already exists) it overwrites the existing meal. Most importantly, it returns the ID of the new meal.
`deleteAll()`	**`void`**	Deletes all meals from database – used when testing the database.
`deleteMealById(long mealId)`	**`void`**	Deletes the meal in the database which has the same ID as that passed.

4.2.5 IngredientDao

The IngredientDao.java file is the Ingredient class' equivalent of the MealDao. Table 6 displays all the methods used in this Data Access Object interface along with their return conditions and a brief description of each:

Table 6: List of Methods; IngredientDao.java

Method Name	**Return Type**	Description
`loadIngredientById(long id)`	`Ingredient`	Selects the **`Ingredient`** which has the ID passed to the method
`findIngredientsOfMeal(long mealId)`	`List<Ingredient>`	Retrieves a list of **`Ingredient`** objects which have a **`meal_id`** parameter that matches the ID passed to the method.
`insertIngredient(Ingredient ingredient)`	**`long`**	Inserts an **`Ingredient`** into the database and returns the ID of the new addition. Replaces **`Ingredient`** if a conflict occurs.
`deleteAll()`	**`void`**	Uses the SQLite "INNER JOIN" to retrieve the list of ingredients from the meal that matches the meal type and date passed.
`deleteIngredientById(long ingredientId)`	**`void`**	Finds the **`Ingredient`** in the database which has the ID passed and deletes it.
`deleteIngredientByMealId(long mealId)`	**`void`**	Deletes any ingredients belonging to the meal with the **`mealId`** passed to the method

4.2.6 AppDatabase

This public abstract class manages the application's database. It is an adaptation of the class provided from the Android Open Source Project and Google Developers codelab tutorial for implementation of the Room Database functionality. Where the provided example code is developed to store the classes "Book", "Loan" and "User" as the database entities, the version used in this application stores the classes **`Meal`** and **`Ingredient`** as described above. This class is annotated using **`@Database`** and its purpose is to establish order between the Data Access Object interfaces, while defining the database version. This version number is incremented when the database is updated and is used when implementing database migrations [76]. The **`MealDao`** and **`IngredientDao`** are assigned public abstract methods which allow them to be accessed throughout the application, name **`mealModel()`** and **`ingredientModel()`** respectively.

4.3 Utilities

Code that is designed to perform standard tasks within an operating system (OS) is often referred to as a part of the "utilities" of the OS [88]. Android, like many other operating systems, has a series of inbuilt utilities for manipulating data, working with different file formats, and many other common functions [89]. It is typical in Android development, and in development on other platforms, that the developer will design small, custom pieces of code that perform a concise task which may be applied throughout the application. These pieces of code will often be considered as custom "utilities" by the developer (or simply "utils" [90]), and it is considered good practice to store these together in a single package/folder within the directory. As can be seen in the directory shown in Figure 6, the custom utilities of this application are all housed within the aptly named package "utilities". These utilities are used at various points throughout the code and hence will be discussed here before the activities that use them.

4.3.1 AppDBUtils

The AppDBUtils.java file contains the utilities created to interact with the application's database. These utilities are the methods which operate between the UI and the database: retrieving, adding, and deleting data while providing the results of these operations to the UI. The activities of the UI can make calls to any of the 16 public static methods of **`AppDBUtils`** to access the application database. In turn, these utilities call methods of either the **`MealDao`** or **`IngredientDao`** Data Access Object interfaces through the public abstract methods **`mealModel()`** and **`ingredientModel()`**, which were declared in the AppDatabase.java file.

One of the first methods of this file which will be used in the UI is the **`makeBlankMeal`** method. This method uses the SQL **`@Insert`** method **`addMeal`** declared in the **`MealDao`** to insert a blank meal into the database. While it may seem fruitless, this action reserves an entry in the database for the meal being created and returns the ID of this reserved location. This is an important design as the UX flows from day to meal to ingredient, and each ingredient must bear the ID of the meal it belongs to. Hence, the meal is reserved, ingredients created, and then the meal updates lastly, before the user is returned to the daily view. The following code shows the **`makeBlankMeal`** method:

```
public static Long makeBlankMeal(AppDatabase db, int mealType, Date
mealTime) {

    Meal meal = makeMeal(0, mealType, mealTime, null, null, null, …
null, null, null, null);
    return addMeal(db, meal);
}
```

In the example above, the first parameter that the method requires is an instance of the **AppDatabase** class. This is the same for all methods which require access to the application database. Following this parameter, the method requires an integer value representing the meal type and a **Date** object, both of which are passed to the method **makeMeal** which creates a **Meal** object, which is then committed to the database via the **addMeal** method.

```
private static Meal makeMeal(final long id, final int mealType, Date
mealTime, final Double calories,
                             final Double fat, final Double sats,
final Double carbs, final Double sugars, final Double protein, final
Double sodium) {

    String mealTitle = getTitleFromInt(mealType);

    Meal meal = new Meal();
    meal.id = id;
    meal.mealType = mealType;
    meal.mealTime = mealTime;
    meal.mealTitle = mealTitle;
    meal.totalCalories = calories;
    meal.totalFat = fat;
    meal.totalSats = sats;
    meal.totalCarbs = carbs;
    meal.totalSugars = sugars;
    meal.totalProtein = protein;
    meal.totalSodium = sodium;
    return meal;
}

public static Long addMeal(final AppDatabase db, final Meal meal){
    return db.mealModel().insertMeal(meal);
}
```

This was designed in this way so that this data would be stored immediately as a meal is created, and then retrieved when updating the meal.

4.3.2 DatabaseInitialiser

This class is utilised to populate the database with "dummy" data for testing purposes. Upon application launch it is triggered to initialise the database and insert a number of ingredients and meals, but as it does not contribute to application functionality (and would not be included in an official release) it will not be discussed. Please refer to the DatabaseInitialiser.java file in the Appendix A for further information.

Table 7: List of Methods; AppDBUtils.java

Method Name	Description
`ingredientsToMeal`	Takes a list of **`Ingredient`** objects, a long meal ID, an int for the meal type and the **`Date`** of the meal and creates a **`Meal`** object.
`makeTimestamp`	Sets the day, month, and year of a **`Calendar`** instance to the int values passed; returns **`Date.`**
`makeBlankIngredient`	Adds an **`Ingredient`** with no data to the database, returns new ID as long.
`makeIngredient`	Creates an **`Ingredient`** object from ID from method above plus the ID of the parent **`Meal`**, along with ndbno, weight, calories, sugar, sodium, and macronutrient values.
`addIngredient`	Adds **`Ingredient`** to database, returns ID as long.
`deleteIngredientWithId`	Deletes the **`Ingredient`** with the provided ID.
`deleteIngredientWithMealId`	Deletes any ingredient belonging to the meal with the ID passed.
`returnIngredientsWithMealId`	Retrieves a List of **`Ingredient`** objects which match the ID passed to their **`meal_id`** parameter.

makeBlankMeal	Adds an empty meal to the database, returns new ID.
makeMeal	Private static method which creates a **Meal** object from inputted values.
addMeal	Adds completed **Meal** to the database, returns ID as **Long**.
deleteMealWithId	Removes the meal with the ID passed from the database.
returnTypeOfMealWithId	Fetches the integer value representing the type of meal; matches meal ID to ID passed.
returnTimeOfMealWithId	Fetches time of meal as a Date object from the provided meal ID.
returnIngredientsOfMealTypeAndDay	Searches **Meal** model by type and day and returns a list of ingredients from matching meal.
returnMealIdFromTypeAndDay	Searches database y type and day and returns ID of matching meal.
getTitleFromInt	Private method which produces a string output for the **mealTitle** parameter based on the **mealType** integer.

4.3.3 NetworkUtils

The **NetworkUtils** are utilities which connect to the internet. As there are only two types of network requests made from this application, for the Food Search and Nutrient Report of the USDA database, only two different URL structures are required. As described in the Background Theory (Section 3.3) on the USDA API, both URLs require the same base URL and the API key, but different queries and parameters. As this is the only API used and no other network requests are currently needed for the application, the base URL of the USDA database was assigned to a **String** named ***BASE_URL*** while the writer's API key was assigned to the **String** ***apiKey.*** If more network requests are needed in future builds these variables will be prefixed with the API they are used by.

A list of the objects used in this class can be seen in Table 8 while the methods are summarised in Table 9. As each object used is a **`static final`** field, where the combination of the **`static`** and **`final`** modifiers defines a constant, the contents of each cannot change. As such, Table 8 shows the contents of each of these **`private static final String`** objects alongside their names. The queries shown do not change between searches, only the parameters passed to the URL-forming methods may differ, i.e. the ndbno number or the text of the food the user is searching for.

Table 8: List of Constants Strings; NetworkUtils.java

Name	Contents
apiKey	Private API key for USDA database.
BASE_URL	**`"https://api.nal.usda.gov/ndb/"`**
REPORTS	**`"reports/?"`**
SEARCH	**`"search/?"`**
SEARCH_QUERY	**`"q"`**
BRAND_QUERY	**`"ds`**
API_QUERY	**`"api_key"`**
TYPE_QUERY	**`"type"`**
FORMAT_QUERY	**`"format"`**
NDBNO_QUERY	**`"ndbno"`**
type	**`"b"`**
format	**`"json"`**
brand	**`"Standard Reference"`**

Table 9: List of Methods; NetworkUtils.java

Method Name	Description
`makeNdbnoUrl`	Uses the Android URI builder to create a URI with the queries required for the USDA Nutrient Report, along with the respective parameters for each query. Returns as **`URL`**.
`makeSearchUrl`	Same function as above, except this method forms the URL for the USDA Food Search.
`getResponseFromHttpUrl`	This method uses the **`HttpURLConnection`** class to retrieve the response of the URL request as a **`String`**.

The first two methods of the class handle the formation of the two URL variants, while the third method converts the response of either to a **`String`** object. The URLs are formed by the Android **`Uri.Builder`** class [91], which is designed here to parse the ***`BASE_URL`*** with either the ***`REPORTS`*** or ***`SEARCH`*** constant, then append the required search queries and parameters and build the Uniform Resource Identifier, or URI. A URI is comparable to a URL except its purpose is internal where a URL is used for external requests. Once the URI is built it is converted to a URL using the **`toString()`** method. This can be seen in the methods shown in the code example below.

```
public static URL makeNdbnoUrl(String ndbno) {
    Uri builtUri = Uri.parse(BASE_URL + REPORTS).buildUpon()
            .appendQueryParameter(NDBNO_QUERY, ndbno)
            .appendQueryParameter(TYPE_QUERY, type)
            .appendQueryParameter(FORMAT_QUERY, format)
            .appendQueryParameter(API_QUERY, apiKey)
            .build();
    URL url = null;
    try {
        url = new URL(builtUri.toString());
    } catch (MalformedURLException e) {
        e.printStackTrace();
    }
    return url;
}

public static URL makeSearchUrl(String search) {
    Uri builtUri = Uri.parse(BASE_URL + SEARCH).buildUpon()
            .appendQueryParameter(SEARCH_QUERY, search)
            .appendQueryParameter(BRAND_QUERY, brand)
            .appendQueryParameter(FORMAT_QUERY, format)
            .appendQueryParameter(API_QUERY, apiKey)
            .build();
    URL url = null;
    try {
        url = new URL(builtUri.toString());
    } catch (MalformedURLException e) {
        e.printStackTrace();
    }

    return url;
}
```

Once either of the URLs have been formed they can be passed to the remaining method **`getResponseFromHttpUrl`** which makes the HTTP connection and returns the response of the request. Once connected to the database, the aim is to retrieve the entire of the result in one operation, and then process this with further utilities. To achieve this, a simple method was made possible by the **`Scanner`** class that was introduced in Java 1.6 [92]. Whereas the

developer would previously need to write a few lines of code to implement an instance of the **BufferedReader** class to format the HTTP result into a single **String**, with **Scanner** this can be achieved in one or two lines of code. By feeding the response to an instance of **Scanner** and using .**useDelimiter("\\A")**, this instructs the **Scanner** instance to parse the result from input start to next input start. With only one input returned from the HTTP request, no "next" input start exists, so the entire result is parsed into one String [93]:

```
public static String getResponseFromHttpUrl(URL url) throws
IOException {
    HttpURLConnection httpURLConnection = (HttpURLConnection)
url.openConnection();
    try {
       InputStream inputStream = httpURLConnection.getInputStream();

        Scanner scanner = new Scanner(inputStream);
        scanner.useDelimiter("\\A");

        boolean hasInput = scanner.hasNext();
        if (hasInput) {
            return scanner.next();
        } else {
            return null;
        }
    } finally {
        httpURLConnection.disconnect();
    }
}
```

4.3.4 UsdaJsonUtils

With the response of the HTTP request stored in a single **String** it is not entirely useful. This single **String** contains all the information from the response, and much of this data is not needed for the application. For the Food Search, the **String** will contain the data of every food that matches or has a similarity with the search query; for the Nutrient Report it contains detail on over 20 nutrients, and the amount of each nutrient present in a multitude of quantity types. For example, the response for a type of cheese will include data such as the presence of nutrients in a cup of diced cheese, a cup of shredded cheese, and so on. This level of detail is not needed in this application. To work with this **String**, which is packed with data, it is necessary to parse it into smaller, usable packets of data in an organised and controlled manner. Therefore the UsdaJsonUtils.java file was created.

The **NetworkUtils.getResponseFromHttpUrl** method produces a single **String** which is formatted as JSON by the database, as described in the Background Theory of this

report, Section 3.3. An example of the output can be seen there. To parse this data, it is divided into JSON objects and JSON arrays via the appropriately named classes **`JSONObject`** and **`JSONArray`**. Once an array of data is found within the JSON data the developer can programmatically iterate through the array and extract the required information from it. The following piece of code covers the method **`getNutrientDataFromJson`** and illustrates the process:

Table 10: List of Methods; NetworkUtils.java

Method Name	Description
`getNutrientDataFromJson`	Produces an array of Nutrient objects from a String of JSON nutrient data.
`getFoodDataFromJson`	Produces an array of Food objects from a String of JSON food search result data.

```
public static Nutrient[] getNutrientDataFromJson(String
nutrientJsonString) throws JSONException{

    final String USDA_REPORT = "report";
    final String USDA_FOOD = "food";
    final String USDA_NUTRIENTS = "nutrients";
    final String USDA_NAME = "name";
    final String USDA_UNIT = "unit";
    final String USDA_VALUE = "value";

JSONObject nutrientJson = new JSONObject(nutrientJsonString);
JSONObject nutrientReport = nutrientJson.getJSONObject(USDA_REPORT);
JSONObject nutrientFood = nutrientReport.getJSONObject(USDA_FOOD);
JSONArray nutrientArray = nutrientFood.getJSONArray(USDA_NUTRIENTS);

    Nutrient[] nutrients = new Nutrient[nutrientArray.length()];

    for (int i = 0; i < nutrientArray.length(); i++) {

        JSONObject results = nutrientArray.getJSONObject(i);

        nutrients[i] = new Nutrient();
        nutrients[i].setName(results.getString(USDA_NAME));
        nutrients[i].setUnit(results.getString(USDA_UNIT));
        nutrients[i].setValue(results.getString(USDA_VALUE));
    }
    return nutrients;
}
```

In the example above, the method takes the **`String`** passed to it and begins segmenting it into JSON objects. Each JSON object is defined by declaring the method **`getJSONObject`** while passing a **`String`** containing the exact text used in the respective heading of the JSON string. In this example, the array that is required is indicated by the titled "nutrients", hence the call **`getJSONArray(USDA_NUTRIENTS)`** is used to isolate this, where the **`String USDA_NUTRIENTS`** contains **`"nutrients"`**. With the JSON array isolated, an array of Nutrient objects is created which is the same length as the JSON array and a for-loop is used to iterate through the arrays while copying the required data. As can be observed, the loop sets the **`name`**, **`unit`**, and **`value`** of a newly created **`Nutrient`** to the respective values of each JSON object within the JSON array for every instance of the integer **`i`**, from 0 to the end of the array.

The method **`getFoodDataFromJson`** operates like the method shown above, but uses its respective, locally stored final **`String`** objects to segregate the JSON objects and array found in the Food Search response. As well as this it returns an array of **`Food`** objects instead of the **`Ingredient`** array produced from the **`getNutrientDataFromJson`** method. A full list of the local constant **`String`** objects used in each method can be found in Tables 11 and 12.

Table 11: List of Local Constants; UsdaJsonUtils.getNutrientDataFromJson

Name	Contents
`USDA_REPORT`	`"report"`
`USDA_FOOD`	`"food"`
`USDA_NUTRIENTS`	`"nutrients"`
`USDA_NAME`	`"name"`
`USDA_UNIT`	`"unit"`
`USDA_VALUE`	`"value"`

Table 12: List of Local Constants; UsdaJsonUtils.getFoodDataFromJson

Name	Contents
`USDA_LIST`	`"list"`
`USDA_ITEM`	`"item"`
`USDA_GROUP`	`"group"`
`USDA_NAME`	`"name"`
`USDA_NDBNO`	`"ndbno"`

4.4 Activities and Fragments

The function and corresponding layout of each Activity is described here, followed by the Fragments that can be launched from each Activity.

