Identification and Classification in Food Images

UNIVERSITY *of* LIMERICK

OLLSCOIL LUIMNIGH

Department of CSIS

Final Year Project - 2018

Name: Tom Barrett

ID: 14171198

Bachelor of Science in Computer Systems (LM051)

Supervised by: J. J. Collins

Abstract

Health conditions in the modern age are progressively getting worse with a high percentage of the population being obese. In order to combat this problem, a dietary assessment smartphone application would be invaluable. The task of recording one's food intake could be quite tedious and therefore, utilising computer vision to automatically classify and calculate nutritional value of a user's food image could help to make the process more practical for use. One approach that could be attempted, is by using Convolutional Neural Networks (CNNs) for the classification of food images. This is due to the recent high success in using CNNs for image classification. Retraining of the Inception-V3 model architecture using TensorFlow and the Food-101+ dataset was carried out, resulting in a classification model. A Top 1 accuracy of 66.6% and a Top 5 accuracy of 85.96% was achieved using a dataset of 108 classes. It was found that CNNs are very successful in classifying images with over 100 classes and that network fine-tuning could result in even better results.

Acknowledgements

Firstly, I would like to thank my supervisor Mr. J.J. Collins for all the support and guidance over the course of this final year project. I came to J.J. with the plan to centre my FYP around image recognition using convolutional neural networks. J.J. provided me with an aim to recognise food images which was both a challenging and relevant topic. Throughout the course of my FYP, J.J. was available to talk through any issues or queries in a helpful manner and always made time for me. This project would not have been possible without him.

Secondly, I would like to thank my course director Dr. Norah Power. Norah has always been open to giving me guidance and support during my time in UL, of which I am very grateful. I would also thank Dr. Patrick Healy who succeeded Norah in my final year as course director.

Many thanks are due to my family and friends who supported me during my time in college. It would not have been possible to complete my degree without my parents Karen and Derek, my sister Kate and my girlfriend Lauren.

Special thanks to Miky, Robbie and PJ for all the projects that we worked on together and the good times that came with them.

Finally, I would to thank all the CSIS staff, especially Dr. Jim Buckley our FYP coordinator, for their support over my four years in UL.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter outlines an introduction to the subject area of this Final year Project (FYP), the objectives of this project, the methodology used to complete this project, an overview of the report and an explanation of the motivation behind this FYP. Also included is a project plan and contribution section.

1.1 Overview

This project explores identification and classification of food images for use in a nutritional assessment android application. Food calorie consumption is a huge problem in the modern world. Over 25% of the population in Ireland is obese and this figure is likely to rise over the coming years. There has been extensive research carried out into nutritional assessment. Many of these studies utilise smart phone applications to record calorie intake while physical food diaries have also been used in the past. A study was carried out by (Goodman et al., 2015) whereby they used a mobile application to keep track of Vitamin D intake for young Canadians ranging from teenagers to early adulthood. (Arens-Volland, Spassova, and Bohn, 2015) also carried out a review of current computer supported nutritional assessment applications. These applications ranged from web-based applications to smart phones. The problem with many of these 'food diary' applications is that they can be very tedious and time

intensive for users. Leveraging artificial intelligence to decrease the effort of the user or dietician would be very beneficial and many even encourage use of said nutritional assessment applications. Artificial intelligence could be utilised to classify food images and calculate the calorie count of an image so that a user would not have to input the information themselves.

The classification of images is the process whereby an algorithm predicts what objects are in an image. As humans, we take vision for granted. Computers (which find what are complex task for us, seemingly simple i.e. mathematical computations) find it very difficult to interpret images. A grayscale image is simply an array of numbers with values of ranging from 0 to 255 as seen in Figure 1.1. For colour images, there are multiple layers of arrays for each of three colour channels red, green and blue. It can be quite difficult to extract feature information from these arrays.

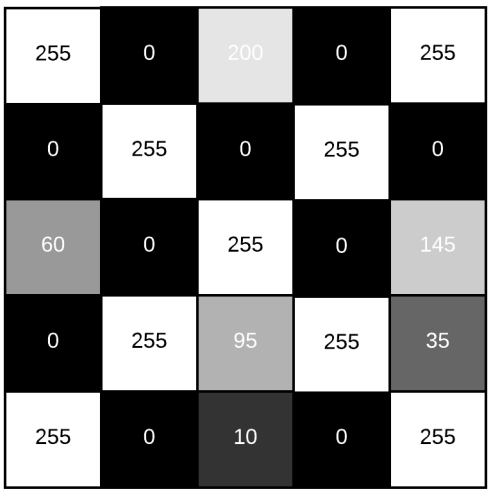


Figure 1.1: Representation of an Image as an Array

Even when classification can be achieved by analysing these arrays of numbers, it is most successful when the object in question takes up most of the image. For example, the banana is clearly a very prominent object in Figure 1.2.

Alternatively, when the object of the image is occluded or not the focus of the image (Figure 1.3 and Figure 1.4), classification becomes very difficult. In contrast it can be clearly seen to humans that a banana is still present in

Figure 1.2: Banana

both images. If we take a filled roll for example, it is difficult for the human eye to see the contents never mind a computer so this is another issue that arises. Illumination is also a factor that influences classification greatly as colour can play an important aspect in classification of food images as discovered by (Pouladzadeh, Villalobos, et al., 2012). Other factors of illumination that affect classification are shading, shadows and specular highlights (Gong, McKenna, and Psarrou, 2000).

Figure 1.3: Occluded Banana Image

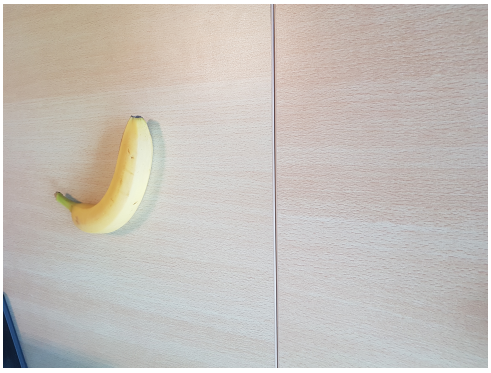


Figure 1.4: Banana at Small Scale

Traditionally, Support Vector Machines (SVMs) have been used to classify

food images, many examples of which are covered in Chapter 2. The aim behind SVMs is to cast the information to a higher dimension, for example from a 2D to 3D space, so that the classes can be separated. SVMs will be explored further in Chapter 2. The features used by SVMs for food image classification are normally colour, texture, shape and size as in (Pouladzadeh, Shirmohammadi, and Al-Maghrabi, 2014) and (Pouladzadeh, Villalobos, et al., 2012). SVMs have their problems however, and these problems of long training times and low accuracy with many classes have resulted in SVMs losing popularity in recent years.

Another reason for the shift from the use of SVMs is due to the recent success of utilising neural networks for image classification. Neural networks are bio-inspired algorithms based off the structure of the human brain. A special type of neural network, a Convolutional Neural Network (CNN), has proven to be very successful in image classification (Krizhevsky, Sutskever, and G. E. Hinton, 2012). CNNs differ from other neural networks due to their extra layers of convolution and pooling (Zeiler and Fergus, 2014). Convolutional layers filter the images for features while pooling refines the information in the arrays to reduce computational time and to make the features in the image clearer. CNN's have been around since the 1980's (Aurélien, 2017) but were not used by many until 2012 when CNNs became popular again due to the work carried out in that time, for example by (Krizhevsky, Sutskever, and G. E. Hinton, 2012). CNNs have been proven to be very successful in facial recognition and have been applied to food images with promising results as outlined in various studies in Chapter 2 such as (Keiji Yanai and Kawano, 2015) and (Mao, Yu, and Wang, n.d.).

For a nutritional assessment application aided by computer vision, there are a few steps that must completed. Firstly, the food images are classified with prior segmentation in some cases. Image segmentation is the process by which an image is segmented into multiple parts with usually a segment for each different object or as in the case of Figure 1.5 each different food item. Secondly, size estimation of each food type is carried out. Once data on the food type and size has been collected, calorie count estimation can be undertaken. Size and calorie estimation is outside the scope of this FYP.

Figure 1.5: Segmented Food Image (*Meanshift Algorithm for the Rest of Us (Python)* 2016)

The full system proposed to solve the problem statement would be able to integrate with an Android mobile phone application. The concept is, that when a user is about to eat their meal, they can simply take a picture of their meal for computation. The application would take an image and send it to a server to be analysed. On the server side the image would be classified using a TensorFlow model. TensorFlow is a machine learning library that can be used to create CNNs. Once the classification step is completed, the size of each food type would be measured and through this, an overall calorie count would be displayed for the user. This could be logged for user metrics. The full system may not be possible to implement due to time constraints so therefore, classification will be the furthest step looked in to.

1.2 Objectives

1.2.1 Primary Objectives

1.2.1.1 Use Convolutional Neural Networks for Food Image Classification

There has been a large paradigm shift in food image recognition in recent years to using convolutional neural networks. This paradigm will be used to answer the problem statement of this FYP.

1.2.1.2 Tune the CNN, replicating previous work

There are many different approaches to food identification and classification, many of which we will see in Chapter 2. An approach will be selected that has shown promising results in the past and replication of these results will be attempted. In addition to this, there are many different network architectures and parameters that can be adjusted when building CNNs and these will be tuned to get the best outcome possible.

1.2.1.3 Develop a mobile application to leverage the CNN

Another objective for this project is to develop an application that can be used for dietary assessment. This application would be able to take an image and then send the image to a server to identify and classify the foods within the image. Size estimation will not be explored for this project. Alternatively, due to time constraints, a previously developed application could be used to demonstrate the CNN created.

1.2.2 Secondary Objectives

1.2.2.1 Understanding of Convolutional Neural Networks

In this project, CNNs will be used for classification in food images. A machine learning library will be used to develop a CNN due to time constraints, but it is a key objective to develop a deep understanding of CNN's as they are quite pivotal in the current computer vision industry and bio-inspired systems are very interesting.

1.2.2.2 Learn about different image identification and classification techniques

Although CNN's will be used for implementation, other methods of identification and classification will not be ignored. It is very important to learn about other methods as different methods are better suited for some situations and

it would be best to know about these methods due to the inevitability of their use.

1.3 Contribution

While many researchers have explored the topic of food image classification, there has not been many comprehensive surveys of the work done. In Chapter 2 of this report, over 20 research papers have been summarised and critiqued. This survey of the literature could be very useful for future researchers.

1.4 Methodology

The following methodology was adapted for this project:

1. Define the research question: The first step to this project was to define the research question. The general area of a computer vision supported nutritional assessment application was known at conception but the scope of this is too broad for an FYP. Therefore, it was decided that the research question would be to look at the food identification and classification aspect.

2. Literature review: Once the research question was defined, finding related work was the next milestone. There are many attempts at food image classification and these were not difficult to find using Google Scholar, but many of these papers glossed over the segmentation aspect and relied on third parties for this step. Because of this, quite a few references to different papers had to be followed that focused solely on image segmentation.

3. Explore different image identification methods: Various image identification methods were collected from the literature review that was carried out, so there were many options to evaluate. Convolutional Neural Networks were the clear choice due to recent popularity and successful

results, so more traditional methods of identification using colour and texture was not so strenuously explored.

4. Select an image identification and classification method: After the various methods to identify and classify food images had been collated in the literature review, one of these had to be selected.

5. Research technologies and develop skills in these technologies: As CNNs are used in this project, many resources have been leveraged to enrich understanding of the process. TensorFlow is the main resource utilised in creating a CNN and on-line tutorials for this technology were greatly beneficial. A deep learning course on Udacity was also used to enhance understanding and skills.

6. Build a prototype of the application: A prototype application had to be built for demonstration for this project.

7. Compare and analyse results to other implementations: Once a model had been successfully trained, comparison to previous work was carried out.

1.5 Project Plan

To plan for this project, a WBS (work breakdown structure) and a Gantt chart were created. A WBS is used to get a list of the tasks required to complete a project. A Gantt chart maps these tasks to dates and manages dependencies between the tasks i.e. does one task have to finish before another starts. Refer to Figure 1.6 for the WBS and Figure 1.7 for the Gantt chart.

1.6 Overview of Report

This report is broken down into various chapters. The introduction chapter is to give on overview of what this project is about, how the project will be approached and why it is being carried out. Following this, some information on the background of the subject of this FYP will be outlined. Once the

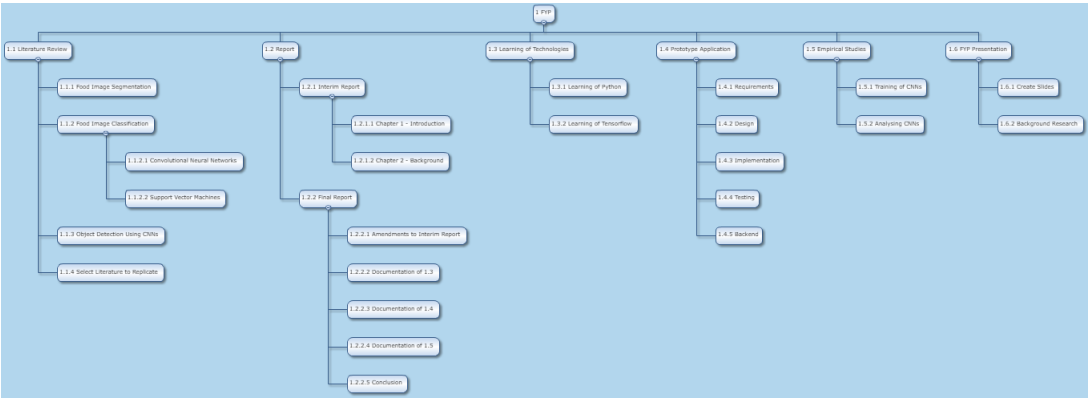


Figure 1.6: Work Breakdown Structure

chapters of introduction and background have been completed, a chapter outlining the learning of TensorFlow will be explored. TensorFlow is a machine learning library that is used to develop CNNs. In this chapter, the goal is to demonstrate the activities undertaken to learn about the technologies used and show tutorials that have been completed to aid with this learning. In the chapter titled 'Empirical Studies Part 1', the purpose is to outline experiments that have been carried out in relation to training TensorFlow models. A TensorFlow model is the term used to describe the neural network produced by TensorFlow. The Inception-V3 architecture is proven to be a very successful architecture for classifying images and will be explained in detail in said chapter. Once the model has been trained, the following chapter 'Empirical Studies Part 2' will consist of experiments that analyse and test the models trained. A prototype smart phone application was developed to demonstrate the efficacy of the system and an overview of the process will be explained in the chapter titled 'Prototype Application.' The final chapter's purpose is to provide a discussion on the results obtained and offer a conclusion to the findings.

1.7 Motivation

I find the topic of computer vision a very interesting one. It excites me, to be able to 'teach' a machine how to see as we do. For this reason, I really wanted to learn about neural networks and this was a large motivator for this project.

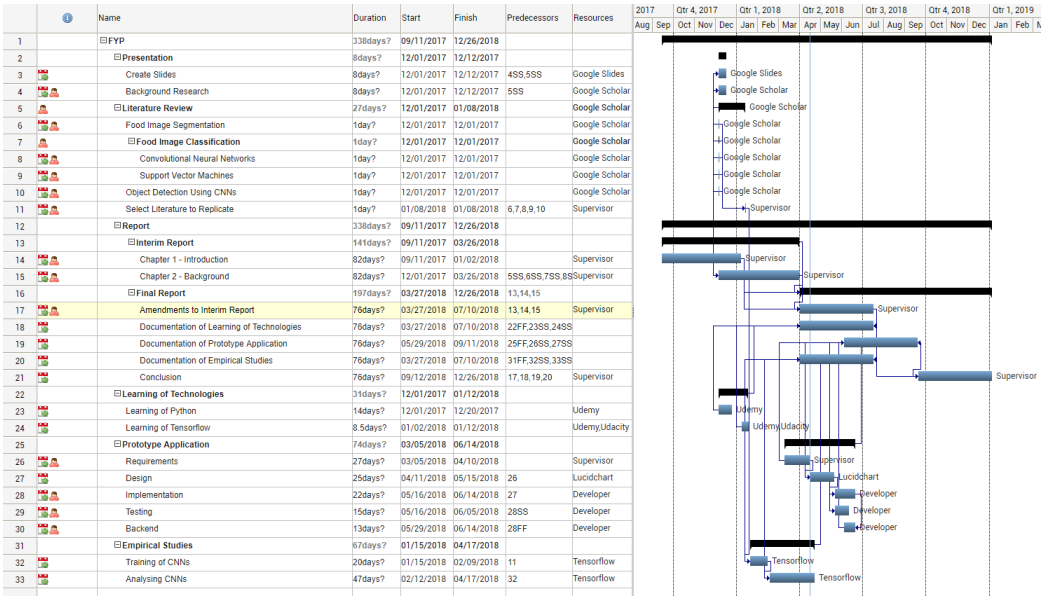


Figure 1.7: Gantt Chart

Once I had a topic that I wanted to research, I needed a focus or problem statement for this research. I find that it is much more rewarding to work on something that positively impacts both myself and other people, so I decided that I wanted to research something that fit this requirement.

Food calorie consumption is a very big problem in the modern world. Over 25% of the population in Ireland are obese. A mobile application that could help keep track of a user's calorie intake by taking a picture of their meals would be helpful in combatting this problem. This problem statement works very well for me because of its application use and because of its complexity. Identifying and recognising food is much more difficult than say recognising faces as it has no uniform shape. Therefore, this problem would also be very beneficial to developing skills in the computer vision area.

I would like to develop these real-world skills so that I can partake in computer vision projects in industry or to do further research in academia. I have also been offered a graduate role in Jaguar Land Rover and these skills would be very relevant in the autonomous driving industry. This is because machine learning has really taken off in the last few years and is used by many in industry. While machine learning has become very popular, computer vision is probably the most prominent that has come to fore. Face detection, for one,

is being researched extensively for use in personalised advertising and secure access to devices and systems.

Chapter 2

Background

This chapter outlines background information about nutritional assessment using computer vision. An overview of neural networks is explored, followed by a literature review of the subject area. Analysis into evaluating the efficacy of CNNs is covered and finally a conclusion to the information gained.

2.1 Introduction to Machine Learning

In (Mitchell, 1997a), machine learning is defined as "the question of how to construct computer programs that automatically improve with experience". Machine learning has blossomed in recent years with applications across multiple domains using vastly different paradigms and technologies. Some of the different approaches used in machine learning are: Artificial Neural Networks, Genetic Algorithms, Decision Tree Learning and Bayesian Learning (Mitchell, 1997a).

There are many ways in which machine learning can be used in the modern world, many of which are being utilised to great effect. Some of these applications are image recognition, natural language processing and many more. These applications can be applied to many different domains such as security (face detection in airports) or object detection (autonomous driving). (Erickson et al., 2017) carried out research into the area of using machine learning to aid diagnosis of medical conditions. Machine learning algorithms could

suggest possible diagnosis for medical professionals to interpret and therefore reduce the time spent diagnosing a patient's condition. There may be fear that machine learning will start to take away many jobs from humans, but this may not be the case as in the example above as computers will only be aiding professionals.

One of the most exciting avenues in machine learning is computer vision. This is due to the application domains mentioned above. Computer vision is the process of extracting high-dimensional data from an image to produce useful information, which in terms of classification usually results in labelling. It can be used in many areas to improve our lives. As mentioned earlier, autonomous cars are only possible when a machine can determine what objects are around it. Computer vision can allow a machine to recognise medical conditions in an image such as breast cancer using mammography images (Erickson et al., 2017). The applications are nearly limitless.

2.2 Neural Computing

The main area of my focus for this project is in Artificial Neural Networks (ANN). This is because extensive research has been carried out into CNNs which are based on ANNs.

2.2.1 Artificial Neural Networks

An ANN is a bio-inspired system that is used to model the human brain in how it learns from experience. ANNs were first introduced in 1943 and were studied until the 1960's when the concept was shelved and revived again in the 1980's (Aurélien, 2017). The ANN uses this model to build a very complex web of connected units called artificial neurons. These neurons are connected by certain weights which determine the processing capacity of the network and these weights are created by learning a dataset (Eaton, 2017). A neuron has a set of inputs that take in a value, sometimes from network outputs and produce a single result or classification.

As stated above, an ANN is bio-inspired from biological neurons (Aurélien, 2017). Biological neurons are cells located in animal's brains. These neurons receive signals (electrical impulses) from other neurons and fire their own signal thereafter. While an individual neuron is very simple, millions of these neurons working together can result in very complex computations. From this, an artificial neuron was proposed through a simple model which had a binary input (1 or more) and a single binary output.

Before CNNs can be explored, which are vital to image processing, the perceptron learning algorithm, the multi-layer perceptron and backpropagation must be analysed.

2.2.1.1 Perceptron Learning - Artificial Neuron

In our ANN, a perceptron is an artificial neuron. It is called an artificial neuron because it is a bio-inspired neuron which models a neuron in the human brain in terms of inputs and output.

In perceptron learning, we can take two inputs which are put towards an activation function with a bias attached as seen in Figure 2.1. These inputs are multiplied by the weights that connect the input to the activation function and depending on the result, the activation function may fire an output. These inputs are either 1 or -1.

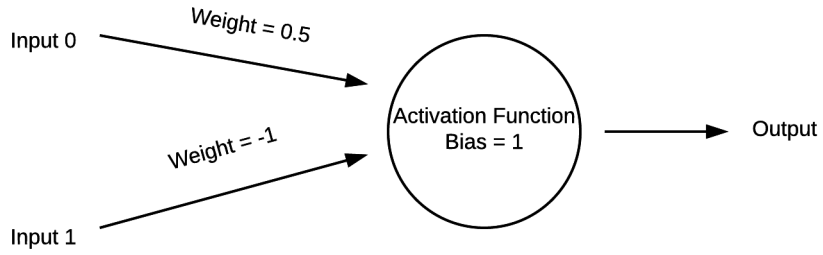


Figure 2.1: Perceptron

The Perceptron Training Rule is how weights are selected to produce the correct output during training. As in (Mitchell, 1997a), a common way to train a

perceptron is to start with random weights and change them during training as per the training rule. This rule follows the formula in Equation 2.1, where xi is the input and Equation 2.2 is valid:

$$w_i \leftarrow w_i + \Delta w_i \tag{2.1}$$

$$\Delta w_i = n(t - o)x_i \tag{2.2}$$

In Equation 2.2, "t is the target output for the current training example, o is the output generated by the perceptron, and n is the positive constant called the learning rate" (Mitchell, 1997a). The output of a neuron is calculated using the activation function. This Perceptron Training Rule assumes that there are two sets of instances, a positive and negative set (class x and - in Figure 2.2), and that they are linearly separable, as in demonstrated in Figure 2.2.

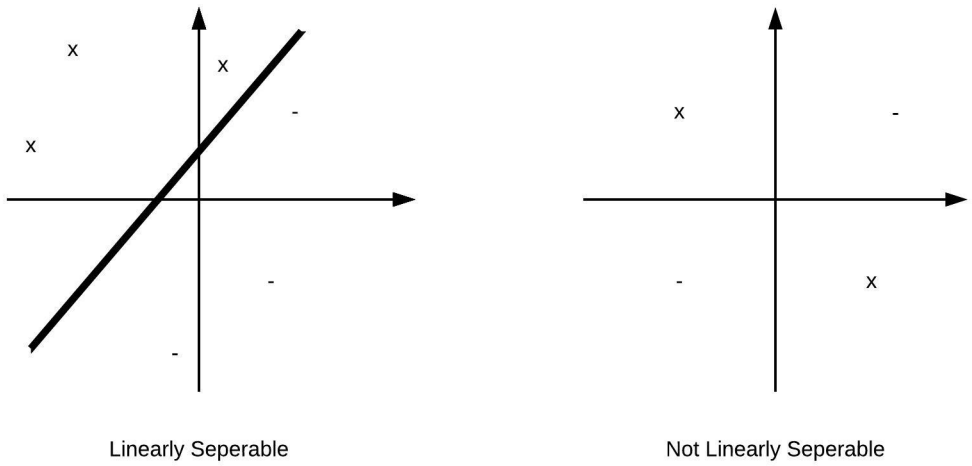


Figure 2.2: Linearly Separable, adapted from (Mitchell, 1997a)

A perceptron is trained using supervised learning. When the perceptron classifies a result, it is told if it is correct or not. If the result is incorrect, weights are changed in value so that this error can be reduced (Luger, 2005).

There is one major problem with perceptron learning and that is, it can't solve a problem if there is not a clear linear separation between the classes. There is

a way in which we can attempt to solve this, through the delta rule. The delta rule utilises gradient descent to find the best weight for the training samples (Mitchell, 1997a). We will discuss gradient descent in the next section.

2.2.1.2 Multi Layered Perceptron

Multi-Layer Perceptrons (MLPs) are made up of multiple layers of perceptrons connected together and are used to combat non-linearly separable classes. While the delta rule can solve problems of non-linearity when there are two classes, MLPs can solve non-linearity when there are more than two classes. Firstly, we have an input layer, followed by one or more hidden layers and then finally an output layer. Any ANN with more than three hidden layers is categorised as a deep neural network.

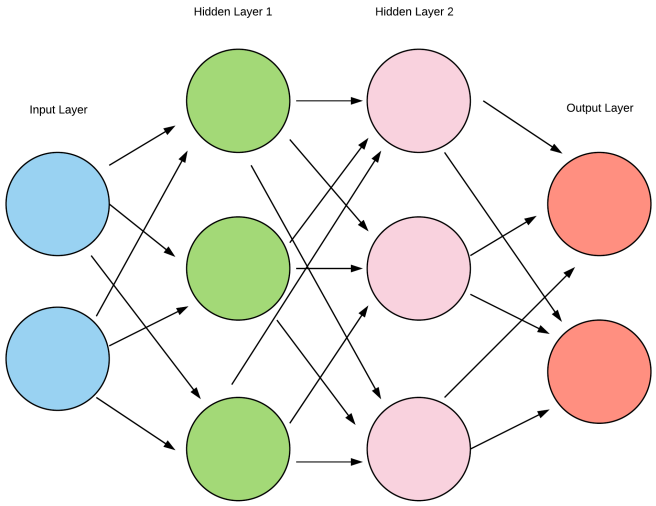


Figure 2.3: Multi Layer Perceptron

The input layer of a network consists of the data that is fed into the network to be classified. The input layer passes this data to a hidden layer whose purpose is to transform this data into something that the output layer can understand. This transformation results in a linearly separable space that can be classified. The output layer normally consists of a class prediction.

