‘5 a day’ tracker: Food logging using Convolutional Neural Network

Hiu Fung Keung

August 2017

Acknowledgement

I would like to thank my supervisor, Professor Majid Mirmehdi, for giving me guidance throughout the project.

Executive Summary

Consumption of fruits and vegetables has shown to be able to prevent a number of chronic diseases [33, 14]. To improve general public's health, NHS started a '5 a day' campaign to encourage individuals to consume at least 5 portions of fruits and vegetables every day [30]. However, survey showed that only a quarter of adults in the UK meets the target [35].

To help address this issue, in this project, I built an Android application that helps user to keep track of their fruits and vegetables consumption through photos taken by them. As CNN has shown to be the state of the art image recognition technique [22, 50, 41, 42], it was used for the image recognition task in the application.

This project consisted of both software development (type 1) and research (type 2).

Research of this project mainly involved looking into how a Convolutional Neural Network (CNN) works and looking into different CNN models. I evaluated different CNN models to decide which one suits my needs. In the end I chose a ImageNet pre-trained *Inception5h* CNN model for my application. The reason behind is that *Inception5h*, which is a version of GoogLeNet, is more computationally efficient than most other CNN I looked into while having a very good accuracy at the same time [41].

Software development of this project involved coding an Android application that keeps tracks of user's daily fruits and vegetables consumption and implementing the *Inception5h* chosen earlier into the application. Since the CNN can only identify one object within an image, I also had to develop and evaluate different strategies to allow recognition of multiple objects in a single image through the CNN.

The main achievements from this projects are as follow:

- Implemented a Convolutional Neural Network (CNN) into a mobile application in an offline setting.
- Developed and evaluated different strategies to identify multiple objects in a single image through a CNN.
- Designed a database for the application.
- Designed an user interface for the application.

Contents

1 Introduction

1.1 Motivation

Fruits and vegetables are high in fibre and contain a wide range of vitamins and minerals. Studies have shown that fruits and vegetables consumption is linked with a lower risk of cardiovascular diseases [33] and even specific types of cancer [14]. As fruits and vegetables are generally low in calories, it can also be used to help individuals to control their weight easier.

To raise awareness of the benefits of fruits and vegetables consumption, NHS has launched a public health campaign called '5 a day' [30]. The idea behind this campaign is to encourage individuals to consume at least 5 portions of fruits and vegetables every day. Despite the campaign effort, Health Survey of England suggested that only a quarter of adults in the UK meets the '5 a day' target [35].

One way to help solve this issue is to make individuals more aware of their daily consumption of fruits and vegetables. In this project, I will attempt to use the latest technology to help individuals to keep track of their '5 a day' consumption easier. With how common it is for individuals to take photos of their food through their smartphones, I decided that my project should be a smartphone application that uses image recognition technique for photos taken by the user to help them to keep track of their fruits and vegetables consumption.

1.2 Technical Background

Accuracy of the state-of-the-art image recognition technique have improved drastically throughout the last decade [22, 42]. One of the main reasons behind this improvement is the use of Convolutional Neural Network (CNN) for this task. CNN is a versatile machine learning algorithm that is used for many different purposes with image recognition being one of them. Recent researches all showed that CNN has a superior image recognition accuracy comparing to other 'traditional' techniques[22]. This is looked into more detail in section 2.5. With how accurate this technique is, I decided to use it for the image recognition for my application.

Although there are currently food logging applications in the market that made use of CNN to identify food in the images taken by the user, as far as I am aware, all these applications require internet connection to use. I aim to address this in my project.

1.3 Aims & Objectives

The aim of this project is to create an Android application that is able to keep track of user's fruits and vegetables consumption solely through images taken by the user in an offline setting. This can be broken down into 4 objectives.

1. Choose a suitable Convolutional Neural Network model and implement it into an Android application in an offline setting.

2. Develop and evaluate different strategies on identifying multiple objects in a single image through the Convolutional Neural Network.

3. Design a database to store user's input.

4. Design and create an user interface for the application.

1.4 How the application should work

As far as the user concerned, logging in the food by the CNN in the application is a three steps process. First, the user will tap on the camera button in the application. This will bring them to a camera screen where the user take a photo of their food. Then there will be a pop up from the screen that will prompt them to select the quantity and confirm if the food is correctly identified. This is illustrated in figure 1.

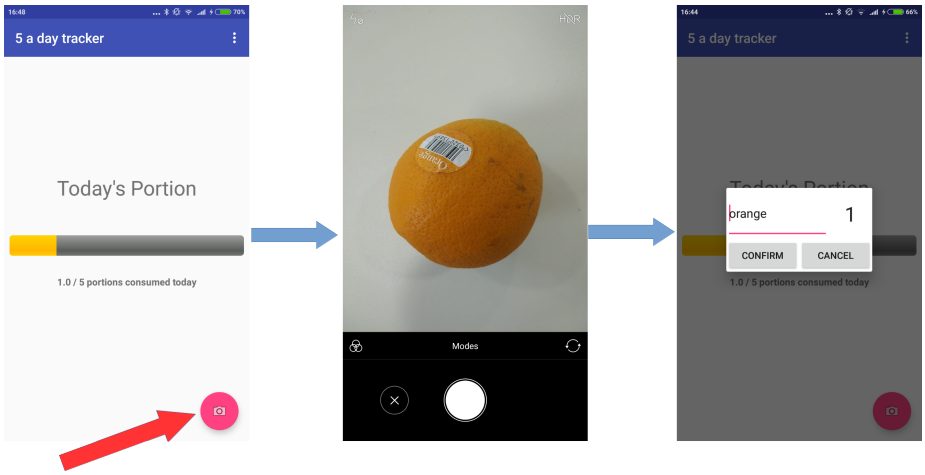


Figure 1: Process of user using CNN to do food logging in the application.