4.4.1 DailyViewActivity.java

The **DailyViewActivity** is the central hub of the application: it allows users to examine an overview of their daily meals, change the day of interest as desired, delete existing meals or progress to other activities to add or edit meals. It is the first interface that the user is presented with and it is the activity that the application returns to after adding or editing meals. As with many activities in modern Android applications, the layout of this activity is defined by two xml layout files in the **res\layout** folder: **activity_daily_view.xml** and **content_daily_view.xml**. The **activity_daily_view.xml** layout file contains the widgets used in the UI, the **fab (FloatingActionButton)** and **Toolbar** in this case, as well as a reference to include the **content_daily_view.xml** layout file. Hence, when the Java class **DailyViewActivity** calls to inflate the **activity_daily_view.xml** file as its layout, the **content_daily_view.xml** layout is automatically inflated within this. These UI elements, the widgets mentioned above, and the call to include the UI content layout can be seen listed in the "Component Tree" on the left side of the layout design window in Figure 7.

The **content_daily_view.xml** layout file contains the main UI components that the user gathers information from and interacts with. It is governed by the ConstraintLayout class, which specifies the position of each UI element in relation to either the parent (the window that UI belongs to) or to other UI elements. This is known as the element's "constraints". At the top of the UI a **TextView** (with ID **tv_day_title**) displays the date of the "observed" day, i.e. the date being observed by the user. This will initialise to the user's current day upon app launch, however the date under observation can be changed by using the directional **ImageButton** objects either side of this **TextView** (IDs: **btn_left** and **btn_right**). Beneath these three elements lies a **ListView** which is used to list the user's meals. The contents of the ListView is populated programmatically by retrieving data stored in the application's database and the process will be described in the latter half of this section. Figure 8 shows the layout design window of the **content_daily_view.xml** layout file, with each of these UI elements listed in a hierarchical manner the "Component Tree" on the left side of the image.

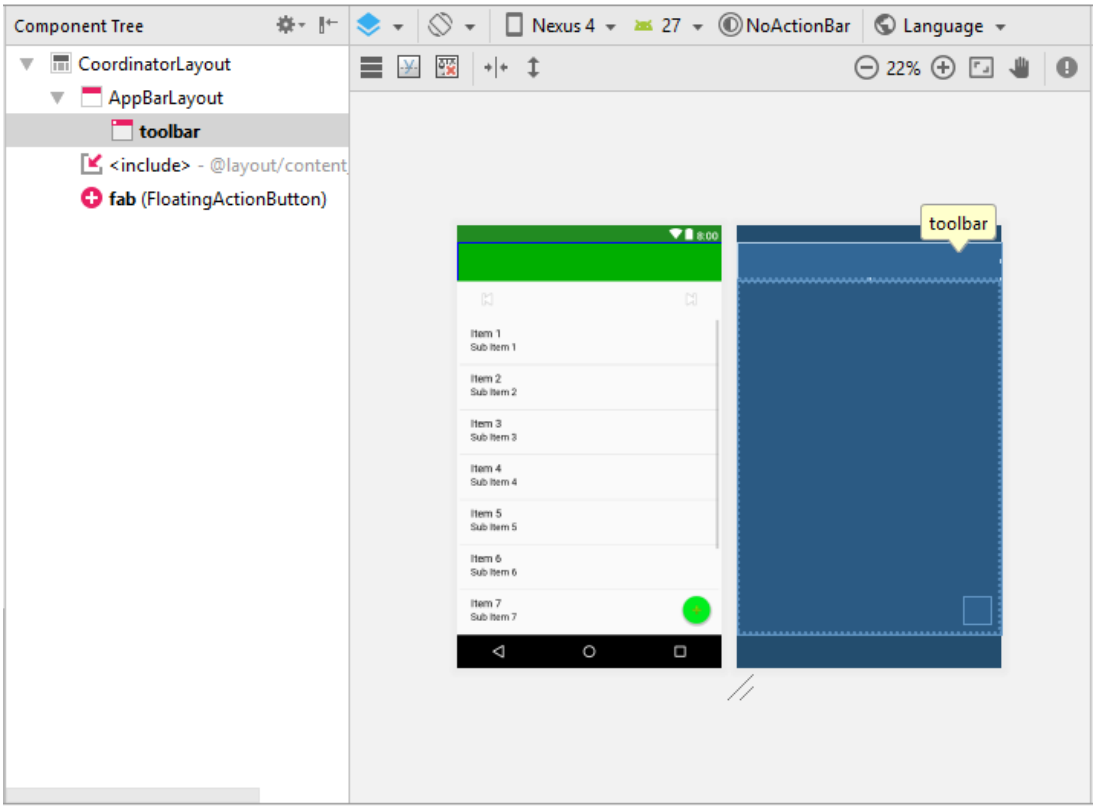

Figure 7: Layout Design Window of activity_daily_view.xml; from Android Studio

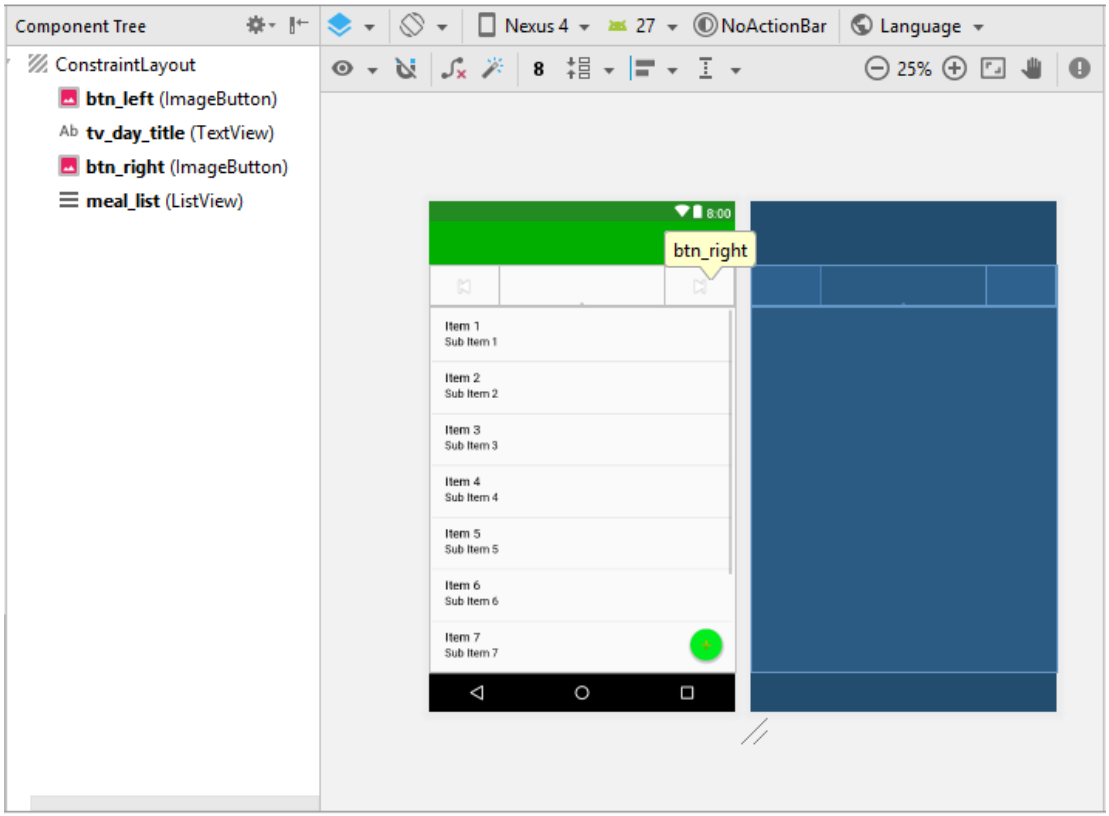

Figure 8: Layout Design Window of content_daily_view.xml; from Android Studio

The directional buttons used in this Activity are **ImageButton** objects, as mentioned above and as can be seen in Figure 8. An **ImageButton** operates exactly like a regular **Button** object, except the text is replaced with an image or icon. The image used can be provided by the developer, however the Android system already contains many standard icons. For these buttons, the author utilised the inbuilt icons typically used for changing media, such as songs or video clips. A close-up of these icons is shown in Figure 9. These icons are indicated in the XML file by assigning the "**src**", or source file, which can be seen in the following example:

```
<ImageButton
        .
        .
        android:id="@+id/btn_left"
        android:src="@android:drawable/ic_media_previous"
        .
        ./>
```

From the above code example it can be observed that source file address for the image to be used in the left-hand button is therefore: "**@android:**", indicating it is an Android stock file; "**drawable/**", pointing to the drawable folder, where the images of an app are held; "**ic_media_previous**", the title of the file used which in this case is the previous media button, however for the right-hand button this reads "**ic_media_next**".

Figure 9: Close-up of Media Previous (Left), Media Next (Right) Icons on ImageButtons

The **fab** (**FloatingActionButton**) was also assigned a stock Android icon: the "add" symbol. This was chosen because the **fab** is used to add new meals, and was implemented with the addition of the following line in the XML element:

```
app:srcCompat="@android:drawable/ic_input_add"
```

As mentioned above, the **fab** is declared in the **activity_daily_view.xml** file, where the UI widgets of DailyViewActivity reside. As the fab is a design widget, it uses the attribute **app:srcCompat** when assigning the source image rather than the **android:src** attribute. This is to support compatibility with older devices. A close-up of the **fab** can be seen in Figure 10 below.

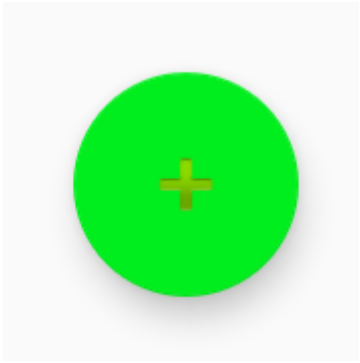

Figure 10: Close-up of Input Add Icon on FloatingActionButton

The left and right **ImageButton** elements allow the user to change the date being observed, while an integer called **changeCount** is used to keep track of the changing date. The following excerpt of code is from with within the **onCreate()** method of the **DailyViewActivity.java** file and shows the buttons of the layout file (which have resource IDs, i.e. **R.id.btn_left**) being assigned to **ImageButton** objects that were declared at the beginning of the Java class, along with the **onClickListener** for each button:

```
changeCount = 1;
previousDayBtn = findViewById(R.id.btn_left);
nextDayBtn = findViewById(R.id.btn_right);
previousDayBtn.setOnClickListener(v -> {
    changeDay(-1);
    changeCount--;
    checkChangeCount(changeCount);
    });
nextDayBtn.setOnClickListener(view -> {
    changeDay(1);
    changeCount++;
    checkChangeCount(changeCount);
});
```

As can be seen in the code, when each button is clicked, it calls the **changeDay(int)** method. This changes the observed day by using the number supplied to the method as an offset of days to add to or subtract from the currently observed day. The **onClick** methods then either increments or decrements the **changeCount**, depending on which button is activated. Each **onClick** method subsequently checks the value of the **changeCount** integer through calling the method **checkChangeCount(int)** while passing the current **changeCount** value as the integer, i.e. **checkChangeCount(changeCount)**. This method compares the value of the integer using an if statement and launches the **DatePickerFragment** if the

changeCount exceeds the limits of a ±4-day offset. The methods mentioned in this paragraph are all shown in the following code excerpt:

```
private void checkChangeCount(int i){
    if(i < -4 | i > 4){
        int x;
        if(i < -4){
            x = -1;
        } else {
            x = 1;
        }
        DialogFragment datePicker = new DatePickerFragment();
        datePicker.show(getFragmentManager(),"Date Picker");
        zeroChangeCount(x);
    }
}

public void zeroChangeCount(int diff){
    changeCount = diff;
}

public void changeDay(int offset){
    Calendar calendar = Calendar.getInstance();
    calendar.set(Calendar.YEAR, currentYear);
    calendar.set(Calendar.MONTH, currentMonth);
    calendar.set(Calendar.DAY_OF_MONTH, currentDay);
    calendar.add(Calendar.DATE, offset);
    currentYear = calendar.get(Calendar.YEAR);
    currentMonth = calendar.get(Calendar.MONTH);
    currentDay = calendar.get(Calendar.DAY_OF_MONTH);
    int weekDay = calendar.get(Calendar.DAY_OF_WEEK);

    startDate = setDateLimits(calendar, 0);
    endDate = setDateLimits(calendar, 1);

    dayOfWeek = getDayName(weekDay);
    monthName = getMonthName(currentMonth);
    dayTitle = findViewById(R.id.tv_day_title);
    String thisString = dayOfWeek + ", " + currentDay + " " +
monthName + " " + currentYear;
    dayTitle.setText(thisString);
    fetchData(startDate, endDate);
}
```

As can be seen above, the **checkChangeCount(int)** method checks the **changeCount** and launches the **DatePickerFragment**, as previously described. It can also be seen that this method declares another integer, **x**, and makes a call to another method **zeroChangeCount(int)**. This method and the integer passed to it were included to improve the user experience. If the user launches **DatePickerFragment** from using the next or previous day buttons, they have the option to the Fragment without picking a date. If

that happens, it was decided that the **changeCount** should be reset to prevent the **DatePickerFragment** from being triggered again immediately. It was also decided that the user should have the ability to return to their starting position/current day without triggering the Fragment again, hence the use of the integer **x**. This integer acts as a buffer for the number of days a user can change.

The **changeDay(int)** method, as shown in the code above, relies upon the use of the **Calendar** Java class. This class is an abstract class which has been included in the utilities of Android since API level 1. It can convert between an instance in time and specific calendar fields, such as those used above: `YEAR`, `MONTH`, `DAY_OF_MONTH`, `DAY_OF_WEEK` [94]. As can be seen, the method first creates a **Calendar** object with the current date and time by calling the method **getInstance()**, and then proceeds to set the date of this **Calendar** object to the date that the user was observing before pressing the button. This is done by individually setting the year, month, and day of the Calendar object to integer values that are initialised in the **onCreate()** method via another method which was designed by the author called **loadMeals()**.

```
Calendar calendar = Calendar.getInstance();
calendar.set(Calendar.YEAR, currentYear);
calendar.set(Calendar.MONTH, currentMonth);
calendar.set(Calendar.DAY_OF_MONTH, currentDay);
```

With the Calendar object set to the date that was previously being observed by the user, the **changeDay(int)** method then takes the integer that was passed to it and adds this to the observed date. This "offset" of days changes the date accordingly, adding 1 to the date if the right-hand next button was pressed or adding -1 if the left-hand previous button was pressed:

```
calendar.add(Calendar.DATE, offset);
```

Once the date has been offset, the **changeDay(int)** method then updates the integers used above with the new date values. Where the set method was used above, setting the new Calendar instance to previously stored values, the get method is used after the offset has been added to retrieve the new value for each integer. While doing so, a call is also made to retrieve the **DAY_OF_WEEK** value from the **Calendar** object, which is stored in a local integer called "**weekDay**" . This value is an integer which ranges from 1 to 7, representing each day of the week from Sunday to Saturday. It is stored in a local integer as it is not needed outside of this method.

```
currentYear = calendar.get(Calendar.YEAR);
currentMonth = calendar.get(Calendar.MONTH);
currentDay = calendar.get(Calendar.DAY_OF_MONTH);
int weekDay = calendar.get(Calendar.DAY_OF_WEEK);
```

To utilise the integer representing the day of the week a simple method was designed to convert the integer to a shortened version of the day's name, i.e. "Sun", "Mon", "Tues" etc. A similar method was then created for converting the **currentMonth** integer into a shortened version of the month's name, and both methods were set to return string objects. With these strings formed, they are concatenated along with the day and year integer value and the **TextView** at the top of the UI (**tv_day_title**, as described above) is populated with this concatenated string. Figure 11 shows an example of this.

```
dayOfWeek = getDayName(weekDay);
monthName = getMonthName(currentMonth);

String thisString = dayOfWeek + ", " + currentDay + " " + monthName
+ " " + currentYear;
dayTitle.setText(thisString);
```

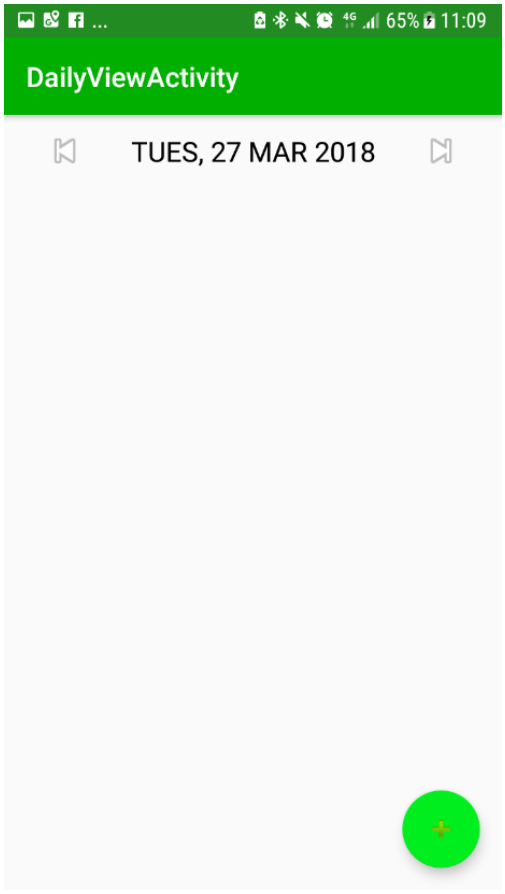

Figure 11: Screenshot of DailyViewActivity; without Meals

Lastly, the **changeDay(int)** method makes calls to other methods to load the data (i.e. meals) for the day being observed. This first of these methods returns a **Date** object using a method that has been declared as **setDateLimits(Calendar, int)** . This method takes a **Calendar** object, adds a day offset using the same method as above, and then sets the time to midnight (00:00:00). The purpose of this is to set a time range to search for meals, by setting the start of the time range to 00:00:00 on the day of interest and the end to 00:00:00 on the following day:

```
startDate = setDateLimits(calendar, 0);
endDate = setDateLimits(calendar, 1);

fetchData(startDate, endDate);
```

For the example shown in the screenshot of Figure 11 above, the time range would therefore be 27/03/2018 @ 00:00:00 to 28/03/2018 @ 00:00:00. This is returned as a Date object as the database was developed with a class which converts Date objects to a timestamp, which will be described towards the end of this chapter. When the start and end dates are passed to the **fetchData()** method, it retrieves the meals for the current day as an object list and organises them in the order: breakfast, lunch, dinner, snacks. This is done using the **Collections** class and its **sort** method and its effect can be seen in the left-hand image of Figure 12. Each meal was added in a random order but were organised as shown below.