MLPs are a class of feed forward ANNs. These means that the output of each perceptron feeds into an input in the next layer of the network, example seen in Figure 2.3.

During training, using backpropagation for each step, the output of every neuron is calculated in each layer and then passed to the next layer. Then the error of the output is calculated, and the network calculates how each neuron in the last hidden layer effected the error. This is continued back through all the layers until the input layer (Aurélien, 2017). The weights are then altered to try and reduce this error.

There is one large problem with MLP's and as a result CNNs were created. If one is attempting to classify images with a MLP then each pixel in that image would have to be a separate input. This creates a massive number of neurons through all the layers and this isn't feasible. CNN's solve this problem which we will discuss later.

2.2.1.3 Gradient Descent and backpropagation

Gradient descent is an algorithm used to find the optimal weights to produce the smallest prediction error. It is used to overcome problems of non-linearly separable classes. Gradient descent search selects a random weight value and then modifies it gradually to minimize the error. "At each step, the weight vector is altered in the direction that produces the steepest descent along the surface" (Mitchell, 1997a). This step is iterated until the lowest value is met.

There is an error function used for the perceptron which finds the lowest error for that neuron, but it can't be used here because, since we have many neurons, there could be an error in multiple neurons. Gradient descent is mathematically based on the derivative of a function. The gradient of a function can be calculated by differentiating it. As the weights are what is being controlled,

"they are what we differentiate in respect to" (Marsland, 2015). The negative gradient of this function is followed to find the lowest possible point, hence the name gradient descent (Marsland, 2015).

One problem with gradient descent is that if we look at Figure 2.4, we may never get to the optimal point, point B. This is because we will find point A without too many problems but when the weights change we will get too high a slope of error and therefore will never reach point B.

Another variation of gradient descent is Stochastic Gradient Descent (SGD). SGD is different because it updates "weights incrementally, following the calculation of the error of each individual example" (Mitchell, 1997a).

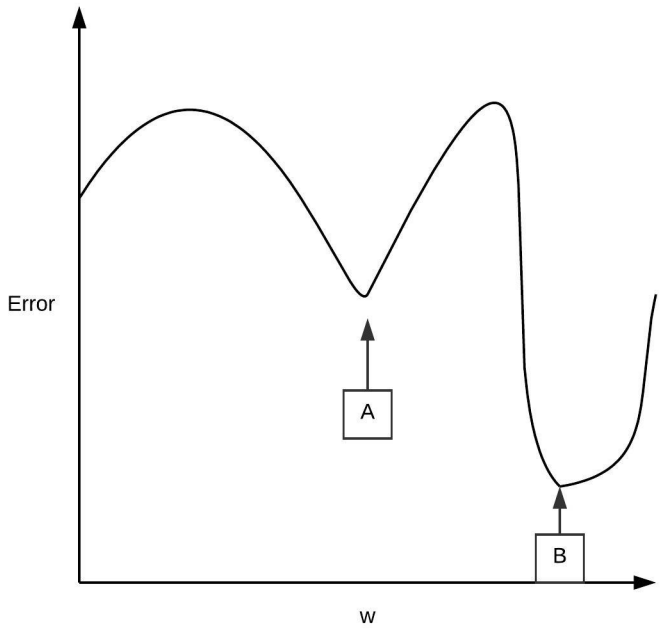


Figure 2.4: Gradient Descent

"The Backpropagation algorithm learns the weights of a multilayer network, given a network with a fixed set of units and interconnections" (Mitchell, 1997a). Backpropagation attempts to minimize the mean squared error between the target output and the actual output of a network.

Backpropagation works by starting at the output layer of the network and

going back through previous hidden layers, updating weights as it goes i.e. it propagates back through the network, updating the weights to try and reduce the error.

(Mitchell, 1997a) defined a walkthrough of the backpropagation algorithm. For every value of Equation 2.3, in the training set where x is a vector of inputs and t is a vector of output values to act as a target:

- Run x through the network and output Equation 2.4.
- For each output k, calculate the error by Equation 2.5.
- For every hidden unit, calculate the error by Equation 2.6.
- Update weights by Equation 2.7 where Equation 2.7 is true.

$$\vec{x}, \vec{t} \tag{2.3}$$

$$o_u \tag{2.4}$$

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k) \tag{2.5}$$

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k \tag{2.6}$$

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji} \tag{2.7}$$

$$\Delta w_{ji} = \delta_j x_{ji} \tag{2.8}$$

2.3 Convolutional Neural Networks Overview

CNNs are essentially a MLP with a special structure (Zeiler and Fergus, 2014). CNNs have one major difference from a MLP, they have extra layers of convolution and pooling. The architecture of a convolution network can be seen in Figure 2.5.

Each convolutional layer in the network extracts features from the images. The closer the image is to the input layer, more primitive features are extracted

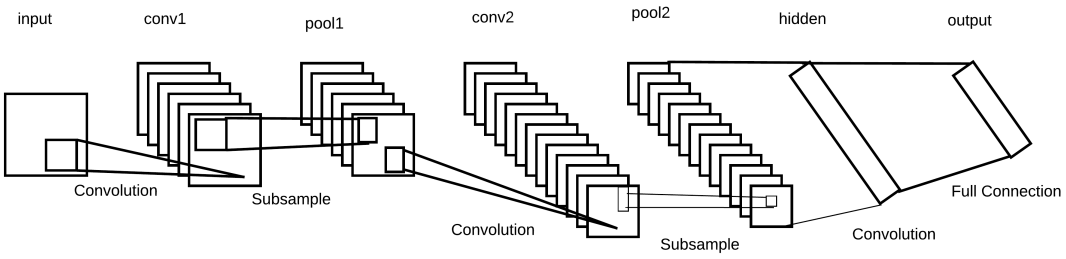


Figure 2.5: CNN Architecture

while the closer the image is to the output layer, more complex features are extracted.

Figure 2.6 (a) shows an image that we want to compare against Figure 2.6 (b). For humans, it is quite easy to determine that these images are very similar but for a computer this task is surprisingly difficult.

So, what a CNN does to combat this problem, is to take a small feature from Figure 2.6 (a) and compare it to a subsection of Figure 2.6 (b). The CNN multiplies the feature and a section of Figure 2.6 (c), adds up the results and divides by 9 (The number of pixels in the feature). This then gives a decimal value of how likely it is that the feature is in the part of the image, as seen in Figure 2.6 (d). This is called filtering. The convolutional layer is composed of carrying out this filtering for every single possible location in Figure 2.6 (a).

Next is the Pooling Layer that takes the convoluted layer output (you can use Figure 2.6 (d) as reference) and from a user defined size i.e. 2x2, gets either the highest decimal value (max pooling), the average value (mean pooling) or the lowest decimal value (min pooling) and records that as the new value for the section. This is then applied to the entire image. As we can see in Figure 2.6 (e) we now have a much smaller image stack in which to classify, thus making the computation easier. Pooling also makes features of the image clearer which helps to result in more accurate classification.

In between the convolution and pooling layer, there is sometimes a normalisation layer. This normalisation layer creates Rectified Linear Units (RLU's). In other words, if we take Figure 2.6 (d), it changes all minus values to zero.

There are some problems with CNN's however. One of the main problems is

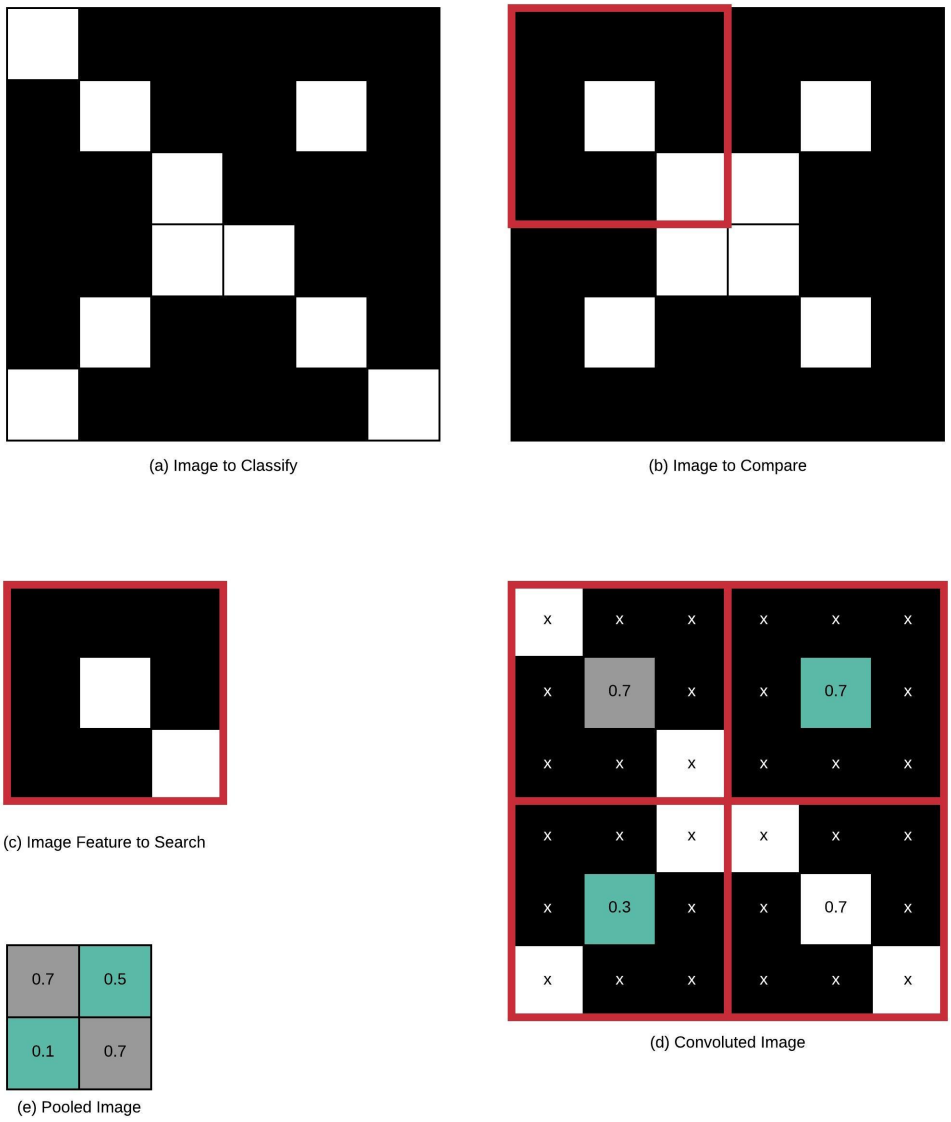


Figure 2.6: CNN Overview

that you need a very large dataset in order to produce an accurate model and the training can be very time consuming even with a GPU. This is because each training image (or batch) must be run through the network and go through backpropagation.

2.3.1 Receptive Field and Kernel

In the previous section where an overview of CNNs was addressed, there was a discussion focused on filtering where features were searched for across the images. This filtering resulted in windows sliding across the image, looking for common features. This window is called the receptive field, and this is the section of the image that is being looked at. As the image is convoluted and processed through the network, the changed image can be called the kernel. The kernel of the CNN represents what features are extracted from the image throughout the network.

2.3.2 Activation and Loss Function

The activation function used in a neural network calculates the output for a neuron. The most commonly used activation function in CNNs is the ReLu activation function or the rectified linear unit (Aurélien, 2017). The SoftMax activation function is sometimes used for the output layer.

The loss function in a neural network simply calculates the magnitude the of the error. It represents how different the result obtained and the result expected are.

2.3.3 Dropout

Dropout is a type of regularisation technique used to prevent overfitting in a network (Aurélien, 2017). Overfitting in a network is when the network cannot generalise to new data after over learning the training data. In dropout, each neuron is given a probability (normally set to 0.5) and this is the likelihood that the neuron will be used for that training step. It has proven quite successful for increasing a networks accuracy (Aurélien, 2017).

2.3.4 Fully Connected Layer

Once the layers of convolutional pooling have detected features in an image, a fully connected layer is added to the end of the CNN (Deshpande, 2018). This layer uses the output of the previous layer to determine a prediction for the network. The fully connected layer outputs a vector of n dimensions where n is the number of possible classes to predict. A probability is calculated for each class to determine how likely it is that the class is present in the image. This layer calculates the probabilities by looking for features in its input that relate to the classes. For example, if the image was of a banana, the fully connected layer would look for features such as curved shape or yellow colour to calculate a high probability.

2.3.5 Different Sizes of Convolutions

Zero padding can be added to an output so that it is the same size as the previous layer by adding zeros around the feature (Aurélien, 2017). Through training, it is "possible to connect a large input layer to a much smaller layer by spacing out the receptive fields" (Aurélien, 2017). The stride between two different receptive fields is the spacing between them (Aurélien, 2017).

2.3.6 Fully Convolutional Networks

A fully convolutional network is one that does not have a fully connected layer and in a fully connected layers place is another convolution layer. This can be achieved by making the size of the receptive field equal to the size of the input (*Image Segmentation Using DIGITS 5* 2018). They are proven to be very successful in object detection.

2.3.7 Inception-V3 Model Architecture

The Inception V3 model network architecture was used for this experiment. The Inception V3 architecture was created by building on the existing Inception model aimed at efficient image classification (Szegedy et al., 2016). This research was carried out due to the popularity of convolutional neural networks. The main aim of this study was to produce a model that would "scale up networks in ways that aim at utilizing the added computation as efficiently

as possible by suitable factorized convolutions and aggressive regularization" (Szegedy et al., 2016). The team did not want to simply rely on larger models for better results.

There are 4 main design principles that the research team followed in the paper (Szegedy et al., 2016):

- Avoid representational bottlenecks.
- It is easier to process higher dimensional representations locally.
- Without much loss in power, spatial aggregation can be carried out over lower dimensional embeddings.
- The width and depth of the network should be balanced and when one is increased it can be beneficial to increase the other.

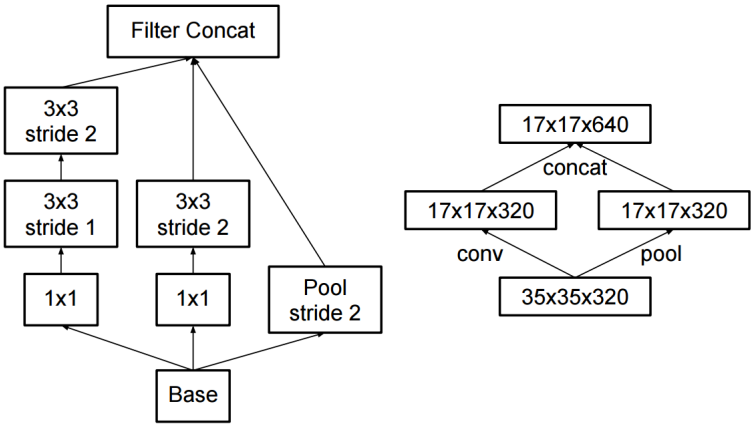


Figure 2.7: Inception module (Szegedy et al., 2016)

The main idea behind the network is that different strands of the network are followed, and the best result is used as seen in Figure 2.7.

In order to combat the problem of low resolution input, three separate experiments were carried out based on the fact that "One simple way to ensure constant effort is to reduce the strides of the first two layer in the case of lower resolution input, or by simply removing the first pooling layer of the network" (Szegedy et al., 2016).

These experiments are as follows, with the first (299 x 299 receptive field) resulting in the most accurate:

- "299 x 299 receptive field with stride 2 and maximum pooling after the first layer." (Szegedy et al., 2016)
- "151 x 151 receptive field with stride 1 and maximum pooling after the first layer." (Szegedy et al., 2016)
- "79 x 79 receptive field with stride 1 and without pooling after the first layer." (Szegedy et al., 2016)

The best results of the Inception-V3 model on the ILSVR 2012 dataset achieved 78.8% Top-1 accuracy and 94.4% Top-5 accuracy (Szegedy et al., 2016). This model also was the least computationally expensive of other published and successful models. The full model can be viewed in Figure 2.8.

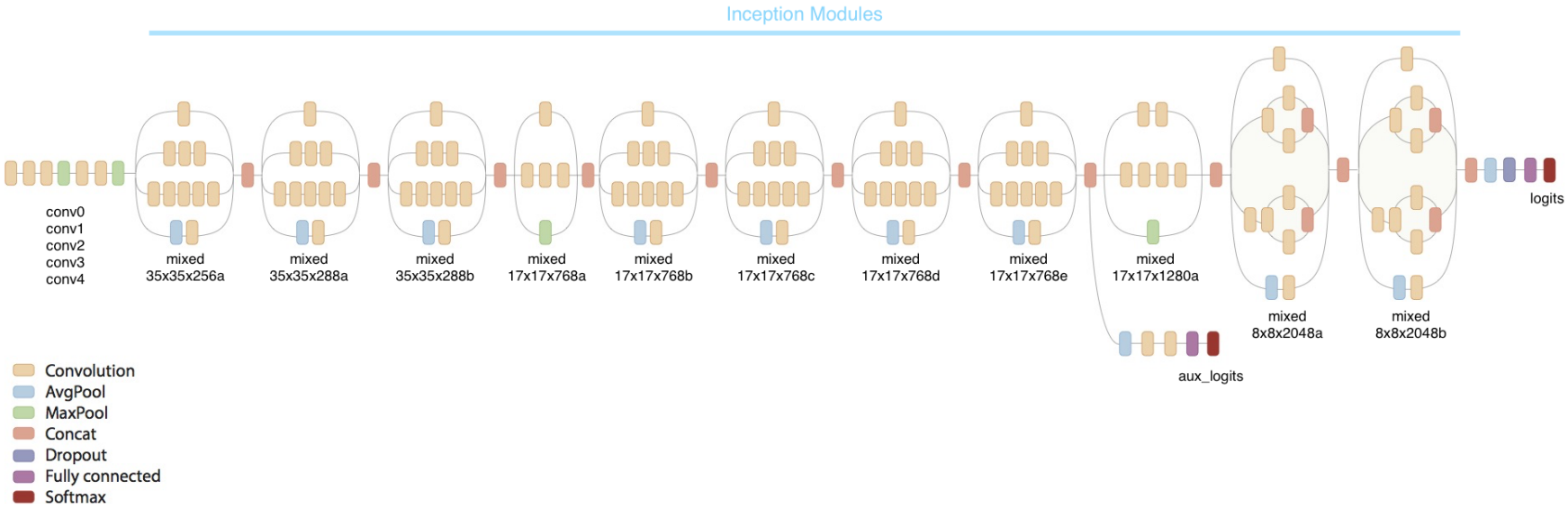


Figure 2.8: Inception V3 Architecture (Szegedy et al., 2016)

2.3.8 MobileNet Model Architecture

This network architecture is called MobileNet, created by (Howard et al., 2017). This architecture is designed to be smaller and much more efficient than others, so that it can be used on smartphones which have less powerful resources available.

MobileNet is designed for various use cases on mobile phones such as image classification, landmark recognition, object detection and face attributes. It is based on "depthwise separable convolutions" whose purpose is the split a convolutional layer into a depthwise convolution and a pointwise convolution (1x1 convolution) (Howard et al., 2017). The purpose of the former is to apply "a single filter to each input channel" (Howard et al., 2017) while the latter combines these outputs. This in turn reduces the computational time. The first layer of the network is a standard convolutional layer and does not get split up.

Unfortunately, there is a significant decrease in accuracy in using MobileNet as it is a much thinner model.

2.3.9 Libraries

There are many libraries available that can be used for the creation of CNNs such as:

- TensorFlow
- Caffe
- Gluon
- MxNet

In this project, TensorFlow will be used due to its reputation, supporting documentation, training resources and prevalence online.

2.3.10 History

CNNs are a bio-inspired approach to solving intensive tasks. They have been applied to image recognition, along with natural language processing and voice

recognition, with image recognition being explored since the 1980's (Aurélien, 2017). In 2012, CNNs increased in popularity due to increased computational power in the hardware, availability in datasets and breakthrough research carried out in the field (Krizhevsky, Sutskever, and G. E. Hinton, 2012). CNNs have been recently applied to many complex tasks with exceptional results such as facial recognition, and many more.

2.4 Support Vector Machine

A Support Vector Machine (SVM) is a machine learning algorithm that has been very popular before the use of a CNN was mainstream. Many of the texts that will be analysed later use SVMs for their classification.

A SVM works by creating an n-dimensional space, with n as the number of inputs you have (*Support Vector Machines for Machine Learning* 2017). The SVM algorithm finds the hyperplane that splits this space. This hyperplane can then be used for classification. An SVM casts the problem to a higher dimensional space and this can solve a problem when the classes are not linearly separable. This can be seen in Figure 2.9, where on the left there is no clear way to separate the classes but once the problem is cast to another dimensional, the separation is clear.

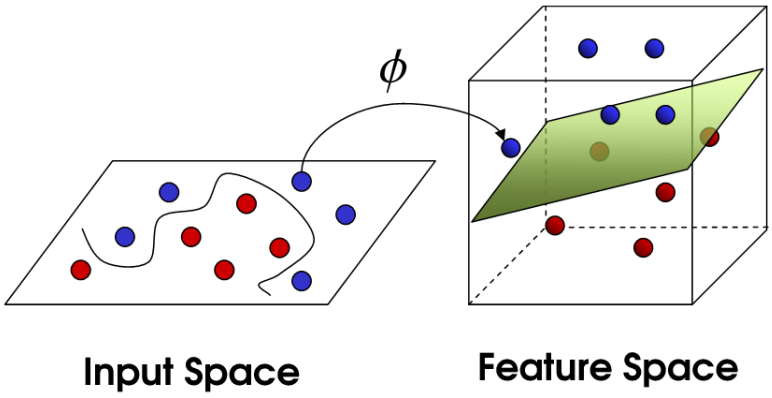


Figure 2.9: Support Vector Machine Applicability sourced from https://stackoverflow.com/questions/9480605/

A kernel is used to implement a SVM. There are a few types of kernels that

can be used in the SVM algorithm such as:

- Linear Kernel
- Polynomial Kernel
- Radial Kernel

The kernel is a vector of how similar two vectors are. In terms of a linear kernel, the dot product of two vectors is the kernel.

There are benefits and liabilities to using a SVM. They can be very accurate and can work efficiently with small datasets but unfortunately it can take a large amount of time to train.

2.5 Evaluating the Output

There are various metrics that can be used for evaluating the output of a classifier or segmentation algorithm, many of which we have seen in previous sections. These metrics will be examined below.

There has been some research into the question of how to evaluate object detectors, one of which will be discussed in detail (Hoiem, Chodpathumwan, and Dai, 2012). This paper in question "analyzes the influences of object characteristics on detection performance and the frequency and impact of different types of false positives" (Hoiem, Chodpathumwan, and Dai, 2012). They found that there were many effects that had influence on detectors as follows:

- occlusion
- size
- aspect ratio
- visibility of parts
- viewpoint
- localisation error

- confusion with semantically similar objects
- confusion with other labelled objects
- confusion with background

The research team goes on to analyse false positives in object detectors. Localisation errors were a large factor. This is where bounding boxes overlap to other objects in the image. Confusion with similar objects had a large influence on false positives also by which, for example, a dog detector had a high score for a cat (Hoiem, Chodpathumwan, and Dai, 2012). Confusion with dissimilar objects and confusion with background are the categories of the rest of the false positives they measured.

In conclusion the team found that "Most false positives are due to misaligned detection windows or confusion with similar objects" (Hoiem, Chodpathumwan, and Dai, 2012). They had some recommendations towards improving detectors as follows:

- Smaller objects are less likely to be detected
- Localisation could be improved
- Reduce confusion with similar categories
- Robustness to object variation
- More detailed analysis

2.5.1 Top-1 and Top-5 Accuracy

When a classifier is given an image, it normally responds with a list of predictions along with a decimal representation of the likelihood that it is of that class. The Top-1 accuracy is the top valued prediction and Top-5 accuracy is the top 5 predictions.

Table 2.1: Confusion Matrix

Actual (n=200)	**Predicted: NO**	**Predicted: YES**
NO	55	18
YES	7	120

2.5.2 Cross-Validation

When cross-validation is utilised, the test set is used for gradient descent while a separate validation set is used to measure error (Mitchell, 1997a). K-fold cross-validation is where k separate validation sets are created. These k different validation sets are used to test the model and the results are averaged.

2.5.3 Confusion Matrix

A proven, very successful way to measure how well a classifier is performing is by using a confusion matrix (Aurélien, 2017). A confusion matrix simply counts the number of times an object is classified correctly or incorrectly as in Table 2.1.

2.6 Dietary Assessment using Computer Vision

2.6.1 Convolution Neural Networks

Many researchers have used convolutional neural networks for image classification with various network architectures and many have used a food image dataset. Some of these papers will be looked at below. A summary of their results can be seen in Table 2.3.

(Mao, Yu, and Wang, n.d.) focused on a deep learning approach to food image recognition and based their neural network architecture on Inception-ResNet and Inception V3. Deep learning is a term usually given to algorithms based on neural networks. They also used the Food-101 dataset. For this system, Google's TensorFlow was used for image pre-processing. Pre-processing was needed as the environmental background is different in many food images. Because of these "Grey World method and Histogram equalization" (Mao, Yu,

and Wang, n.d.) were used. Amazon Web Services (AWS) Graphics Processing Units (GPU) instances were used for training. AWS instances are cloud servers. The results on completion were quite impressive with a Top-1 Accuracy of 72.55% and a Top-5 Accuracy of 91.31%.