The work behind the scenes will be more complicated and involves more steps. After the user take the photo, the photo will be processed by one of the strategies that I developed. The processed images will then be passed into a Convolutional Neural Network (CNN) and the output of the CNN will be read and interpreted. The relevant item found from the CNN output will then be shown as a pop up on user's screen and user will be prompted to confirm it and enter its quantity. Finally, the confirmed item will be put into the database so the user can see their record when they desire.

1.5 Structure of the thesis

This thesis is divided into 6 parts.

1. The first part of the thesis is on the related work and the technical background of the project. In this part, I will first look into some of the previous work done on food image recognition. Then I will look into the structure of CNN and different models of CNN and evaluate the reasons on how viable it is for me to implement each model into my application. Since I am implementing the CNN on an offline mobile setting, I will also look into researches on improving computational performance of CNN on smartphones.

2. In the second part of the thesis, I will talk about different factors that I considered in order to choose the ideal CNN for the application. I will also mention how I implement the CNN into the application in this section.

3. The third part of the thesis will be on the proposed methodology for identifying multiple objects in a single image. In this part, I will first describe all the strategies that I developed, then I will evaluate the results and make a decision on which one to implement in my application.

4. Design and implementation of user interface and database will be the fourth part of the thesis. As the name suggests, in this part, I will go through the design decisions I made when making the user interface and database. I will also show some screenshots of my user interface in the relevant sections.

5. In the fifth part of the thesis, I will be evaluating the final application that I made. I will refer back to the aims and objectives as described above and see if I have met them or not. I will also discuss about different limitations of my application and what could be done to overcome it. The added value and future work that could be done on this project will also be discussed in this part.

6. The last part of the thesis is to conclude and summarise everything that I did throughout the project.

2 Background on technologies used/Related Work

2.1 Introduction

In the sections below, I will first look into studies that used conventional methods for food image recognition. Then, I will look into the latest image recognition technology, Convolutional Neural Network, which has shown to have a higher accuracy rate than conventional methods.

2.2 Food Image Recognition with Conventional Methods

Conventional methods used for food image recognition is usually done in two steps. Firstly, specific information is extracted from the image that contained food. This can be a colour histogram, bag of features or something else. These extracted information is then passed into a classifier, which is usually a Support Vector Machine (SVM), in order to determine what type of food it is.

In study by Kawano and Yanai [20], the authors developed a real-time mobile food recognition system for 50 different categories of food. User first had to segment the food in the phone application by drawing a bounding box. Then colour histogram and bag-of-SURF features were extracted from the area inside the bounding box. These extracted information were then passed into 50 linear SVM classifiers that are trained for the task. Result from the study showed that it can achieve a 68.2% of top-5 accuracy rate. The same study also tried to extract different information from bounding box by using Histogram of Oriented Gradients (HOG) and colour patch with Fisher Vector. Using this alternative method, result showed that it can achieve a 77.6% top-5 accuracy rate.

In a study by Kitamura et al. [21], author tried to determine the group of food in the image (e.g. grains, vegetables, milk, fruits and meat & beans). This was done by dividing the image into 300 separate windows. Colour histogram and DCT coefficients were then extracted from each window and passed into a SVM to determine which food group the window belonged to. Using this approach, the study achieved a 73% accuracy and performed particularly well on windows that was in the 'grain' category. However, there was trouble distinguishing 'meats & beans' and 'vegetables' from each other.

Study by Hoashi et al. [15] went further by extracting 17 different features, which included color histogram, bag-of-features, Gabour features and gradient histogram, to classify 85 categories of food by using a SVM with the help of Multiple Kernel Learning (MKL). Result showed that this method can achieve 62.52% of classification rate in their test images, and 45.3% classification rate using images uploaded by trial users.

Other innovate methods were also explored. By using pixel level image labelling in combination of exploiting the spatial characteristics of food, study by Yang et al. [49] extracted the distance between pixels that represent different ingredients and passed it into a SVM in order to determine what the food is. This method was tested on Pittsburgh fast-food image dataset, a challenging data set that consisted of different fast food [4]. The result showed that

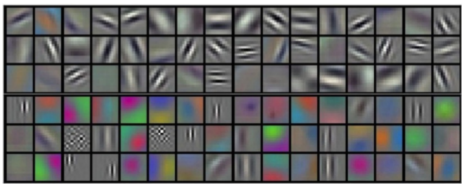
this method has nearly 80% of accuracy on identifying different types of food e.g. salad, sandwiches, which is nearly 30% higher than the accuracy obtained through the baseline approach, which used colour histogram with SVM and bag-of-SIFT features with SVM. However, the use of this spatial exploitation method to classify food is also quite limited, as it only works with food group that consists of multiple distinct ingredients.