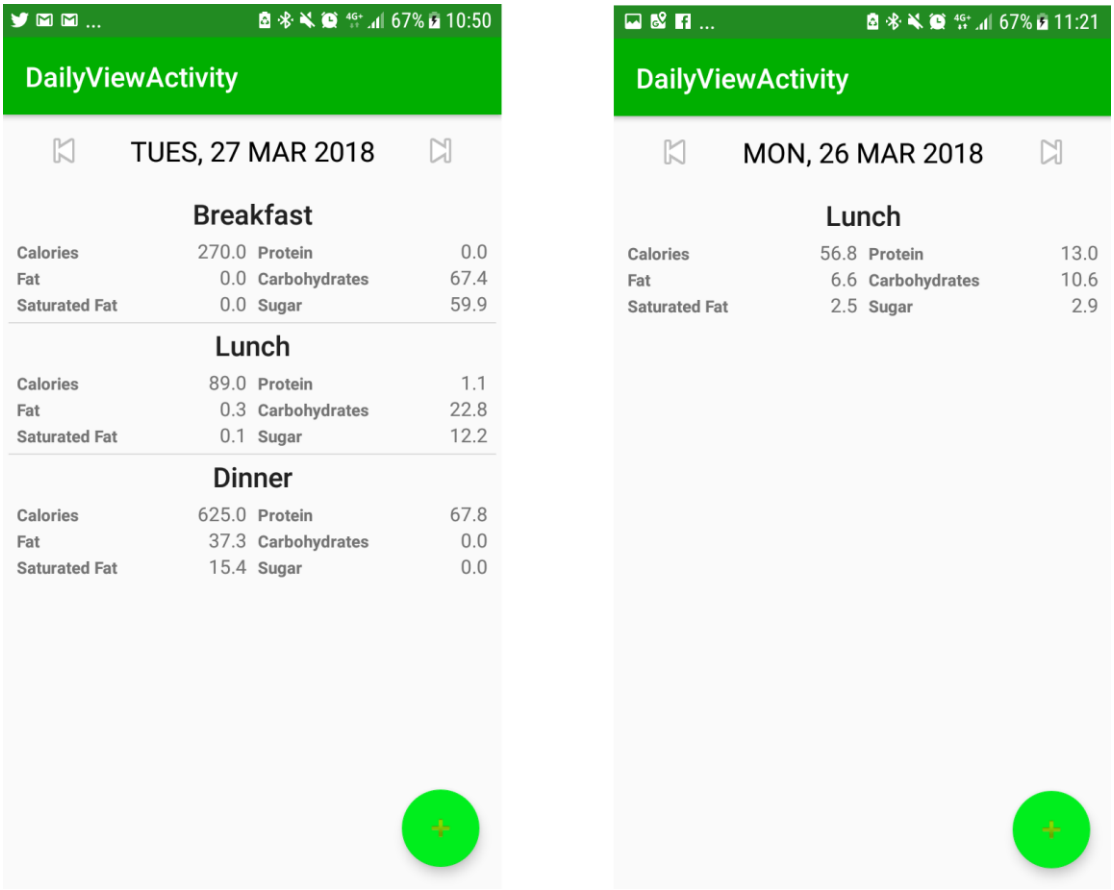

Figure 12: Screenshots of DailyViewActivity: 3 Meals, Organised (Left); 1 Meal (Right)

After the **`fetchData`** method retrieves the meals from the database and organises them, it makes a call to another method **`checkMeals(List<Meal>)`**, where **`Meal`** is the class that was designed to store meal data, as discussed above. This method takes the list of meals passed to it and checks each meal's type, i.e. breakfast, lunch etc. A Boolean value is created for each **`mealType`** (named B, L, D, and S respectively), and each is set to true if a meal of that type is already stored for the day being observed. This was implemented to prevent the user from creating multiple meals of the same type on one day, however it could be expanded in further versions to facilitate those who eat multiple small meals per day.

```
public void fetchData(Date start, Date end) {
  final List<Meal> meals = mDb.mealModel() …
                                .findAllMealsByDay(start, end);
  Collections.sort(meals);
  checkMeals(meals);
```

Once the meals have been retrieved, organised, and had their meal types checked, the **`fetchData`** method creates an instance of the **`MealAdapter`** class and passes the list of meals it has retrieved to this adapter. The adapter populates the **`ListView`** element with some of the data of each meal and displays it within the **`DailyViewActivity`**. This data adapter

is responsible for retrieving the appropriate data from each **Meal** object and using this to populate the UI elements of the **ListView**. Please refer to the **MealAdapter** file in the Java code of Appendix A for further details of its operation.

```
MealAdapter mealAdapter = new MealAdapter(DailyViewActivity.this,
meals);
  mealList.setAdapter(mealAdapter);
```

Once the adapter has been set with the **setAdapter()** command, the methods **setOnItemClickListener** and **setOnItemLongClickListener** are called. These are public methods that the **ListView** inherits from the **AdapterView** class [95] [96], and allow the developer to create actions to respond to an item within the list being clicked or long-clicked (i.e. pressed and held momentarily) respectively. In this application, it was designed so that a single click on a meal item would raise a **Snackbar** to ask if the user would like to edit this meal, which includes an action to launch the **MealBuilderActivity**. The appearance of this **Snackbar** is shown in Figure 13.

Figure 13: Screenshot of DailyViewActivity; Snackbar for Single Click/Edit Item

The following code is the section of the **fetchData** method which dictates the response of the activity to the user clicking on an item within the **ListView**:

```
mealList.setOnItemClickListener(new …
AdapterView.OnItemClickListener() {
@Override
public void onItemClick(AdapterView<?> adapterView, View view, int …
i, long l) {
 Meal meal = meals.get(i);
 String mealTitle = meal.mealTitle;
 Snackbar.make(view, mealTitle + ": would you like to edit this …
meal?", Snackbar.LENGTH_LONG)
          .setAction("Edit", new EditMealListener(meal.id)).show();
    }
});
```

As can be seen in the code above, the action of the **Snackbar** is performed by the **EditMealListener**, which listens for clicks on the text showing "EDIT". When this listener is triggered it creates an **Intent** to transition from the **DailyViewActivity** to the **MealBuilderActivity**. Using the **putExtra(String, long)** method, the ID of the meal to be edited is passed to the **MealBuilderActivity**. This allows the ingredients of the meal to be retrieved in the next activity as each ingredient is tagged with the meal it belongs to in the application's database. The following code is the action taken from within the **EditMealListener** class:

```
@Override
        public void onClick(View v) {
            Intent intent = new Intent(DailyViewActivity.this,
MealBuilderActivity.class);
            intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK);
            intent.putExtra(EXTRA_MEAL_ID, id);
            startActivity(intent);
        }
```

The **setOnItemLongClickListener** uses code that produces a similar response for long clicks. It raises a **Snackbar** like for the case above, however the text and action are different. The message which previously included the meal title and asked the user if they would like to edit the meal now asks if they would like to delete the meal and clicking the action button within the **Snackbar** now launches an **AlertDialog**. This **AlertDialog** askes the user to confirm that they intend to delete the meal and was implemented to prevent the erroneous loss of data. An example of the **Snackbar** launched from a long click is shown in Figure 14, along with the **AlertDialog**. The following code shows the **onClick** method of the

DeleteMealListener class, which is triggered from the **Snackbar** and launches the **AlertDialog**:

```
@Override
public void onClick(View view) {
 new AlertDialog.Builder(DailyViewActivity.this)
  .setTitle("Delete Meal")
  .setMessage("Are you sure you want to delete this meal?")
  .setNegativeButton("Cancel", null)
  .setPositiveButton("Delete", new DialogInterface.OnClickListener()
{

                public void onClick(DialogInterface arg0, int arg1) {
                     mDb.mealModel().deleteMealById(id);
                     fetchData(startDate, endDate);
                }
            }).create().show();
}
```

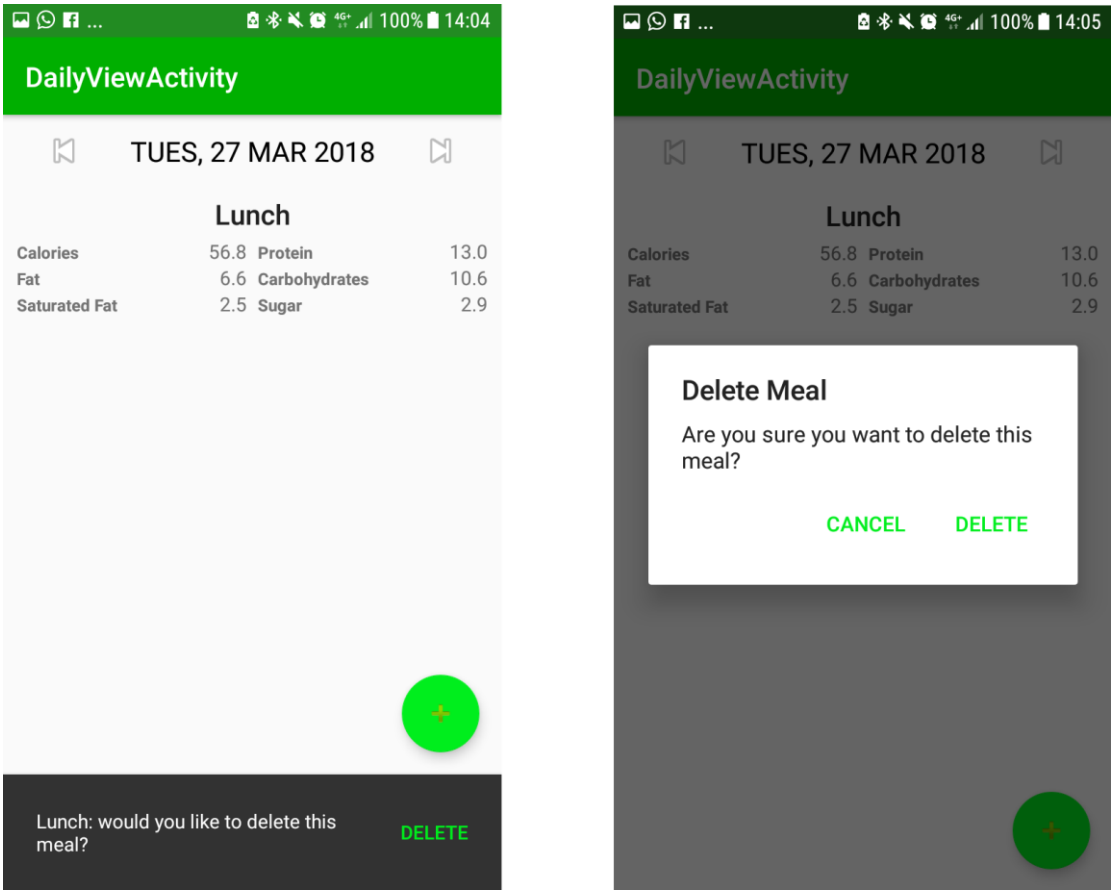

Figure 14: Screenshots of DailyViewActivity; Snackbar for Long Click/Delete Item (Left); AlertDialog for Deleting Meal (Right)

The remaining UI element, the **fab** (**FloatingActionButton**), is controlled by a click listener that is assigned within the **onCreate** method. The **onclickListener** responds to the user interacting with the **fab** by launching the **MealTypeDialogFragment**, a

DialogFragment which contains a list of the 4 meal types: “Breakfast”, “Lunch”, “Dinner” and “Snacks”. The following code comes from the **onCreate** method of **DailyViewActivity** and shows the **fab** being assigned to its corresponding layout element, along with the assignment of the onClickListener and the call to create an instance of **MealTypeDialogFragment**:

```
FloatingActionButton fab = findViewById(R.id.fab);
fab.setOnClickListener(view ->
MealTypeDialogFragment.newInstance().show(getSupportFragmentManager(
), "dialog"));
```

The remaining code in the **DailyViewActivity.Java** file consists of **public** *getters* ad setters for the int values of the year, month, and day which allow other activities and fragments to retrieve the date of the day being observed by the user. The file also includes **public static Boolean** objects which are used in **MealTypeDialogFragment** to check the meal types stored for the current day i.e. if a Breakfast is already stored for today, **Boolean *B*** will be True, and when the **public static Boolean *checkB()*** is called from **MealTypeDialogFragment** it will return True also. The code mentioned appears as follows:

```
//getters & setters
public int getCurrentYear(){
    return currentYear;
}

public int getCurrentMonth(){
    return currentMonth;
}

public int getCurrentDay(){
    return currentDay;
}

public void setDayStorage(int year, int month, int day){
    currentYear = year;
    currentMonth = month;
    currentDay = day;
}

//Boolean state checks
public static Boolean checkB(){
    return B;
}
public static Boolean checkL(){
    return L;
}
```

```
public static Boolean checkD(){
    return D;
}
public static Boolean checkS(){
    return S;
}
```

4.4.1.1 DatePickerFragment

This Fragment is launched to allow the user to select a date that is 5 or more days ahead or behind the current day. As discussed above, it is launched by the user through interacting with the directional, day-changing buttons at the top of the **`DailyViewActivity`** UI. The appearance of this Fragment's UI is shown in Figure 15.