Another research team in Japan, (Keiji Yanai and Kawano, 2015) researched this topic. This was built off previous research they had carried out in the field (Kawano and Keiji Yanai, 2014). They were aware of how difficult the problem was and therefore employed many techniques to solve the problem such as "pre-training with the large-scale ImageNet data, fine-tuning and activation features extracted from the pre-trained DCNN". In conclusion, they found that the "fine-tuned DCNN which was pre-trained with 2,000 categories" from ImageNet was the best method. A DCNN is a Deep Convolution Neural Network. A network can become a DCNN when the number of hidden layers is larger than three. While many of the CNNs discussed have been DCNN, most are just labelled as CNNs. They achieved results of 78.77% for Top-1 accuracy in the UECFOOD100 dataset.

(Kagaya, Aizawa, and Ogawa, 2014) also employed the use of convolutional neural networks for image detection. They used a CNN for the "tasks of food detection and recognition through parameter optimization". They found that a CNN is much better suited to the task than a SVM. They achieved an overall classification accuracy of 93.8% against their baseline accuracy of 89.7%. This accuracy was calculated using a dataset that they created specifically for this task. When they had completed the task, they analysed the trained convolutional kernels and came to an important conclusion. They found that "colour features are essential to food image recognition".

The second last paper that will be analysed, (Liu et al., 2016), was oriented around using a CNN for food image recognition and focused on developing a dietary assessment application for use on a smart phone. They used the UEC-256 and Food-101 dataset for their experiments and achieved impressive results. They used a CNN but "with a few major optimizations, such as optimized model and an optimized convolution technique". They used the Inception module for their CNN. After the inception module was complete, they made the GoogleNet architecture by combining modules. In total, the

network had 22 layers. They achieved the results shown in Table 2.2.

Table 2.2: DeepFood Results

Dataset	**Top-1**	**Top-5**
UEC-256	54.7%	81.5%
UEC-100	76.3%	94.6%
Food-101	77.4%	93.7%
UEC-256 With Bounding Box	63.8%	87.2%
UEC-100 With Bounding Box	77.2%	94.8%

(Mezgec and Koroušić Seljak, 2017) developed a new neural architecture specifically for detecting food and drink images using deep convolutional neural networks called NutriNet. The trained network was to be used to aid patients with Parkinson's disease in monitoring their diet. NutriNet was created based off the AlexNet architecture. The dataset used for this study consisted of approximately 500 images for each of over 500 classes. Through this dataset a Top-1 accuracy of 86.72% and a Top-5 accuracy of 94.47% was recorded. A smart phone application was used for real world testing which brought a Top-5 accuracy of 55%. The application also saved these real-world images from the smart phone to increase their dataset size. In conclusion, the team found that there were modifications that could be made to the NutriNet architecture as real-world images didn't perform incredibly well due to occlusion and background noise in the images. The detection and recognition steps were separated in this architecture. The team also acknowledges that joining these steps into a single DCNN may be successful and should be explored.

Table 2.3: Summary of results in CNN based methods

Title	**Dataset**	**Top-1 Accuracy**
(Mao, Yu, and Wang, n.d.)	Food 101	72.6%
(Keiji Yanai and Kawano, 2015)	UECFood101	78.8%
(Kagaya, Aizawa, and Ogawa, 2014)	Own dataset	93.8%
(Liu et al., 2016)	Food 101	77.4%
(Mezgec and Koroušić Seljak, 2017)	Own dataset	86.7%

2.6.2 Support Vector Machines

While CNNs have proven very successful in recent years, there are many other methods of food image identification and classification that have been employed by food image recognition researchers, such as SVMs. A summary of the results of these methods can be seen in Table 2.4.

In a study conducted by (Joutou and Keiji Yanai, 2009), a practical use for food image recognition in the form of a mobile phone application was proposed. In order to classify the images, multiple kernel learning (MKL) was used. MKL is similar to a SVM expect that instead of a single kernel during training, MKL "treats with a combined kernel which is a weighted linear combination of several single kernels" (Joutou and Keiji Yanai, 2009). The idea behind this is that different food types are distinguishable by different factors and using this method, the best of these factors can be used for classification of that food type. In the experiments carried out, three different factors were used for learning:

- Colour Histograms
- Gabor Texture Features
- Bag-of-Features using Scale Invariant Feature Transformation (SIFT)

According to (Szeliski, 2010), "SIFT features are formed by computing the gradient at each pixel" in a window and then using the correct level of the Gaussian filters where the window was detected. 50 different classifiers were created in a SVM using MKL with "one category as a positive set and other 49 categories as a negative set" (Joutou and Keiji Yanai, 2009). For each of these categories, a web scrape was carried out and then the best 100 images for each scrape was manually selected. Five-fold cross validation was utilised in the paper.

MKL proceeded to yield Top-1 results of 61.34% on the 50 food types and a Top-3 accuracy of 80.05%. The prototype mobile phone application resulted in a 37.55% user accuracy. While the Top-1 and Top-3 accuracies for this model show promising results, the user accuracy is very poor. This would suggest that the classifier does not work well with real life images and is possibly

overfitting to the training and testing dataset. MKL classifiers may not be promising for generalisation.

Another quite successful study was carried out using a SVM. (Pouladzadeh, Villalobos, et al., 2012) had established that both colour and texture are very important, but they also decided that shape and size are vital features to analyse. The proposed system has two main parts, segmentation followed by classification. To create a 'robust' system, a 'Robust Handling of Different Lighting Conditions' module is added to the system (Pouladzadeh, Villalobos, et al., 2012). This is so that various lighting conditions don't cause colour data to be distorted.

Since this paper calls for calorie estimation, the first step of the system calculates the size of the food portion. In order to do this, a coin or the users thumb is included in the image taken so that the pixel count of the thumb and the food can be compared to estimate the size. Following this the image is segmented into various portions. The following step classifies each segment of the image by extracting colour, texture and shape features and inputting these into a SVM.

12 different food types were trained for this SVM with an average classification accuracy of 92.6%. (Pouladzadeh, Villalobos, et al., 2012) concluded that it would be difficult to use their algorithm with real data and no evidence of real-world testing was recorded. This is unfortunate as it is difficult to get an idea of the success of the system. Unfortunately, only 12 food types were trained, and no evidence is given of how the classifier scales to more classes.

Another study that employed both a SVM and an emphasis on colour, texture and shape features, was carried out by (Pouladzadeh, Shirmohammadi, and Al-Maghrabi, 2014). Size was also a factor in the calorie measurement module of the system. It was found that using all four of these features increases the overall accuracy.

To segment the image successfully, Gabor filters were applied to separate texture features while colour was also utilised. For each segment established, size, shape, colour and texture features were extracted and using a SVM, a classification was made. The SVM used the radial basis function kernel. Calorie estimation was also a large part of this paper, and the users thumb was taken

with the food to calculate food size.

In the prototype application, once the classification had been made, the user can confirm or change the prediction. Another feature of the application was in regards to "Partially Eaten Food" (Pouladzadeh, Shirmohammadi, and Al-Maghrabi, 2014). This was accomplished by taking a picture before and after consumption and as a result only calculated the size of the food eaten and therefore more accurate calorie counts can be produced.

15 food types were trained using the SVM with 3,000 images. The accuracy for the classifier averaged at 90.41% using 10-fold cross-validation. There was also a calorie count accuracy of 86%. The best classification results were on single foods followed by non-mixed and finally mixed foods produced the worst results.

Even though (Pouladzadeh, Shirmohammadi, and Al-Maghrabi, 2014) segmented the image before classification, they still had poor results on mixed foods. The low number of classes the classifier was trained on makes it difficult to see how effective the classifier is. It would also be beneficial to know how long classification took on a new image as the classifier used features of colour, texture, size and shape which would be quite time consuming.

(Zhu et al., 2011) had a strong focus on the segmentation aspect of a dietary assessment system. The segmentation of the food images was achieved "using Normalized Cuts based on intensity and colour" (Zhu et al., 2011). Normalized Cuts is a graph-based segmentation method. To aid the segmentation aspect of this study, a common background colour was introduced to the images. Segmentation refinement was also an important module in the experiment. This is the process by which neighbouring segments with the same classification label are merged together. This also helps calculate a more accurate size estimation. The classification of the segmented image was processed by using a SVM calculating colour and texture features. Gabor filters were used for the texture feature extraction.

In the experimental results for this study, it was found that segmentation was not always successful "when the region of interest is camouflaged by making its boundary faint" (Zhu et al., 2011). In their case, it was a can of coke that wasn't segmented correctly. The classification accuracy was of 56.2% and

95.5% with ground truth segmentation data. 19 classes of food were used in this study with approximately 60 images per class. Very little information was given in this paper on the classifier used. The low classification results do not yield promising results for this method.

There was a study carried out on food identification through a smart phone application by (M.-Y. Chen et al., 2012). This study resulted in an application that allows a user to send an image of their food to a server which can give them an automatic response in 12 seconds. This back-end service can have 34 threads working concurrently as stated at the time the paper was published.

A SVM is used to classify the image across 50 categories trained on around 100 images each. The SVM uses SIFT and Local Binary pattern feature extractors. A separate SVM was trained for each of these extractors and was merged together using a " Multi-class AdaBoost algorithm" (M.-Y. Chen et al., 2012).

The study produced a Top-1 accuracy of 68.3%. Accuracy of 80.6%, 84.8% and 90.9% were recorded using Top-2, Top-3 and Top-5 accuracy respectively. Unfortunately, real-world image classification results were not recorded from the smartphone application. Due to this, it is difficult to measure the efficacy of the classifier, even more so because they only had 50 classes.

(Villalobos, Almaghrabi, Pouladzadeh, et al., 2012) researched the question of using a computer vision approach to this topic. They focus mostly on the segmentation and region of interest calculation of the system in their study.

The system in question requires two images of the food, one from above and one from the side. This helps with size estimation. The users thumb is required to be in the image for accurate size estimation. The application also requires an image after consumption as to not calculate calories for uneaten food. The system segments the image and then extracts colour, size and shape information from each segment. This data is then used by a SVM for classification along with a nutritional database for calorie information. Multiple segmentation methods were tested such as:

- Semi-automatic contour definition
- Watershed transformation

Table 2.4: Summary of accuracy in dietary assessment methods

Title	**Classes**	**Accuracy**
Food Image Recognition with Multiple Kernel Learning	50	61.3%
A Novel SVM Based Food Recognition Method	12	92.6%
Measuring Calorie and Nutrition from Food Image	15	90.4%
Segmentation Assisted Food Classification	19	56.2%
Large Scale Leaning for Food Image Classification	11	78.0%
Toward Dietary Assessment via Mobile Phone Video Camera	20	92.0%
Automatic Chinese Food Identification and Quantity Estimation	50	68.3%
Food Recognition and Nutrition Estimation on a Smartphone	15	85.0%

- Colour rasterization
- Edge accentication

The first two were dismissed due to poor results but the second two were used in conjunction for the segmentation aspect of the system. (Villalobos, Almaghrabi, Pouladzadeh, et al., 2012) did not provide extension information on the classifier used in this study and would be therefore very difficult to replicate. However, impressive segmentation results were obtained.

An application called "Snap-n-Eat" was proposed by (Zhang et al., 2015). When an image is taken using this application, the system finds saliency regions to remove the background of the image. If the image has multiple food types present, hierarchical segmentation takes place before proceeding to a SVM. Similar segments are merged together. These are found by using colour, texture and size.

SIFT and Histogram of Oriented Gradients (HOG) feature extractors are used on the image and these features are used by the SVM for classification. The SVM is trained using Scholastic Gradient Descent. (Zhang et al., 2015) also uses a Bag of Visual Words model along with k-means clustering. An accuracy of 85% on 15 classes was recorded using this method.

2.6.3 Other Methods

Methods outside of CNNs and SVMs are outlined below. A summary of the results can be seen in Table 2.4 along with SVM method results.

(Abbirami.R.S et al., 2015) proposed a food image recognition system using a Bag of Features model. This study used over 5,000 images separated into 11 classes. A clustering algorithm was employed on this study before classification. For the classification step, experiments were carried out using different methods:

- SVM
- ANN
- Random Forests

The final accuracy of the system was 78%. An accuracy of 78% isn't very promising for such a small number of classes. Approximately 500 images were used per class for each of the 11 classes. (Abbirami.R.S et al., 2015) also utilised a k-means clustering algorithm but very little information on the reasoning and results of using this was documented.

Similar to other approaches seen thus far, (Villalobos, Almaghrabi, Hariri, et al., 2011) employs the use of the users thumb in the image for size estimation. Once a photo has been taken by the user with their thumb present, the system segments the food on the plate using shape, colour and texture detectors. The system then classifies the food type based on these features.

In this paper, it was decided to allow the users to change the prediction by the system. The thumb of each user is calibrated upon first use of the application so that size estimation can be as accurate a possible. Very little information on the classification aspect of this paper was documented which would make the study very difficult to replicate.

Another study into using computer vision for dietary assessment was carried out by (N. Chen et al., 2010). They had a unique medium for the topic by using a video of the dishes in question and extracting frames from these videos to get the food from different angles.

(N. Chen et al., 2010) then formed a region of interest in the image, where there were the most food items and extracted colour and image features. These image features were extracted using Maximally Stable Extremal Regions (MSER), Speeded Up Robust Features (SURF) and Star detector. MSER is a region detector that is computed by thresholding an image in all areas that are grey and because of this, MSER only works on grayscale images (Szeliski, 2010). The Star detector and SURF are both used for the detection of "interesting keypoints in images" (N. Chen et al., 2010).

This research team also uses k-means clustering to build a bag-of-words model. Very little information on the implementation for this clustering algorithm was documented. The system had results as seen below across 20 categories using five images out of each video taken of the food:

- MSER - 95%
- SURF - 90%
- STAR - 90%

While the results seem quite impressive for this classifier, it was not documented in detail and it could be speculated that this system would be time consuming due to classifying five images and collating the results.

(Schap et al., 2014) proposed a system used by smart phones which sends an image of a user's food to a back-end system for computation. Once this has been completed, the image is segmented, features are extracted from each segment and these segments are classified. Colour and texture features are used for classification. The user can confirm or amend predictions of the food type.

Size estimation is also an important aspect of this system. In contrast to previous studies, (Zhang et al., 2015) uses food type shape and then those shape's geometric properties to estimate size.

(Schap et al., 2014) produced results of 94% out of 32 test cases. Documentation of the classification module is very sparse in this paper.

2.7 Object Detection Using CNNs

Due to the issues with composite images (images with multiple items) in food image recognition, an object detection approach may prove to be successful. If the objects (different food items) in an image could be classified separately then the overall efficiency of a nutritional assessment system, aided by deep learning, could be improved. Ross Girshik and other contributors had some very positive results in the area of object detection using region based convolutional neural networks. There were four iterations of papers based on this work by Ross and groups in UC Berkley, Microsoft and Facebook. A PHD student at the time of Ross's first paper also completed his dissertation on the subject. These papers, their results (Table 2.5) and the changes made through each iteration will be analysed thoroughly in the proceeding sections.

In the first paper written by Ross Girshik, while researching at UC Berkeley, focused on two main insights. These were that "one can apply high-capacity convolutional neural networks (CNNs) to bottom-up region proposals in order to localize and segment objects" and that "when training data is scarce, supervised pre-training for n auxiliary task, followed by domain-specific fine-tuning, yields a significant performance boost" (Girshick et al., 2014).

The system that they developed followed these steps:

- Take image as input.
- Extract approximately 2,000 region proposals from the image.
- Compute fixed length vectors of features for the regions using a convolutional neural network.
- Use a SVM to classify these regions.
- Bounding box regression for final region proposals.

This system utilised selective search to gather these region proposals, but they mention that a sliding-window detector is also an option. Ross Girshik and his team used the open source Caffe CNN library for this system. The system is quite efficient and scalable. It is scalable because of the fixed length vector

of features which will remain constant regardless of inputs and additional outputs. The team evaluated their results on a few metrics and test sets as seen in Table 2.5. Explanation of the datasets used can be seen in Table 2.6.

Ross Girshik's next iteration of work on region-based convolution neural networks took place in Microsoft Research. This paper was titled 'Fast R-CNN' as its aim was to decrease training and testing time "while also increasing detection accuracy" (Girshick, 2015).

This paper analyses why RCNN (Girshick et al., 2014) was slow and therefore how it could be improved. RCNN was classified to be slow because of three main factors:

- There are multiple stages to training as both a CNN and a SVM need to be trained.
- In training of the SVM, each region proposal must be written to disk and is therefore expensive.
- Object detection takes 47 seconds per image.

Due to these problems with RCNN, a new algorithm, titled Fast RCNN was proposed. The architecture is as follows. An image is taken as input along with a proposal for regions. The image is pushed through convolutional and pooling layers (using max pooling). A fixed-length vector of features is then extracted from each region proposal. These vectors are inputted to fully connected layers for bounding box location prediction. At detection time, a pass through of the net is all that is needed so this runtime is significantly less than RCNN.

Due to the success of RCNN and Fast RCNN, Faster RCNN was introduced to combat the problem of region proposal computation (Ren et al., 2015). The architecture for this system comprises of two modules. These consist of a convolutional neural network for region proposals (RPN) which the feeds into a Fast RCNN detector. These combine to produce a single neural network for object detection.

Instead of training these networks separately, the team had to look at how to share layers between the two networks. There were three options available:

- Alternating training whereby RPN is trained, and then used to train Fast RCNN. The Fast RCNN network is then used to initialise RPN and the process is iterated (Ren et al., 2015). This paper follows this approach.
- Approximate joint training.
- Non- approximate joint training.

Table 2.5: Results from Region Based CNN Research

	VOC07	**VOC10**	**VOC11**	**VOC12**	**COCO15**	**COCO16**
RCNN	58.5%	53.7%	47.9%	N/A	N/A	N/A
Fast RCNN	70.0%	68.8%	N/A	68.4%	N/A	N/A
Faster RCNN	78.8%	N/A	N/A	75.9%	42.7%	N/A
Mask RCNN	N/A	N/A	N/A	N/A	N/A	63.1%

Table 2.6: Datasets

Table	**Explanation**
VOC07	The PASCAL VOC dataset is used for the PASCAL (Pattern Analysis, Statistical Modelling and Computational Learning) Visual Object Classes Challenge. (Everingham et al., 2007) was used for the 2007 challenge.
VOC10	The PASCAL VOC 2010 dataset was used for the 2010 challenge (Everingham et al., 2010).
VOC11	The PASCAL VOC 2011 dataset was used for the 2011 challenge (Everingham et al., 2011).
VOC12	The PASCAL VOC 2012 dataset was used for the 2012 challenge (Everingham et al., 2012).
COCO15	The COCO (Common Objects in Context) dataset created by Microsoft ((Lin et al., 2014)) is used to measure the efficacy of object detection algorithms. The COCO 2015 dataset was released in 2015 for training.
COCO16	An updated version of the COCO dataset was released in 2015.

The most recent paper on this topic was also written by Ross Girshik while working with Facebook AI Research (K. He et al., 2017). Mask RCNN "extends Faster RCNN by adding a branch for predicting an object mask in parallel with the existing branch for bounding box regression" (K. He et al., 2017). Mask RCNN has two modules, like Faster RCNN, where the first module is the Region Proposal Network. In the second module, in parallel to classification, a binary mask is outputted for each region. Bounding box regression and classification are done in parallel.

2.8 APIs and Libraries

2.8.1 TensorFlow

TensorFlow is a deep learning software library for various machine learning paradigms. TensorFlow will be used to create neural networks. TensorFlow uses a data structure called a tensor which is basically an array of n dimensions. TensorFlow has two utilisations, through a Graphics Processing Unit (GPU) and through a Central Processing Unit (CPU). GPU computation is recommended for CNN training.

2.8.1.1 Central Processing Unit Computation

It is quite easy to get TensorFlow up and running if you are only using a CPU to train. TensorFlow CPU has been successfully installed on both Windows and Ubuntu, for this project. For Windows you can download and install using the TensorFlow website and on ubuntu you can using apt-get. Once installed, TensorFlow can be imported into any python shell or script for use. TensorFlow can also be used in C++. There will be various python implementations of neural networks in Chapter 3.

2.8.1.2 Graphics Processing Unit Computation

For use with a GPU, the set up for TensorFlow is a bit more complicated. Firstly you must check that the GPU in your machine is compatible for CUDA

8.0 using the NVIDIA website. If your GPU is compatible, you must install CUDA after signing up as an NVIDIA developer. CUDA 8.0 is compatible with TensorFlow. You also need to install cudnn6. The NVIDIA website contains tutorials to install these. Once these are installed, download and install tensorflow-gpu. This can be imported into python similar to CPU computation.

2.8.2 OpenCV

OpenCV is an industry wide, open source library for computer vision and machine learning (*About - OpenCV library* 2018). It has over 2500 algorithms that are available for use (*About - OpenCV library* 2018). OpenCV is supported across multiple languages and platforms such as Python, C++, C, Java, MATLAB, running on Windows, Android, Mac OS and Linux (*About - OpenCV library* 2018).

There is not much of the library utilised in this project due to the nature of TensorFlow but some algorithms for image reading, writing and resizing were used due to the ease of use.

```
image = cv2.imread('image.jpg')

resized = cv2.resize(image, (299, 299))

cv2.imwrite('imageResized.jpg', resized)
```

2.8.3 NumPy

NumPy is a package for Python that is used for scientific computing (developers, 2018). In the context of this FYP, it will be used for multi-dimensional array manipulation.

2.9 Public Domain Datasets

Some public domain datasets were used in this FYP while learning TensorFlow and conducting empirical studies. These datasets are outlined below.

2.9.1 Food-101

(Bossard, Guillaumin, and Van Gool, 2014) created the Food-101 dataset which consists of 101 different food types. Each food type in the dataset has 1,000 images associated. These images can be divided up into a training, test and validation set as seen fit by the user. The Food-101 dataset is open for public use if it is used for research purposes and not for commercial use. Examples from the Food-101 dataset can be seen below.

Figure 2.10: Hot-dog

Figure 2.11: Chocolate Cake

Figure 2.12: Apple Pie

Figure 2.13: Pizza

2.9.2 MNIST

The MNIST dataset (created by (LeCun and Cortes, 2010)) consists of 70,000 hand written digits. It is widely used in introductory CNN tutorials. The dataset is split into 60,000 training images and 10,000 test images. All the images are of the same dimensions. Examples from the MNIST dataset can be seen below.

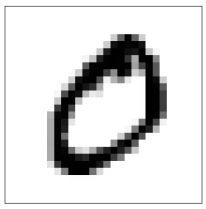


Figure 2.14: MNIST 0

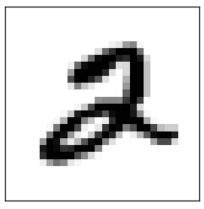


Figure 2.15: MNIST 2

Figure 2.16: MNIST 4

Figure 2.17: MNIST 5

2.9.3 CIFAR-10

(Krizhevsky, Nair, and G. Hinton, n.d.) created the CIFAR-10 dataset which consists of 10 classes. Each of the 10 classes has 6,000 32x32 colour image associated and these are split between a 5,000 image training set and a 1,000 image test set. There are five training batches and one test batch in the dataset. The test set contains 1,000 randomly selected images from each class. The training batches consist of random orderings of the remaining images. Examples from the CIFAR-10 dataset can be seen below.

Figure 2.18: CIFAR-10 Truck

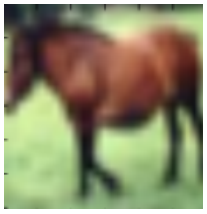


Figure 2.19: CIFAR-10 Horse

Figure 2.20: CIFAR-10 Boat

Figure 2.21: CIFAR-10 Frog

2.10 Conclusion

This chapter has covered a comprehensive survey of literature along with an overview of the underlying concepts to the approaches used in the literature. This survey has covered the topics of using CNNs, SVMs and other methods for nutritional assessment. Topics also covered include evaluating the output, object detection using CNNs, neural computing, an overview of CNNs and SVMs and an overview of the technologies used in this project.

From the literature review, there are many different approaches that have been promising in the area of using deep learning for nutritional assessment. It has been decided to attempt to replicate the work carried out by (Keiji Yanai and Kawano, 2015) with some modifications that will be outlined in the following chapters. While this approach will be attempted, there are many viable options that can still be investigated for future work.

Chapter 3

Introduction to Using TensorFlow

The purpose of this chapter is to give an overview of what has been completed to gain understanding of how to create CNNs and also how to work with datasets. Three experiments are explored in this chapter, the first two of which are tutorials completed as an introduction to TensorFlow and CNNs. The final experiment explores how to feed datasets from file directories into CNNs. In addition to these experiments a Deep Learning course was completed on Udacity (an online nanodegree tutorial site) (Udacity, 2018).

3.1 Template for Experiments

The template followed for all experiments in chapters three, four and five is outlined in Table 3.1.

3.2 Udemy Tutorial

3.2.1 Objective

There are many on line resources that are geared towards helping deep learning novices. One of these resources is a course on Udemy titled 'Complete Guide

Table 3.1: Experiment Template

Section	**Rationale**
Objective	An explanation of the purpose of the experiment along with how it is carried out.
Network Architecture	An explanation of the network architecture used in the experiment. If the architecture has been explained in another experiment, the architecture will only be referenced by name.
Dataset	The dataset used for the experiment, with its source and an overview of its details. A brief mention will be made in following experiments.
API's and Libraries	Reference to the technologies used as outlined in Chapter 2.
Script	Snippets of the script used for the experiment.
Results	The results acquired from the experiment, usually in the form of a percentage accuracy.
Analysis	States any information gained from the experiment and speculation for the reasoning of the results.

to TensorFlow for Deep Learning with Python' (*Complete Guide to Tensorflow for Deep Learning with Python* 2017). This course has a section on Convolutional Neural Networks which has been followed and completed. The CNN used in this section is very simple as it is an introductory CNN. It consists of two convolutional layers, two max pooling layers and a fully connected layer.

Table 3.2: Udemy Tutorial

Network Architecture	A simple architecture was used for prototyping purposes
Dataset	MNIST dataset
APIs and Libraries	TensorFlow

3.2.2 Script

The following Figures 3.1, 3.2, and 3.3 convey key aspects of the script needed to create and train this model. Figure 3.1 defines helper functions that are used to initialise weights and biases, create convolutional layers, create pooling layers, and create fully connected layers. Figure 3.2 uses these helper functions

to create the layers for this network. Finally, the code required to train the model is outlined in Figure 3.3.