2.3 Food Image Recognition with Convolutional Neural Network

More recent food image recognition studies have shifted away from the use of conventional method, as described in the section above, into the use of Convolutional Neural Network (CNN). In this section, I will look into different researches that used CNN for food image recognition. I will also look into two different papers that attempted to implement the CNN for food image recognition in a mobile setting.

2.3.1 Researches that used CNN

Most researches has shown that CNN has a superior accuracy comparing to the conventional methods used for food image recognition, with a top-5 accuracy rate that ranges from over 70% to over 90% [18, 1, 48, 19, 6, 27]. It is also worth mentioning that the CNN in these studies are only 4-7 layers deep, which is relatively shallow comparing to most modern CNN models. With CNN being the state of the art image recognition technique in general (this will be discussed in section 2.5), the superior accuracy shown was to be expected.



(a) Filters trained using ImageNet dataset.

(b) Filters trained using food images.

Figure 2: Convolutional filters trained from different dataset, from [18]

Research from Kagaya et al. looked at filters for ImageNet trained CNN and food images trained CNN and suggested that colour is the most dominant feature in food recognition task [18]. The filters of the two differently trained CNN was shown in figure 2. You can see that lines and edges are more dominant in the ImageNet trained CNN, while filters for the food images trained CNN are more colour-specific. This is interesting as the difference in the filters suggested that there are fundamental difference between ImageNet trained CNN and food trained CNN. As result, it is possible that training a CNN from scratch using only food images may result in a significant difference in accuracy comparing to a CNN that is pre-trained from ImageNet and fine-tuned for food recognition.

2.3.2 Implementation examples

On top of academic researches, there are also a few papers on the implementation of CNN on smartphones for food recognition.

Im2Calories is an ambitious attempt by Google to use CNN for calorie estimation [28]. This is done by replacing the 1000 way softmax classifier on an ImageNet pre-trained GoogleLeNet with a 201 way food classifier. In this study, CNN was also used for image segmentation and depth estimation in order to estimate the volume of the identified food. The paper claimed that they had only managed to port the food classifier into a smartphone setting so far.

DeepFoodCam is another example application that implemented a CNN for food image recognition [48]. DeepFoodCam utilised a CNN that has no fully connected layer. Author suggested that this is because most of the parameters in the network lies in the fully-connected layer. Removing them will reduce the memory overhead of the network, making the speed of the application much faster. Result in the paper showed that the application run on an iphone SE has an inference speed of 77.6ms with 95.2% top-5 error rate. It is interesting to note that author was not able to achieve the same speed in Android smartphones. Author suggested that this was because the BLAS library in the iOS Accelerate Framework was able to speed up the application effectively, but the BLAS implementation for Android was not able to do the same.

2.4 Structure of CNN

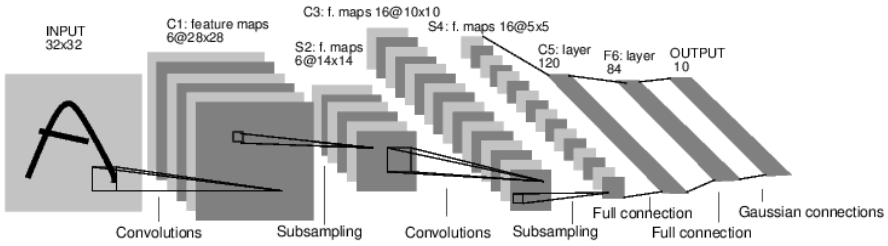


Figure 3: Architecture of LeNet-5, from [25]

Since CNN has shown to have very high accuracy on food image recognition, I decided to look into the structure of CNN and how it works in this section.

In figure 3, you can see the architecture of LeNet. LeNet [25] can be thought of as the first CNN. Although newer CNN models may have a more complicated structure, the basic architecture and ideas behind LeNet still applies to the modern day CNN models.

The structure of a CNN can roughly be divided into two parts. The first part is made up of convolutional layers stacking on top of each other, with subsampling layer in between them. The second part is a fully-connected layer (which includes a output layer). In sections below, I will look into the different layers and describe what they do briefly.

2.4.1 Convolutional Layer

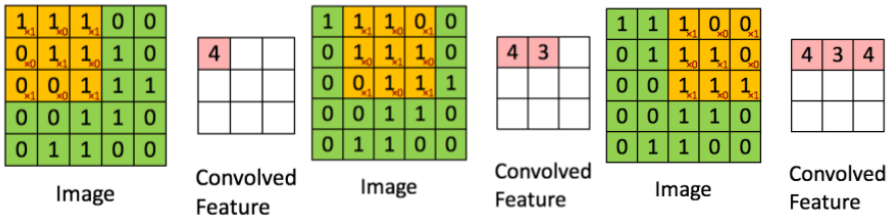


Figure 4: Visualisation of a 3×3 filter sliding across a 5×5 image, from *deeplearning.stanford.edu*

Filter and feature map Every convolutional layer consists of a filter (or multiple filters in most cases). The filter is simply a matrix that slides across the image. In figure 4 you can see a 3×3 filter sliding across a 5×5 image. Different convolutional layers can have filters with different size.