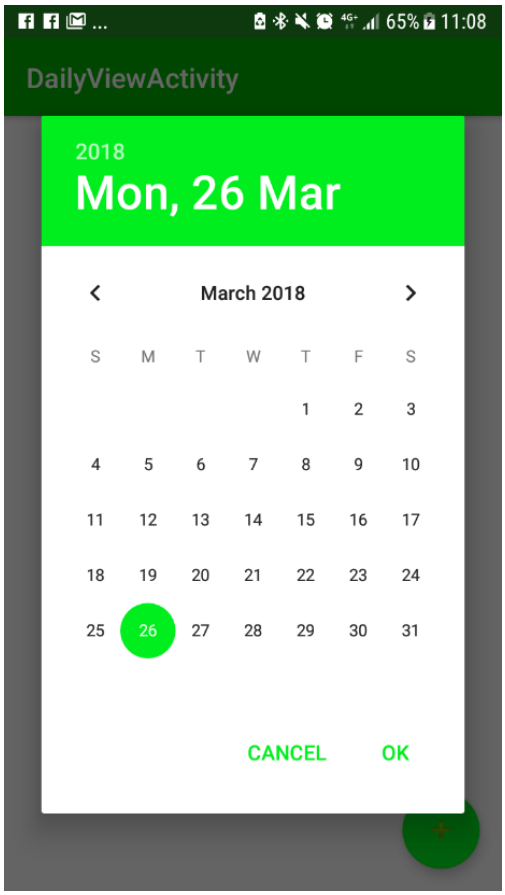

Figure 15: Screenshot of DatePickerFragment

The **`DatePickerFragment`** inherits its functionality from the **`DialogFragment`** class, while monitoring user interaction by the implementation of the public static interface **`DatePickerDialog.OnDateSetListener`**. This interface was added in API level 1 [97] and can be easily implemented to produce a responsive feature. The only edits made to the inherited and implemented code was to set the current date as the default date of the

DatePicker in the **onCreateDialog** method, and to also implement some methods of the **DailyViewActivity** once the user has set a date. The new code can be seen below:

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    final Calendar c = Calendar.getInstance();
    int year = c.get(Calendar.YEAR);
    int month = c.get(Calendar.MONTH);
    int day = c.get(Calendar.DAY_OF_MONTH);
    dayOfWeek = c.get(Calendar.DAY_OF_WEEK);
  return new DatePickerDialog(getActivity(),this, year, month, day);
}

@Override
public void onDateSet(DatePicker datePicker, int year, int month,
int day) {
  final Calendar calendar = Calendar.getInstance();
  calendar.set(Calendar.YEAR, year);
  calendar.set(Calendar.MONTH, month);
  calendar.set(Calendar.DAY_OF_MONTH, day);
  dayOfWeek = calendar.get(Calendar.DAY_OF_WEEK);

  ((DailyViewActivity)getActivity()).setDayStorage(year, month,
day);

  TextView tv = getActivity().findViewById(R.id.tv_day_title);
  String monthName =
((DailyViewActivity)getActivity()).getMonthName(month);
  String dayName =
((DailyViewActivity)getActivity()).getDayName(dayOfWeek);


  String thisString = dayName + ", " + day + " " + monthName + " " +
year;
    tv.setText(thisString);

  Date startDate =
((DailyViewActivity)getActivity()).setDateLimits(calendar, 0);
  Date endDate =
((DailyViewActivity)getActivity()).setDateLimits(calendar, 1);
   ((DailyViewActivity)getActivity()).fetchData(startDate, endDate);
}
```

The methods shown above, initiated once the user selects a date, updates the fields of the **DailyViewActivity** and loads the data for the chosen day.

4.4.1.2 MealTypeDialogFragment

The **`MealTypeDialogFragment`** is launched from the **`fab`** of **`DailyViewActivity`** and is used when creating a new meal. It allows the user to select which meal type they would like to assign to the meal being created by displaying a list of options for the user to choose from. This **`Fragment`** extends the **`BottomSheetDialogFragment`** class, a version of **`DialogFragment`** that uses an interface which expands from the bottom of the display. This **`Fragment`** class is relatively new, being added in version 23.4.0 of Android [98]. From this class, the **`MealTypeDialogFragment`** inherits its appearance, animation, and functionality.

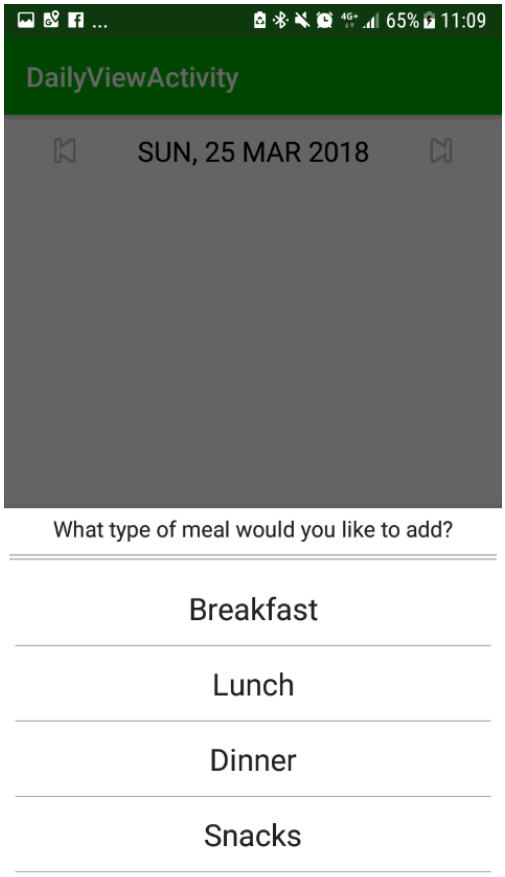

Figure 16: Screenshot of MealTypeDialogFragment

The standard appearance of the **`BottomSheetDialogFragment`** is a white list of items. When the user creates an instance of the stock **`Fragment`**, the developer must include a method for passing an integer value to indicate how many list items the **`Fragment`** should create. The **`Fragment`** then displays these items via a **`RecyclerView`**, which is useful for performance when handling long lists. As there are only four meal types in this application the stock **`Fragment`** was altered so that each instance of the **`Fragment`** that is created would

populate a dialog with 4 list items, and without the need for an integer to be passed. The results of this can be seen in Figure 16, while the following code comparison shows the edits that were made to produce this:

```
//Stock code
public static ItemListDialogFragment newInstance(int itemCount) {
    final ItemListDialogFragment fragment = new
ItemListDialogFragment();
    final Bundle args = new Bundle();
    args.putInt(ARG_ITEM_COUNT, itemCount);
    fragment.setArguments(args);
    return fragment;
}

//Edited code
public static MealTypeDialogFragment newInstance() {
    int itemCount = 4;
    final MealTypeDialogFragment fragment = new
MealTypeDialogFragment();
    final Bundle args = new Bundle();
    args.putInt(ARG_ITEM_COUNT, itemCount);
    fragment.setArguments(args);
    return fragment;
}
```

In the above comparison, aside from the difference in Fragment names, the only difference relates to the **itemCount** integer: rather than being passed to the Fragment in the stock code, the edited code always sets itemCount to 4. While the **RecyclerView** is not necessary for this application, as no views are recycled in this list, it was decided to not alter this inherited functionality. This was because the **RecyclerView** is an efficient view regardless, and it works. Within the **RecyclerView.ViewHolder** , further edits were made to the stock code as seen in the following:

```
//Stock code
@Override
public void onBindViewHolder(ViewHolder holder, int position) {
    holder.text.setText(String.valueOf(position));
}

//Edited code
@Override
public void onBindViewHolder(ViewHolder holder, int position) {
    if(position == 0){
        holder.text.setText(R.string.meal_breakfast);
    }
    if(position == 1){
        holder.text.setText(R.string.meal_lunch);
```

```
        }
        if(position == 2){
            holder.text.setText(R.string.meal_dinner);
        }
        if(position == 3){
            holder.text.setText(R.string.meal_snacks);
        }
    }
```

The edited code above simply changes the **`TextView`** within each item of the list, instructing the items to display the name of each of the meal types. The final edit to note is the changes made to the **`onClick`** method of the **`onClickListener`** applied to each list item. In the stock code, the **`onClick`** method is provided however no action is taken. The **`onClick`** method calls **`dismiss();`** which simply dismisses the click action. In the edited code, the **`onClick`** method is adapted to respond to the user's meal selection:

```
//Stock code
text.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (mListener != null) {
            mListener.onItemClicked(getAdapterPosition());
            dismiss();
        }
    }
});

//Edited code
text.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
       if (mListener != null) {
           mListener.onItemClicked(getAdapterPosition());
           int mealType = getAdapterPosition() + 1;
//Checks if mealType already in database - if statement simplified
           boolean duplicateB = mealType == 1 && checkB();
           boolean duplicateL = mealType == 2 && checkL();
           boolean duplicateD = mealType == 3 && checkD();
           boolean duplicateS = mealType == 4 && checkS();
           if(duplicateB || duplicateL || duplicateD || duplicateS){
               postNotice(text.getText().toString(), mealType);
           } else {
               launchBuilder(mealType, false );
           }
    }
}
```

In the edited code, the **`onClick`** method begins by using the position of the selected item in the adapter's list to determine which **`mealType`** was chosen. This is done by retrieving the item's position in the adapter, which is 0-indexed (also known as "zero-based" or "zero-

indexed"), meaning it starts its list or count from 0. As the **mealType** integers stored in the application database are 1-indexed, the integer value 1 is added to the adapter position integer to translate this to the mealType the user selected:

```
int mealType = getAdapterPosition() + 1;
```

With the **mealType** stored in an **int** object, the code is then designed to check whether a meal of the same type has been previously created. In the first iteration of this application this was achieved through a series of nested if statements, however this became complicated and difficult to debug. To simplify this, a single **boolean** was created for each meal which is only **True** if the user is attempting to duplicate the meal. As can be seen above, the if statement works by checking if the user is trying to duplicate any meal (i.e. if **duplicateB = True** etc.) and, if so, it makes a call to the **void** method **postNotice(String, int)**. This method takes the name of the meal (Breakfast, Lunch etc.) as its String input, which is used in the message displayed in an **AlertDialog**:

```
private void postNotice(String mealTitle, int mealType) {
   new AlertDialog.Builder(getContext())
    .setTitle("Overwrite Meal")
    .setMessage("You already have a " + mealTitle + " saved. Would …
you like to overwrite it?")
    .setNegativeButton("Cancel", null)
    .setPositiveButton("Overwrite", new …
DialogInterface.OnClickListener() {

        public void onClick(DialogInterface arg0, int arg1) {
           launchBuilder(mealType,true);
        }
    }).create().show();
}
```

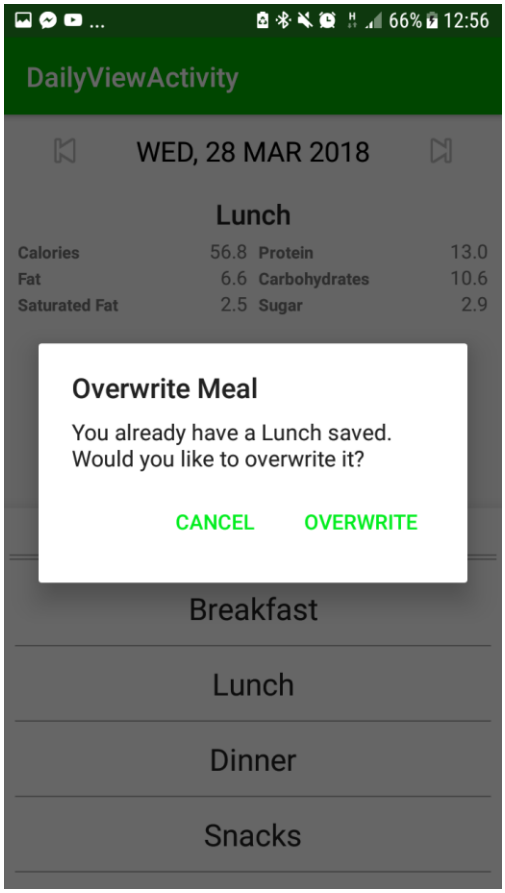

Figure 17: Screenshot of DailyViewActivity; MealTypeDialogFragment; Overwrite Meal AlertDialog

This **`AlertDialog`** alerts the user that they must overwrite the existing meal of that type if they wish to proceed, and the an **`onClickListener`** is set to the "OVERWRITE" option. If the user selects to overwrite the meal, the **`launchBuilder(int, boolean)`** is called to start the **`MealBuilderActivity`**, with the integer value representing the type of meal being passed along with the boolean value true. This **`boolean`** value is used to indicate to the activity if it is overwriting a meal (**`true`**) or creating a new meal (**`false`**), and both the **`boolean`** and the **`mealType`** integer are passed to the new activity, along with integers for the day, month, and year of the meal:

```
private void launchBuilder(int mealType, boolean overwrite){
   int year = ((DailyViewActivity)getActivity()).getCurrentYear();
   int month = ((DailyViewActivity)getActivity()).getCurrentMonth();
   int day = ((DailyViewActivity)getActivity()).getCurrentDay();


    Intent intent = new Intent(getActivity(), …
MealBuilderActivity.class);
    intent.putExtra(EXTRA_OVERWRITE_CASE, overwrite);
    intent.putExtra(EXTRA_MEAL_ID, 0);
    intent.putExtra(EXTRA_MEAL_TYPE, mealType);
    intent.putExtra(EXTRA_MEAL_YEAR, year);
    intent.putExtra(EXTRA_MEAL_MONTH, month);
    intent.putExtra(EXTRA_MEAL_DAY, day);
    startActivity(intent);
}
```

As can be seen in the above, the day, month, and year values are retrieved by calling the *getters* defined in the fragment's parent activity, **`DailyViewActivity`** . These methods are shown at the end of the **`DailyViewActivity`** section. These are passed along with the **`mealType`** integer, the overwrite boolean, and 0 as the long value for the **`mealId`**. The above method is also called from the if statement shown previously. In the case that no duplicates are present, the overwrite boolean is set to false to indicate a new meal is being created.

4.4.2 MealBuilderActivity

As the name implies this is the Activity that the user utilises to build meals. This activity can be launched for a few reasons: in the creation of a new meal; when editing a current meal; or when overwriting an existing meal. When a new meal is created, the activity is told the type of meal the user wants to build, as well as the date the user was previously observing in the **`DailyViewActivity`** before this activity was started. In this scenario, a position in the meal database is reserved by the activity when it is launched, using the methods described in Section 4.2. This action generates a **`mealId`** which is used for the **`meal_id`** field of each ingredient added, as well as being used when the meal is updated at the end of the meal-building process.

If a meal is being edited the correct **`mealId`** must be passed to the activity with the intent. With this ID, the activity can retrieve the existing ingredients of the meal so that they can be edited or more ingredients can be added to the meal. In the case of a meal being overwritten, however, a different process occurs. As this case occurs when a user attempts to create a duplicate meal, the system knows what type of meal is being overwritten and the date of the meal. This information is passed to the **`MealBuilderActivity`**, along with a true value for the **`overwrite boolean`**. When this boolean is true, the program retrieves the ID of the meal which matches the date and type, and then deletes all its ingredients. This provides a blank meal that the user can add ingredients to, while using the same database position for the meal. The conditional statement which ensures the correct applications for the different methods is shown in the following excerpt:

```
Intent myIntent = getIntent();

          long mealId =
myIntent.getLongExtra(MealTypeDialogFragment.EXTRA_MEAL_ID, 0);
          if (mealId != 0) {
          loadIngredients(mealId);
          ingredientAdapter = new IngredientAdapter(ingredientList);
          ingredientAdapter.setClickListener(this);
          itemList.setAdapter(ingredientAdapter);
          currentMealId = mealId;
            } else { //dates
              mealType = myIntent.getIntExtra(EXTRA_MEAL_TYPE, 0);
              if (mealType == 0) {
              throw new NullPointerException("Houston, we have a
problem: mealType must be set for new meal creation");
              }
              int year = myIntent.getIntExtra(EXTRA_MEAL_YEAR, 0);
              int month = myIntent.getIntExtra(EXTRA_MEAL_MONTH,
```

```
13); //Jan-Dec = 0 - 11; 12 reserved for lunar calendars
                int day = myIntent.getIntExtra(EXTRA_MEAL_DAY, 0);
                if (year == 0 | month == 13 | day == 0 ) {
                    throw new NullPointerException("Houston, we have
a problem: date must be passed");
                }
                boolean overwrite =
myIntent.getBooleanExtra(MealTypeDialogFragment.EXTRA_OVERWRITE_CASE
,false);
                if(overwrite){
                    mealId = overwriteIngredients(mealType, day,
month, year);
                    deleteIngredientWithMealId(mDb, mealId);
                    currentMealId = mealId;
                } else {

                Date mealTime = makeTimestamp(year, month, day);
                currentMealId = makeBlankMeal(mDb, mealType,
mealTime);
            }
        }
```

With a blank meal or a meal being edited, the user has two options for adding new ingredients: using the text search features or the image classifier. The image classifier is launched using the button marked “SEARCH BY IMAGE”, whereas the text search features are controlled using the **`EditText`** field and **`ImageButton`** at the top of the UI. If the image classifier is selected, the camera is activated and, through processes described in Section 4.5, a String containing the result of the classification is returned to the **`MealBuilderActivity.`** The classifier calls a method which creates a database request using the same method employed by the text-search feature, by using **`NetworkUtils`** to form a URL and establish the connection with USDA database.

With the text-search feature, the user can use the **`EditText`** field to input a search query, with the **`ImageButton`** beside this launching the **`AsyncTask`** called **`FoodSearchTask`**. This **`AsyncTask`** uses a background thread to search for food types from the USDA food database API. It performs this by calling the **`NetworkUtils.getResponseFromHttpUrl`** method which, as discussed above, returns a JSON response. This JSON is not processed immediately, as the activity is designed to pass the entire JSON string to the following activity, **`SearchResultsActivity`** as one Intent extra. This process is performed regardless of the user’s method for search, i.e. through text or image classification. With a text search, the value entered into This can be observed in the **`onPostExecute`** void method of the **`FoodSearchTask ASyncTask,`** as it passes the entire String with the line:

```
intent.putExtra(SearchResultsActivity.EXTRA_FOOD_DATA, searchResults);
```

This data is transmitted along with a multiple items of data such as the ID of the meal and the ID of the newly created ingredient. To use the image classification features to search for nutritional content, the button beneath the **`EditText`** field and **`ImageButton`** triggers an Intent to the **`CameraActivity`**. Figure 18 shows the appearance of this activity with and without ingredients in the current meal.

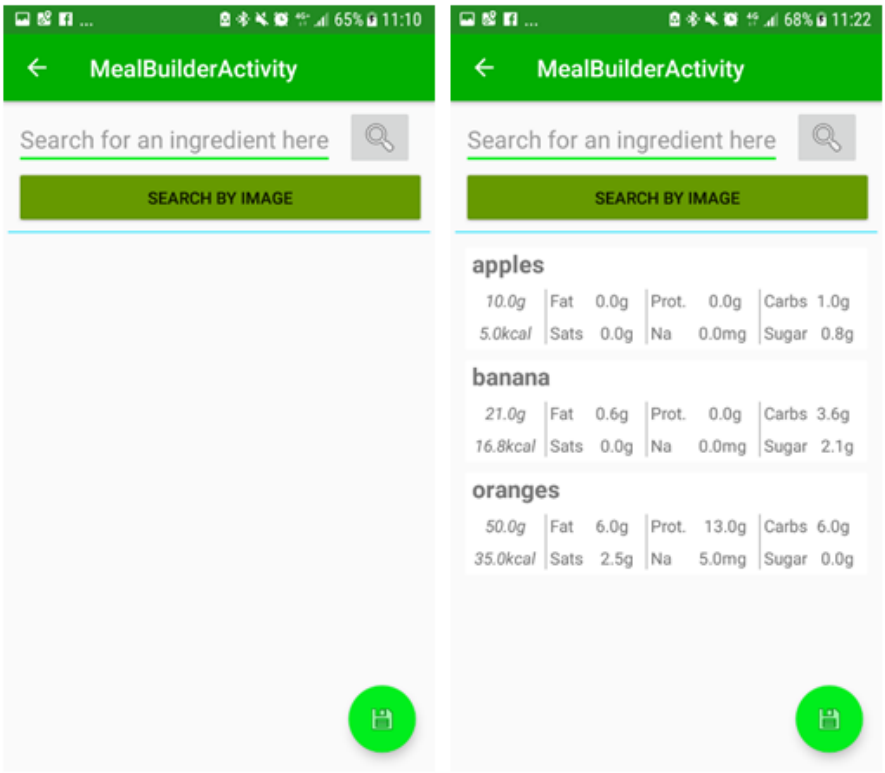

Figure 18: MealBuilderActivity; No Ingredients (Left); 3 Ingredients (Right)

When Ingredients are present, the activity uses the adapter **`IngredientAdapter`** to maintain the UI and control the data representation of the ingredients. Most of the code used here is similar in function to that used in the **`DailyViewActivity`** to display meals. In the MealBuilderActivity the fab has a different purpose however. Here it is used to save progress rather than to add new items. Some other notable code used here is in the use of the **`AlertDialog`** seen in Figure 19 which alerts a user to select if they would like to edit or delete the current ingredient.

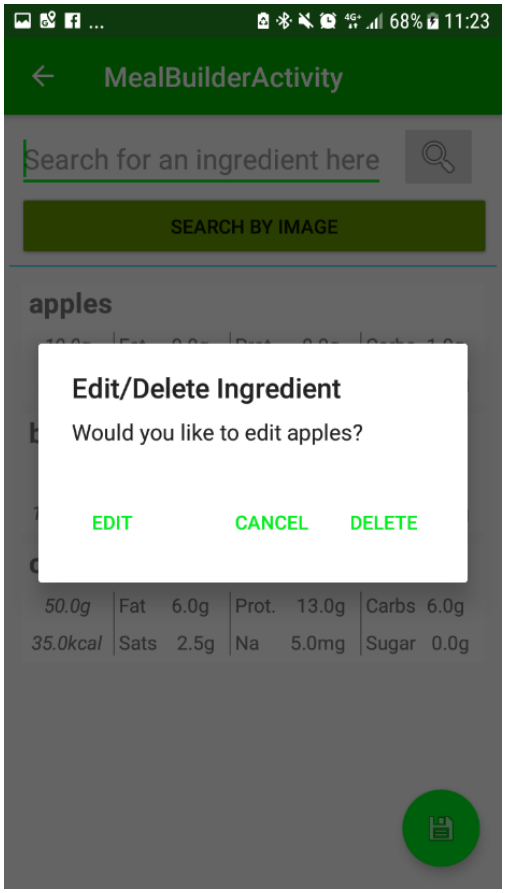

Figure 19: MealBuilderActivity; AlertDialog

4.4.3 SearchResultsActivity

`SearchResultsActivity` is launched from the **`MealBuilderActivity`** via an Intent. Upon launching, the activity uses the **`Intent.getExtra(String)`** method to retrieve the ingredient name that was passed to it as an extra attached to the Intent. This String value is passed to the **`NetworkUtils`** class to form a URL request to the USDA database on a background thread using the **`ASyncTask`** method. The JSON results of this request is passed to the **`UsdaJsonUtils.getFoodDataFromJson`** method which parses the data and stores it in an array of **`Food`** class objects. The resultant array of **`Food`** objects are then passed to the **`SearchResultAdapter`** which binds the information of each **`Food`** object in the array to an item of a **`RecyclerView`**. If a user selects one of these items, the name and ndbno of the food that the user selected is used to form a request for a USDA Nutrient Report, the results of which are passed to the **`WeightActivity`**. Figure 20 shows an example of this activity's UI.

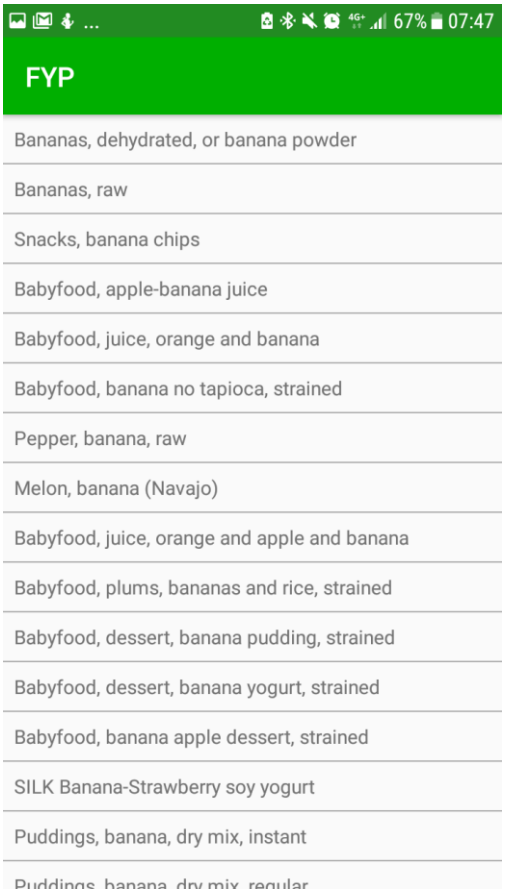

Figure 20: SearchResultsActivity; UI Example

4.4.4 WeightActivity

The **`WeightActivity`** is responsible for recording the weight of the user's ingredient, which it then uses to dynamically calculate the values of each nutrient of the ingredient. For example, the **`WeightActivity`** initially loads the 100-gram index values for each nutrient, i.e. the amount of each nutrient present in 100g of the ingredient. The current weight value is then multiplied with the index values to calculate the current nutrient values. An example of this could be an instance where an ingredient has 260 calories per 100g: if the user changes the weight input value to 50g, the calorie value will dynamically update to 130 calories. This is achieved by the data adapter **`NutritionalInfoAdapter`**, which takes the weight value as a **`Double`** object and uses this in the following code:

```
@Override
public void onBindViewHolder(ViewHolder holder, int position) {
    Nutrient nutrient = nutrientList.get(position);
    Double multiplier = foodWeight/100;
    nutrientName = nutrient.getName();
    nutrientVal = nutrient.getValue();
```

```
    nutrientUnit = nutrient.getUnit();

    Double quantityPer100g = Double.parseDouble(nutrientVal);
    dynamicVal = quantityPer100g * multiplier;

    holder.nutrientNameTV.setText(nutrientName);
    double testvar = new BigDecimal(dynamicVal).setScale(1,
RoundingMode.HALF_UP).doubleValue();
   String val =  Double.toString(testvar);
    holder.nutrientValTV.setText(val);
    holder.nutrientUnitTV.setText(nutrientUnit);
}
```

The above code is taken from the **`NutritionalInfoAdapter`** and shows the data binding used to dynamically calculate each nutrients value in the UI of the **`WeightActivity`**. Figure 21 shows an example of this UI, where the user enters the weight value in the EditText field and uses the confirm button to save the ingredient. With the Bluetooth scale implemented, this process is performed automatically through Bluetooth receivers and the Skale 2 development kit [99].

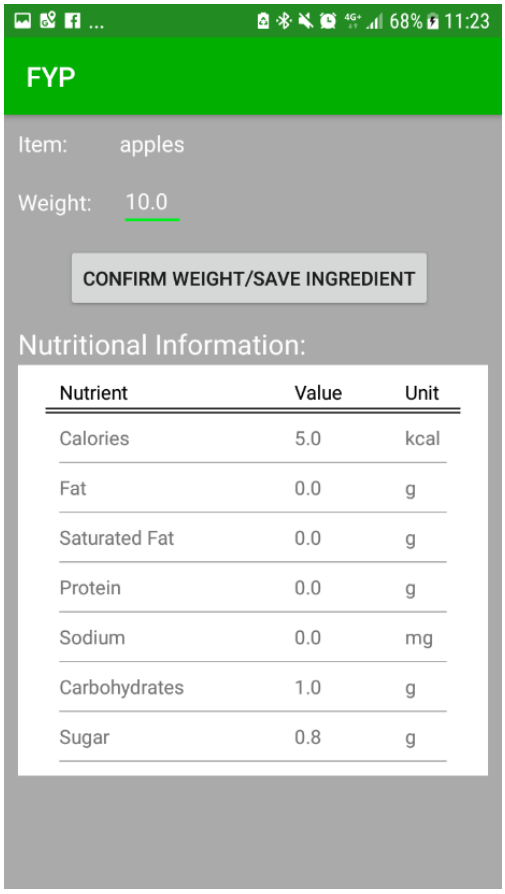

Figure 21: WeightActivity; UI Example

4.5 Classifier

This package contains all the Java files responsible for the TensorFlow image classifier. Much of the code is used on an as-is basis from what is provided by Google under the Apache 2.0 open-source license [83]. Due to the lack of design involved in this Section, an overview of how the system operates will be provided, and the edits that were made to the code will be highlighted.

The image classification process is performed using four classes:

- CameraActivity, which is launched from the MealBuilderActivity using an Intent. This is the parent activity to the Camera2BasicFragment.
- Camera2BasicFragment, a Fragment which is instantiated by the CameraActivity and is used to access the device's camera. This class manages the calls to the classifier and, in the unedited open source version, it literally "runs" the operation. This will be explained below.
- AutoFitTextureView, a view which is applied to the camera feed to process it. It formats the camera feed to the appropriate dimensions and quality for TensorFlow classification.
- ImageClassifier, the class which runs the TensorFlow classification methods and implements the data labels and classification model stored in the application's "assets" folder.

These four classes cooperate to implement an image classification feature which continually classifies the incoming camera feed. In doing so it posts a list of the top three closest matches to the current object in the camera feed. It does this by ranking the result with what can only be described as a "running" list, whereby the results of each classification are appended to a continuous stream of classification results. The classifier then posts the top three results, but due to the memory involved in this running order and the sorting features used, it was common for data to be retained when it was not needed. This led to some misclassifications, or the perception of misclassifications in early testing. An example of this would be if a user successfully classified an apple with a result of 90% confidence. If the user then attempted to classify an orange and, due to the lighting conditions or otherwise, the classifier had difficulty classifying the orange, the result would remain as an apple classification. If the classifier was 60% confident that the object was an orange but 40% confident that it may be a lemon, in this

scenario the highest result would prevail with the classifier stating a 90% confidence result of an apple.

For the writer's application, it was decided to edit to this code to include a button which allows single classification processes, rather than the continual classification included in the stock model. It was also desirable to clear the list of previous results after each classification. The original code used in the TensorFlow example, but absent from the writer's application, is seen in the following excerpt from the **`Camera2BasicFragment`**:

```
/** Starts a background thread and its {@link Handler}. */
private void startBackgroundThread() {
  backgroundThread = new HandlerThread(HANDLE_THREAD_NAME);
  backgroundThread.start();
  backgroundHandler = new Handler(backgroundThread.getLooper());
  synchronized (lock) {
    runClassifier = true;
  }
  backgroundHandler.post(periodicClassify);
}
/** Stops the background thread and its {@link Handler}. */
private void stopBackgroundThread() {
  backgroundThread.quitSafely();
  try {
    backgroundThread.join();
    backgroundThread = null;
    backgroundHandler = null;
    synchronized (lock) {
      runClassifier = false;
    }
  } catch (InterruptedException e) {
    e.printStackTrace();
  }
}
  static final int REQUEST_IMAGE_CAPTURE = 1;

  private void dispatchTakePictureIntent() {
      Intent takePictureIntent = new
Intent(MediaStore.ACTION_IMAGE_CAPTURE);
      if
(takePictureIntent.resolveActivity(getActivity().getPackageManager()
) != null) {
          startActivityForResult(takePictureIntent,
REQUEST_IMAGE_CAPTURE);
      }
  }
  @Override
  public void onActivityResult(int requestCode, int resultCode,
Intent data) {
      if (requestCode == REQUEST_IMAGE_CAPTURE && resultCode ==
RESULT_OK) {
          Bundle extras = data.getExtras();
```

```
            Bitmap imageBitmap = (Bitmap) extras.get("data");
            mImageView.setImageBitmap(imageBitmap);
        }
    }
/** Takes photos and classify them periodically. */
private Runnable periodicClassify =
    new Runnable() {
      @Override
      public void run() {
        synchronized (lock) {
          if (runClassifier) {
            classifyFrame();
          }
        }
        backgroundHandler.post(periodicClassify);
      }
    };
```

In the above code, the original version of this system continually classifies the incoming camera feed through use of the **Runnable** class, which enables continual processes in the Java language. Once the **run()** method of the **Runnable periodicClassify** is called it continually makes calls to the method **classifyFrame()**, which is seen directly above, following the **@Override** annotation. In the writer's work, the aim is to allow the user sufficient time to get their food item in view of the camera before classification begins. Concurrently, only the top result is needed. To achieve this a button was added to the UI with the following code in the **Camera2BasicFragment**:

```
mBtn = view.findViewById(R.id.btn);
mBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        String search = classifyFrame();
        Context context = getContext();
        MealBuilderActivity.searchFoodDatabase(context, search);
    }
});
```

As can be seen in this excerpt, the **classifyFrame()** method was modified to return a string, with this result being used in a database search request of the **MealBuilderActivity**. This modification simply altered the output of the filtering process used within the **ImageClassifier.java** file to ensure it would only post the highest confidence match. That single result is then passed back to the **MealBuilderActivity** which uses it to find the nutritional content of the food.

Chapter 5 - Optimisation, Testing, and Evaluation

This chapter focuses on the optimisation of the application, testing and the evaluation of the application's performance. As a user-centric product, the evaluation is based on the efficiency and efficacy with which its users can perform the designated task, as well as the user's level of satisfaction with the product. A user study was conducted over the course of two months, whereby one user analysed the application at various stages and performed efficiency analysis at each point. The aim of this was to generate feedback during development, and to also have preliminary indicators of the application's efficiency. The results of this User Study will be discussed first, following the UI improvements that developed from this. Following this, the results of the Usability Testing will be presented, ultimately evaluating the project. Some experiments were also performed on the training of the TensorFlow image classifier and the results of this will be included at the end of this section.

5.1 User Study

Over the course of 2 months, from the 28th of January the 8th of April, a hyper-focused user study was conducted with only one participant. As only one participant was involved in this study, the results of the efficiency tests are not transferrable, i.e. the results are not indicative to how the application will perform with a wider audience. The study did produce valuable results however. The user is an active meal logger who currently uses the Android application "Lose it!" to track their food. The user partakes in the "trendy" [100] diet plan known as the "keto-diet". This diet involves restricting carbohydrate consumption to allow the body to enter a state known as ketosis, whereby the body burns lipid-based ketones for fuel instead of glucose derived from carbohydrates [101] [102]. Considering how readily available carbohydrates are in food sources, it is a diet that must be closely monitored for proper execution. As such, every Sunday evening the user prepares her main meal for the weekdays of the coming week in one large batch. Each weeknight she consumes one chicken breast, approximately 250 grams of broccoli, and 20 grams of butter, with each meal being prepared, weighed, and logged the previous Sunday. This was opportune for gathering feedback on the application as it progressed.

Due to the efforts in retraining the image classifier focusing mostly on fruits, the classifier was not applied in these tests. All tests of this section involved analysing the UI of the application at various stages of development, while the use of the Bluetooth kitchen scale was tested in the final user study. For an evaluation of the overall system please refer to Section 5.3.

5.1.1 Study No. 1

The first set of tests conducted with the user took place on the 28th of January. This was performed with an early version of the application. This set a baseline for the application's performance throughout this study, while some points were raised on how the application could be improved.

Table 13: User Study Results: Test 1

Meal No.	Efficiency – Time to Log (s)	Efficacy – No. of Errors
1	185.23	3
2	115.44	4
3	102.35	0
4	102.14	1
5	119.11	1
Avg:	104.43	1.8

5.1.2 Study No. 2

The second study with this user was performed with a version of the application with some improvements made to the user interface. This study took place on Sunday the 25th of February. The effects of the improvements made in the month since the previous study can be seen in the results of Table 14, which shows a reduction in the error rate and an improvement in the efficiency. The user credited this to the implementation of the points raised in the feedback of the previous study

Table 14: User Study Results: Test 2

Meal No.	Efficiency – Time to Log (s)	Efficacy – No. of Errors
1	121.01	2
2	79.12	0
3	69.80	0
4	68.96	1
5	57.48	1
Avg:	79.28	0.8

In comparison to the results of Study No. 1, these results show a 24.1% improvement in the average time taken for this user to log a meal. Similarly, a 55.56% reduction in the average error rate occurred, based on the measurements taken.

5.1.3 Study No. 3

The final study of this section took place on Sunday the 8th of April. In this study, the Bluetooth kitchen scale was implemented as well as some UI improvements. The effects of these developments are noticable in the efficiency recordings of Table 15. The final study showed a 32.29% improvement in average efficiency recordings when compared to the previous study, and a 48.67% improvement in comparison to the initial study. Alternatively, the average efficacy of the application was reduced by 15% in comparison to the previous study, but an overall improvement of one third when compared to the initial study. This increase in application error was due to a Toast message which was included as part of the implementation of the Bluetooth kitchen scale. Unbeknownst to the writer during development, this Toast message was implemented in a way which allowed it to "linger" on the screen of the application, blocking the use of the keyboard in some instances. The Toast message is included in the UI to notify the user of the status of connectivity between the phone and the Bluetooth scale. Details of this issue and its resolution is described in the following section.

Table 15: User Study Results: Test 3

Meal No.	Efficiency – Time to Log (s)	Efficacy – No. of Errors
1	54.26	1
2	52.82	1
3	56.98	2
4	50.88	1
5	53.04	1
Avg:	53.60	1.2

5.2 UI/UX Improvements

Prior to usability testing, the writer used the feedback gained from the User Study to improve the design of the application. This was performed to ensure that the UI was working properly and effectively, and to ensure that bias was removed from the usability testing. For example, if a bug was present in testing which only affected the text-based features of the application, this would produce biased results in the usability testing. Many improvements were made due to the feedback provided, however most of these were minor improvements or purely aesthetic improvements. As such, this section will only address the improvements made which were pertinent to producing an unbiased and accurate result in the usability testing.

5.2.1 Text-based Weight Input

When testing the app with text-based features, it was noted that the **EditText** field of the **WeightActivity** should be overwritten as soon as the user starts entering digits. The **EditText** was previously set to be populated with the value 100 upon creation. This was because the **NutritionalInfoAdapter** updates the nutrient values based on the **EditText** value, and it was designed to initially show the nutrient values per 100g. To implement the suggested improvement the adapter was altered to initially accept 100 as its weight input value and only read the **EditText** value after the user begins entering numbers. The following code shows the final configuration while the commented code is what was previously used:

```
//      weight = Double.valueOf(weightIn.getText().toString());
      weight = 100.0;

      infoAdapter = new NutritionalInfoAdapter(nutrientList, weight);
      itemList.setAdapter(infoAdapter);
```

The XML element of the **EditText** can be seen below. The previously used code included the attribute in comments beneath the element (i.e. **android:text)**, whereas the updated version includes the **android:hint** attribute. The difference between these two attributes is that the text attribute sets the literal value of the field to the value of the assigned string (**default_weight** in this case), whereas the hint attribute shows the value of the string as grey, inactive text behind the field. It is essentially a placeholder which indicates to the user where they should enter values. The string **default_weight** used here is stored in the strings.xml file and contains the text value of “100”. The following is the **EditText** XML element:

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/et_weight"
    android:inputType="number"
    android:layout_margin="5dp"
    android:hint="@string/default_weight"
    android:textSize="16sp"
    android:textColor="@android:color/white"/>
    <!--android:text="@string/default_weight"-->
```

Upon testing this, however, it was noticed that the activity was not working as intended when editing a pre-existing ingredient. Rather than use the stored weight value to calculate the nutrient values on launch, the calculations were defaulting to 100g values. To circumvent this, the following if statement was included:

```
if(weightIn.getText().toString().length() != 0) {
        weight = Double.valueOf(weightIn.getText().toString());
    } else {
        weight = 100.0;
    }

infoAdapter = new NutritionalInfoAdapter(nutrientList, weight);
itemList.setAdapter(infoAdapter);
```

Another suggestion for **WeightActivity** was to allow the "DONE" button of the keypad to save the ingredient as well as the confirm button included in the UI. This was easily implemented by using the in-built Android method **setOnKeyListener** and using the correct keycode for the enter button, as seen in the following code:

```
weightIn = findViewById(R.id.et_weight);
weightIn.setOnKeyListener(new View.OnKeyListener() {
    public boolean onKey(View v, int keyCode, KeyEvent event) {
      if ((event.getAction() == KeyEvent.ACTION_DOWN) && (keyCode ==
KeyEvent.KEYCODE_ENTER)) {
            saveIngredient();
            return true;
        }
        return false;
    }
});
```

With this code implemented, when the "DONE" button of the keypad is pressed, the listener above calls the method **saveIngredient()** which saves the ingredient and returns to the **MealBuilderActivity.**

5.2.2 Obstructive Toast Message

As mentioned above, the user study revealed an instance which could adversely affect the usability testing. While the possibility for affecting the efficacy of the application was apparent, this was a relatively simple problem. In the implementation of the Skale Bluetooth kitchen scale, a **`Toast`** message is declared to inform the user that the application is attempting to pair with the device. The initial build can be seen in the following piece of code:

```
if(mSkaleHelper.isBluetoothEnable()){
    boolean hasPermission = SkaleHelper.hasPermission(this);
    if(hasPermission){
        mSkaleHelper.resume();
        String toastString = "finding skale...";
        Toast.makeText(this,toastString, LENGTH_LONG).show();
    }else{
        SkaleHelper.requestBluetoothPermission(this,
REQUEST_BT_PERMISSION);
    }
}else{
    Intent turnOn = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(turnOn, REQUEST_BT_ENABLE);
}
```

In this code above, it can be seen that the Toast message is set to display for the time "***LENGTH_LONG***", which results in the Toast message not disappearing after this activity is closed. The effect this has on the usability of the application is that it blocks a small section of the keyboard of the device, potentially preventing the user from searching for their next ingredient momentarily.

5.3 Usability Testing

The testing of the application was performed with a set of users from various backgrounds and demographics. With the criteria of usability in mind: efficiency, efficacy, and satisfaction, as proposed by the International Organization for Standardization; the use of the app was analysed with these users to quantify the results of the project. three main tests were conducted with the users:

1. A comparison of food-searching methods: using a general, broadly-trained image classifier to search for foods Vs. the use of text-search features.
2. A comparison of the use of the Bluetooth kitchen and the use of text for recording weight values.
3. A comparison of application approaches: the use of the writer's application Vs. the use of text the popular "Lose It!" application.

Anonymised information of the users is presented in Table 16, while the results of the first test can be seen in Table 17. The first test was conducted with an unoptimized TensorFlow instance of the MobileNet image classifier, trained on 1000 classes from the ImageNet Challenge. This was used for the testing as it was deemed as the most appropriate model for testing purposes. In future builds of this application, with fully optimised models trained for large arrays of food types, the computational times should be similar to the model used here. If the models developed in the retraining process were used here, where the models never exceeded nine classes, this could lead to an imbalanced result. The computational speed of a classifier with only 9 classes would be significantly faster than one with 1000 classes.

Table 16: Anonymised Participant Information

User ID	Age Group	Gender	Mobile OS	Profession	Level of Education Attained – [Pursuing]
1	55+	M	Android	Sales	N/A
2	45 – 54	F	iOS	IT Clinical Lead	MSc – [M.Sc]
3	18 – 26	F	Android	Product Manager	LL.B. Pol. Sc. – [M.Sc]
4	18 – 26	F	Android	University Student	BSc – [M.Sc]
5	18 – 26	M	iOS	Secondary School Student	Junior Cert - [Leaving Cert]
6	18 – 26	F	Android	University Student	B.A
7	27 - 35	M	Android	University Student	Leaving Cert – [B.Sc]

When measuring the efficiency of the image classifier, it was analysed in comparison to relying on text input to search for the food items. Each tester was assigned with the task of logging the same meal twice: once while using the image classifier and once while using the smartphone's keyboard. The order in which each user logged the meals was randomised to compensate for the testers learning how to use the application. For simplicity, the meal logged was the same for both tests: 100g of apple, 200g of oranges, and 300g of banana. These items were chosen as it was confirmed that the classes of apple, oranges and banana were included in the pretrained version of the MobileNet classifier.

Table 17: Results U&A Test 1

User ID	Logging Time - Text (s)	Logging Time – Classifier (s)	Time Difference (%)
1	49.87	43.44	-12.89
2	58.55	40.43	-30.95
3	48.20	36.40	-24.48
4	55.47	43.53	-21.53
5	39.98	43.42	8.6
6	31.33	28.23	-9.89
7	34.19	30.78	-9.97

The second set of tests were held three weeks after the first. This was performed with the same group as the first test, with a focus on the effect of the Bluetooth scale.

Table 18: Results U&A Test 2

User ID	Logging Time - Text (s)	Logging Time – BT Scale (s)	Time Difference (%)
1	123.21	58.85	-52.24
2	137.08	68.88	-49.75
3	87.72	48.84	-44.32
4	N/A	N/A	N/A
5	56.44	37.75	-33.11
6	79.80	88.81	11.29
7	N/A	N/A	N/A

The final set of tests were a comparison of the writer's application to the popular diet management application, "Lose It!".

Table 19: Results U&A Test 3

User ID	Logging Time – Lose It!	Logging Time – BT Scale (s)	Time Difference (%)
1	144.98	54.21	-62.60
2	157.83	59.98	-61.99
3	96.27	49.82	-48.24
4	N/A	N/A	N/A
5	63.02	51.21	-18.74
6	102.17	51.18	-49.9
7	N/A	N/A	N/A

5.4 Retraining the Image Classifier

The process of retraining the image classifier's final layer was analysed to develop an understanding of the minimum requirements for accurate classification. The aim of these experiments was to find a minimum viable number of samples which could be used in an initial build of a broader, food-focused database. Testing began by incrementally increasing sample sizes for each class, with initial tests focusing on five classes of food: apple, orange, banana, tomato, and bell pepper. From initial testing it was found that a minimum of approximately 100 samples should be used. If less than 100 samples are used in each class, the results proved to be unusable. It was common for the training process to fail with less than 100 samples also. This is due to TensorFlow's training process, whereby an initial analysis of the images splits each class into 3 sets: train, test, and validation. Without enough samples the validation set would often be assigned none of the samples, which would end the training process.

Tests were then done where the classifier was trained with only 100 samples of each class, and then the number of food classes was incrementally increased. The writer developed an independent sample set with which this was tested. The process was split into 5 experiments with 5 tests in each. In each of the 5 tests within these experiments was a sample set of training data. For Experiment 1, the training data had 5 classes, for Experiment 2 there were 6 classes. This was incremented up to a fifth experiment, which had 9 classes in each of its five sample sets. The first five classes are those mentioned above: apples, oranges, banana, tomato and bell pepper. By Experiment 5 this was increased to include 100 samples each of raspberries, blackberries, blueberries, and strawberries. In total, 4500 sample pictures were used in the training process, while a further 45 samples were used in the testing process (5 samples of each of the 9 classes).

This was a monumental task which required a substantial amount of processing power and produced a significant amount of data. The findings, however, were not conclusive enough to warrant an analysis here, so they will be summarised in the upcoming section. A link to a folder containing this data will be provided in Appendix B.

5.4.1 TensorFlow Training Summary

These tests were all performed on an unoptimized version of the image classifier. The one conclusion that can be made from the process is that, with more classes added to the classifier, the computational performance decreases. Each set of experiments increased the computational time, but the analysis of training methods of the system for accuracy were somewhat inconclusive. The amount of data used is relatively arbitrary, as the quality of data appears to be the biggest influence in the accuracy and reliability of the training. For instance, the most accurate training set was the one which showed the greatest variety amongst each class. This indicated that the quality of samples may be more relevant than the number of samples. Further testing was done whereby the sample rate was increased, from the standard 500 steps up to 8000 steps. These training methods produced more accurate results on the same sample sets, however they are computationally expensive, with some training processes taking over six hours to compute.

Chapter 6 - Discussion

The work of this project has shown a novel and effective approach to diet management applications. The methods used have provided a measurable improvement in the user experience and confirmed the hypothesis set forth by the writer. It should be noted that it is only a confirmation of a hypothesis however, with the analysis performed in this study being equivalent to an alpha-testing phase. To further this work, a beta-testing phase is required. The results are promising in terms of accessibility also, but further work would be required to confirm this.

This project progressed from early work which was solely focused on developing the meal-logging features, such as local device storage and the retrieval of nutritional information, to the later work of implementing the image classification and Bluetooth capabilities. Overall, the main objectives of the project were achieved, however further analysis is needed regarding the training of image classifiers. The resources required to compute these training systems were much more significant than initially anticipated. Ultimately, the classifier retraining was outside of the scope of this project, and this should have been identified sooner. That said, the knowledge gained through the experience of the process was useful, even though the data collected from the process was not. The training process required Linux based programming, which was a new area for this writer. Better still, the lessons learned from the retraining process could be implemented to better use, on a grander scale in future. And, with this basis of research and a prototype developed, further work can and will be performed.

6.1 Ethical Considerations

In the design of a product an engineer must consider all appropriate ethical implications and ensure that the design meets the necessary requirements and guidelines. As this product intersects with the healthcare industry, the ethical guidelines and principles of this industry should also be considered. Within engineering, ethical considerations are a prominent feature of the profession. Without the proper ethical considerations, the risk of producing unethical results is apparent. Take for example the recent Facebook-Cambridge Analytica data scandal. This is a case where software was engineered to mine the data of Facebook users and the entirety of their network [103]. The data breach was originally reported to have leaked the personal information of over 50 million Facebook users, which was used to influence political campaigns such as the Trump campaign [104] and the Brexit Vote Leave campaign [105]. Facebook has since announced that the number of people affected by the data breach could be up to 87 million people [106]. One report, which emphasises the virality of Cambridge Analytica's engineering design, illustrates how the interaction between this software and only 335 users may have led to the exposure of the personal information of over 550,000 users [107].

The Institution of Engineers of Ireland (or Engineers Ireland) has developed a Code of Ethics which was updated on the 1st of January 2018 [108]. This Code of Ethics outlines the requirements of the engineering professional to meet social and environmental obligations, as well as guidelines for maintaining a professional standard. It also addresses the responsibility of the engineering professional in regard to their relations with colleagues, clients, employers, and society in general. In the section regarding an engineer's relations with others, Code 1.9 states:

> *"At all times in their relations with the public, Members shall apply their skill and experience to the common good and the advancement of human welfare with proper regard for the safety, health and welfare of the public. A Member shall not engage in any activity which he/she knows or has reasonable grounds for believing is likely to result in a serious detriment to any person or persons."*

This code is relevant to all mobile applications which are publicly released and openly available. Similarly, Code 2.1 bears relevance to mobile applications:

> *"Members shall at all times be conscious of the effects of their work on the health and safety of individuals and on the welfare of society. While acting as designers, operators or managers on projects, members shall strive to eliminate risks to health and safety during all project stages. Members shall also undertake to minimise or eliminate any adverse impact on the natural environment arising from the design and execution of all project work that they are engaged in."*

The above Codes draw attention to the responsibility and obligation of the engineering professional to produce their work with others in mind. Those affected or who have the potential to be affected by the result of an engineer's efforts should always be considered. Code 2.1 ends with reference to protecting the environment. Code 2.3 expands upon this:

> *"Members shall strive to accomplish the objectives of their work with the most efficient consumption of natural resources which is practicable economically, including the maximum reduction in energy usage, waste and pollution."*

This Code is particularly relevant to mobile applications and software development. Improper practices and poor software development principles could potentially lead to energy waste. For example, excessive programmatic procedures could perform unnecessary computations, or an application could be poorly designed so that data requests are made, or other services are used when they are not required. These actions would cause the mobile device's battery to drain faster, requiring more frequent recharging. In today's energy generation climate, this is likely to result in greenhouse gas emissions and an adverse effect on the environment. It also has another, "knock-on" effect in that it reduces the lifespan of the smartphone's battery unnecessarily, as each recharge affects battery life [109]. This could result in a replacement being needed or, in the case of smartphones with non-replaceable batteries, an entirely new device. Due to the necessity for adequate recycling of Lithium-ion batteries, this could result in additional harm to the environment if the batteries are improperly disposed of. Every measure was taken in the development of this application with this ethical consideration in mind. The implementation of software which is new to the writer involved researching the best practices for their use. An example of this is in the use of the new Android Room data persistence library, as the computational effect of annotating single methods or the entire class with the **`@TypeConverter`** was considered. These actions were performed with respect to another ethical obligation outlined by Engineers Ireland in Code 3.4:

> *"Members shall carry out their work with due care, skill, diligence and expedition consistent with good practice."*

Best practices in software development were deployed throughout this project, and careful consideration was made to the design and execution of the system. This was considered as the due diligence required to perform this project. In other instances, the lack of due diligence in the engineering process has led to serious ethical implications. In the Health Service Executive's (HSE) final report of what is known as the "NIMIS '<' Symbol Incident", poor software development practices and a lack of care and diligence produced a significant incident which "had the potential to cause patient harm" [110]. The National Integrated Medical Imaging System, or NIMIS, is a national system that was developed by the HSE as an internal platform for requesting medical imaging examinations. NIMIS is comprised of multiple subsystems for managing the medical examination requests, and the technical issue which caused the incident arose from a "less than" symbol (<) being recorded in some aspects of the system, but not transferring to others. The HSE gives an example of the inherent risk of this technical issue: if a clinician was to report that a patient had a < 50% stenosis in their internal carotid artery, this would appear downstream as a report that the patient had a 50% stenosis, which could result in the incorrect recommendation to undergo immediate surgery. This incident, which is reported by the HSE as being the first worldwide instance where a technical issue had such a substantial effect on a national imaging system, resulted in 24,275 erroneous examination reports. This incident hearkens to the incidents involving the Therac-25 radiation therapy machine between 1985 and 1987 [111]. This was a software-controlled machine which, due to poor software design, resulted in multiple deaths and injuries due to radiation overdose. The scale and potential for harm of the NIMIS incident, as well as the unfortunate incidents produced by the Therac-25, accentuate the responsibility of the engineer to ensure that they perform their work with due care and diligence.