Figure 3.1: Helper Functions (*Complete Guide to Tensorflow for Deep Learning with Python* 2017)

```
#INIT WEIGHTS
def init_weights(shape):
    init_random_dist = tf.truncated_normal(shape, stddev = 0.1)
    return tf.Variable(init_random_dist)

#INIT BIAS
def init_bias(shape):
    init_bias_vals = tf.constant(0.1, shape = shape)
    return tf.Variable( init_bias_vals )

#CONV2D
def conv2d(x, W):
    #x -> [batch, H, W, Channels]
    #W -> [filterH, filterW, ChannelsIn, ChannelsOut]
    return tf.nn.conv2d(x, W, strides = [1,1,1,1], padding = 'SAME')

#POOLING
def max_pool_2by2(x):
    #x -> [batch, H, W, Channels]
    return tf.nn.max_pool(x, ksize = [1,2,2,1], strides = [1,2,2,1],
        padding = 'SAME')

#NORMAL (FULLY CONNECTED)
def normal_full_layer(input_layer, size):
    input_size = int(input_layer.get_shape()[1])
    W = init_weights([input_size, size])
    b = init_bias([size])
    return tf.matmul(input_layer, W) + b
```

3.2.3 Results

The final Top-1 accuracy for this experiment was of 97.3%, calculated from a batch of unseen test images.

Figure 3.2: Layer Creation (*Complete Guide to Tensorflow for Deep Learning with Python* 2017)

```
#32 features for every 5 x 5 kernel with 1(grayscale)
convo_1 = convolutional_layer(x_image, shape = [5,5,1,32])
convo_1_pooling = max_pool_2by2(convo_1)

convo_2 = convolutional_layer(convo_1_pooling, shape = [5,5,32,64])
convo_2_pooling = max_pool_2by2(convo_2)

convo_2_flat = tf.reshape(convo_2_pooling, [-1,7*7*64])
full_layer_one = tf.nn.relu( normal_full_layer (convo_2_flat , 1024))
```

Figure 3.3: Training the Model (*Complete Guide to Tensorflow for Deep Learning with Python* 2017)

```
with tf.Session() as sess:
    sess.run(init)

    #for every steo, get a batch and run it through the model
    for i in range(steps):
        batch_x, batch_y = mnist.train.next_batch(32)
        batch_test = mnist.test.next_batch(32)
        sess.run(train, feed_dict = {x:batch_x, y_true:batch_y,
            hold_prob:0.5})
        if i%500 == 0:
            print("ACCURACY: ")
            match = tf.equal(tf.argmax(y_pred, 1), tf.argmax(y_true, 1))
            acc = tf.reduce_mean(tf.cast(match, tf.float32))
            print(sess.run(acc, feed_dict = {x:batch_test,
                y_true:mnist.test.labels, hold_prob:1.0}))
            print('\n')
    sess.run(acc, feed_dict = {x:mnist.test.images} )
```

3.3 Udemy Tutorial 2

3.3.1 Objective

Similar to the previous experiment, this experiment is a Udemy course exercise (*Complete Guide to Tensorflow for Deep Learning with Python* 2017). In contrast, this does not use a dataset built into TensorFlow so therefore, there is extra configuration to be done on the dataset.

Table 3.3: Udemy Tutorial 2

Network Architecture	Same as 3.2
Dataset	CIFAR-10 dataset
APIs and Libraries	TensorFlow

3.3.2 Script

Key features of the script used to create the CNN for this tutorial are outlined in Figures 3.4, 3.5 and 3.6. Figure 3.4 unpickles the dataset from the files that it is stored in. A pickled file is a Python object that has been serialised. Once the data has been unpickled, the data must be set up to be the correct size and shape and it must also be accessible by batches. This is defined in Figure 3.5. Finally, Figure 3.6 documents the code required to train the model.

3.3.3 Results

A final Top-1 accuracy of 71% was reached.

3.3.4 Analysis

Two tutorials have been carried out training models, using similar architectures, on two different datasets. The first model was trained on the MNIST dataset and resulted in a Top-1 accuracy of 97.3% while the second model was trained on the CIFAR-10 dataset with a Top-1 accuracy of 71%. Both

Figure 3.4: Unpickle Images (*Complete Guide to Tensorflow for Deep Learning with Python* 2017)

```
def unpickle( file ):
    import pickle
    with open( file , 'rb') as fo:
        cifar_dict = pickle.load(fo, encoding='bytes')
    return cifar_dict

dirs = ['batches.meta','data_batch_1',
'data_batch_2','data_batch_3','data_batch_4'
,'data_batch_5','test_batch']
all_data = [0,1,2,3,4,5,6]

for i,direc in zip(all_data,dirs):
    all_data[i] = unpickle(CIFAR_DIR+direc)

batch_meta = all_data[0]
data_batch1 = all_data[1]
data_batch2 = all_data[2]
data_batch3 = all_data[3]
data_batch4 = all_data[4]
data_batch5 = all_data[5]
test_batch = all_data[6]
```

datasets had 10 classes associated. There are three main reasons why the Top-1 accuracy of these two models has a difference of 26.3% which are:

1. Dataset size: In MNIST, there are 60,000 images used for training as opposed to 50,000 in CIFAR-10. The difference here would be minuscule, but it would have some impact.

2. Colour channels: The images in MNIST are in black and white while the CIFAR-10 images have three colour channels associated.

3. Object complexity: The objects in the the MNIST dataset are simpler shapes with a plain background. The CIFAR-10 images are more complex such as a horse and the background is noisy.

Figure 3.5: Set Up Images for Training (*Complete Guide to Tensorflow for Deep Learning with Python* 2017)

```
def set_up_images(self):
    # Vertically stacks the training images
    self.training_images = np.vstack([d[b"data"] for d in
        self.all_train_batches])
    train_len = len(self.training_images)

    # Reshapes and normalizes training images
    self.training_images =
        self.training_images.reshape(train_len,3,32,32).transpose(0,2,3,1)/255
    # One hot Encodes the training labels (e.g.  [0,0,0,1,0,0,0,0,0,0])
    self.training_labels = one_hot_encode(np.hstack([d[b"labels"] for d in
        self.all_train_batches]), 10)

    # Vertically stacks the test images
    self.test_images = np.vstack([d[b"data"] for d in self.test_batch])
    test_len = len(self.test_images)

    # Reshapes and normalizes test images
    self.test_images =
        self.test_images.reshape(test_len,3,32,32).transpose(0,2,3,1)/255
    self.test_labels = one_hot_encode(np.hstack([d[b"labels"] for d in
        self.test_batch]), 10)

#get the next batch
def next_batch(self, batch_size):
    x = self.training_images[self.i:self.i+batch_size].reshape(100,32,32,3)
    y = self.training_labels[self.i:self.i+batch_size]
    self.i = (self.i + batch_size) % len(self.training_images)
    return x, y
```

Figure 3.6: Train the Model (*Complete Guide to Tensorflow for Deep Learning with Python* 2017)

```
#loss function
cross_entropy =
    tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels =
    y_true, logits = y_pred))
#activation function
optimizer = tf.train.AdamOptimizer(learning_rate = 0.001)
train = optimizer.minimize(cross_entropy)
init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    #for each step, get the next batch and run through the model
    for i in range(10000):
        batch = ch.next_batch(100)
        sess.run(train, feed_dict={x: batch[0], y_true: batch[1],
            hold_prob: 0.5})

        # PRINT OUT A MESSAGE EVERY 100 STEPS
        if i%1000 == 0:
            matches = tf.equal(tf.argmax(y_pred,1),tf.argmax(y_true,1))
            acc = tf.reduce_mean(tf.cast(matches,tf.float32))
            testSet = ch.next_test_batch(100)
            print('Accuracy: ')
            print(sess.run(acc,feed_dict={x:testSet[0]
                ,y_true:testSet[1], hold_prob:1.0}))
```

3.4 Using the Food-101 Dataset

3.4.1 Objective

For the next experiment, it was decided to use the Food-101 dataset. An on line tutorial was used to create a dataset in TensorFlow using image directories on disk (*Build an Image Dataset in Tensorflow* 2018). Tutorials used previously were also utilised for this experiment such as (*Complete Guide to Tensorflow for Deep Learning with Python* 2017).

Table 3.4: Using the Food-101 Dataset

Network Architecture	Similar to 3.2
Dataset	Food-101 dataset
APIs and Libraries	TensorFlow

3.4.2 Script

Figures 3.7, 3.8 and 3.9 document the key aspects of the script used in this experiment. The activities documented include reading in the dataset (Figure 3.7), create batches from the dataset (Figure 3.8) and training the model (Figure 3.9).

3.4.3 Results

The final accuracy for this model was 18.8% after 5,000 steps.

3.4.4 Analysis

The poor accuracy of this model is to be expected due to the simple architecture of the network.

Figure 3.7: Read in the Dataset (*Build an Image Dataset in Tensorflow* 2018)

```
# Reading the dataset
def read_images(dataset_path, mode, batch_size):
    imagepaths, labels = list(), list()
    # An ID will be given to each sub-folder alphabetically
    label = 0
    classes = sorted(os.walk(dataset_path).__next__()[1])
    # List each sclass
    for c in classes:
        c_dir = os.path.join(dataset_path, c)
        walk = os.walk(c_dir).__next__()
        # Add each image to the training set
        for sample in walk[2]:
            if sample.endswith('.jpg') or sample.endswith('.jpeg'):
                imagepaths.append(os.path.join(c_dir, sample))
                labels.append(label)
        label += 1
```

Figure 3.8: Create Batches (*Build an Image Dataset in Tensorflow* 2018)

```
# Convert to Tensor data strcuture
imagepaths = tf.convert_to_tensor(imagepaths, dtype=tf.string)
labels = tf.convert_to_tensor(labels, dtype=tf.int32)

# Build a queue
image, label = tf.train.slice_input_producer([imagepaths, labels],
                                             shuffle=True)

# Read images
image = tf.read_file(image)
image = tf.image.decode_jpeg(image, channels=CHANNELS)

# Resize images
image = tf.image.resize_images(image, [IMG_HEIGHT, IMG_WIDTH])

# Normalize the images
image = image * 1.0/127.5 - 1.0

# Create batches
X, Y = tf.train.batch([image, label], batch_size=batch_size,
                      capacity=batch_size * 8,
                      num_threads=4)
return X, Y
```

Figure 3.9: Train the Model (*Build an Image Dataset in Tensorflow* 2018)

```
with tf.Session() as sess:
    sess.run(init)
    tf.train.start_queue_runners()

    #For each step in range, run batch through the model
    for step in range(1, num_steps+1):
        if step % display_step == 0:
            # Run optimization and calculate batch loss and accuracy
            _, loss, acc = sess.run([train_op, loss_op, accuracy])
            print("Step " + str(step) + ", Minibatch Loss= " + \
                  "{:.4f}".format(loss) + ", Training Accuracy= " + \
                  "{:.3f}".format(acc))
        else:
            sess.run(train_op)
```

3.5 Conclusion

This chapter has outlined the work done in learning about using TensorFlow to create CNNs and working with datasets.

Chapter 4

Empirical Studies Part 1

This chapter consists of experiments which train various TensorFlow models. There are parameters that can be changed in the code that creates a TensorFlow model based on the Inception-V3 model architecture. These parameters were changed to see how the model accuracy would be affected. In addition to this, change in dataset size was explored in relation to model accuracy.

4.1 Experiment 1: Retrain ImageNet Inception V3 Model

4.1.1 Objective

For this experiment, it was decided to take inspiration from (Keiji Yanai and Kawano, 2015), where pre-training was used for training a model for food classification. In order to achieve this, the final layer of the Inception V3 model which was trained on the ImageNet dataset had to be retrained. This is called transfer learning. A tutorial created by Google, on the TensorFlow website, was followed for direction on this process (*How to Retrain Inception's Final Layer for New Categories* 2018).

Firstly, to retrain the final layer of a model, a dataset must be prepared in the correct way. The Food-101 dataset (Bossard, Guillaumin, and Van Gool, 2014) was used for this experiment, which will be analysed below.

Figure 4.1: Retrain Inception Command

```
python TensorFlow/examples/image_retraining/retrain.py \ --image_dir
~/dataset_directory
```

This dataset contrasts with (Keiji Yanai and Kawano, 2015) as they used the UECFOOD100 dataset. The Food-101 dataset was chosen due to larger number of images per class. The UECFOOD100 dataset has 100 images per class while the Food-101 dataset has 1,000 images per class. The dataset must be structured so that there is a separate directory for each class with the directory name as the class name. These directories should contain all the images for this class.

Once this dataset has been set up correctly, a directory can be found on GitHub which contains the necessary files for this tutorial. When the directory has been downloaded, the command in Figure 4.1 can be executed.

The first thing that the script will do is create bottleneck files for the images. A bottleneck is a term used to define the final layer before the output layer. This is so that for each image, we do not have to push it through the entire network during training (*How to Retrain Inception's Final Layer for New Categories* 2018).

After, the bottlenecks are created, the training can be completed. The images are split into three sub directories of training, testing and validation. By default, these images are split into percentages of 80%, 10% and 10% respectively. The model is trained at a default of 4,000 steps.

At the final stage of the script, the model is run on a batch of test images not yet seen and a final test accuracy is displayed. This can be seen in the Script section below.

The command used for using this model once it is trained can be seen in Figure 4.2.

4.1.2 Script

The following snippets of code in Figure 4.3 and Figure 4.4 are from the

Figure 4.2: Label Image Command

```
python TensorFlow/examples/label_image.py
    --graph=/tmp/output_graph.pb
--labels=/tmp/output_labels.txt --input_layer=Mul
    --output_layer=final_result
--input_mean=128 --input_std=128 --image=~/image_directory
```

Table 4.1: Retrain ImageNet Inception V3 Model

Network Architecture	Inception-V3 architecture (*Inception in Tensorflow* 2018)
Dataset	Food-101 dataset
APIs and Libraries	TensorFlow and NumPy

retrain.py script. Figure 4.3 defines how to add a new layer to the top of the network while Figure 4.4 defines how to evaluate the accuracy of the new layer.

4.1.3 Results

The final test Top-1 accuracy for this retrained model was 54.8%. These results are calculated by passing an unseen batch of test images through the model. The 95% confidence interval of the Top-1 accuracy of this model is between 45.1% and 64.5%. This confidence interval was calculated by Equation 4.1.

$$e \pm 1.96\sqrt{\frac{e(1-e)}{n}} \tag{4.1}$$

An image of pizza Figure 4.5, was fed into the model with the following results:

- pizza 0.925
- pancakes 0.008
- nachos 0.007
- beef carpaccio 0.006
- tiramisu 0.004

In contrast, Figure 4.6 was not classified as a pizza.

Figure 4.3: Add New Layer (*How to Retrain Inception's Final Layer for New Categories* 2018)

```
# Add new layer to the network
(train_step, cross_entropy, bottleneck_input, ground_truth_input,
final_tensor) = add_final_training_ops(
        len(image_lists.keys()), FLAGS.final_tensor_name,
        bottleneck_tensor,
        model_info['bottleneck_tensor_size'],
        model_info['quantize_layer'])

# Create operations to evaluate the accuracy of the model
evaluation_step, prediction = add_evaluation_step(
final_tensor, ground_truth_input)

# Set up weights to initial default values.
init = tf.global_variables_initializer()
sess.run(init)
```

Figure 4.4: Evaluate Model (*How to Retrain Inception's Final Layer for New Categories* 2018)

```
# Run final test evaluation
test_bottlenecks, test_ground_truth, test_filenames = (
    get_random_cached_bottlenecks(
        sess, image_lists, FLAGS.test_batch_size, 'testing',
        FLAGS.bottleneck_dir, FLAGS.image_dir, jpeg_data_tensor,
        decoded_image_tensor, resized_image_tensor, bottleneck_tensor,
        FLAGS.architecture))
test_accuracy, predictions = sess.run(
    [evaluation_step, prediction],
    feed_dict={bottleneck_input: test_bottlenecks,
        ground_truth_input: test_ground_truth})
tf.logging.info('Final test accuracy = %.1f%% (N=%d)' %
                (test_accuracy * 100, len(test_bottlenecks)))
```

Figure 4.5: Pizza - sourced from https://www.cicis.com/

Figure 4.6: Pizza not classified correctly by the model

4.1.4 Analysis

These poor results are not that surprising. This is because we have many classes to train for, 101, and no parameter tuning has been carried out on this code. Figure 4.6 was not classified correctly and this is most likely due to the fact that the pizza does not take up much of the image.

4.2 Experiment 2: Retrain with Extended Dataset

4.2.1 Objective

As the Food-101 dataset mostly consisted of meals (Bossard, Guillaumin, and Van Gool, 2014), it was decided to extend the dataset slightly by including some single foods such as:

- cheese
- grapes

- banana
- apple
- orange
- spaghetti
- roll

To collect these images, the ImageNet repository was utilised to search for these foods individually and then download the subset of images to be included with the Food 101 dataset (Deng et al., 2009). This new extended dataset was named Food-101+ which has 108 classes. The retrain.py script was run on the Food-101+ dataset as in 4.1.

Table 4.2: Retrain with Extended Dataset

Network Architecture	Inception-V3
Dataset	Food-101+ dataset
APIs and Libraries	TensorFlow and NumPy
Script	As seen in 4.1

4.2.2 Results

For this model, a Top-1 accuracy of 55.3% was achieved. The 95% confidence interval of the Top-1 accuracy of this model is between 45.9% and 64.7%. A image of a banana (Figure 4.7) was fed into the model with the following results:

- banana 0.9962
- orange 0.0009
- cheese 0.0003
- frozen yoghurt carpaccio 0.0002
- churros 0.0001

Figure 4.7: Banana - sourced from http://www.ciaoimports.com/

4.2.3 Analysis

The slight increase in accuracy from 4.1, from 54.8% to 55.3%, makes sense. Since the model was pre-trained using the ImageNet dataset and all the new images used were from ImageNet, we would expect a higher classification accuracy on the new additions to the dataset. This would increase the overall average classification accuracy.

4.3 Experiment 3: Retrain with Parameter Tuning

4.3.1 Objective

Within the retrain.py script (*How to Retrain Inception's Final Layer for New Categories* 2018), as mentioned in previous experiments, there are various parameters that can be set and changed. Various combinations of these parameters were changed to see if it would increase the test accuracy of the model.

Table 4.3: Retrain with Parameter Tuning

Network Architecture	Inception-V3
Dataset	Food-101+ dataset
APIs and Libraries	TensorFlow and NumPy

4.3.2 Script

Script as seen in 4.1 but with some additions to calculate Top-5 accuracy as seen in Figure 4.9. Also documented is the code used to label a new image using the model (Figure 4.10).

Figure 4.9 takes 10 images for each class and runs each through the model. The script then checks if the expected value is within the top 5 predictions returned from Figure 4.10. If it is, the amount of correct top 5 predictions is incremented by one. To calculate the Top-5 accuracy, the number of correct top 5 predictions is divided by the total number of images tested (10 images per class x number of classes) and multiplied by 100 to obtain a percentage value.

Figure 4.8: Retrain Inception with Parameter Tuning Command

```
1 python retrain_top5.py \ --image_dir
2 ~/dataset_directory \ --how_many_training_steps 4000 \ --learning_rate
    0.01 \
3 --testing_percentage 10 \ --validation_percentage 10
```

Some further parameters could be set such as:

- –flip_left_right
- –random_crop
- –random_scale
- –random_brightness

4.3.3 Results

The results of each set of parameters can be seen in Table 4.4. Using the parameters of 10,000 steps, and a learning rate of 0.1, the model achieved a Top-1 accuracy of 66.3% and a Top 5 accuracy of 85.96%. This Top-5 accuracy was calculated from 1090 images in the test dataset and the average probability

Table 4.4: Comparison of parameters

Parameter Tuning	**Steps**	**Learning Rate**	**Test %**	**Validation %**	**Results**
Configuration 1	8,000	0.01	10	10	59.1%
Configuration 2	8,000	0.10	10	10	65.8%
Configuration 3	10,000	0.10	10	10	66.3%
Configuration 4	12,000	0.10	10	10	66.6%
Configuration 5	10,000	0.20	10	10	66.0%
Configuration 6	10,000	0.10	15	15	66.3%

of the predictions was 0.62. The 95% confidence interval of the Top-1 accuracy of this model is between 57.4% and 75.2%.

4.3.4 Analysis

The set of parameters that seem to be the most effective are 10,000 steps with a 0.1 learning rate. Graphs of this model can be seen in Figures 4.12 and 4.11. These are based on the validation set and then the test set respectively. A side by side comparison can also be seen in Figures 4.13 and 4.14 where orange is for the test set and blue is for the validation set.

There were two separate factors that each increased classification accuracy of about 5% each. These were training steps and learning rate.

Training steps are related to the number of images so before, when the training steps were at 4,000, not all of the training images were being used. As the steps were increased twofold we saw a 3.8% increase in accuracy.

Another parameter that increased accuracy significantly was learning rate. The default learning rate is 0.01 which was increased to 0.1. This resulted in an increase of 6.7%. This is most likely since we are only looking at the last layer, we can afford to change the weights more significantly.

The Top 5 accuracy of the model was quite good but it was not run on the same number of images as the final test accuracy. This is because TensorFlow does not have an API for calculating Top 5 accuracy. As a result, a script had to be created to calculate it. The average probability of 0.62 is also quite close to the overall Top-1 accuracy which may imply some correlation between the two, but this is merely speculation.

Figure 4.9: Top-5 Accuracy Calculation

```
#Variables used to store a list of all classes, the amount of images tested,
# the amount of images with a top 5 accuracy and the sum of the highest probabilities
classes = list(image_lists.keys())
image_count = 0
top_5_count=0
total_of_top1_probs = 0
i = 0
class_counter = 0

#Loops through all test images, selecting 10 images per class
#and running them through the method in 'label_image.py'.
while(i < len(test_bottlenecks)):
  class_counter += 1

  if class_counter > 10:
    i = int(round((i + len(test_bottlenecks)/len(classes)) - 10))
    class_counter = 0
  else:
    image_count += 1
     results_from_classifier = label_image.runModel(test_filenames[i])
    results = results_from_classifier[0]
    probabilities = results_from_classifier[1]

    if classes[test_ground_truth[i]] in results:
      top_5_count += 1
    else:
      print("Expected: " + classes[test_ground_truth[i]])
      for result in results:
        print("Classes: " + result)
      print("")

    total_of_top1_probs += max(probabilities)
    i = i + 1
print(str(top_5_count))
average_probabilities = total_of_top1_probs/(len(classes*10))

#Prints out the amount of test images used, the top 5 accuracy
# and the average probability of predictions.
print("Amount of test images: " + str(image_count))
print("Top 5 Accuracy: " + str((top_5_count/image_count)*100))
print("Average probability: " + str(average_probabilities))
```

Figure 4.10: Label Image - adapted from (*How to Retrain Inception's Final Layer for New Categories* 2018)

```
def runModel(file_name):
    model_file = \
      "/tmp/output_graph.pb"
     label_file  = "/tmp/output_labels.txt"
    input_height = 299
    input_width = 299
    input_mean = 128
    input_std = 128
    input_layer = "Mul"
    output_layer = " final_result "

    graph = load_graph(model_file)
    t = read_tensor_from_image_file(file_name,
                                    input_height=input_height,
                                    input_width=input_width,
                                    input_mean=input_mean,
                                    input_std=input_std)

    input_name = "import/" + input_layer
    output_name = "import/" + output_layer
    input_operation = graph.get_operation_by_name(input_name)
    output_operation = graph.get_operation_by_name(output_name)

    with tf.Session(graph=graph) as sess:
        results = sess.run(output_operation.outputs[0],
                           {input_operation.outputs[0]: t})
    results = np.squeeze(results)

    top_k = results.argsort() [-5:][::-1]
    labels = load_labels( label_file )

    setIndex = False

    top5_results = [None] * 5
    index = 0
    for i in top_k:
        top5_results[index] = labels[i]
        index += 1

     final_results  = [top5_results, results]
    return  final_results
```

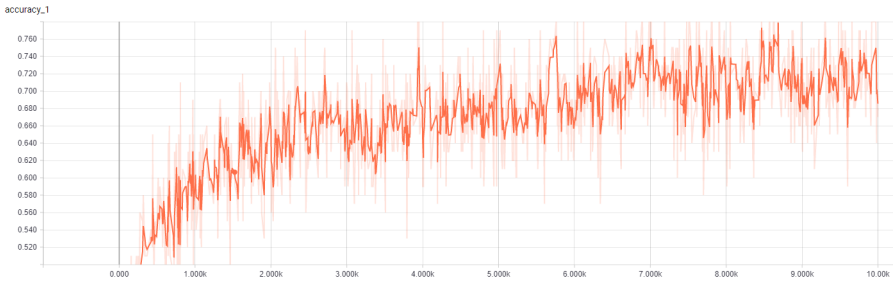


Figure 4.11: Graph of accuracy of the test dataset during training

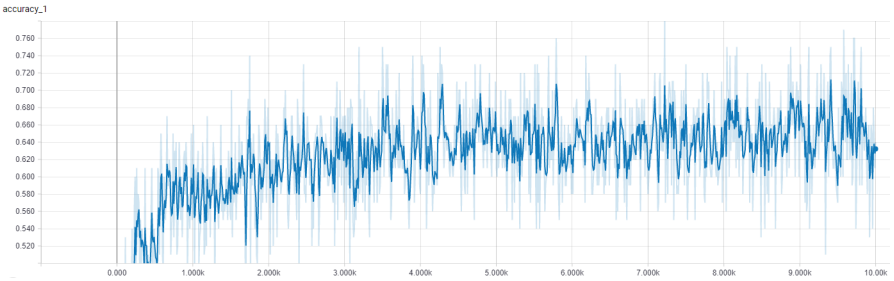


Figure 4.12: Graph of accuracy of the validation dataset during training

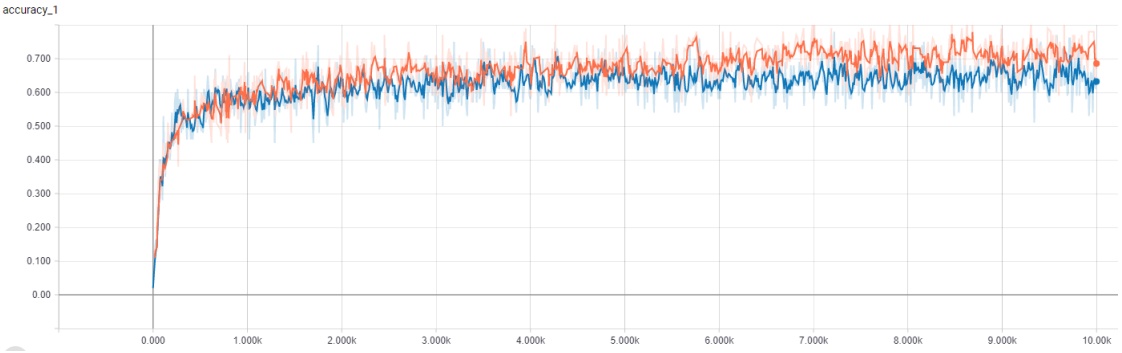


Figure 4.13: Comparison of accuracy

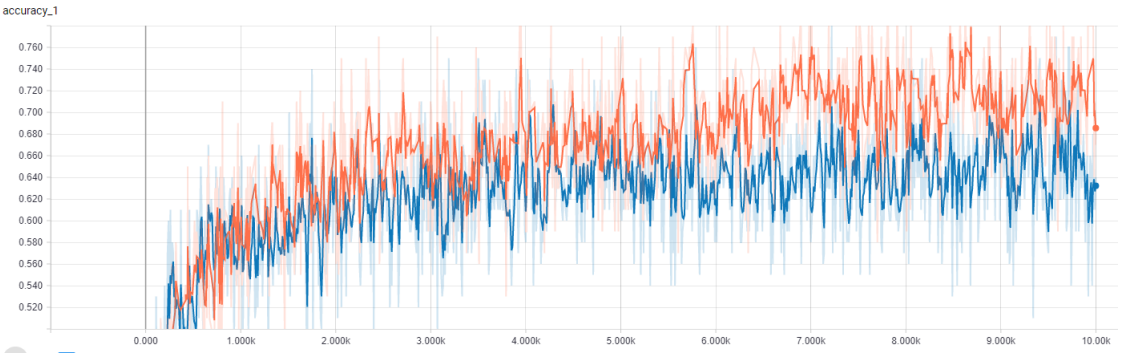


Figure 4.14: Comparison of accuracy

4.4 Experiment 4: MobileNet

4.4.1 Objective

Since the end goal for this project is to have a smartphone application that a user can use to keep track of their calorie measurement, there are a couple of options in how to achieve this. Firstly, an image can be taken on the phone and sent to a server to run a classification algorithm. Secondly, a model can be stored on the phone for computation. Transfer learning was once again used but on a different, smaller architecture called MobileNet (Howard et al., 2017).