Initially, the numbers inside the filter matrix (also called weights) is random, and is determined by training the CNN. (Training of the CNN is described in more detail in section 2.4.4.) The way that the filter works is that the weights multiplies the pixel value that it covers in the image. The multiplied values are then summed up into a single number. This new number becomes one of the new pixels in the new 'image'. This new 'image' is called a feature map, also sometimes referred to as an activation map. In other words, the purpose of a convolutional layer is to turn its input into a new feature map. It should be noted that it is likely that the resulted feature map will have a smaller size than the input image. This is because the number of possible locations that the filter can be placed on the original image is less than the number of pixels in the original image. Using figure 4 as an example, although there are 25 pixels in the 5×5 image, the resulted feature map is only 3×3, as there are only 9 possible locations that the filter can fit into the 5×5 image.

Stride On top of the size of the filter, the stride also affects the size of the feature map. Stride is another term for describing how far the filter slides each time. If the stride is 1, the filter moves by 1 pixel each time, as shown in figure 4. If the stride is 2, the filter moves by 2 pixels each time. Therefore, the bigger the stride is, the smaller the resulting feature map will be.

Rectified Linear Units (ReLU) After a feature map is produced from the convolutional layer, a non-linear activation function is applied to it. This is because all the convolutional layer did was element-wise matrix multiplication and addition. These operations are linear operations. To make the output more meaningful, nonlinearity is introduced to the feature map output. This is important as most real-world data are non-linear. Rectified Linear Units

(ReLU) is usually used for the task. ReLU simply replaces negative values in feature map with 0. The equation of ReLU is shown below.

$$f(x) = max(0, x)$$

Although there are other non-linear activation functions available, research showed that ReLU is superior to other activation functions as it is able to speed up training [22]. ReLU also do not have vanishing gradient issue that is common in some other non-linear activation function[10].

Multiple layers It is important for a CNN to have multiple convolutional layers. This is because the first few convolutional layers will only extract lower-level features e.g. lines, curves, while the latter layers will extract higher-level features e.g. faces. This is illustrated in figure 5. The reason behind this is because as the original image advances through the convolutional layer, it will result in a smaller feature map. This means that the filter's receptive field is relatively larger to the image and therefore the filter will be able to look at a wider region of the image and extract higher-level features.

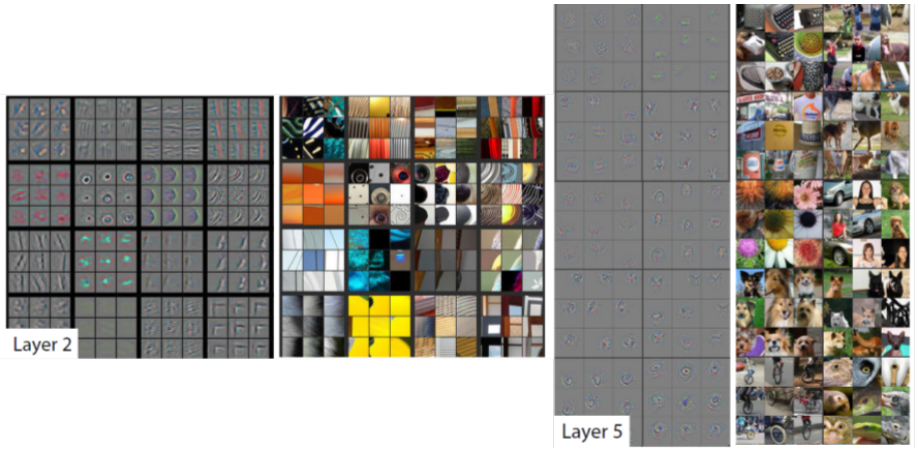


Figure 5: Feature maps from different convolutional layers, from [50]

Overfitting Although it may seems like having a large number of small filters and stacking as many convolutional layers will lead to a more accurate CNN model, in reality, that will only lead to overfitting.

Overfitting means that the CNN is looking at the images in too much detail therefore making its prediction worse. Figure 6 shows an example of overfitting. The overfitted function was represented as the polynomial line, and correct function was represented as the linear line. Even though the polynomial line fits every data, the linear line will do a better job at predicting results.

In general, decreasing the number of parameters in the network and increasing the training data are used to combat overfitting. A technique called *dropout* is also often used in modern CNN models to help to deal with overfitting [40].