Regarding the ethical guidelines of the healthcare industry, multiple sources were researched and analysed. The Declaration of Helsinki [112] outlines the ethical principles for research with human subjects. While it is directed at the medical community and experiments which focus on the research of humans in clinical settings, it has relevance to this project. The document outlines that the researcher must "ensure respect for all human subjects", while protecting the "privacy and confidentiality of the personal information of the research subjects". This amplifies the importance of anonymising the data of the users involved in the testing process. The Nuremberg Code [113] is another document which is relevant to this project. It was

originally developed over the course of the Nuremberg Trials in 1947 where members of the Nazi party were made accountable for war crimes they had committed [114]. The document consists of 10 points of ethical considerations for research involving human participants, however only one of these points are prominently relevant to the research performed in this project. The first point made in the document underlines the responsibility and duty of the researcher to ensure that the participants have informed consent regarding their participation in the research. As such, the writer explained the study to the participants and obtained a signed consent form for each participant.

Information governance is another area of focus in the ethical considerations of both the healthcare and software development industries. The American Health Information Management Association (AHIMA) sets forth a list of ethical principles for organisations which deal with health information [115]. They insist that any organisation which deals with a person's health information, regardless of the organisations role, must make a "firm commitment" to responsibly and ethically handle the personal information of its users. The ethical principles they propose are to ensure accountability, transparency, and reliability when dealing with user's data, while taking measures to uphold the integrity of that data. AHIMA asserts that an organisation which stores health information has an ethical responsibility to ensure that the user's data is available to them. They highlight the responsibility of an organisation to both effectively preserve and dispose of data where requested, and that appropriate data administration efforts should be made. This includes routinely backing up data to avoid losses in the cases of malfunctions, data corruption, or disaster, as well as the effective migration of data when upgrading systems or replacing obsolete technologies. They also highlight the importance of a well-designed and properly maintained database with regard to making a user's data actively and effectively accessible.

The remaining ethical principles which AHIMA promotes, the principles of protection and compliance, are relevant to virtually all mobile applications, regardless of the industry or setting in which they are applied. AHIMA calls attention to the responsibility of the engineer and organisation which ensures that any user's data is protected from the moment it is created to the moment it is destroyed, regardless of interruptions in service or otherwise. A particular emphasis is placed upon personally identifiable information as this is the most sensitive user data, with the greatest potential for ethical misuse. Taylor [116] accounts of how the UK National Health Service had sold the data of 47 million patients to insurance companies, with which the insurance companies used to adjust premiums. Taylor also discusses a case of how,

in 2011, Harvard researcher Latanya Sweeney bought access to anonymised patient data, however she could "confidently identify" 35 patients within this database from news reports of the area containing the word "hospitalisation". In ten of these cases, the anonymised data contained sensitive information, such as "a patient's venereal disease, drug dependency, alcohol use or payment issues".

The area of data protection intersects with the AHIMA's principle for compliance, and Engineers Ireland's Code of Ethics, with Code 3.5 stating:

> *"Members shall be familiar with the substance and intent of national, European Union and other legislation relevant to their field of engineering practice."*

The Irish Data Protection Act of 1988 [117], the Amendment to this Act in 2003 [118], as well as the ePrivacy Regulations of 2011 [119] are the national legislations relevant to this project. The incoming General Data Protection Regulation (GDPR) [120]is also highly relevant to the future of this project. These legislations state that the legal obligation which a data controller has to protect and maintain the data of the users it serves, as well as requirements for the disposition of data. These documents focus on the legal requirements regarding the collection and handling of personal data.

The application developed in the writer's work, in its current form, does not collect any user information. However, this type of application typically includes profile creation for saving data to online databases, or to allow social interactions through the application. With this implemented, the onus is upon the service provider (i.e. the writer or a subsequent company if the application is publicly launched) to take the necessary measures to ensure that this data is not compromised. The utmost care and consideration must be taken when designing and implementing security systems for the databases. That said, even without user profile creation and user information being stored in online databases, this application could be manipulated to compromise a user's data.

In the current version of the writer's application, no security features have been implemented. This lack of security is accompanied by a series of sensitive permission requirements which are needed to use the application: to save data, the application requires both read and write access to the device's storage; to retrieve information from the USDA database, the application requires internet access; for use with the Bluetooth kitchen scale the application requires both Bluetooth services access, as well as the coarse location of the device. While the coarse location of the device is necessary for the Bluetooth implementation method used in the current version

of the application, this may be removed in future versions. Nevertheless, in its current form, this application has the potential to leak the information of any user. A hacker could potentially connect to the application over Bluetooth, access the entire contents of the smartphone's storage, and transmit this over the internet.

Point 3.3 of the Engineers Ireland Code of Ethics:

> *"Members shall accept and perform only work for which they are qualified and competent to undertake and shall obtain whatever advice and assistance is necessary to discharge this responsibility."*

In line with the writer's ethical obligation, the writer acknowledges the limitations of this project, as well as the writer's ability. The writer has no experience in the design or implementation of security systems and, as such, it is paramount that adequate assistance is sought when this project is further developed to a commercial level. The aim of the writer is for this project to progress to a stage where it is feasible to hire another engineer with experience in the field of security and data protection. In the absence of this capability, it will be necessary for the writer to find a supervisor who can advise on this topic while the writer undertakes training to account for this deficit, if the application is to be published. Another limitation of the project which should be noted is the potential for misuse. All products designed have a potential for misuse, however misuse of this application could arise amongst users with eating disorders. For example, an anorexic person could misuse this application by ensuring they maintain a caloric intake which is below their basal metabolic rate. Further research is needed to identify the optimal method for addressing this issue, and the writer will collaborate with Irish Platform for Patient Organisations, Science and Industry (IPPOSI) has in this research. For anorexic users, the current proposition is to implement a warning that triggers if the abovementioned misuse is detected, alerting the user of the harm of their actions. The efficacy of this will hopefully be confirmed in the coming months, and the writer also hopes to work with IPPOSI to conduct a study with Dyslexic users.

Chapter 7 - Conclusions

This work produced an application which has met the intended project objectives. The writer's hypothesis was confirmed by successfully implementing image classification and a Bluetooth kitchen scale with a typical calorie-counting application format. The evaluation of the application, which is a user-centred product, was performed through the usability testing. The results of these tests proved that the innovative technologies used can improve upon existing solutions in the field of diet-management applications. In the process of this work, the relevant literature was addressed as well as the ethical implications of the work.

7.1 Future Work

The work completed for this project has produced the first iteration of this application, and the writer intends to further improve the application and prepare it for a commercial launch. The application is far from finished, and the work performed in this project is essentially a proof of concept. With the prototype built and some user feedback received, the writer intends to iteratively improve upon the current version. The first improvement will be to remove requests to the application's database from the main thread of the application. While a relatively trivial data is currently retrieved from the device's storage, it is still considered best practice to perform all database requests on background threads. This is because database requests made on the main thread can affect the UI [121]. Performing these requests on main threads could affect the performance in future builds also, as the writer intends to implement localised food and nutrient databases to facilitate offline meal logging, or meal logging in areas with poor internet connectivity.

Additional features are planned for the application, such as visual displays of daily and weekly nutritional values. As discussed in the literature review, bar charts are an effective means if transmitting health data, particularly for diabetic users. As such bar charts will be implemented, but the use of these will be compared to pie charts in a user-focused case study. The writer also intends to extend the meal logging capabilities of the application to facilitate those who eat more than 3 main meals a day. A diet of 6, smaller meals is common amongst the bodybuilding and weight-loss communities [122] [123] and some clinicians recommend 6 – 10 small meals to alleviate the symptoms of some diseases [124].

Once the application has been tested, improved, and optimised, case studies and customer validation will be performed in other countries. With North America currently encapsulating the largest proportion of the market share for calorie-counting and meal-logging applications the writer plans to conduct research there initially. A focus will be placed upon finding active users of other application in this category to conduct the case studies. The aim will be to gather valuable and precise feedback on the writer's work, as well as the points of frustration that exist in current applications. While this work has made the meal-logging process more efficient and user-friendly, there could be other problems within this category of application that could be solved. Efforts will also be made to develop an understanding of the user's perspective and the context of use for these applications. Following user validation, research will be performed on

the market and a route to market for North America will be developed, before researching and performing customer validation in other markets.

For each market entered, a localised version will need to be created. For North American markets it could be reasonably assumed that the same version as produced in Ireland could be applicable and effective there, however the writer aims to research this first. An objective that will be tackled before other markets, however, will be to improve the user experience for Dyslexic and illiterate users. For these users, the removal of text where possible is a necessity. In terms of purely technical work, the application will be expanded to support a broader range of API levels, allowing the application to be used on older or lower-level devices. The use of a cross-platform framework such as Google's Flutter or Facebook's React, which allow simultaneous deployment on both Android and iOS, will be explored. Further tests will also be performed on the retraining of image classifiers, and a complete, food-focused image classification model will be developed in the coming months.

Appendices

Appendix A - Android Project Files

The entire project can be found at the following Github link:

https://github.com/LukeScales1/FYP

Appendix B - Classifier Retraining Results

All data pertaining to the retraining process can be found at:

https://drive.google.com/drive/u/0/folders/1sYbecSiDOrZB7wUMuiajunt19htCPhbm

References

[1] Reuters, “mHealth Market Worth $23 Billion in 2017 and Estimated to Grow at a CAGR of more than 35% over the next three years,” Orbis Research, 18 April 2017. [Online]. Available: https://www.reuters.com/brandfeatures/venture-capital/article?id=4640. [Accessed 2018 March 13].

[2] Statista, “mHealth (mobile health) industry market size projection from 2012 to 2020 (in billion U.S. dollars),” 2013. [Online]. Available: https://www.statista.com/statistics/295771/mhealth-global-market-size/. [Accessed 2018 March 13].

[3] Zion Market Research, “Global mHealth Market Set for Rapid Growth, to Reach USD 102.43 Billion by 2022,” 12 December 2016. [Online]. Available: https://www.zionmarketresearch.com/news/global-mhealth-market. [Accessed 08 April 2018].

[4] World Health Organisation, “Controlling the global obesity epidemic,” [Online]. Available: http://www.who.int/nutrition/topics/obesity/en/. [Accessed 12 March 2018].

[5] World Health Organisation, “Overweight and obesity,” 2017. [Online]. Available: http://www.who.int/gho/ncd/risk_factors/overweight/en/. [Accessed 23 March 2018].

[6] A. Must, J. Spadano and E. H. Coakley, “The Disease Burden Associated With Overweight and Obesity,” 27 October 1999. [Online]. Available: https://jamanetwork.com/journals/jama/fullarticle/192030. [Accessed 24 March 2018].

[7] World Health Organisation, “mHealth: New horizons for health through mobile technologies,” 2011. [Online]. Available: http://www.who.int/goe/publications/goe_mhealth_web.pdf. [Accessed 20 March 2018].

[8] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM,* vol. 60, no. 6, pp. 84-90, 2017.

[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going deeper with convolutions," ArXiv, 2014.

[10] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy and e. al, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision,* vol. 115, no. 3, pp. 211-252, 2015.

[11] L. Columbus, "2017 Roundup Of Internet Of Things Forecasts," Forbes, 10 December 2017. [Online]. Available: https://www.forbes.com/sites/louiscolumbus/2017/12/10/2017-roundup-of-internet-of-things-forecasts/#3ddc2e51480e. [Accessed 8 April 2018].

[12] C. Hill, "The 5 most (and least) popular diet and fitness apps," Market Watch, 20 January 2015. [Online]. Available: https://www.marketwatch.com/story/the-5-most-and-least-popular-diet-and-fitness-apps-2015-01-06. [Accessed 8 April 2018].

[13] Google Play, "Lose It! - Calorie Counter," Google, [Online]. Available: https://play.google.com/store/apps/details?id=com.fitnow.loseit&hl=en. [Accessed 8 April 2018].

[14] P. Langley, "Editorial: On Machine Learning," *Machine Learning,* vol. 1, no. 1, pp. 5-10, 1986.

[15] R. J. Passonneau, V. Bhardwaj, A. Salleb-Aouissi and N. Ide, "Multiplicity and word sense: evaluating and learning from multiply labeled word sense annotations," *Language Resources and Evaluation,* vol. 46, no. 2, pp. 219-252, 2012.

[16] A. C. o. Pearls, "Machine Learning: What and Why?," Tumblr, San Francisco, 2013.

[17] Genetic Programming, "What We Mean by "Machine Intelligence"," Stanford, 2003.

[18] A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers," *IBM Journal,* vol. III, no. 3, pp. 535-, 1959.

[19] S. Byford, “AlphaGo retires from competitive Go after defeating world number one 3-0,” The Verge, 27 May 2017. [Online]. Available: https://www.theverge.com/2017/5/27/15704088/alphago-ke-jie-game-3-result-retires-future. [Accessed 1 April 2018].

[20] C. Cadell, “Google AI beats Chinese master in ancient game of Go,” Reuters, 23 May 2017. [Online]. Available: https://www.reuters.com/article/us-science-intelligence-go/google-ai-beats-chinese-master-in-ancient-game-of-go-idUSKBN18J0PE. [Accessed 30 March 2018].

[21] J. Tromp and G. Farneback, “https://tromp.github.io/go/gostate.pdf,” Github, Princeton, 2016.

[22] R. Kohavi and F. Provost, “On Applied Research in Machine Learning,” *Machine Learning,* vol. 30, no. 2-3, pp. 127-132, 1998.

[23] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, et al, “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems,” Google Research, 9 Novemeber 2015. [Online]. Available: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45166.pdf. [Accessed 2018 March 24].

[24] P. Brazdil and C. Giraud-Carrier, “Metalearning and Algorithm Selection: progress, state of the art and introduction to the 2018 Special Issue,” *Machine Learning,* vol. 107, no. 1, pp. 1-14, 2018.

[25] D. Tomé, F. Monti, L. Baroffio, L. Bondi, M. Tagliasacchi and S. Tubaro, “Deep Convolutional Neural Networks for pedestrian detection,” *Signal Processing: Image Communication,* vol. 47, no. 1, pp. 482-489, 2016.

[26] Y. Taigman, et al., “Deepface: closing the gap to human-level performance in face verification,” in *Conference on Computer Vision and Pattern Recognition (CVPR),* Columbus, 2014.

[27] C. Lu and X. Tang, “Surpassing human-level face verification performance on LFW with GaussianFace,” in *29th AAAI Conference on Artificial Intelligence (AAAI),* Austin, 2014.

[28] Y. Sun, X. Wang and X. Tang, “Deep learning face representation from predicting 10,000 classes,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, Columbus, 2014.

[29] M. Matsugu, K. Mori, Y. Mitari and Y. Kaneda, “Subject independent facial expression recognition with robust face detection using a convolutional neural network,” *Neural Networks*, vol. 16, no. Special Issue, pp. 555-559, 2003.

[30] H. Ting, B. Yong and S. M. Mirhassani, “Self-Adjustable Neural Network for speech recognition,” *Engineering Applications of Artificial Intelligence*, vol. 26, no. 9, pp. 2022-2027, 2013.

[31] O. Abdel-Haimid, A. Mohamed, H. Jiang, L. Deng, G. Penn and D. Yu, “Convolutional Neural Networks for Speech Recognition,” *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 22, no. 10, pp. 1533-1545, 2014.

[32] Y. LeCun and Y. Bengio, “Convolutional networks for images, speech, and time series.,” 1995. [Online]. Available: http://yann.lecun.com/exdb/publis/pdf/lecun-bengio-95a.pdf. [Accessed 6 April 2018].

[33] D. H. Hubel and T. N. Wiesel, “Receptive fields of single neurons in the cat's striate cortex,” *The Journal of Physiology*, vol. 148, no. 3, pp. 574-591, 1959.

[34] K. Fukushima, “Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position,” *Biological Cybernetics*, vol. 36, no. 4, pp. 193-202, 1980.

[35] K. Fukushima, “Neocognitron for handwritten digit recognition,” *Neurocomputing*, vol. 51, no. 1, pp. 161-180, 2003.

[36] K. Fukushima, “Neocognitron trained with winner-kill-loser rule,” *Neural Networks*, vol. 23, no. 7, pp. 926-938, 2010.

[37] K. Fukushima, “Training multi-layered neural network neocognitron,” *Neural Networks*, vol. 40, no. 1, pp. 18-31, 2013.

[38] D. G. Lowe, “Distinctive Image Features from Scale-Invariant Keypoints,” *International Journal of Computer Vision,* vol. 60, no. 2, pp. 91-110, 2004.