Table 4.5: MobileNet

Network Architecture	MobileNet
Dataset	Food-101+ dataset
APIs and Libraries	TensorFlow and NumPy

4.4.2 Script

The retrain.py script (*How to Retrain Inception's Final Layer for New Categories* 2018) was used with a different command parameter as in Figure 4.15.

Figure 4.15: Train Model using MobileNet Architecture Command

```
python TensorFlow/examples/image_retraining/retrain.py \ --image_dir
~/dataset_directory \ --architecture mobilenet_1.0_224 \
--how_many_training_steps 10000 \ --learning_rate 0.1
```

4.4.3 Results

The final Top-1 accuracy of this model came to 50.2%. The 95% confidence interval of the Top-1 accuracy of this model is between 40.8% and 59.6%.

4.4.4 Analysis

There was a decrease of 16.1% in this model to the highest accuracy from 4.4. This is due to the smaller architecture which is aimed to be faster and smaller with an expected decrease in accuracy.

4.5 Experiment 5: Food-101+ subset

4.5.1 Objective

In many of the papers that have been researched where food image classification was carried out, they attempted to classify a lot less than 108 food types as has been the case for experiments previously shown. (Pouladzadeh, Villalobos, et al., 2012) used 12 classes, (Pouladzadeh, Shirmohammadi, and Al-Maghrabi, 2014) had 15, (Abbirami.R.S et al., 2015) attempted to classify 11 classes of food, 20 classes were used in (N. Chen et al., 2010) and (Zhang et al., 2015) predicted 15 classes. Due to the lower number of classes in these papers, it was decided to retrain Inception-V3 on a subset of the Food-101+ dataset to benchmark results. 13 classes were selected from the Food-101+ dataset for training.

Table 4.6: Food-101+ subset

Network Architecture	Inception-V3
Dataset	Food-101+ subset
APIs and Libraries	TensorFlow and NumPy

4.5.2 Results

A Top-1 accuracy of 92.6% was recorded for this experiment which performs quite high in comparison to the data in Table 4.7. A Top-5 accuracy of 100% was calculated with an average prediction probability of 0.89 using 130 images. This was calculated using the script defined in 4.3. The 95% confidence interval of the Top-1 accuracy of this model is between 78.4% and 100%.

Table 4.7: Accuracy of other studies

Reference	Classes	Accuracy
(Pouladzadeh, Villalobos, et al., 2012)	12	92.6%
(Pouladzadeh, Shirmohammadi, and Al-Maghrabi, 2014)	15	90.4%
(Abbirami.R.S et al., 2015)	11	78.0%
(N. Chen et al., 2010)	20	91.7%
(Zhang et al., 2015)	15	85.0%

Table 4.8: Summary of Results of Chapter 4

Experiment	Key Findings
Experiment 1	A model was trained on the Food-101 dataset using the Inception-V3 architecture with a Top-1 accuracy of 54.8%.
Experiment 2	A model was trained on the Food-101+ dataset using the Inception-V3 architecture with a Top-1 accuracy of 55.3%.
Experiment 3	A model was trained on the Food-101+ dataset using the Inception-V3 architecture with a Top-1 accuracy of 66.3% and a Top-5 accuracy of 85.96%.
Experiment 4	A model was trained on the Food-101+ dataset using the MobileNet architecture with a Top-1 accuracy of 50.2%
Experiment 5	A model was trained on a subset Food-101+ dataset using the Inception-V3 architecture with a Top-1 accuracy of 92.6% and a Top-5 accuracy of 100%.

4.5.3 Analysis

It would make sense the accuracy of our model would increase when the number of classes are reduced as the margin of error is decreased. The Top 5 accuracy of the model was very successful. There were only 10 images per class tested though (totalling at 130 images) so if the number of images increased, this accuracy would probably decrease slightly. If the code is run again, the same results cannot even be replicated for sure. On another run of this code a Top 5 accuracy of 96% was recorded. While this is still a very good accuracy, it does not compare to the first run of the script.

4.6 Conclusion

The focus of this chapter has been to train TensorFlow models using the Inception-V3 architecture under various constraints. These models have been trained using transfer learning. The highest accuracy for a model trained using the Food101+ dataset yielded a Top-1 accuracy of 66.6% and a Top-1 accuracy of 85.96%. Alternatively, a model was trained using a subset of the Food101+ dataset. 13 classes from Food101+ were used to benchmark the efficacy of using CNNs against other methods found in the literature review. This CNN reached a Top-1 accuracy of 92.6% and a Top-5 accuracy of 100%. This exceeded or was comparable to the other methods researched.

Chapter 5

Empirical Studies Part 2

The purpose of this chapter is to analyse the models trained previously to gain understanding of them and to gauge applicability to the problem statement.

5.1 Experiment 6: Sliding Window

5.1.1 Objective

In the previous experiments, the one-shot approach to food image classification has been looked at. That is, the model will give a prediction of the most likely food item in that image. This is a problem when there is a composite image i.e. there are multiple foods in an image, see Figure 5.2. There are a few options to combat this problem. Firstly, one could detect objects in the image, segment the image according to these objects and then run each segment through the model. A simple approach to this would be to segment the image into several sections and then run each section through the model. To follow the latter approach, a sliding window approach was used. This sliding window would move across the image and classify the segment of the image in the window. There are three options for window sliding shape as defined by a command line argument.

These three options for window shape are:

- Grid based window as per Figure 5.3.

Figure 5.1: Sliding Window Command

```
python sliding_window.py --image=~/image_dir --window_shape grid
```

- Row based window as per Figure 5.4.
- Column based window as per Figure 5.5.

5.1.2 Libraries

For this experiment, TensorFlow provided the classification of each segment while also helping with resizing along with NumPy. OpenCV was used to implement the sliding window as per (*Sliding Windows for Object Detection with Python and OpenCV* 2018).

5.1.3 Script

There were four main elements to the script. Firstly, extracting a window to be classified (Figure 5.8). Secondly, resizing the image to be compatible with the TensorFlow model (Figures 5.10 and 5.11). Thirdly, running the window through the TensorFlow model (Figure 5.12) and finally saving a new image with a coloured overlay of classifications (Figure 5.13). In regard to Figure 5.13, as seen in Figure 5.6, each square represents a window and each colour is for a different classification. Blue is for an apple, yellow for banana, green for grape, white for orange and black if an unexpected prediction is made.

5.1.4 Results

5.1.4.1 Grid based window

The grid-based window resulted in fifteen separate classifications. As seen in Figure 5.2, there are multiple fruits in the image. Of these fruits, our model is trained on four, apple, banana, orange and grapes. This method classified all four to Top-1 accuracy at least once each. Figure 5.7 represents this same result but only for the apples classified. It can be seen from this image that there is an apple present in every square that was classified as an apple. These

Figure 5.2: Bowl of fruit (Hy-Vee, 2018)

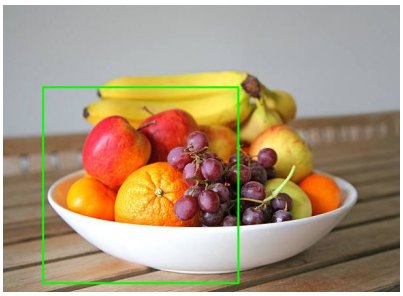


Figure 5.3: Grid based window

Figure 5.4: Row based window

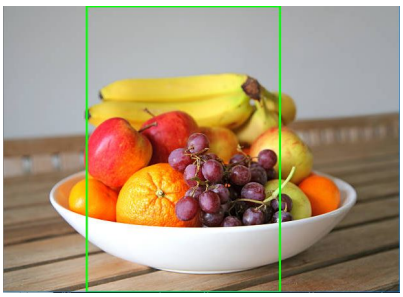


Figure 5.5: Column Based Window

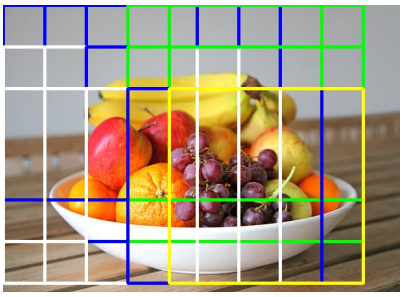


Figure 5.6: Fruit with Colour Overlay

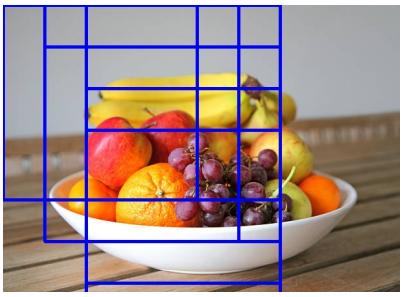


Figure 5.7: Fruit with Apple Overlay

Figure 5.8: Extracting Window from the Image

```
for (x, y, window) in sliding_window(resized, stepSize=64,
    windowSize=(winW, winH)):
    # ignore the window if it is not the correct size
    if window.shape[0] != winH or window.shape[1] != winW:
        continue
```

Figure 5.9: Moving the Window Across the Image

```
def sliding_window(image, stepSize, windowSize):
    # slide a window across the image
    for y in xrange(0, image.shape[0], stepSize):
        for x in xrange(0, image.shape[1], stepSize):
            # yield the window
            yield (x, y, image[y:y + windowSize[1], x:x + windowSize[0]])
```

Figure 5.10: Resizing the Window

```
window = cv2.resize(window, (299, 299))
```

Figure 5.11: Resizing the Image - adapted from (*How to Retrain Inception's Final Layer for New Categories* 2018)

```
resized_image = tf.reshape(image, [1, input_height, input_width, 3])
resized = tf.image.resize_area(resized_image, [input_height, input_width])
normalized = tf.divide(tf.subtract(resized, [input_mean]), [input_std])
```

Figure 5.12: Running the TensorFlow Model - adapted from (*How to Retrain Inception's Final Layer for New Categories* 2018)

```
with tf.Session() as sess:
    numpy_image = sess.run(normalized)

with tf.Session(graph=graph) as sess:
     results = sess.run(output_operation.outputs[0],
                        {input_operation.outputs[0]: numpy_image})
    probabilities = np.squeeze(results)
```

Figure 5.13: Save Image with Colour Overlay

```
cv2.rectangle(display_image, (x, y), (x + winW, y + winH),
    colour_dict.get(top1,
(0,0,0)), 4)
```

windows that classified apples in Figure 5.7 also classified bananas, grapes and oranges to Top-5 accuracy in expected windows. For this window method, all classifications were correct in the sense that the Top-1 prediction was located somewhere in the window. This method took 42.8 seconds to run.

Table 5.1: Grid Based Sliding Window Results

Food type	**No. of Top-1 Classifications**
Apple	5
Banana	1
Grape	4
Orange	5

5.1.4.2 Row based window

The row-based method resulted in four predictions as follows in Table 5.2. Out of these four predictions, only one classified a known fruit at Top-1 accuracy, an apple. An apple was also predicted to Top-5 accuracy in another instance. The runtime of this method was 16.1 seconds.

5.1.4.3 Column based window

The column-based window approach had five total predictions and ran for a total of 13 seconds. As seen in Table 5.3, two out of four known fruits were classified to a Top-1 accuracy with all other fruits predicted to Top-5 accuracy. Only one Top-1 prediction did not contain a correct fruit.

5.1.5 Analysis

These results raise some questions because while a banana was only predicted

Table 5.2: Row Based Sliding Window Results

Food type	**No. of Top-1 Classifications**
Apple	1
Banana	0
Grape	0
Orange	0
Other	3

Table 5.3: Column Based Sliding Window Results

Food type	**No. of Top-1 Classifications**
Apple	3
Banana	1
Grape	0
Orange	0
Other	1

to Top-1 accuracy once in grid based, once in column based and zero times in row-based approach, if the whole image is passed through the model, a banana is at the Top-1 accuracy. A possible reason for this could be that the overall shape of a banana is very distinctive but when sliding windows are used, the whole banana is not present in the window.

5.2 Experiment 7: Recursive Refinement

5.2.1 Objective

After the sliding window code was run on Figure 5.2 in 5.1, it was observed that a sliding window was predicting grapes correctly in regions that contained a bunch of grapes. Since it would make sense that the model would be able detect an individual grape, it was decided that recursive refinement would be executed on a window that contained a grape. Due to the model requiring a 299 x 299 image size, the window could only be refined once as very small segments could not be resized up to 299 x 299. A window of 70 x 70 size was used.

5.2.2 Script

As you would think with recursive refinement, a recursive function would be used, but this was unnecessary due to image size restrictions. Instead, a conditional for loop was added to the existing code to break a window down further i.e. perform a sliding window approach on a window. This can be seen in Figure 5.14.

Figure 5.14: Recursive Refinement Code

```
if top1 == "grape" and window_shape == "grid" and rr_grape:
            for (x_grape, y_grape, grape_window) in
                sliding_window(window_resized, stepSize=64, windowSize=(70,
                70)):
                #reshape to square
                grape_window_resized = cv2.resize(grape_window, (299, 299))
                top1_grape = subSample.classify(grape_window_resized,
                    window_shape)
                if top1_grape == "grape":
                    cv2.rectangle(display_image, (x_grape + x, y_grape + y),
                        (x_grape + x + 70, y_grape + y + 70),
                        colour_dict.get(top1, (0,0,0)), 4)
                    #cv2.imshow("Window", grape_window_resized)
                    cv2.waitKey(1)
time.sleep(0.025)
```

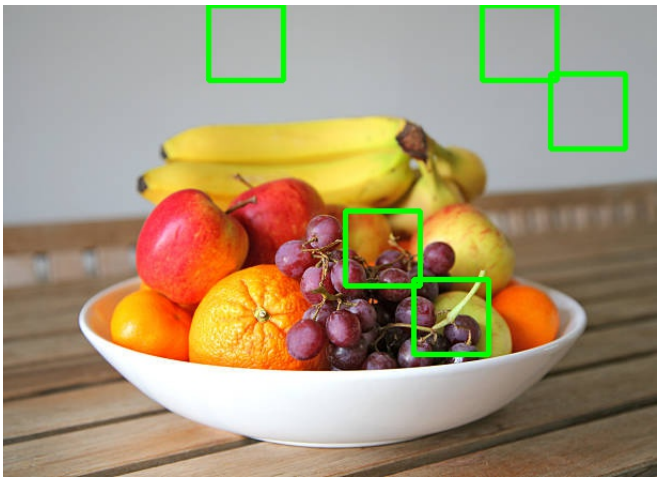


Figure 5.15: Recursive refinement 1

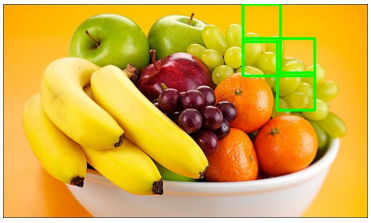


Figure 5.16: Recursive refinement 2 - sourced from http://www.travispta.org/

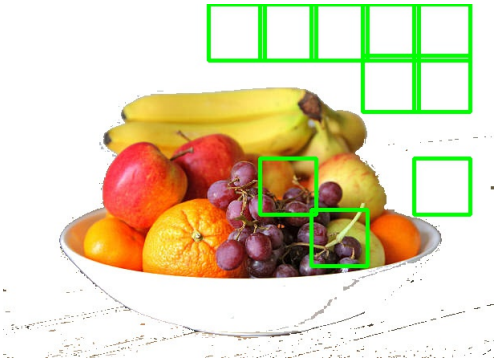


Figure 5.17: Recursive refinement 3

5.2.3 Results

Some very similar results were recorded on three separate images. Two new images are seen here which we will explore in future experiments. In all three images, while the script is calculating some expected predictions, grapes are being classified in locations that have nothing resembling a grape. These can be viewed in Figures 5.15, 5.16 and 5.17.

5.2.4 Analysis

The instances where false positives were recorded may indicate issues with the recursive refinement approach as every window must be classified and it is possible that this just happens to be a grape in some instances. Further analysis to why this may be occurring is outlined in the section 5.5.

5.3 Experiment 8: Impact of Background

5.3.1 Objective

As we can see in section 5.1, using sliding windows to classify many sections of an image, there were some cases where some unexpected predictions were made. Due to this, the decision was made to analyse the effect the background of the image on its classification. The sliding window code was then ran on a new image. This new image was the same fruit bowl as used previously but the background was filled in as white as per Figure 5.18.

Figure 5.18: Bowl of fruit with background removed

Table 5.4: Comparison of fruit image sliding window results with and without background

Food type	**Grid**	**Row**	**Column**	**White Grid**	**White Row**	**White Column**
Apple	5	1	3	10	0	4
Banana	1	0	1	0	0	0
Grape	4	0	0	1	0	1
Orange	5	0	0	3	0	0
Other	0	3	1	1	4	0

5.3.2 Results

5.3.2.1 Grid

For the grid-based sliding window approach, the results turned out to be less successful than with the background. In this experiment, fourteen out of fifteen Top-1 classifications were of an expected food type rather than fifteen out of fifteen with the background present. We expected the food types of apple, orange, grape and banana to appear in this image but while a banana was detected to a Top-5 accuracy on a few occasions it was never predicted to a Top-1 accuracy. The contrast between the image results can be seen in Table 5.4.

5.3.2.2 Row

The row-based sliding window again had worse results than its counterpart, with zero out of four correct classifications as opposed to one. In this case, an orange appeared at Top-5 accuracy once. The most common prediction was ice-cream which appeared at Top-1 accuracy in three out of four instances.

5.3.2.3 Column

In contrast to our previous two methods of sliding window, this method outperformed its counterpart with correct predictions of all five windows while before we only had four out of five. In this experiment, an apple was predicted four times and a grape once, with all correct fruits appearing to Top-5 accuracy.

5.3.3 Analysis

Surprisingly, removing the background of the image reduced the prediction accuracy. Many white foods were classified instead which makes sense due the impact of colour expected.

Table 5.5: Comparison of fruit bowl images

Food type	Grid	Row	Column	New Grid	New Row	New Column
Apple	5	1	3	4	1	0
Banana	1	0	1	5	0	5
Grape	4	0	0	2	1	0
Orange	5	0	0	0	0	0
Other	0	3	1	1	2	1

5.4 Experiment 9: Alternative Test Image

5.4.1 Objective

In our previous sliding window-oriented experiments, we had only used a single image. In order to see whether this image had biases unknown to us, another fruit bowl image had to be tested. This image was selected as fruit took up a larger portion of the image as seen in Figure 5.19.

Figure 5.19: Alternative Bowl of fruit

5.4.2 Results

5.4.2.1 Grid

The performance of this experiment was slightly worse than with the previously used image. When the grid based sliding window was executed on Figure 5.19,

fourteen out of fifteen predictions had an expected value. Out of the fourteen predictions orange was not predicted to Top-1 accuracy at all. This can be seen, in comparison to the previously used image, in Table 5.5.

5.4.2.2 Row

In the row based window for the new fruit image, the results were not very successful as has been the trend for most row based classification. Two out of four predictions had an expected value at Top-1 accuracy.

5.4.2.3 Column

The column based approach had a similar result to its counterpart in that only one of its predictions was unexpected. Although, due to the size of the new image, another column was created and thus has a better overall accuracy.

5.4.3 Analysis

A possible reason that an orange was not classified in any of these images is because in Figure 5.19, a mandarin is displayed.

5.5 Experiment 10: Analysing Results of Recursive Refinement Further

5.5.1 Objective

In Section 5.2, some false positives were obtained when attempting to run recursive refinement on an image with grapes in it. Originally it was meant to see if individual grapes could be recognised. On evaluation of the images used for training, it was found that bunches of grapes were used for training, not individual grapes. Therefore, the results expected where never feasibly going to be obtained. Even though this was the case, the script resulted in finding grapes in portions of the background, as seen in Figure 5.15. The purpose of

this experiment is to investigate why this is occurring. A new image of a fruit bowl (Figure 5.20), taken on a mobile phone was selected for this experiment. The image was used to run the script as defined in 5.2 but on two separate models. The model with tuned parameters, trained on 108 classes, with a final test accuracy of 66.3% and the model trained on 13 classes, with a final accuracy of 92.6%.

Figure 5.20: Fruit image taken on a mobile phone

5.5.2 Results

The outputs of the script can be seen in Figure 5.21 and Figure 5.22. The output using the model trained on 13 classes had less false positive predictions. As the script ran (on both models), the segments predicted as grapes were saved to disk and then were each classified using 'label_image.py'. The probabilities of the predictions were recorded with results as seen in Table 5.6.

5.5.3 Analysis

5.5.3.1 Comparing the two models

As we can see evidence of in Table 5.6, along with Figure 5.21 and Figure 5.22, the model trained on 13 classes predicted only four false positives while the

Table 5.6: Results of recursive refinement segment classifications using two models

Fruitbowl Segment	**13 class model**	**108 class model**
False Positive 1	grape 0.31476045 apple 0.25866747 orange 0.11228518 apple pie 0.10234257 chocolate cake 0.09062083	grape 0.31820437 panna cotta 0.091400646 macarons 0.08560496 roll 0.08188102 apple 0.03981942
False Positive 2	grape 0.29391274 apple 0.2371384 orange 0.16196558 chocolate cake 0.10039616 apple pie 0.096121624	panna cotta 0.21783036 grape 0.105718106 apple 0.087030284 macarons 0.07753955 orange 0.037592527
False Positive 3	grape 0.22402291 chocolate cake 0.16412543 apple 0.14523567 orange 0.12525927 apple pie 0.099948086	grape 0.2835378 panna cotta 0.088551655 macarons 0.06166248 orange 0.041896477 roll 0.040348493
False Positive 4	apple 0.37422368 grape 0.28592423 orange 0.10230055 chocolate cake 0.085367255 apple pie 0.053731006	grape 0.20476209 macarons 0.17529227 panna cotta 0.16139281 roll 0.04359601 orange 0.036991216
False Positive 5	N/A	grape 0.22574392 macarons 0.14840269 panna cotta 0.08670294 roll 0.08650105 orange 0.046900786
False Positive 6	N/A	grape 0.31369162 panna cotta 0.11833602 macarons 0.09010604 apple 0.057512555 roll 0.047116004

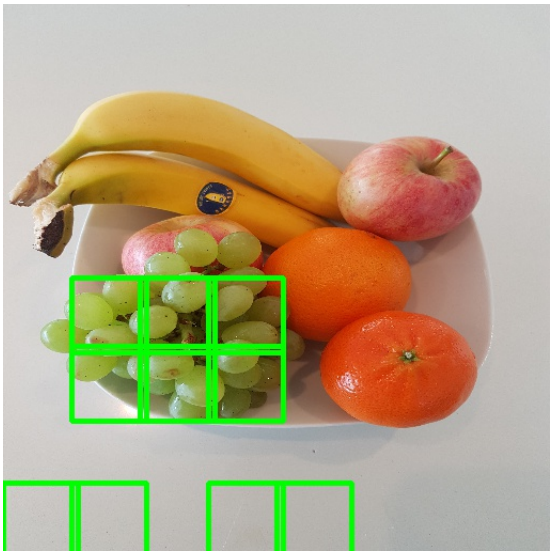


Figure 5.21: Image after sliding window - 13 classes

larger model predicted 6. This is most likely due to the larger model not being able to separate grapes as well as the smaller model. It is worth mentioning that the grape class has the lowest number of training images of all the other classes.