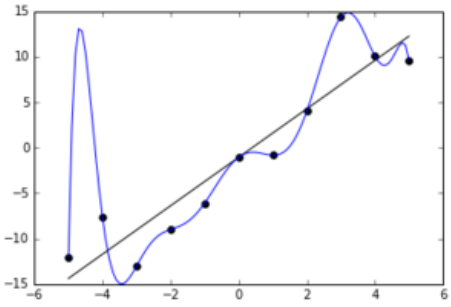


Figure 6: Example of overfitting, from Wikipedia

Higher level perspective In a higher level perspective, a convolutional layer is simply a feature extractor. If the input image has the feature that the filter is trying to extract, it will result in a strong feature map. Therefore, the higher the number of filters, the higher number of features can be extracted from the input image.

2.4.2 Subsampling Layer

Subsampling layer, also referred to as pooling layer, is often placed in between convolutional layers.

Pooling filter Similar to convolutional layer, subsampling layer also has a 'filter' that slides across the input image. The size of the filter determines the amount of information that will be lost through the pooling process. The bigger the filter size, the more information will be lost.

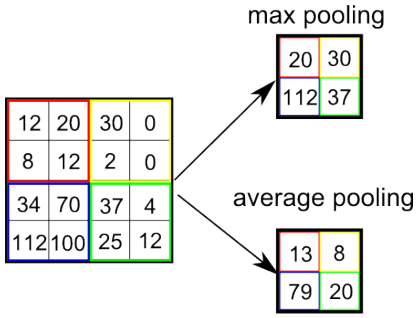


Figure 7: Example of max pooling & average pooling

Max & Average Pooling Besides choosing the size of the filter in subsampling layer, the designer of the CNN will also have to decide if the layer uses max pooling or average pooling. As shown in figure 7, max pooling is taking the highest number that is covered by the pooling filter, while average pooling is taking the average of all the numbers covered by the pooling filter. There are pros and cons for each pooling method and both pooling methods are used in modern CNN.

Overlapping & non-overlapping Pooling After choosing max or average pooling for the subsampling layer, designer of the CNN also have to choose if the pooling is overlapping or non-overlapping. As the name suggests, overlapping

pooling is when the pooling filters overlap each other and non-overlapping is when the filters do not overlap. The example of pooling shown in figure 7 was an example of non-overlapping pooling. In the paper by Krizhevsky et al., the authors suggested that the use of overlapping pooling led to a decrease in error rate in their CNN model [22].

Higher level perspective In a higher level perspective, the purpose of the subsampling layer is to reduce the size of the feature map. As result, this will lead to a lower number of parameters in the network, and reduces the computational cost of the model.

2.4.3 Fully-Connected Layer

Following layers of convolutional and subsampling layers are the fully-connected layers. Fully connected layer in a CNN have the same structure as a 'traditional' regular neural network. It can be divided into 3 parts, an input layer, hidden layer(s) and an output layer. All the nodes in each layer is fully connected to all the nodes from the previous layer. This is demonstrated in figure 8. Below I will briefly describe the 3 types of layers that make up a fully connected layer.

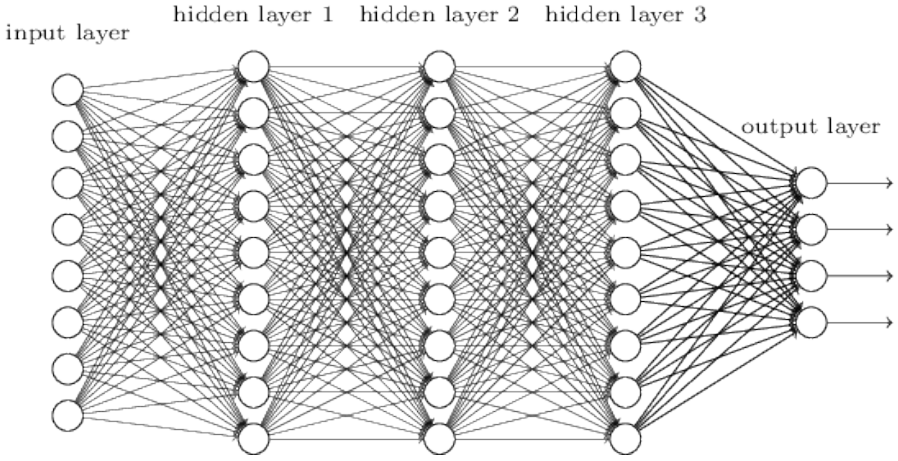


Figure 8: Visualisation of a fully-connected layer, from *neuralnetworksanddeeplearning.com*

Input layer Input layer is the first layer of the fully connected layer. It takes in the output of the previous layer (which can be a ReLU output from the convolutional layer or a output from a pooling layer). The number of nodes in this layers should be the same as the number of features in the training data.

Hidden layer Hidden layer is the middle layer of the fully connected layer. The number of hidden layers and the number of nodes in the hidden layer can differs between different models. The weight of the nodes in a 'fresh' neural

network is initially random and will be determined during training. I will discuss more about training in section 2.4.4.

Output layer The last component that make up the fully connected layer is the output layer. In a CNN that is carrying out a classification task, a softmax regression layer is usually used as the output layer. The softmax function in the layer takes in all the values gathered from the previous layer to give the probability of all the possible outputs in a range of 0 to 1 with the total of all outputs being 1.