[39] L. Zheng, Y. Yang and Q. Tian, “SIFT Meets CNN: A Decade Survey of Instance Retrieval,” *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 40, no. 5, pp. 1224-1244, 2018.

[40] J. Sánchez, F. Perronnin, T. Mensink and J. Verbeek, “Image Classification with the Fisher Vector: Theory and Practice,” *International Joutrnal of Computer Vision,* vol. 105, no. 3, pp. 222-245, 2013.

[41] J. Wang, P. Liu, M. She, S. Nahavandi and A. Kouzani, “Bag-of-words representation for biomedical time series classification,” *Biomedical Signal Processing and Control,* vol. 8, no. 6, pp. 634-644, 2013.

[42] G. Csurka, C. R. Dance, L. Fan, J. Willamowski and C. Bray, “Visual categorization with bags of keypoints,” Xerox Research Centre Europe, Meylan, 2004.

[43] S. Xu, T. Feng and D. Li, “Object Classification of Aerial Images With Bag-of-Visual Words,” *IEEE Geoscience and Remote Sensing Letters, 2010, Volume 7, Issue 2,* vol. 7, no. 2, pp. 366-370, 2010.

[44] E. Karakasis, A. Armanatiadis and A. Gasteratos, “Image moment invariants as local features for content based image retrieval using the Bag-of-Visual-Words model,” *Pattern Recognition Letters, 04/2015, Volume 55,* vol. 55, no. 1, pp. 22-27, 2015.

[45] J.M. dos Santos, E.S. de Moura, A.S. da Silva, R. da Silva Torres, “Color and texture applied to a signature-based bag of visual words method for image retrieval,” *Multimedia Tools and Applications,* vol. 76, no. 15, pp. 16855-16872, 2016.

[46] H. Wu, B. Liu, W. Su, Z. Chen, W. Zhang, X. Ren and J.Sun, “Optimum Pipeline for Visual Terrain Classification Using Improved Bag of Visual Words and Fusion Methods,” *Journal of Sensors,* vol. 2017, no. 1, pp. 1-25, 2017.

[47] Y. LeCun, “"Hi Serge",” 26 June 2013. [Online]. Available: https://plus.google.com/+YannLeCunPhD/posts/gurGyczzsJ7. [Accessed 23 March 2018].

[48] Y. LeCun, “Yann LeCun's Publications,” 2014. [Online]. Available: http://yann.lecun.com/exdb/publis/index.html#fulllist. [Accessed 2018 April 7].

[49] Y. LeCun, L. Najman, C. Couprie and C. Farabet, “Scene Parsing with Multiscale Feature Learning, Purity Trees, and Optimal Covers,” in *Proc. International Conference on Machine learning (ICML'12),* Edinburgh, 2012.

[50] A. Krizhevsky, I. Sutskever and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” 3 December 2012. [Online]. Available: https://www.cs.toronto.edu/~fritz/absps/imagenet.pdf. [Accessed 23 March 2018].

[51] A. G. Howard, et. al, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision,” ARXIV, 2017.

[52] J. J. Garrett, The elements of user experience: user-centered design for the Web, Indianapolis: New Riders, 2002.

[53] Google Scholar, “Citations for J.J. Garret's "The elements of user experience: user-centered design for the web and beyond",” [Online]. Available: https://scholar.google.com/scholar?um=1&ie=UTF-8&lr&cites=5423227805243159339. [Accessed 6 April 2018].

[54] K. Harris, S. Baron and C. Parker, “Understanding the Consumer Experience: It's 'Good To Talk',” *Journal of Markerting Management,* vol. 16, no. 3, pp. 111-127, 2000.

[55] S. Smith, G. C. Smith and Y. Chen, “A KE-LSA approach for user-centered design,” *Journal of Intelligent Manufacturing,* vol. 24, no. 5, pp. 919-933, 2013.

[56] J. A. Bacha, “Mapping Use, Storytelling, and Experience Design: User-Network Tracking as a Compnent of Usability and Sustainability,” *Journal of Business and Technical Communication,* vol. 32, no. 2, pp. 198-228, 2018.

[57] C. Lin and L. Cheng, “Product attributes and user experience design: how to convey product information through user-centered service,” *Journal of Intelligent Manufacturing,* vol. 28, no. 7, pp. 1743-1754, 2017.

[58] ISO (the International Organization for Standardization) , “ISO 9241-11:2018(en): Ergonomics of human-system interaction — Part 11: Usability: Definitions and

concepts," March 2018. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-2:v1:en. [Accessed 06 April 2018].

[59] J. Applen and S. Stephens, "Digital Humanities, Middleware, and User Experience Design for Public Health Applications," *Communication Design Quarterly Review,* vol. V, no. 3, pp. 24-34, 2017.

[60] J. Jones, "Information Graphics and Intuition: Heuristics as a Techne for Visualization," *Journal of Business and Technical Communication,* vol. 29, no. 3, pp. 284-313, 2015.

[61] B. Choi, I. Lee and K. Kim, "Culturability in Mobile Data Services: A Qualitative Study of the Relationship Between Cultural Characteristics and User-Experience Attributes," *INTERNATIONAL JOURNAL OF HUMAN–COMPUTER INTERACTION,* vol. 20, no. 3, pp. 171-206, 2006.

[62] A. Marcus and E. W. Gould, "Crosscurrents: Cultural dimensions and global Web user-interface design," *Interactions,* vol. 7, no. 4, pp. 32-46, 2000.

[63] K. Saint-Amant, "Introduction to the special issue: Cultural considerations for communication design: integrating ideas of culture, communication, and context into user experience design," *Communication Design Quarterly Review,* vol. 4, no. 1, pp. 6-22, November 2015.

[64] Z. Hildon, D. Allwood and N. Black, "Impact of format and content of visual display of data on comprehension, choice and preference: a systematic review," *International Journal for Quality in Health Care,* vol. 24, no. 1, pp. 55-64, 2012.

[65] Android Developers, "Architecture Components > Release Notes," Google, [Online]. Available: https://developer.android.com/topic/libraries/architecture/release-notes.html. [Accessed 29 March 2018].

[66] Android Developers, "Android Studio: The Official IDE for Android," Google, [Online]. Available: https://developer.android.com/studio/index.html. [Accessed 29 March 2018].

[67] JetBrains, "IntelliJ IDEA: The Java IDE for Professional Developers by JetBrains," [Online]. Available: https://www.jetbrains.com/idea/.

[68] Android Developers, “Meet Android Studio,” Google, [Online]. Available: https://developer.android.com/studio/intro/index.html. [Accessed 2018 March 28].

[69] Android Developers, “Android Studio Release Notes > Older Releases,” Google, [Online]. Available: https://developer.android.com/studio/releases/index.html#older-releases. [Accessed 29 March 2018].

[70] Android Developers, “Android Studio Release Notes,” Google, [Online]. Available: https://developer.android.com/studio/releases/index.html. [Accessed 29 March 2018].

[71] M. Cleron, “Android Announces Support for Kotlin,” Google, 17 May 2017. [Online]. Available: https://android-developers.googleblog.com/2017/05/android-announces-support-for-kotlin.html. [Accessed 25 March 2018].

[72] Android Developers, “Activity,” Google, [Online]. Available: https://developer.android.com/reference/android/app/Activity.html. [Accessed 25 March 2018].

[73] Android Developers, “android.database.sqlite,” Google, [Online]. Available: https://developer.android.com/reference/android/database/sqlite/package-summary.html. [Accessed 29 March 2018].

[74] The SQLite Development Team, “About SQLite,” [Online]. Available: https://www.sqlite.org/about.html. [Accessed 29 March 2018].

[75] w3schools, “SQL,” [Online]. Available: https://www.w3schools.com/sql/default.asp. [Accessed 29 March 2018].

[76] Google Developers, “Android Persistence codelab,” Google, [Online]. Available: https://codelabs.developers.google.com/codelabs/android-persistence/#0. [Accessed 29 March 2018].

[77] S. Pichai, “TensorFlow: smarter machine learning, for everyone,” Google, 9 November 2015. [Online]. Available: https://googleblog.blogspot.ie/2015/11/tensorflow-smarter-machine-learning-for.html. [Accessed 23 March 2018].

[78] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang and A. Y. Ng, “Large Scale Distributed Deep Networks,”

2012. [Online]. Available: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/40565.pdf. [Accessed 14 March 2018].

[79] TensorFlow, "TensorFlow In Use," [Online]. Available: https://www.tensorflow.org/about/uses. [Accessed 24 March 2018].

[80] TensorFlow, "Welcome to the TensorFlow Community," [Online]. Available: https://www.tensorflow.org/community/welcome.

[81] TensorFlow, "Computation using data flow graphs for scalable machine learning," Google, [Online]. Available: https://github.com/tensorflow/tensorflow. [Accessed 24 March 2018].

[82] TensorFlow, "Introduction to TensorFlow Lite," [Online]. Available: https://www.tensorflow.org/mobile/tflite/.

[83] TensorFlow, "tensorflow/tensorflow/contrib/lite/," [Online]. Available: https://github.com/tensorflow/tensorflow/tree/master/tensorflow/contrib/lite. [Accessed 24 March 2018].

[84] Nutrient Data Laboratory, USDA, "Welcome to the USDA Food Composition Databases," United States Department of Agriculture, [Online]. Available: https://ndb.nal.usda.gov/ndb/. [Accessed 28 March 2018].

[85] Android Developers, "The Activity Lifecycle," Google, [Online]. Available: https://developer.android.com/guide/components/activities/activity-lifecycle.html. [Accessed 29 March 2018].

[86] Oracle, "Interface Comparable<T>," [Online]. Available: https://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html. [Accessed 29 March 2018].

[87] Food Safety Authority of Ireland, "The Science of Salt & Health," 22 April 2016. [Online]. Available: https://www.fsai.ie/science_and_health/salt_and_health/the_science_of_salt_and_health.html. [Accessed 30 March 2018].

[88] PC Mag, “Definition of: utility program,” [Online]. Available: https://www.pcmag.com/encyclopedia/term/53588/utility-program. [Accessed 28 March 2018].

[89] Android Developers, “android.util,” Google, [Online]. Available: https://developer.android.com/reference/android/util/package-summary.html. [Accessed 28 March 2018].

[90] CodePath, “Organizing your Source Files,” [Online]. Available: https://guides.codepath.com/android/Organizing-your-Source-Files. [Accessed 28 March 2018].

[91] Android Developers, “Uri.Builder,” Google, [Online]. Available: https://developer.android.com/reference/android/net/Uri.Builder.html. [Accessed 31 MArch 2018].

[92] Oracle, “Class Scanner,” [Online]. Available: https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html. [Accessed 31 March 2018].

[93] pat-522181 (Oracle Developer Community Username), “Stupid Scanner tricks... Blog,” 23 October 2004. [Online]. Available: https://community.oracle.com/blogs/pat/2004/10/23/stupid-scanner-tricks. [Accessed 31 March 2018].

[94] Android Developers, “Calendar,” Google, [Online]. Available: https://developer.android.com/reference/java/util/Calendar.html. [Accessed 27 March 2018].

[95] Android Developers, “ListView,” Google, [Online]. Available: https://developer.android.com/reference/android/widget/ListView.html. [Accessed 2018 March 27].

[96] Android Developers, “AdapterView,” Google, [Online]. Available: https://developer.android.com/reference/android/widget/AdapterView.html. [Accessed 27 March 2018].

[97] Android Developers, “DatePickerDialog.OnDateSetListener,” Google, [Online]. Available: https://developer.android.com/reference/android/app/DatePickerDialog.OnDateSetListener.html. [Accessed 31 MArch 2018].

[98] Android Developers, “BottomSheetDialogFragment,” Google, [Online]. Available: https://developer.android.com/reference/android/support/design/widget/BottomSheetDialogFragment.html. [Accessed 27 March 2018].

[99] Skale 2, “The bluetooth electronic scale with an open source SDK,” [Online]. Available: http://www.skale.cc/en/skale_open_sdk.html.

[100] M. Deliso, “Keto: What to know about the trendy low-carb, high-fat diet,” AM New York, 18 March 2018. [Online]. Available: https://www.amny.com/eat-and-drink/keto-diet-recipes-1.17484228. [Accessed 6 April 2018].

[101] I. Strychar, “Diet and weight loss,” *Canadian Medical Association Journal*, vol. 175, no. 11, p. 1407, 2006.

[102] A. Halevy, L. Peleg-Weiss, R. Cohen and A. Shuper, “An Update on the Ketogenic Diet,” *Rambam Maimonides Medical Journal,* vol. 3, no. 1, pp. 1-9, 2012.

[103] O. Ryan, “Data Commissioner 'actively supervising Facebook’s progress in cleaning up its act',” The Journal, 5 April 2018. [Online]. Available: http://www.thejournal.ie/zuckerberg-facebook-3940999-Apr2018/. [Accessed 6 April 2018].

[104] M. Rosenberg, N. Confessore and C. Cadwallad, “How Trump Consultants Exploited the Facebook Data of Millions,” The New York Times, 17 March 2018. [Online]. Available: https://www.nytimes.com/2018/03/17/us/politics/cambridge-analytica-trump-campaign.html. [Accessed 2018 April 6].

[105] C. Cadwalla, “Facebook suspends data firm hired by Vote Leave over alleged Cambridge Analytica ties,” The Guardian UK, 7 April 2018. [Online]. Available: https://www.theguardian.com/us-news/2018/apr/06/facebook-suspends-aggregate-iq-cambridge-analytica-vote-leave-brexit. [Accessed 7 April 2018].

[106] H. Kozlowska, “The Cambridge Analytica scandal affected nearly 40 million more people than we thought,” Quartz, 4 April 2018. [Online]. Available: https://qz.com/1245049/the-cambridge-analytica-scandal-affected-87-million-people-facebook-says/. [Accessed 7 April 2018].

[107] I. S. Punit, “335 Indians installed a Cambridge Analytica app, exposing the Facebook data of 560,000,” Quartz India, 5 April 2018. [Online]. Available: https://qz.com/1245515/facebook-admits-cambridge-analytica-may-have-accessed-the-data-of-over-560000-users-in-india/. [Accessed 5 April 2018].

[108] Engineers Ireland, *Code of Ethics*, Dublin, 2018.

[109] R. Goldsborough, “Understanding the Common Battery,” *Teacher Librarian; Bowie*, vol. 42, no. 3, pp. 69-71, 2015.

[110] Health Service Executive (HSE), “NIMIS ‘<’ SYMBOL INCIDENT,” HSE, Dublin, 2018.

[111] N. G. Leveson, “The Therac-25: 30 Years Later,” *Computer*, vol. 50, no. 11, pp. 8-11, 2017.

[112] The World Medical Association, Inc., “WMA DECLARATION OF HELSINKI – ETHICAL PRINCIPLES FOR MEDICAL RESEARCH INVOLVING HUMAN SUBJECTS,” World Medical Association (WMA), October 2008. [Online]. Available: https://www.wma.net/policies-post/wma-declaration-of-helsinki-ethical-principles-for-medical-research-involving-human-subjects/. [Accessed 6 April 2018].

[113] L. Alexander and A. Ivy, “THE NUREMBERG CODE,” [Online]. Available: https://history.nih.gov/research/downloads/nuremberg.pdf.

[114] G. J. Annas and M. A. Grodin, The Nazi Doctors and the Nuremburg Code, Oxford: Oxford University Press, 1992.

[115] American Health Information Management Association (AHIMA), “Information Governance - Principles for Healthcare (IGPHC),” AHIMA, Chicago, 2014.

[116] P. Taylor, “Whose Property?,” *London Review of Books,* vol. 30, no. 3, pp. 25-26, 2018.

[117] The Irish Data Protection Commissioner, “DATA PROTECTION ACT,” Dublin, 1988.

[118] The Irish Data Protection Commissioner, “DATA PROTECTION (AMENDMENT) ACT,” Dublin, 2003.

[119] Irish Data Protection Commissioner, “EUROPEAN COMMUNITIES (ELECTRONIC COMMUNICATIONS NETWORKS AND SERVICES) (PRIVACY AND ELECTRONIC COMMUNICATIONS) REGULATIONS,” Dublin, 2011.

[120] The Irish Data Protection Commissioner, “General Data Protection Regulation (GDPR),” Dublin, 2018.

[121] Android Developers, “Accessing data using Room DAOs,” Google, [Online]. Available: https://developer.android.com/training/data-storage/room/accessing-data.html. [Accessed 7 April 2018].

[122] J. Corleone, “Meal Plans for 6 Meals a Day,” Livestrong, 3 October 2017. [Online]. Available: https://www.livestrong.com/article/206656-meal-plans-for-6-meals-a-day/. [Accessed 7 April 2018].

[123] A. Paturel, “6 Meals a Day for Weight Loss,” WebMD, 28 June 2014. [Online]. Available: https://www.webmd.com/diet/obesity/features/6_meals_a_day. [Accessed 7 April 2018].

[124] H. S. Dashti and K. M. Mogensen, “Recommending Small, Frequent Meals in the Clinical Care of Adults: A Review of the Evidence and Important Considerations,” *Nutrition in Clinical Practice,* vol. 32, no. 3, 2016.