5.5.3.2 Analysing Probabilities

The probabilities of predictions for grapes in these false positive classifications are all low, in both models. In contrast, some of the correctly classified segments were run through the model and the results all had probabilities in the high nineties. When these background segments are run through the model, some prediction has to be made and none of these false predictions have a probability of over 0.4. The average probability of these segments being grapes is in fact 0.257 (rounded to three decimal places). A counter measure to this problem could be to disregard any predictions with a probability under a certain threshold, perhaps 0.4.

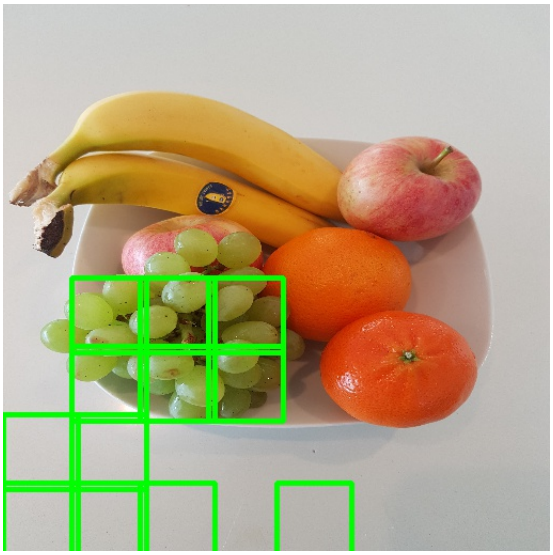


Figure 5.22: Image after sliding window - 108 classes

5.6 Experiment 11: Scaling Down Images

5.6.1 Objective

In this experiment, it was decided to look into how scaling down the size of the image would influence its classification. Three images were scaled down to 10% of their size for this purpose, Figure 5.23, Figure 5.24 and Figure 5.25. Each of these images were run through the model before and after image resizing and the predictions were recorded.

Figure 5.23: Banana Image Pre-Resolution

Figure 5.24: Apple Pie - sourced from https://www.bettycrocker.com/

Figure 5.25: Pizza Image Pre-Resolution

5.6.2 Results

The results of the images before and after were somewhat contrasting as seen in Table 5.7. The 'Top 1' notation in the table suggests that the food was predicted as the number one prediction. The decimal value is the probability of that food type being the correct classification.

Table 5.7: Results of Down Scaled Images

Food Image	**Pre-Scaled Image**	**Scaled Image**
Banana	Top 1 : 0.9971	Top 1 : 0.9995
Apple Pie	Top 1 : 0.7986	Top 5 : 0.0836
Pizza	Top 1 : 0.9753	Top 1 : 0.9705

5.6.3 Analysis

As we can see in Table 5.7, the banana (Figure 5.23) has a very similar prediction and probability. This would indicate that the classifier does not need clear quality images for bananas but maybe shape, texture and colour are important factors.

The apple pie (Figure 5.24) has drastically different results in Table 5.7. Originally the apple pie was correctly predicted with a probability of 0.7986 but after downscaling, the apple pie was the fifth food predicted with a likelihood of 0.0836.

The image of the pizza (Figure 5.25), had very similar results before and after resizing. This might indicate that pizza is determined more by shape, texture and colour that fine quality.

5.7 Experiment 12: Effect of Colour

5.7.1 Objective

As seen in the last experiment, it is possible that colour plays a significant part in the overall classification of some food types, along with shape and texture. Due to this, what would happen if colour was removed from the image? The three images, Figure 5.23, Figure 5.24 and Figure 5.25, were all converted to greyscale to test this and the resulting images can be seen in Figure 5.26, Figure 5.27 and Figure 5.28. Each of these images were ran through the model before and after converting to greyscale and the results were recorded.

5.7.2 Results

The results for this experiment can be seen in Table 5.8. The 'Top 1' notation in the table suggests that the food was predicted as the number one prediction. The decimal value is the probability of that food type being the correct classification.

Figure 5.26: Banana Image Greyscale

Figure 5.27: Apple Pie Image Greyscale

5.7.3 Analysis

5.7.3.1 Colour

In regard to the images of the banana (Figure 5.23 and Figure 5.26), there is very little difference in classification. They were both classified to a Top 1 accuracy and their probabilities differing by only 0.0015 with the grey scale image being of a higher probability. This would indicate that colour is not an important factor for classifying bananas and the coloured image may even have noise. This can be seen in Table 5.8.

Table 5.8: Effect of Colour

Food Image	**Pre-Scaled Image**	**Greyscale**
Banana	Top 1 : 0.9971	Top 1 : 0.9986
Apple Pie	Top 1 : 0.7986	Top 1 : 0.3462
Pizza	Top 1 : 0.9753	Top 2 : 0.1299

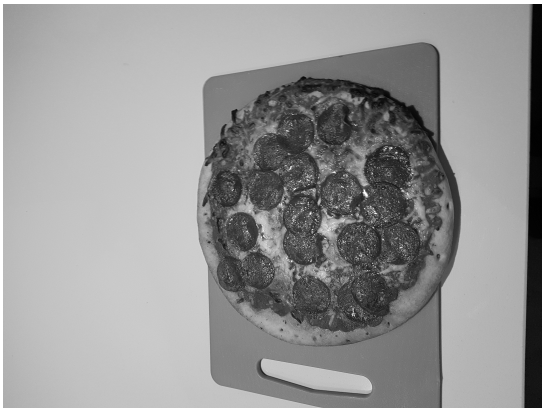


Figure 5.28: Pizza Image Greyscale

The apple pie images (Figure 5.24 and Figure 5.27) were also both classified correctly but with a large gap in the likelihood of that classification being correct. As per Table 5.8, the coloured image had a probability of 0.7986 as opposed to one of 0.3462. This would indicate that while colour is important, there is enough unique data from the rest of the image to result in correct classification.

Finally, the pizza images (Figure 5.25 and Figure 5.28) were the most contrasting in their results. While the original image was classified to Top 1 accuracy with a likelihood of 0.9753, the coloured image was classified to Top 2 accuracy with a probability of 0.1299. From this we can deduce that colour is vital for the classification of pizza.

Since this experiment only compared three images, we cannot say for sure if this analysis is biased or not.

5.7.3.2 Colour vs Scale

In the last experiment, it was seen that the pizza and banana images (Figure 5.25 and Figure 5.23) were not really affected by image quality. While this seems to be true, the image of apple pie, Figure 5.24, was greatly influenced by only being classified to a Top 5 accuracy when down scaled.

For the greyscale images, bananas were not effected greatly and neither was an apple pie but a pizza was greatly influenced.

From this, it can be said that the prominent unique features for each of these food types are different. The banana (Figure 5.23) may have focus on shape and texture, the pie (Figure 5.24) on image quality and some influence of colour while the pizza (Figure 5.25) may have a focus on shape, texture and colour.

5.8 Conclusion

This chapter has analysed the models trained in Chapter 4. Emphasis was placed on dealing with composite images through a sliding window approach. Once this had been completed, analysis into the results of the sliding window approach was carried out by using multiple test images and removing the background from the image. Recursive refinement on the sliding window approach for images containing grapes was carried out and the results were analysed. Finally, the effect of colour and image size on the classifier was measured. It was found that different food types required different levels of features for classification.

Chapter 6

Prototype Application

In order to demonstrate practical use of the models trained in Chapter 4, a lightweight proof of concept prototype application was developed. This application was developed for a smartphone running on the Android operating system in the Java programming language. A backend service was created to receive an image from the smartphone application, process the image using the CNN previously developed and return a response to the application in the form of a prediction.

6.1 Requirements

Lightweight requirements elicitation was carried out for this FYP. Since prototyping itself is an elicitation technique (Tiwari, Rathore, and Gupta, 2012), the requirements for this prototype application did not need to be extensive. Myself and my supervisor acted as stakeholders for this project as users of the system. Two methods of elicitation were used for this project sourced from (Tiwari, Rathore, and Gupta, 2012). These elicitation techniques were data gathering from an existing system and brain storming.

In a previous FYP, a mobile application was developed for nutritional assessment. The use of this application provided the opportunity to elicit requirements based on data gathered from this system. Brainstorming also took place between the stakeholders for requirements elicitation.

6.1.1 Functional Requirements

The functional requirements for this application can be seen in Table 6.1.

ID	**Title**	**Description**	Dependencies
R1	Open application from Android phone	A user can open the application from a list of applications on their Android phone.	
R2	Choose to take a photo	A user can choose to take a photo of a food item.	R1
R3	Take a photo	A user can take a photo using the application.	R1, R2
R4	Confirm the photo	A user can confirm the photo captured is sufficient.	R1, R2, R3
R5	Retry photo capture	A user can retry the capturing of an image.	R1, R2, R3
R6	Send the image to be classified	A user can send an image of a food item to be classified.	R1, R2, R3, R4
R7	Cancel sending an image	A user can cancel sending an image.	R1, R2, R3, R4
R8	Classify the food image	The system can classify an image as a food type.	R1, R2, R3, R4, R6
R9	View the food image classification	A user can view the classification made of the image.	R1, R2, R3, R4, R6, R8
R10	View the calorie information of the food classification	A user can view the calorie information of the classified food type.	R1, R2, R3, R4, R6, R8
R11	Change the food classification data	A user can change the food classification data if they are unhappy with it.	R1, R2, R3, R4, R6, R8, R9
R12	Change the calorie information	A user can change the calorie information if they are unhappy with it.	R1, R2, R3, R4, R6, R8, 10

R13	Submit the food classification and calorie data for logging	A user can submit the data calculated such as classification data and calorie information to be logged.	R1, R2, R3, R4, R6, R8, R9, R10
R14	Save food log data	The system can save information for logging.	R1, R2, R3, R4, R6, R8, R9, R10, R13
R15	Choose to classify an image from the phone's storage	A user can choose to select an image from the phone's storage to be classified.	R1
R16	Choose an image from phone's storage	A user can choose an image from the phone's storage to classify.	R1, R14
R17	Choose to view food logs	A user can choose to view the food logs saved on their device.	R1
R18	View food logs by day	A user can view the food logs saved on the device by day.	R1, R2, R3, R4, R6, R8, R9, R10, R13, R17
R19	View food logs by week	A user can view the food logs saved on the device by week.	R1, R2, R3, R4, R6, R8, R9, R10, R13, R17
R20	View food logs by month	A user can view the food logs saved on the device by month.	R1, R2, R3, R4, R6, R8, R9, R10, R13, R17
R21	View total calories recorded per day	A user can view the total calorie count recorded per day.	R1, R2, R3, R4, R6, R8, R9, R10, R13, R17

R22	View total calories recorded per week	A user can view the total calorie count recorded per week.	R1, R2, R3, R4, R6, R8, R9, R10, R13, R17
R23	View total calories recorded per month	A user can view the total calorie count recorded per month.	R1, R2, R3, R4, R6, R8, R9, R10, R13, R17
R24	Delete a food log	A user can delete an individual food log.	R1, R2, R3, R4, R6, R8, R9, R10, R13, R17, R18

Table 6.1: Functional Requirements

6.1.2 Non-Functional Requirements

The nature of a prototype application does not call for non-functional requirements. Despite this, some non-functional requirements were kept in mind during the development of this application such as:

- Performance - for example, how long it takes for an image to be classified.
- Extensibility - is it easy to extend the functionality of the system.
- Maintainability - how easy it is to maintain the system.
- Usability - how easy it is to use the system.

6.2 Design

The design process for this prototype application consisted of user interface design, android application system design and the activity of choosing resources for a backend host.

6.2.1 User Interface

The user interface mock ups were created using (*fluid* 2018). The aim of this design was to provide a simple, easy to use interface so that a user could use the application with minimal effort. The main benefit of using computer vision for this type of application is to reduce the effort of the user or dietician keeping track of food intake. Therefore, this application had to be very quick and easy to use. The application is called NutriLog.

Three main activities were needed for this application. The first activity, as seen in Figure 6.1, would consist of options to either take an image of food or to view the food logs of the user. If the user decides to take an image of food, they will be brought to the second activity which can be seen in Figure 6.2. The user is required to take a picture before reaching the activity in Figure 6.2. Once the user presses the send button the image is classified and the content of the activity changes to display the classification and calorie count as in Figure 6.3. Alternatively, the user can cancel the process and return to the first activity. The user would then submit the food classification for logging and automatically return to the first activity. The final activity is displayed when a user views their food logs. This activity has the ability to view a list of the food logs taken by the user by day, week or month. The calorie count of the selected time frame is to be displayed on this activity also.

6.2.2 System Architecture

6.2.2.1 Architectural Patterns

The Model-View-Presenter (MVP) architectural pattern was adapted for this application (*Model-View-Presenter: Android guidelines* 2017). This pattern is

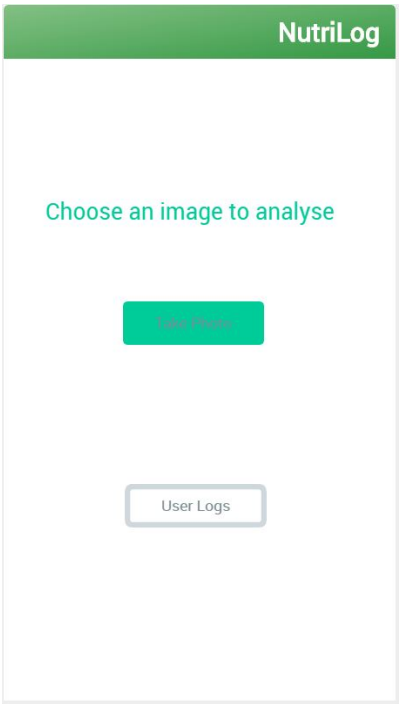


Figure 6.1: Landing Activity

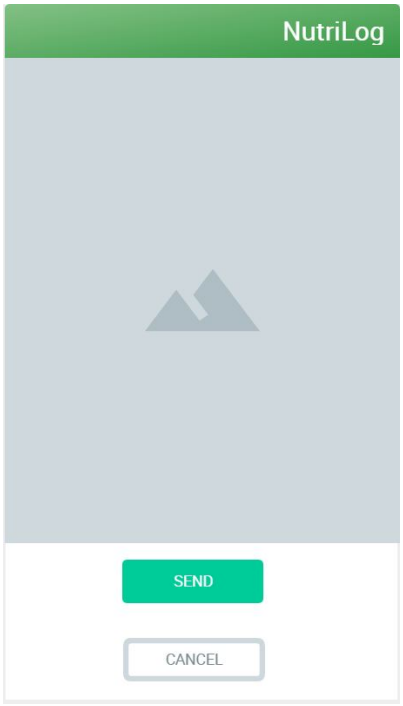


Figure 6.2: Image Submission Activity

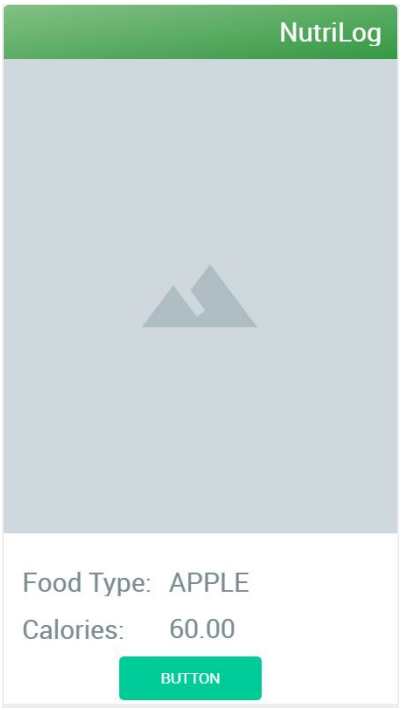


Figure 6.3: Classification Activity

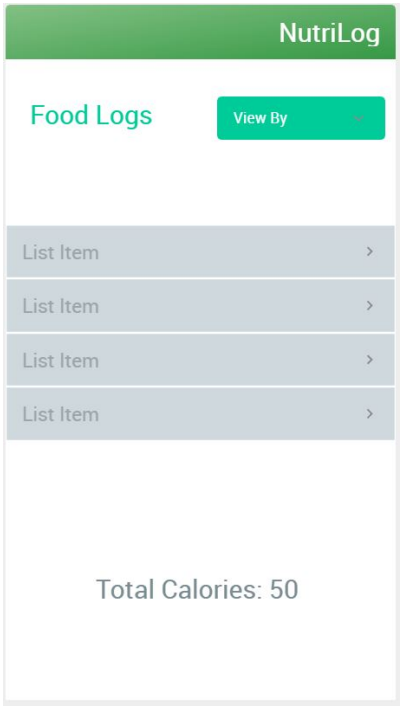


Figure 6.4: Food Logs Activity

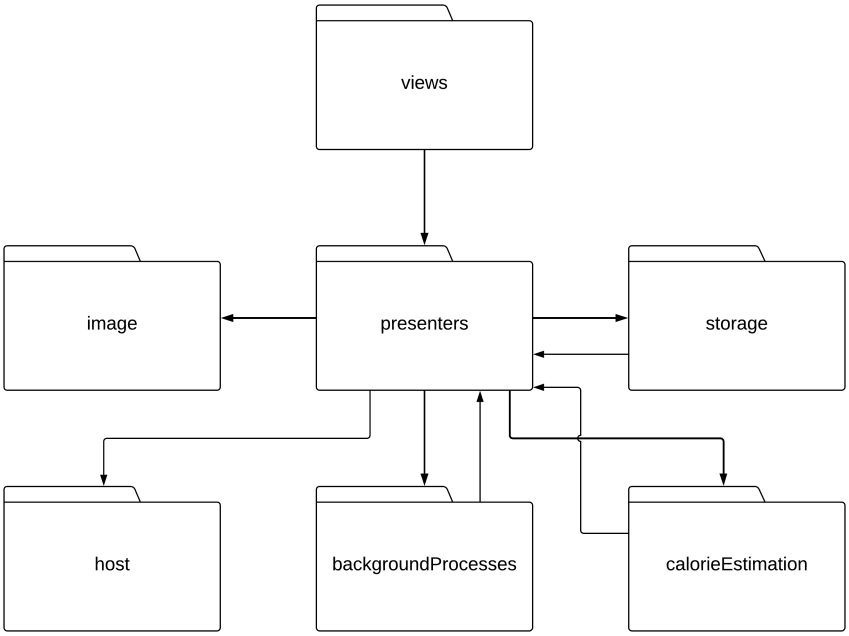


Figure 6.5: Package Diagram

very similar to the Model-View-Controller (MVC) architecture which is quite popular in the software development industry. The main difference between MVC and MVP is that whereas in MVC the controller is responsible for which view is used, in MPV the presenter is called through the view. This is due to the architecture of Android applications in general, where the view takes primary control. In the view classes, there should be no logic whatsoever in an MVP architecture and all logic should be called in the presenters. There is also a one to one dependency between views and presenters.

As illustrated in Figure 6.5, the views in the application have a dependency only on their corresponding presenter. The presenters contain the business logic of the application and have a dependency on all other packages. The packages of backgroundProcesses, calorieEstimation and storage have a dependency on the presenters package for callback purposes and also database creation in Android requires context of the activity information.

6.2.2.2 Marchitecture

In Figure 6.6, a marchitecture diagram can be seen. This diagram represents the architecture of the system. NutriLog is an Android application that sends an image to an AWS (Amazon Web Services) instance using the API OkHttp. The AWS instance is running a Python FLASK application that classifies the image using a TensorFlow model and sends a response back to NutriLog. The Nutritionix API is used to collect nutritional information of this classification.

The technologies used for this application were used for the following reasons:

AWS

Past experience was a large factor when it came to choosing a cloud provider to host the backend instance for this prototype application. AWS instances are very easy to set up once the process has been carried out a few times and this was a contributing factor to why AWS was chosen. AWS also has a free tier which provides all the necessary server space to host the backend service for this application.

OkHttp

OkHttp seemed to be popular for Android applications as seen on websites

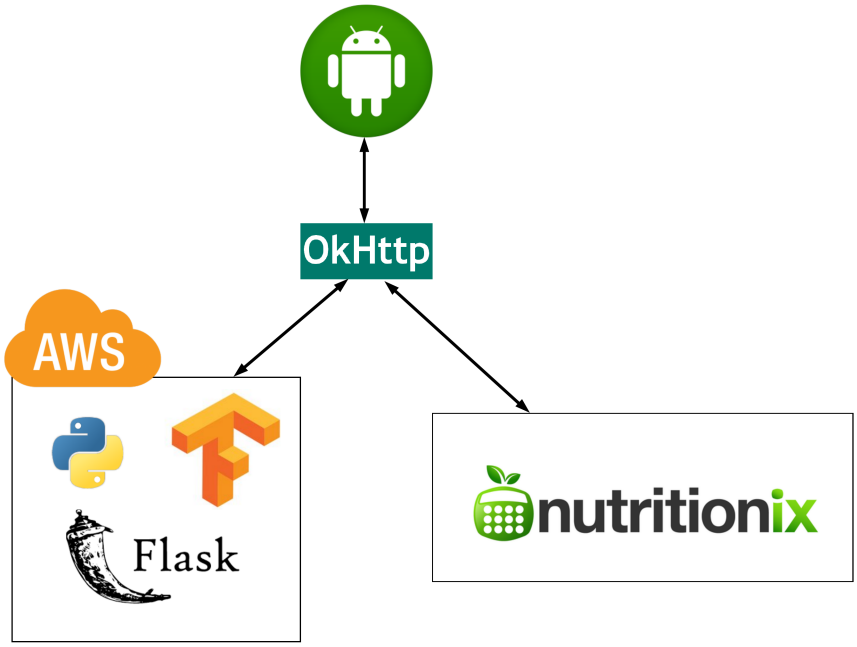


Figure 6.6: Marchitecture Diagram

such as Stack Overflow. OkHttp also has extensive documentation and is a very simple API to import into Android. The learning curve for OkHttp is quite small and it doesn't require much code to carry out a simple task like sending a HTTP POST request to a backend instance.

FLASK

Python FLASK was used for the backend service in this FYP due to experience and its simplicity.

TensorFlow

TensorFlow was chosen as the library to create neural networks for this FYP mostly because of its reputation. TensorFlow has been used by many researches in the computer vision industry. TensorFlow also has extensive documentation and a wide array of resources and tutorials.

Nutritionix

Nutrionix was the best nutritional information API that was also free to use and this is why it was chosen.

6.3 Android Development

The prototype application developed for this FYP was created for use on an Android device.

6.3.1 Activity Lifecycle

An activity in an Android application represents a single UI screen. As a user navigates an application, the activities within the application go through different stages of their lifecycle (*Developer Guides* 2018). The stages in an activities lifecycle can be seen in Figure 6.7.

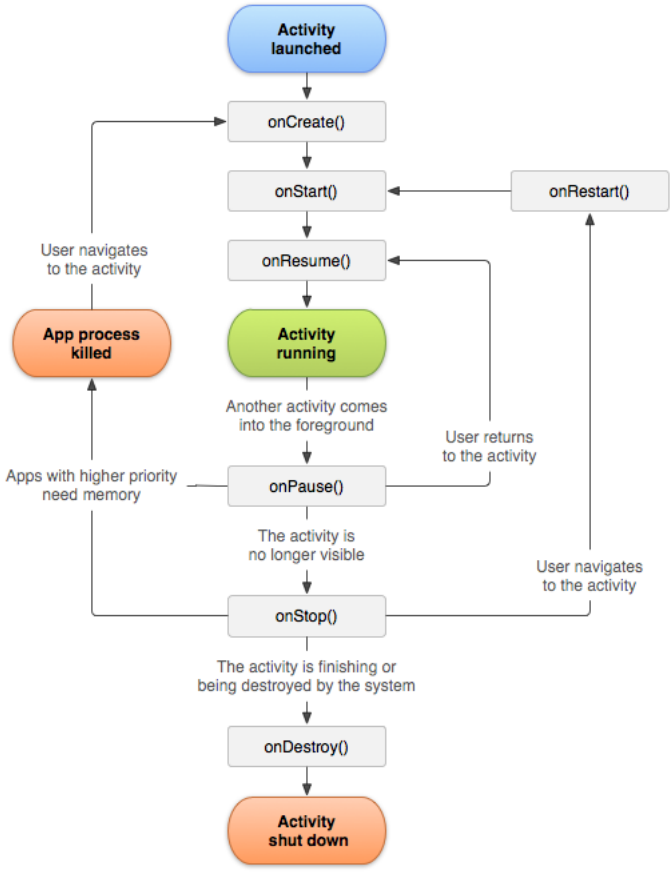


Figure 6.7: Android Activity Lifecycle (*Developer Guides* 2018)

Android provides call back mechanisms for each stage of the lifecycle. These call backs are:

- onCreate(): This callback is executed when the activity is first created and must be implemented.
- onStart(): This callback is triggered when the activity is in the Started state. This callback executes the code that prepares the UI for use.
- onResume(): This callback is triggered when the activity is resumed. This could occur if the user temporarily leaves the activity and returns.
- onPause(): This callback is invoked when a user is leaving an activity to be returned to later.
- onStop(): This callback is executed when the activity is finished running or is no longer accessible by the user.
- onDestroy(): The onDestroy() callback is invoked before the activity is destroyed and it is the final callback executed.

6.3.2 Android Studio

Android Studio is an IDE that is very common for the development of Android applications. It is developed by JetBrains who also created the InteliJ IDE and there are many similarities between the two. Android Studio provides a nice interface for the creation of the user interface of applications with a drag and drop mechanism.

6.3.3 Languages

There are two main languages used for Android development in the Android Studio IDE which are Java and Kotlin. Kotlin is a language developed by JetBrains with the tag line of "Statically typed programming language for modern multi-platform applications" (JetBrains, 2018). It is defined as being safe, tool-friendly, inter-operable and concise by the reduction of boilerplate (repetitive) code.

6.3.4 Key Concepts

In the development of Android applications an intent can be used as an ab-

straction to execute operations. Intents can launch activities, communicate with services running in the background and can also initiate broadcast receivers (allows use of system resources).