For example, if the input to the CNN is an image of a cat, the softmax regrssion layer may output [cat 0.7, dog 0.1, lion 0.1, tiger 0.1]. This means that there is a 70% chance the image is a cat, and 10% that it is a dog, lion or a tiger. Notice that the possibilities is represented as a range between 0 to 1, and all the possibilities add up to 1 in total.

Higher level perspective In a higher level perspective, the purpose of the fully-connected layer is to take the feature map output from the previous convolutional and subsampling layers and interpret it to give a meaningful output.

2.4.4 Training

Although technically training is not part of the structure of the CNN, it is essential for a CNN to function properly. Different parameters like the weights in the filters in convolutional layer and weight in the nodes inside the fully connected layers are determined by training the CNN. In simple terms, training a CNN is the process of adjusting these weights and parameters in the network to find the optimal 'setting'.

Training process Below is a simplified version on the steps it takes to train a CNN.

1. An input (i.e. an image) is passed through the whole network. Since all the filters and weights are random initially, the output of the CNN will also be random.

2. The error between the output of the CNN and the correct result is calculated. Loss function is used to calculate this error. Mean squared error (MSE) and cross-entropy are two of the most common loss functions used for the task.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Correct - Predicted)^2$$

   MSE is often used as a linear regression loss function, while cross entropy is preferred as a logistic regression loss function. Therefore in a CNN that do image classification, cross entropy is the preferred loss function to use for training.

$$\text{Cross-Entropy} = \sum_x p(x) \log q(x)$$

with p being the correct result vector and q being the predicted one. In simple words, the purpose of this formula is to calculate the distance between the predicted vector and the correct one.

3. A backward pass is performed in order to determine the weights that contributed to the loss functions.

4. The weights are updated and the loss function is minimised through the use of gradient descent. Because the weights are updated, the next time when the same image is passed through the network, the error will be smaller than last time and the weight will be closer to the optimal weight.

5. Steps 1-4 are repeated for all the training data.

Training data Since the weight is continuously optimised during training, the more training data there is, the more optimised the weights in the network will become. Therefore it is no suprise that different data augmentation techniques (e.g. random cropping, horizontal reflection) are often used to artificially increase the amount of training data.

2.4.5 Newer CNN Networks

Although a traditional CNN has a relatively linear structure, newer and better performing CNN models like Inception [41] and ResNet [12] added a lot more non-linearity into their structure. I will look into these CNN models briefly in section 2.5.5 and 2.5.6 respectively.

2.5 Why CNN

In the previous section, I mentioned how the structure of a typical CNN looks like. In the following section, I will explain why CNN is considered the state of the art while looking into different CNN models. Since I will be using CNN for image recognition for my project, for each CNN model, I will also explain how viable it is for me to implement it into my application.

To understand why CNN is the current state of the art image recognition technique, I have to first introduce ImageNet Large Scale Visual Recognition Challenge (ILSVRC).

2.5.1 ImageNet Large Scale Visual Recognition Challenge

Since 2010, every year there is a computer vision 'Olympic' called ImageNet Large Scale Visual Recognition Challenge (ILSVRC) where academics and research teams around the globe compete for the most accurate algorithm across different computer vision related tasks [36]. Therefore to determine what is

the state of the art image recognition technique, we can look at the winners of ILSVRC.

In sections 2.5.2 - 2.5.6 below, I will look into the winners and different innovative algorithms that achieved great results in the classification task in ILSVRC.

2.5.2 AlexNet

AlexNet was the winner of the 2012 ILSVRC. Comparing with other competitors in ILSVRC-2012, AlexNet achieved a top-5 error rate of 15.3% while the next-best performing algorithm had a top-5 error rate of 26.2% [22]. With how staggering the accuracy difference was between AlexNet and its competitors, it was no surprise that the algorithms used in AlexNet quickly became the new state of the art image recognition technique. In fact, all the ILSVRC winners from 2012 onwards implemented different techniques and ideas that were used and introduced in the AlexNet paper. Increasingly powerful GPU, availability of larger training sets and novel strategies like dropout all contributed to the up rising of CNN [50].

AlexNet is a deep convolutional neural network that was trained with 15 million images across 22,000 categories. Although there are researches on convolutional neural network before AlexNet [25], what sets the AlexNet paper apart was that it introduced a few different novel features. Some of the novel techniques included using Rectified Linear Units (ReLU) for faster learning, training CNN model across multiple GPUs and using dropout to combat overfitting. These techniques are all widely used in different CNN models that are developed later.

For my project In my project, I will not be implementing the AlexNet, as there are newer CNN models that are more computationally efficient, and has a lower error rate. Even though I am not using the AlexNet, novel techniques that are introduced in the paper, like ReLU, will still be used in the model that I am implementing.

2.5.3 ZFNet

ZFNet, the winner of ILSVRC-2013, had a very similar architecture to AlexNet. One of the few differences was that ZFNet used a `7×7` filter for the first convolutional layer instead of `11×11` as proposed in AlexNet [50]. They also changed the stride from 4 to 2 and expanded the size of the middle convolutional layer. The author suggested that these changes allowed the CNN model to retain more information from the original input images and therefore achieve a better classification performance. This was shown as the ZFNet achieved a top-5 error rate of 14.8%.

Although the architecture for ZFNet was not ground-breaking, the paper introduced a novel visualisation technique called Deconvolutional Network [51]. Deconvolutional Network is a reversed convolutional network where it maps

features to pixels. This visualisation technique gave researchers a better understanding on how each layer in the model works as it allows them to see which pattern activates which feature map. Figure 5 is an example of the information that can be gathered by using Deconvolutional Network.

For my project In my project, I will not be implementing the ZFNet. Although the accuracy was an improvement from AlexNet, there are still better performing models available nowadays that are less computationally intensive.

2.5.4 VGG-Net

Although VGG-Net was only the runner-up of ILSVRC-2014, this CNN model is very important to the field. VGG-Net reinforced the idea that depth is critical for the accuracy of a CNN. Comparing VGG-Net to AlexNet and ZFNet, VGG-Net used significantly more convolutional layers than the two. There was 16 convolutional layers in VGG-Net while AlexNet and ZFNet only had 5. VGG-Net also used a much more linear and simple design, with all 16 convolutional layers using `3×3` filters and all pooling in the model using `2×2` windows. This simple and deep design achieved a 7.3% top-5 error rate [38], which was a significant improvement comparing to all the previous ILSVRC winners. However, this design also led to a lot more parameters in the network. Using AlexNet as a comparison, AlexNet had 60 million parameters in its network, while VGG-Net had around 140 million.

For my project In my project, I will not be implementing the VGG-Net. Although the accuracy is impressive, the number of parameters in the network is too high. For that reason, I do not think that it is viable to implement it on a smartphone setting, where the computational power is limited.

2.5.5 GoogleLeNet

With a 6.67% top-5 error rate, GoogleLeNet was the winner of ILSVRC-2014. It did not only had a lower top-5 error rate than VGG-Net, it also had significantly less number of parameters in the network. Comparing to VGG-Net which had 140 million parameters, GoogleLeNet only had 4 million [41]. This was due to the innovative design used in GoogleLeNet. While VGG-Net used a 'simple and deep' design, GoogleLeNet was 'deep' but not simple. Instead of stacking convolutional layers on top of each other like previous CNN models, GoogleLeNet introduced something called an 'Inception module'. In section 2.6, I will be looking into the 'Inception module' in more details.

For my project In my project, I will be implementing a variation of GoogleLeNet. This is because this CNN model is very computationally efficient while being accurate at the same time.

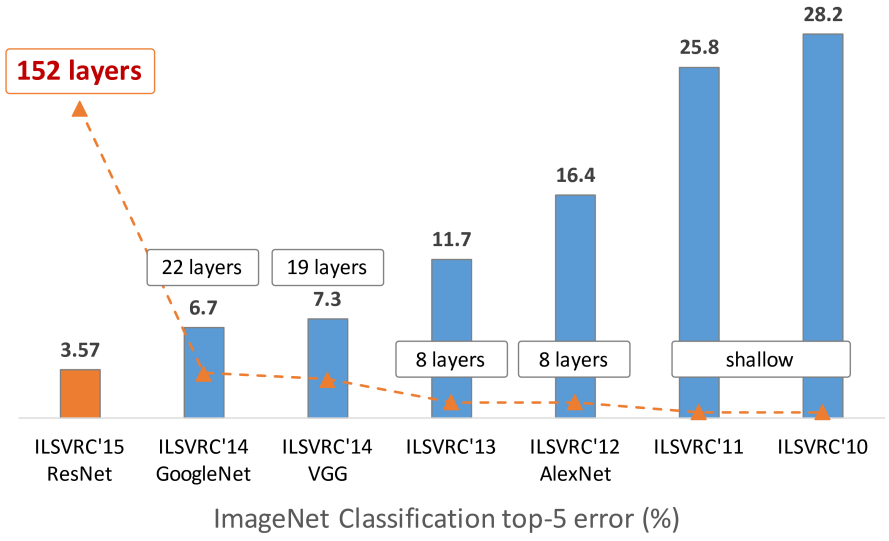


Figure 9: Chart showing the correlation of depth of CNN and accuracy, from [13]

2.5.6 ResNet

ResNet won the ILSVRC-2015 by having a top-5 error rate of 3.57% [12]. According to [37], this is a lower error rate than human performance. With 152 layers in the network, this again proved the importance of depth to the accuracy of CNN. This is further illustrated in figure 9. Another secret behind why this network performed so well was because of the use of a novel technique called 'Residual block'. This is illustrated in figure 10.

In a traditional neural network, the original input, x, goes through the weight layer and is transformed into $F(x)$. The original input, x, is not preserved. However, in a residual block, the original x is preserved. x is transformed into $F(x) + x$ and therefore preserving the original x. The reason behind why this technique was used is that it is easier to optimise residual mapping, $F(x) + x$, then unreferenced mapping, $F(x)$.