An AsyncTask in an abstract class in Android development. Its enables the performance of background operations and to call the UI thread with ease. They are aimed towards short operations which take a few seconds at most.

In Android development, an onClickListener is used to define actions for when a view is clicked. It is an interface that is implemented as a callback for when views are clicked.

6.4 Implementation

Outlined below are some coding fragments categorised under: Design Patterns, Asynchronous Tasks, Presenters, Views and Interesting Coding Fragments.

6.4.1 Design Patterns

Various design patterns were used in the implementation of this application such as the Factory, the Builder and the Singleton which are outlined below.

The Factory design pattern was used to retrieve a DAO (Data Access Object) as seen in Figure 6.8. This was used so that if at a future time the storage type of the application is to be changed, the developer would only have to change one instance of the codebase and return a new implementation of the DAO interface in the method getDAO().

Figure 6.8: DAO Factory Class

```
1 public class DAOFactory {
2
3     public DAO getDAO(Context context){
4         return SqlLiteDAO.getInstance(context);
5     }
6 }
```

A Host builder was used to create a Host object. This builder is a static inner class in the Host class file as documented in Figure 6.9

A Singleton instance of the DAO implementation was used in this application. This was mainly due to the best practice of keeping database access objects as singleton instances. This class can be seen in Figure 6.10.

6.4.2 Asynchronous Tasks

Asynchronous Tasks in Android are used to run tasks in the background or are sometimes used to simply distribute processes off the main thread. The AsyncTask class must be extended on creation of the background processes. An AsyncTask was used to send the food image to the backend host as in Figure 6.11.

6.4.3 Presenters

In the MPV architecture as described earlier, the presenter for each activity contains all the logic for that view. The presenter in Figure 6.12 was tasked with creating intents to new activities (views).

6.4.4 Views

In the MPV architecture that this application uses, the views are responsible for interacting directly with the user interface. The view for the MainActivity in Figure 6.13 responds to button clicks and calls to its presenter to carry out the logic of the required task.

6.4.5 Storage

SQLite was used for storing data in this application and the class that handled creation, input and output to this database implemented the DAO interface. An interface for storage devices was created so that each implementation would have the ability to add food logs, retrieve food logs, delete food logs and remove the data store completely. This is documented in Figure 6.14.

The method in Figure 6.15 was used to execute a query and return a List of

FoodLog objects.

6.4.6 Interesting Coding Fragments

Some interesting coding fragments are outlined below which merit inclusion in this report.

The method in Figure 6.16 was used to encode the image taken on the phone to a string. This was used to send the image to the backend with minimal latency. The bitmap size was also reduced as in line 8 to decrease response time also.

A map of strings to runnables was used to query the database for food logs as documented in Figure 6.17. This was used as the select query for the database had to dynamically change depending on how the food logs were being viewed, by day, week, or month.

6.4.7 User Interface

User interface implementation as in Figures 6.18, 6.19, 6.20, 6.21, 6.22, 6.23, 6.24 and 6.25.

Figure 6.9: HostBuilder Class

```
public static class HostBuilder {
    private final String ipv4;
    private String dns;
    private int port;
    private String route;

    public HostBuilder(String ipv4) {
        this.ipv4 = ipv4;
    }

    public HostBuilder withDns(String dns) {
        this.dns = dns;
        return this;
    }

    public HostBuilder withPort(int port) {
        this.port = port;
        return this;
    }

    public HostBuilder withRoute(String route) {
        this.route = route;
        return this;
    }

    public Host build() {
        return new Host(this);
    }

}
```

Figure 6.10: Singleton DAO Object

```
public static DAO getInstance(Context context) {
    if(instance == null) {
        instance = new SqlLiteDAO(context);
    }
    return instance;
}
```

Figure 6.11: UploadImage Class

```
@Override
protected String doInBackground(Void... params) {
    String result = "";
    OkHttpClient client = new OkHttpClient();
    String imageToSend = image;
    RequestBody requestBody = new
       MultipartBody.Builder()
            .setType(MultipartBody.FORM)
            .addFormDataPart("image", imageToSend)
            .build();

    Request request = new
       Request.Builder().url(host.getUrl())
            .post(requestBody).build();

    Response response = null;
    try {
        response = client.newCall(request).execute();
        result = response.body().string();
        response.body().close();
    } catch (IOException e) {
        e.printStackTrace();
    }

    return result;
}
```

Figure 6.12: MainPresenter Class

```
1  public class MainPresenter {
2
3      private Intent intent;
4      private Context context;
5
6      public MainPresenter(Context context) {
7          this.context=context;
8      }
9
10     public void takePhoto() {
11         intent = new Intent(context,
               CaptureImageActivity.class);
12         context.startActivity(intent);
13     }
14
15     public void userLogs() {
16         intent = new Intent(context,
               FoodLogsActivity.class);
17         context.startActivity(intent);
18     }
19
20 }
```

Figure 6.13: MainActivity Class

```
public class MainActivity extends AppCompatActivity {

    private MainPresenter mainPresenter;

    @Override
    protected void onCreate(Bundle
       savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mainPresenter = new MainPresenter(this);
    }

    public void onClickTakePhoto(View view) {
        mainPresenter.takePhoto();
    }

    public void onClickUserLogs(View view) {
        mainPresenter.userLogs();
    }
}
```

Figure 6.14: DAO Interface

```
public interface DAO {
    void addFoodLog(FoodLog foodLog);
    List<FoodLog> getLogsByDay(Date date);
    List<FoodLog> getLogsByWeek(Date date);
    List<FoodLog> getLogsByMonth(Date date);
    void deleteFoodLogs(List<FoodLog> foodLogs);
    void deleteDb();
}
```

Figure 6.15: Get Food Logs Per Date

```
@Override
private List<FoodLog> selectQuery(String query) {
        foodLogs = new ArrayList<>();
        Cursor resultSet = database.rawQuery(query,
           null);
        while(resultSet.moveToNext()) {
            try {
                foodLogs.add(new FoodLogImpl
                .FoodLogBuilder(resultSet.getString(1))
                .withId(resultSet.getInt(0))
                .withCalories(resultSet.getDouble(2))
                .withTimestamp(dateFormat
                    .parse(resultSet.getString(3)))
                .build());
            } catch (ParseException e) {
                e.printStackTrace();
            }
        }
    return foodLogs;
}
```

Figure 6.16: Encode Bitmap to base64 String

```
public String toBase64() {
    BitmapFactory.Options options = new
       BitmapFactory.Options();
    options.inSampleSize = 8;

    Bitmap imageBitmap =
       BitmapFactory.decodeFile(image.getAbsolutePath(),
       options);

    //resize image for faster upload to server
    Bitmap.createScaledBitmap(imageBitmap, 300, 400,
       false);

    ByteArrayOutputStream byteArrayOutputStream =
       new ByteArrayOutputStream();
    imageBitmap.compress(Bitmap.CompressFormat.PNG,
       100, byteArrayOutputStream);
    byte[] byteArray =
       byteArrayOutputStream.toByteArray();

    return Base64.encodeToString(byteArray,
       Base64.DEFAULT);
}
```

Figure 6.17: Map of Runnables

```
listViewOptions = new HashMap<>();
listViewOptions.put("Day", () -> getLogsByDay());
listViewOptions.put("Week",() -> getLogsByWeek());
listViewOptions.put("Month", () -> getLogsByMonth());
```

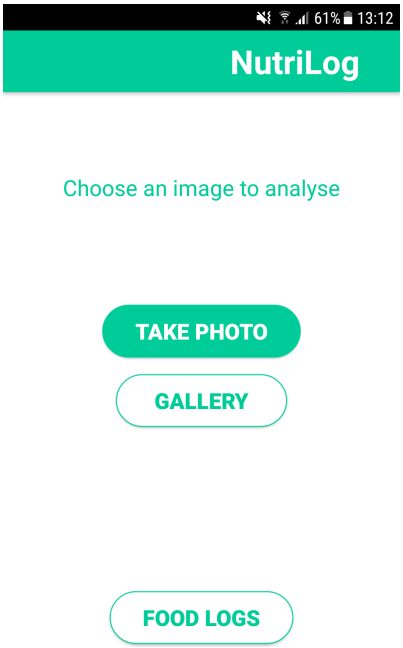


Figure 6.18: Landing Activity

Figure 6.19: Image Capture Activity

Figure 6.20: Image Send Activity

Figure 6.21: Image Submit Activity

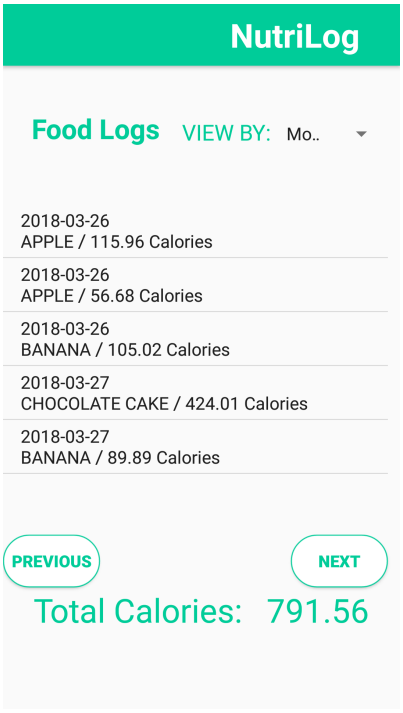


Figure 6.22: FoodLog Month Activity

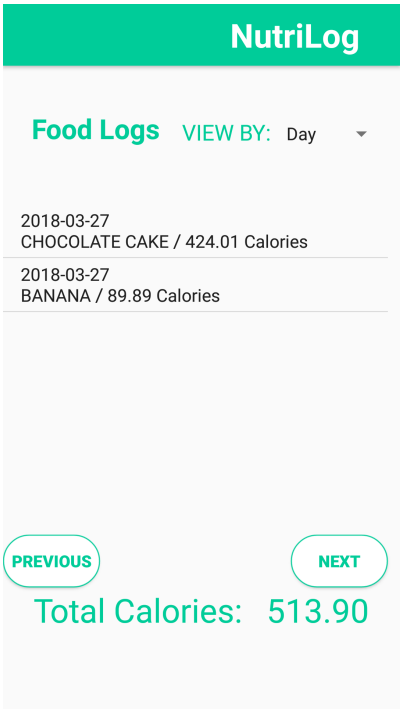


Figure 6.23: FoodLog Day Activity

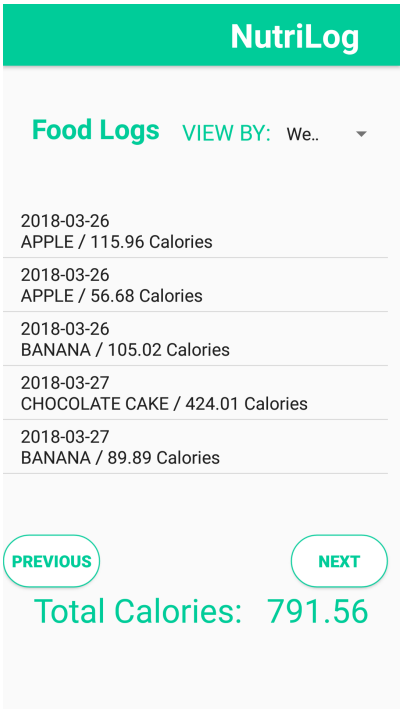


Figure 6.24: FoodLog Week Activity

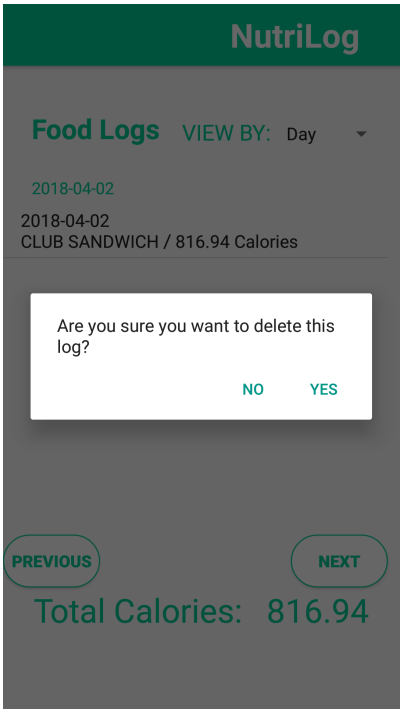


Figure 6.25: Food Log Deletion

6.4.8 Technologies

6.4.8.1 GitHub

GitHub was used as the version control backup system for this project. Commit history can be seen in Figure 6.26

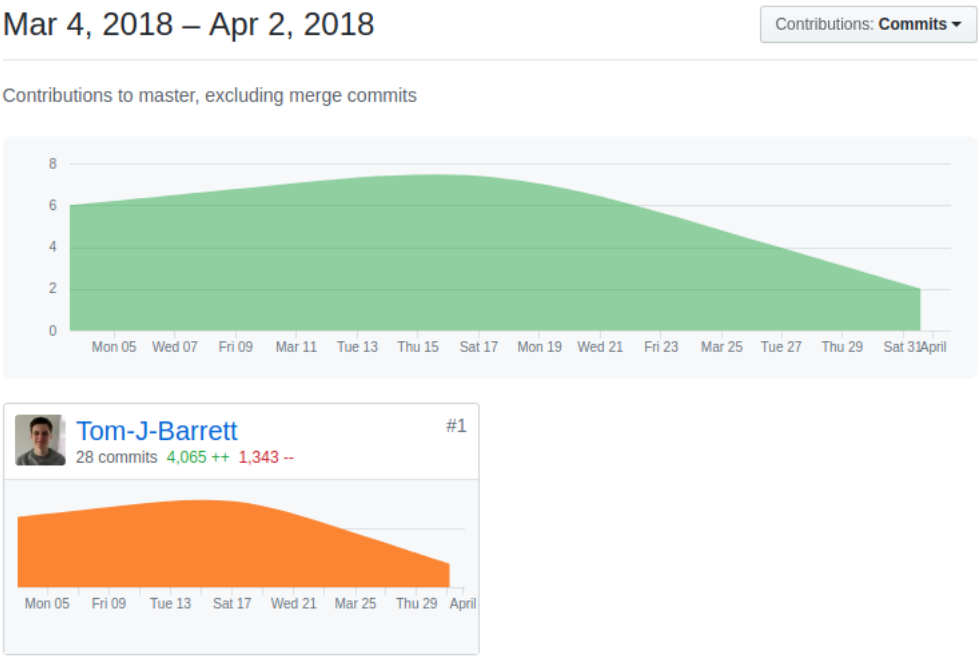


Figure 6.26: GitHub Contributions to Master

6.4.8.2 JSON

JSON (JavaScript Object Notation) is a text format used to store data (*Introducing JSON* 2018). It is both easy for machines to parse and for humans to read and write. It is a data-interchange format as it is language independent. JSON data structures are built on key pairs values and are normally parsed into arrays, lists or vectors.

6.4.8.3 OkHttp

OkHttp is a HTTP client created specifically for Java and Android applications (Square, 2018). It is a very efficient HTTP client with inbuilt defense against

troublesome networks (recovers from common problems silently). Some of the efficiency components are as follows:

- HTTP/2 support.
- Reduced request latency due to connection pooling.
- Caching of responses.

6.4.8.4 Nutritionix API

The Nutritionix API is an API used to collect nutritional information (Nutrionix, 2018). This project used this API to collect calorie information on a food item. The free 'Hacker' account can support up to 10 active users.

6.4.9 Software Quality Attributes

Two main software quality attributes were focused on during the implementation of this prototype application. These were extensibility and maintainability.

Extensibility defines how easy it is to extend the functionality of the system. There were certain places where the system developed is highly extensible. A factory class was used to get the data access object that was used to store information in the application. This factory class has a factory method that returns a DAO object which is an interface. If the developer of the system needs to create an alternative way to store data in the application, they can create an object that implements the DAO interface and replace one line of code in the factory class. A factory class was also used to create a Host object with the same rationale.

Maintainability defines how easy it is to maintain a system. The system is maintainable for many reasons. These reasons include low coupling, high cohesion and readability.

6.5 Testing

JUnit tests were used to test this prototype application. JUnit is a unit testing framework. Mockito is a framework for JUnit that was also utilised. Mock-

ito makes mocks of objects for testing. This is very useful to test database instances as you can mock an SQLite Database object.

6.5.1 Unit Tests

Evidence of some unit tests written for this prototype application can be seen in Figures 6.28, 6.29 and 6.30. Figure 6.27 also displays the automated tests that were run from Android Studio.

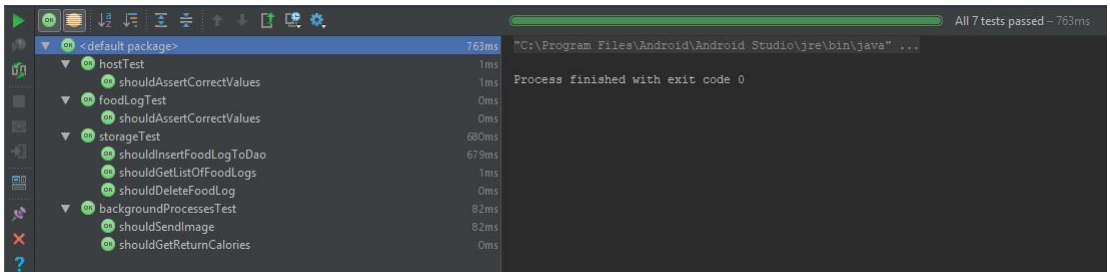


Figure 6.27: Automated Unit Tests

Figure 6.28: Test to ensure FoodLog object stores information correctly

```
@Before
public void setUp() {
    foodLog = new FoodLogImpl(0, "banana", 59.03,
       new Date());
}

@Test
public void shouldAssertCorrectValues() {
    assert food.equals(foodLog.getFood());
    assert calories == foodLog.getCalories();
    assert timestamp.equals(foodLog.getTimestamp());
    assert id == foodLog.getId();
}
```

Figure 6.29: Test to ensure Host object stores information correctly

```
@Before
    public void setUp() {
        host = new Host.HostBuilder("52.214.205.157")
                .withDns("ec2-52-214-205-157
                .eu-west-1.compute.amazonaws.com")
                .withPort(5000)
                .withRoute("/classifyImage/")
                .build();
    }

@Test
public void shouldAssertCorrectValues() {
    assert this.ip.equals(host.getIpv4());
    assert this.dns.equals(host.getDns());
    assert this.port == host.getPort();
    assert this.route.equals(host.getRoute());
    assert this.urlString.equals(host.getUrl());
}
```

Figure 6.30: Test to ensure DAO object stores and removes data correctly

```
@Before
public void setup() {
    dao = mock(SqlLiteDAO.class);
    date = new Date();
    foodLog = new FoodLogImpl(0, "test", 0.0, date);
}

@Test
public void shouldGetListOfFoodLogs() {
    foodLogs = dao.getLogsByDay(date);
    foodLogs = dao.getLogsByWeek(date);
    foodLogs = dao.getLogsByMonth(date);
}

@Test
public void shouldInsertFoodLogToDao() {
    dao.addFoodLog(foodLog);
    setTestLog();
    assert foodLogs.contains(testLog);
}

@Test
public void shouldDeleteFoodLog() {
    List<FoodLog> foodLogs = new ArrayList<>();
    foodLogs.add(testLog);
    dao.deleteFoodLogs(foodLogs);
    setTestLog();
    assert testLog.equals(null);
}

private void setTestLog() {
    foodLogs = dao.getLogsByDay(date);
    foodLogs.forEach(f -> {
        if(f.getFood().equals("test")) {
            testLog = f;
        }
    });
}
```

6.6 Backend

6.6.1 Implementation

Figures 6.31 and 6.34 document the code used to implement the backend service.

Figure 6.31: Backend service code

```
import base64
import label_image
import time

app = Flask(__name__)
#add a route for the flask app and what HTTP methods are allowed
@app.route('/classifyImage/', methods=['POST'])
def classify():
    #decode image and save it
    imgdata = base64.b64decode(request.form.get('image'))
    filename = 'images/' + str(time.strftime("%Y%m%d-%H%M%S")) +
        '.jpg'
    with open(filename, 'wb') as f:
        f.write(imgdata)

    #classify the image
    results = label_image.runModel(filename)
    predictions = results[0]

    #retrun a String of the results
    result_String = ""
    for i in predictions :
        result_String += str(i) + ","
    return result_String

#runs the app on port 5000
if __name__ == '__main__':
    app.run(host='0.0.0.0', threaded=True)
```

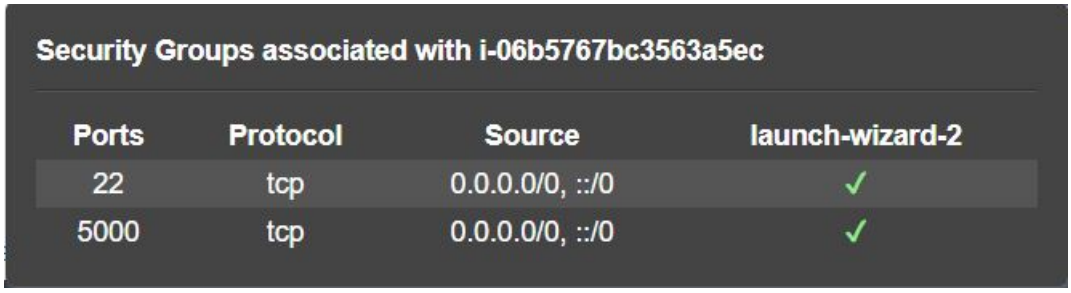


Figure 6.32: AWS Security Group

6.6.2 Deployment

In order to deploy the Flask application to an AWS EC2 instance the following steps had to be followed:

- Upload the output_graph.pb file (TensorFlow model), the output_labels.txt file and the source code to the server using SFTP.
- SSH into the server.
- Create a folder called 'images' in the home directory to store uploads images.
- Place model and label files into folders called 'models' and 'labels' respectively.
- Ensure TensorFlow, Python and Flask are installed on the server.
- Ensure no processes are running on port 5000 using the command 'lsof -i :5000'.
- Use the command 'screen'.
- Run the server.py code using the command 'python server.py'.
- Use the commands CTRL+A and CRTL+D to exit screen.

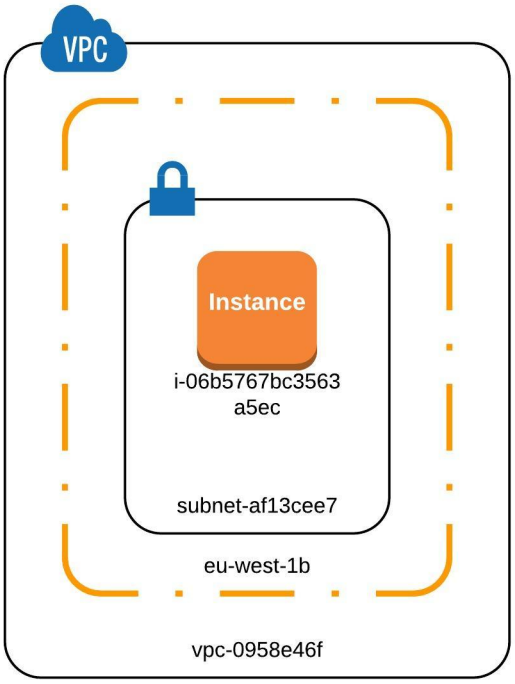


Figure 6.33: AWS Network Diagram

6.6.2.1 AWS Architecture

The application interfaces with an Amazon Web Services (AWS) server called an EC2 instance. To keep the application secure AWS was used to provide a secure architecture. A Virtual Private Cloud (VPC) was created to host the instances. In a production environment a load balancer could be used to evenly distribute traffic across a fleet of backend instances but for the purposes of this prototype application, a single instance as in Figure 6.33 could be used. In Figure 6.33 the instance is located inside a subnet which is in turn located in a VPC. The purpose of a VPC is to have a virtual network dedicated to a users individual account.

Security groups were utilised to restrict the type of traffic that would be allowed to reach the instance, so that the instance would only receive HTTP requests at port 5000 and receive SSH connections at port 22 as per Figure 6.32.

Figure 6.34: Method called from 'label_image.py' - adapted from (*How to Retrain Inception's Final Layer for New Categories* 2018)

```
def runModel(file_name):
  #load the graph using load_graph() provided be Google
  graph = load_graph(model_file)
  #read in tensor
  t = read_tensor_from_image_file(file_name,
                                  input_height=input_height,
                                  input_width=input_width,
                                  input_mean=input_mean,
                                  input_std=input_std)

  input_name = "import/" + input_layer
  output_name = "import/" + output_layer
  input_operation = graph.get_operation_by_name(input_name)
  output_operation = graph.get_operation_by_name(output_name)

  #run the tensor through the graph
  with tf.Session(graph=graph) as sess:
    results = sess.run(output_operation.outputs[0],
                       {input_operation.outputs[0]: t})
  results = np.squeeze(results)

  #get the top 5 results
  top_k = results.argsort() [-5:][::-1]
  labels = load_labels( label_file )

  setIndex = False

  #return results
  top5_results = [None] * 5
  index = 0
  for i in top_k:
    top5_results[index] = labels[i]
    index += 1

  final_results = [top5_results, results]
  return final_results
```

6.7 Issues

Some minor issues were overcame during the development of this prototype application such as:

1. Delay in image transfer: The time taken for an image to be sent to the server, be classified and return a response to the Android application was at around 12 seconds on a 4G network. This time delay was too long. In order to solve this issue, the size of the image was reduced from approximately 3,000 x 4,000 pixels to an image size of 300 x 400 pixels. This resulted in the time delay reducing to 3 or 4 seconds.

2. Service failure on backend: The backend instance service was stopping in a seemingly random fashion. This was due to an incompatibility with Python FLASK and the nohup (ignores the hangup signal in Linux) command used to deploy the FLASK application. To resolve this issue screen was used to run the app using a simple command 'python server.py'.

3. Performance issues: The Android application was facing some performance issues when displaying the food logs. The UI would temporarily freeze. This was because too much work was being carried out on the main thread. To combat this, another thread was created to complete the task.

Chapter 7

Discussion and Conclusion

7.1 Summary

The purpose of this final year project was to explore the use of deep learning for nutritional assessment. This exploration was carried out firstly by conducting a comprehensive literature review of research already undertaken in the area of food image recognition and also some research into the methods used for object detection with CNNs. Once this survey of literature was completed, a method from these papers was selected to be replicated. The work done by (Keiji Yanai and Kawano, 2015) yielded promising results in retraining models using transfer learning and because of this, their work was to be replicated.

Some deviations were made from (Keiji Yanai and Kawano, 2015) in regard to the dataset used. The Food-101 dataset was used for this FYP, in conjunction with some additional classes from ImageNet (Deng et al., 2009). In contrast, (Keiji Yanai and Kawano, 2015) used the UECFOOD100 dataset (Matsuda, Hoashi, and K. Yanai, 2012). The change in dataset was used due to the availability of a larger dataset in Food-101 which has 1,000 images per class.

The Food-101+ dataset was created by adding seven additional classes to the Food-101 dataset. This Food-101+ dataset was used to train a model with the final Top-1 accuracy of 66.6% and a Top-5 accuracy of 85.96%. This model was created by retraining the final layer of the Inception-V3 (Szegedy et al., 2016) model using transfer learning and TensorFlow. A subset of the Food-

101+ dataset was also trained using 13 classes in the same way. This model achieved a Top-1 accuracy of 92.6% and a Top-5 accuracy of 100%.

Once the models were trained, they were analysed under various topics such as dealing with composite images using a sliding window approach, the effect of colour on classification of food types and the effect of image quality on the classification of food types.

After the above empirical studies had been carried out, a lightweight prototype application was created to demonstrate the use of the TensorFlow model for food image classification. This prototype application was created for a smart phone running on Android. This application is called NutriLog. NutriLog is able to send an image of food (acquired either from the camera or gallery of the user's phone) to a server to be analysed. The server that the image is sent to is running a Python FLASK application that runs the image through a TensorFlow model and returns a prediction to NutriLog. NutriLog then collects nutritional information on the prediction using the Nutritionix API and logs information for user metrics. The user can then view their daily, weekly or monthly calorie intake. NutriLog could be very useful for those who wish to keep track of their calorie intake.

7.2 Reflections

The purpose of this section is to give a reflection upon the technologies I have used throughout this project. I will outline my feelings on each of these technologies in the following sections.

7.2.1 TensorFlow

There are many alternative machine learning libraries used for the development of CNNs such as Caffe, Gluon and MXNet. I used TensorFlow due to Google's reputation and prevalence on the Internet. While TensorFlow has a large learning curve, I believe this is due to the complexity of CNNs and machine learning in general rather than the library. I used Python in conjunction with TensorFlow in this project.

The online support for TensorFlow is superb. From resources such as Stack Overflow to Google's forums for questions relating to the topic. While I don't have experience with the other libraries, I would definitely recommend TensorFlow to beginners in machine learning as the documentation and support is outstanding.

7.2.2 Python

Python is a very simple and generic programming language supporting the paradigms of object oriented, functional, imperative and procedural. I enjoyed working with the language for the most part. It seems like the language is written to make things as easy as possible for the developer with many helpful in-built functions.

Python is a dynamically typed language meaning that type checking is carried out at runtime as opposed to statically typed when it is carried out at compile time. Personally, I prefer statically typed languages. This could be partly due to my experience in Java (which is statically typed) by I find when dealing with code that has been written by another developer, something as simple as knowing the types of function parameters is taken for granted.

7.2.3 Flask

Flask is a web framework that integrates with Python. I cannot stress how nice it is to use. It has a very small learning curve and its makes prototyping web services very easy.

7.2.4 AWS

I have had previous experience with Amazon Web Services on a summer internship and while working on college projects. Due to this past experience, there was no hesitation in choosing a cloud provider for my prototype application. A simple EC2 instance is very easy to setup using AWS.

7.2.5 Android Studio

Android Studio is the IDE used for the development of Android applications.

It is made by JetBrains who also developed the IDE InteliJ so there are many similarities there. For the most part it is like InteliJ which is very nice to use for Java development. The UI design in Android Studio is also very good as most of the UI development can be done in a drag and drop fashion which is very helpful for prototyping.

7.2.6 Java

I have been using Java for four years and it is a very nice language to use especially with the introduction of streams and other functional paradigms addition.

7.2.7 Latex

I used Latex for the report for this FYP due to the suggestion from my supervisor. Overall, I find it a very good resource to use. The fact that you focus more on content than presentation is very helpful for such a large report and the use of version control is very beneficial. The positioning of images and tables is sometimes a bit frustrating, but I found Latex the most useful when I wanted to move sections around in the report and for references.

7.3 Future Work

While retraining the Inception-V3 model yielded promising results for food image classification there are some problems associated. The model trained does not deal well with composite images and while a sliding window could be used to classify different portions of the image, it is not a viable method as it is very time intensive. Another issue is that there is a clear correlation between the number of classes and the accuracy of the model. This is clearly seen in the 108 class model yielding a Top-1 accuracy of 66.6% while the 13 class model yielded a Top-1 accuracy of 92.6%. Possible solutions to these issues are outlined below.

7.3.1 Resolving Issues with Composite Images

The model trained for this project relies on the one-shot approach where an image is passed straight to the model preferable containing only one food item. A possible solution to this problem is to segment the image before using the classifier. There has been extensive research into the segmentation of food images carried out, but it was out of scope for this FYP. Colour and texture are the most common features to segment food images by. If it was possible to separate each food item in a food image and feed each of these individual items through the classifier then the problem of composite images would be resolved.

Another option would be to take an object detection approach to food image classification. In Chapter 2 there was a survey of literature carried about using CNNs for object detection as in (K. He et al., 2017). If the objects (food items) could be detected in this way and then passed through the classifier then dealing with composite images would not be such a challenge.

While both of these methods are promising approaches to solving this issue, object detection seems to be a good approach. This is due to the success of (K. He et al., 2017), although the method has not been used for food images before.

7.3.2 Resolving Issues with Class Size

As stated above, the accuracy of the Inception-V3 model declines the more classes are in the training set. In order to solve this problem changes to the architecture of the model or changes to the dataset may have to be introduced. There are many CNN model architectures that could be viable options for food image recognition. (Mezgec and Koroušić Seljak, 2017) created a model architecture aimed specifically towards food image recognition called Nutrinet and experiments on this architecture could be carried out. The dataset used for training could also be extended with more images per class to increase the accuracy.

References

Abbirami.R.S et al. (2015). "Large Scale Learning for Food Image Classification". In: *International Journal on Recent and Innovation Trends in Computing and Communication* 3.3, pp. 973–978. URL: http://dx.doi.org/10.17762/ijritcc2321-8169.150317.

About - OpenCV library (2018). URL: https://opencv.org/about.html (visited on 02/18/2018).

Arens-Volland, Andreas G, Lübomira Spassova, and Torsten Bohn (2015). "Promising approaches of computer-supported dietary assessment and management-Current research status and available applications". In: *International journal of medical informatics* 84.12, pp. 997–1008.

Aurélien, Géron (2017). *Hands-On Machine Learning with Scikit-Learn & TensorFlow*. O'reilly.

Bossard, Lukas, Matthieu Guillaumin, and Luc Van Gool (2014). "Food-101 – Mining Discriminative Components with Random Forests". In: *European Conference on Computer Vision*.

Build an Image Dataset in Tensorflow (2018). URL: https://github.com/aymericdamien/TensorFlow-Examples/blob/master/examples/5_DataManagement/build_an_image_dataset.py (visited on 02/15/2018).

Chen, Mei-Yun et al. (2012). "Automatic chinese food identification and quantity estimation". In: *SIGGRAPH Asia 2012 Technical Briefs*. ACM, p. 29.

Chen, Nicholas et al. (2010). "Toward dietary assessment via mobile phone video cameras". In: *AMIA Annual Symposium Proceedings*. Vol. 2010. American Medical Informatics Association, p. 106.

Complete Guide to Tensorflow for Deep Learning with Python (2017). URL: https://www.udemy.com/complete-guide-to-tensorflow-for-deep-learning-with-python/learn/v4/overview (visited on 02/15/2017).

Daugherty, Bethany L et al. (2012). "Novel technologies for assessing dietary intake: evaluating the usability of a mobile telephone food record among adults and adolescents". In: *Journal of medical Internet research* 14.2.

Deng, J. et al. (2009). "ImageNet: A Large-Scale Hierarchical Image Database". In: *CVPR09*.

Deshpande, Adit (2018). *A Beginner's Guide To Understanding Convolutional Neural Networks*. URL: https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/ (visited on 04/12/2018).

Developer Guides (2018). URL: https://developer.android.com/guide/ (visited on 04/08/2018).

developers, NumPy (2018). *NumPy*. URL: http://www.numpy.org/ (visited on 04/13/2018).

Donahue, Jeffrey (2017). "Transferrable Representations for Visual Recognition". PhD thesis. EECS Department, University of California, Berkeley.

Easterbrook, Steve et al. (2008). "Selecting empirical methods for software engineering research". In: *Guide to advanced empirical software engineering*. Springer, pp. 285–311.

Eaton, Dr. Malachy (2017). *Lecture notes in Intelligent Systems*.

Erickson, Bradley J et al. (2017). "Machine learning for medical imaging". In: *Radiographics* 37.2, pp. 505–515.

Everingham, M. et al. (2007). "The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results". In:

— (2010). "The PASCAL Visual Object Classes Challenge 2010 (VOC2010) Results". In:

— (2011). "The PASCAL Visual Object Classes Challenge 2011 (VOC2011) Results". In:

— (2012). "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results". In:

fluid (2018). URL: https://www.fluidui.com/internify-android-app-prototype (visited on 03/27/2018).

Girshick, Ross (2015). "Fast R-CNN". In: *Proceedings of the International Conference on Computer Vision (ICCV)*.

Girshick, Ross et al. (2014). "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Gong, Shaogang, Stephen J McKenna, and Alexandra Psarrou (2000). *Dynamic vision: from images to face recognition*. Imperial College Press.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press.

Goodman, Samantha et al. (2015). "Vitamin D intake among young Canadian adults: validation of a mobile vitamin D calculator app". In: *Journal of nutrition education and behavior* 47.3, pp. 242–247.

He, Kaiming et al. (2017). "Mask R-CNN". In: *arXiv preprint arXiv:1703.06870*.

He, Ye et al. (2013). "Food image analysis: Segmentation, identification and weight estimation". In: *Multimedia and Expo (ICME), 2013 IEEE International Conference on*. IEEE, pp. 1–6.

Hoiem, Derek, Yodsawalai Chodpathumwan, and Qieyun Dai (2012). "Diagnosing Error in Object Detectors". In: *Computer Vision – ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part III*. Ed. by Andrew Fitzgibbon et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 340–353.

How to Retrain Inception's Final Layer for New Categories (2018). URL: `https://www.tensorflow.org/tutorials/image_retraining` (visited on 01/24/2018).

Howard, Andrew G et al. (2017). "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv preprint arXiv:1704.04861*.

Image Segmentation Using DIGITS 5 (2018). URL: `https://devblogs.nvidia.com/image-segmentation-using-digits-5/` (visited on 03/06/2018).

Inception in Tensorflow (2018). URL: `https://github.com/tensorflow/models/tree/master/research/inception` (visited on 02/05/2018).

Introducing JSON (2018). URL: `https://www.json.org/` (visited on 04/08/2018).

JetBrains (2018). *Statically typed programming language for modern multiplatform applications*. URL: `https://kotlinlang.org/` (visited on 04/08/2018).

Joutou, Taichi and Keiji Yanai (2009). "A food image recognition system with multiple kernel learning". In: *Image Processing (ICIP), 2009 16th IEEE International Conference on*. IEEE, pp. 285–288.

Kagaya, Hokuto, Kiyoharu Aizawa, and Makoto Ogawa (2014). "Food detection and recognition using convolutional neural network". In: *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, pp. 1085–1088.

Kawano, Yoshiyuki and Keiji Yanai (2014). "Food image recognition with deep convolutional features". In: *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*. ACM, pp. 589–593.

Khanna, Nitin et al. (2010). "An overview of the technology assisted dietary assessment project at Purdue University". In: *Multimedia (ISM), 2010 IEEE International Symposium on*. IEEE, pp. 290–295.

Krizhevsky, Alex, Vinod Nair, and Geoffrey Hinton. "CIFAR-10 (Canadian Institute for Advanced Research)". In: URL: `http://www.cs.toronto.edu/~kriz/cifar.html`.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*, pp. 1097–1105.

LeCun, Yann and Corinna Cortes (2010). "MNIST handwritten digit database". In: URL: `http://yann.lecun.com/exdb/mnist/`.

Lin, Tsung-Yi et al. (2014). "Microsoft coco: Common objects in context". In: *European conference on computer vision*. Springer, pp. 740–755.

Liu, Chang et al. (2016). "Deepfood: Deep learning-based food image recognition for computer-aided dietary assessment". In: *International Conference on Smart Homes and Health Telematics*. Springer, pp. 37–48.

Luger, George F. (2005). *Artificial Intelligence. Structures and Strategies for Complex Problem Solving/Luger GF*. Pearson Education Limited.

Mao, Dongyuan, Qian Yu, and Jingfan Wang. "Deep Learning Based Food Recognition". In:

Marsland, Stephen (2015). *Machine learning: an algorithmic perspective*. CRC press, pp. 71–110.

Matsuda, Y., H. Hoashi, and K. Yanai (2012). "Recognition of Multiple-Food Images by Detecting Candidate Regions". In: *Proc. of IEEE International Conference on Multimedia and Expo (ICME)*.

Meanshift Algorithm for the Rest of Us (Python) (2016). URL: http://www.chioka.in/meanshift-algorithm-for-the-rest-of-us-python/ (visited on 03/30/2018).

Mezgec, Simon and Barbara Koroušić Seljak (2017). "Nutrinet: A deep learning food and drink image recognition system for dietary assessment". In: *Nutrients* 9.7, p. 657.

Mitchell, Tom M. (1997a). *Machine Learning*. International Edition 1997. New York: The McGraw-Hill Companies, Inc., pp. 81–126.

— (1997b). *Machine Learning*. International Edition 1997. NewYork: The McGraw-Hill Companies, Inc., pp. 52–78.

Model-View-Presenter: Android guidelines (2017). URL: https://medium.com/@cervonefrancesco/model-view-presenter-android-guidelines-94970b430ddf (visited on 03/27/2018).

Nutrionix (2018). *nutritionix*. URL: https://www.nutritionix.com/business/api (visited on 04/08/2018).

Pouladzadeh, Parisa, Shervin Shirmohammadi, and Rana Al-Maghrabi (2014). "Measuring calorie and nutrition from food image". In: *IEEE Transactions on Instrumentation and Measurement* 63.8, pp. 1947–1956.

Pouladzadeh, Parisa, Shervin Shirmohammadi, and Abdulsalam Yassine (2014). "Using graph cut segmentation for food calorie measurement". In: *Medical Measurements and Applications (MeMeA), 2014 IEEE International Symposium on*. IEEE, pp. 1–6.

Pouladzadeh, Parisa, Gregorio Villalobos, et al. (2012). "A novel SVM based food recognition method for calorie measurement applications". In: *Multimedia and Expo Workshops (ICMEW), 2012 IEEE International Conference on*. IEEE, pp. 495–498.

Ren, Shaoqing et al. (2015). "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *Neural Information Processing Systems (NIPS)*.

Schap, TE et al. (2014). "Merging dietary assessment with the adolescent lifestyle". In: *Journal of human nutrition and dietetics* 27.s1, pp. 82–88.

Shelhamer, Evan, Jonathan Long, and Trevor Darrell (2017). "Fully Convolutional Networks for Semantic Segmentation". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 39.4, pp. 640–651.

Sliding Windows for Object Detection with Python and OpenCV (2018). URL: https://www.pyimagesearch.com/2015/03/23/sliding-windows-for-object-detection-with-python-and-opencv/ (visited on 01/30/2018).

Square (2018). *OkHttp An HTTP & HTTP/2 client for Android and Java applications*. URL: http://square.github.io/okhttp/ (visited on 04/08/2018).

Support Vector Machines for Machine Learning (2017). URL: https://machinelearningmast
com/support-vector-machines-for-machine-learning/ (visited on 12/20/2017).

Szegedy, Christian et al. (2016). "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2818–2826.

Szeliski, Richard (2010). *Computer vision: algorithms and applications*. Springer Science & Business Media.

Tiwari, Saurabh, Santosh Singh Rathore, and Atul Gupta (2012). "Selecting requirement elicitation techniques for software projects". In: *Software Engineering (CONSEG), 2012 CSI Sixth International Conference on*. IEEE, pp. 1–10.

Udacity, Inc (2018). *Deep Learning*. URL: https://eu.udacity.com/course/deep-learning--ud730 (visited on 04/03/2018).

Hy-Vee (2018). *11 heathy ways to eat more fruit and vegtables*. URL: https://celebrate.hy-vee.com/health/articles/26142/11-ways-to-eat-more-fruits-and-vegetables.aspx (visited on 04/14/2018).

Villalobos, Gregorio, Rana Almaghrabi, Behnoosh Hariri, et al. (2011). "A personal assistive system for nutrient intake monitoring". In: *Proceedings of the 2011 international ACM workshop on Ubiquitous meta user interfaces*. ACM, pp. 17–22.

Villalobos, Gregorio, Rana Almaghrabi, Parisa Pouladzadeh, et al. (2012). "An image procesing approach for calorie intake measurement". In: *Medical Measurements and Applications Proceedings (MeMeA), 2012 IEEE International Symposium on*. IEEE, pp. 1–5.

Yanai, Keiji and Yoshiyuki Kawano (2015). "Food image recognition using deep convolutional network with pre-training and fine-tuning". In: *Multimedia & Expo Workshops (ICMEW), 2015 IEEE International Conference on*. IEEE, pp. 1–6.

Zeiler, Matthew D and Rob Fergus (2014). "Visualizing and understanding convolutional networks". In: *European conference on computer vision*. Springer, pp. 818–833.

Zhang, Weiyu et al. (2015). "Snap n Eat - Food Recognition and Nutrition Estimation on a Smartphone". In: *Journal of diabetes science and technology* 9.3, pp. 525–533.

Zhu, Fengqing et al. (2011). "Segmentation assisted food classification for dietary assessment". In: *Computational Imaging IX*. Vol. 7873. International Society for Optics and Photonics, 78730B.

Appendix A

Segmentation

Table A.1: FCN Results (Shelhamer, Long, and Darrell, 2017)

	FCN
VOC11 mean IU	62.7
VOC12 mean IU	62.2
PASCAL VOC10 pixel acc.	67.0
PASCAL VOC10 mean acc.	50.7
PASCAL VOC10 mean IU	37.8
PASCAL VOC10 f.w. IU	52.5

A.1 Image Segmentation

A.1.1 Fully Convolutional Neural Networks for Semantic Segmentation

There has been a very interesting paper from UC Berkeley focused on using Full Convolutional Networks for semantic segmentation (Shelhamer, Long, and Darrell, 2017). Fully Convolutional Networks (FCN) do not have any fully connected layers. They are replaced with more filtering layers. Nvidia Digits have a semantic segmentation implementation based off the work of this paper.

They took this approach because "feedforward computation and backpropagation are much more efficient when computer layer-by-layer over an entire image instead of independently patch-by-patch" (Shelhamer, Long, and Darrell, 2017). This was also because they were focused on object detection. Normal classifiers do not work very well when they are to classify more than one subject in an image and image segmentation was a way to solve this.

There are a set of steps you can follow to turn a CNN into a FCN for semantic segmentation as follows i.e. change to a convolutional layer from a fully connected one:

- The size of the filters must be set to the size of the input layers.
- For every neuron in the fully connected layer, have a filter.

Table A.2: Results

	Single Food Portion	Non-mixed Food	Mixed Food
Colour and Texture	92.21	N/A	N/A
Graph Based	95 (3% increase)	5% increase	15% increase

A.1.2 Segmentation

A.1.2.1 Graph Based Segmentation

Graph cut segmentation has been used extensively in image segmentation. OpenCV has an implementation of a graph cut algorithm called grabcut which has been used to segment food on occasion (Pouladzadeh, Shirmohammadi, and Yassine, 2014).

According to (Pouladzadeh, Shirmohammadi, and Yassine, 2014), "Graph cut based method is well-known to be efficient, robust, and capable of finding the best contour of objects in n image, suggesting it to be a good method for separating food portions in a food image for calorie measurement". Along with the graph cut segmentation algorithm, this research team also used colour and texture segmentation. Gabor filters were used to measure texture features (Pouladzadeh, Shirmohammadi, and Yassine, 2014). When colour and texture segmentation was applied, the method came into difficulty with mixed foods but by applying graph cut segmentation, clearer object boundaries were shown.

In conclusion, the accuracy of the classification increased when using graph based segmentation rather than colour and texture as seen in A.2.

A.1.2.2 Local Variation Framework

Another paper was published in which the research team attempted to create a food calorie estimation system (Y. He et al., 2013). This system would compromise of three steps, image segmentation, image classification and weight estimation. For the segmentation module, a local variation approach to segmentation was performed. Local variation is by which intensity differences between neighbouring pixels is measured. This is a type of graph based segmentation.

The team also carried out some segmentation refinement when the segmenta-

tion algorithm had been performed. This consisted of removed small segments (defined as less than 50 pixels) and trying to prevent over and under segmentation. After classification was performed on each segment, segments with low confidence values were removed (Y. He et al., 2013).

A.1.2.3 Conclusion

Both of the above papers of (Pouladzadeh, Shirmohammadi, and Yassine, 2014) and (Y. He et al., 2013) used a graph based segmentation. The first paper used a more generic implementation while the second used a local variation framework. Both methods provided successful results in the image segmentation process.

Appendix B

Poster

Identification and Classification in Food Images

Name: Tom Barrett **ID**: 14171198 **Supervisor:** J.J Collins

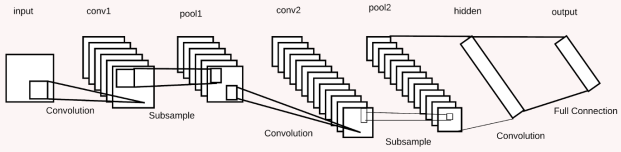
Objective

This work explores the use of deep learning for nutritional assessment. A Convolutional Neural Network (CNN) (created using Google's Tensorflow) is utilised to classify food images. A smart phone application will leverage this CNN to track a users calorie intake. Extensive research has been carried out in this subject area (Yanai and Kawano, 2015).

References: Yanai, Keiji and Yoshiyuki Kawano (2015). "Food image recognition using deep convolutional network with pre-training and fine-tuning". In: Multimedia & Expo Workshops (ICMEW), 2015 IEEE International Conference on. IEEE, pp. 1–6.

Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a deep neural architecture with layers for the extraction of image characteristics, dimensionality reduction, and classification (Zeiler and Fergus, 2014).



References: Zeiler, M.D. and Fergus, R., 2014, September. Visualizing and understanding convolutional networks. In *European conference on computer vision* (pp. 818-833). Springer, Cham.

Dataset

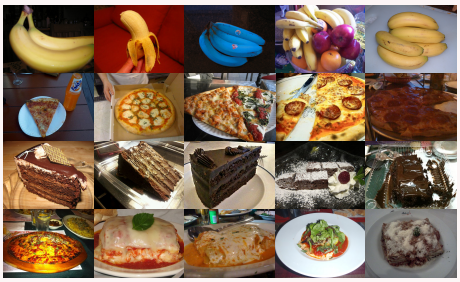
The Food-101 dataset was used for this FYP with 7 extra classes added using the ImageNet dataset to create Food-101+.

Banana:

Pizza:

Cake:

Lasagne:



Results

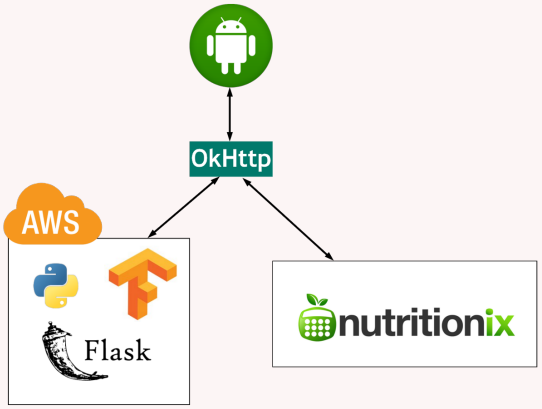
Various models were trained using the Inception-V3 architecture for this FYP. A summary of their results can be seen below.

Experiment	Dataset	Training Steps	Learning Rate	Top-1 Accuracy
1	Food-101	4,000	0.01	54.8%
2	Food-101+	4,000	0.01	55.3%
3	Food-101+	10,000	0.1	66.3%
4	Food-101+ subset (13 classes)	2,000	0.1	92.6%

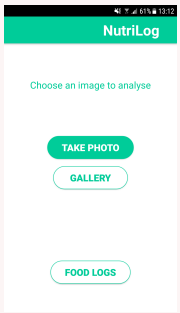
Prototype Application - NutriLog

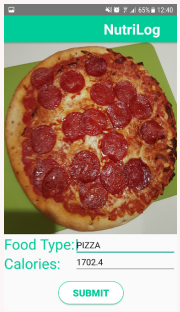
Architecture

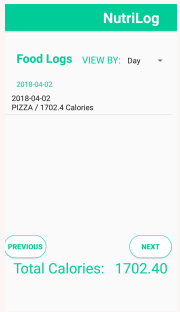
NutriLog is an Android application that sends an image to an AWS instance using OkHttp. The instance is running a Python Flask application that classifies the image using Tensorflow and sends a response back to NutriLog. The Nutritionix API is used to collect nutritional information of this classification.



User Interface







Conclusions

The Inception-V3 architecture has proven to yield promising results for training models using the Food-101+ dataset. NutriLog could greatly benefit those who would like to keep track of their calorie intake. Future work could consist of analysing the results of different architectures and segmenting the image before classification to combat composite images.