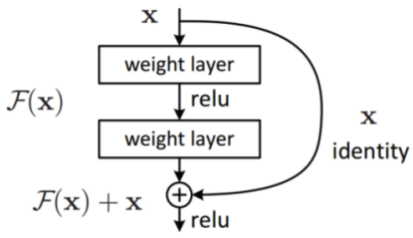


Figure 10: Visualisation of a residual block in ResNet, from [12]

Looking at the accuracy of ResNet, it was clear that 'Residual block' had a positive impact to the accuracy of a CNN. It is therefore no surprise that newer state of the art CNN models, like Inception-ResNet, also incorporates residual blocks in the network's architecture [42].

For my project In my project, I will not be using ResNet. Although the accuracy of the network is outstanding, the computational cost is too high. Under a smartphone setting, where power is limited, I do not believe that the trade off between performance and accuracy is worth it.

2.5.7 Conclusion

As described in section 2.5.2 - 2.5.6, all the winners of the ImageNet Large Scale Visual Recognition Challenge from 2012 onwards used some sort of CNN. It is obvious that CNN is the state of the art image recognition technique and I should be implementing it in my application. Among all the CNN models I looked into, GoogleLeNet, that was discussed in 2.5.5, looked to be the one that is most viable for me to implement into a mobile setting due to its low number of parameters. Therefore, I will be looking into it in more details in the next section.

2.6 Inception Architecture

As discussed in section 2.5.5, GoogleLeNet was chosen to be the CNN of my choice for the application. In this section, I will be looking into the architecture of the network in more details. The main difference between this network and other CNN is that instead of stacking convolutional layers on top of each other, GoogleLeNet stacks 'Inception module' on top of each other. Therefore GoogleLeNet is often simply referred to as 'Inception' [41]. Since the introduction of 'Inception module' in 2014/2015, there are multiple design changes made to it. For example, the newer version of Inception incorporated Residual block to improve its accuracy [42]. In this section, I will only look into the original Inception module that was introduced in the paper 'Going Deeper with Convolutions' [41], as all the newer version of Inception modules are all built upon the original one.

As can be seen on figure 11, the Inception module consists of multiple convolution layers with different filter sizes. The reason behind this is that `1×1`, `3×3` and `5×5` filters all have their merits. `1×1` convolutions is able to cover more concentrated clusters while `5×5` convolutions can cover more spread out clusters with `3×3` convolutions covering clusters in between. Therefore, using all 3 types of filter will allow more features from different scales to be extracted.

The downside of using all 3 convolutional filter is that it will lead to a tremendous increase in computational cost. This issue is solved by the use of `1×1` convolutional layers. As seen in figure 11, `1×1` convolution layers are added before the `3×3` and `5×5` convolution operations. The purpose of these `1×1` convolutional layers is to reduce the dimension of the input before passing into other convolution layers. Moreover, since ReLU is performed after every convolutional layer, adding additional `1×1` layers in the network will also help the network performance. The reason behind why ReLU improves performance of the network was briefly discussed in section 2.4.1.

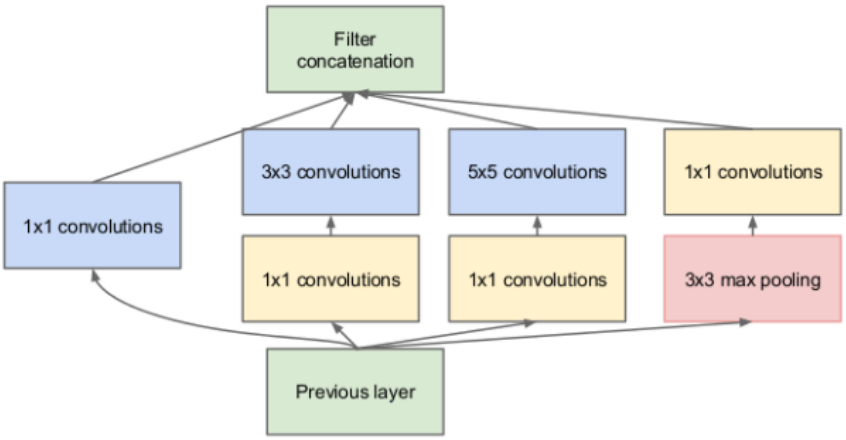


Figure 11: An Inception module, from [41]

The design of the Inception module in general took a lot of inspiration from a paper called 'Network in Network' [26].

2.7 CNN in mobile application

As I am implementing my CNN into a mobile setting. I decided to look into details on different researches that has been done and technologies that are available to make CNN run more smoothly on an offline mobile setting.

Memory storage, computational power and energy are all bottlenecks that prevent most CNN models to be implemented into smartphones in an offline setting. Therefore, a trade-off between depth of neural network (which often equates to accuracy), and speed has to be made when designing a CNN that is capable of running on smartphones. Researches have been done on both hardware and software side in an attempt to increase viability of running a CNN in a smartphone setting.

2.7.1 Hardware

In hardware side, the biggest smartphone processor company, QualComm, has developed processors that are built to optimise for TensorFlow, a machine learning framework that is developed by Google [34]. In my project, I will be using TensorFlow as the framework to integrate the CNN into my application. Nvidia has also developed mobile GPUs that are capable of running AlexNet in Caffe [46, 2]. However, these mobile GPUs are only available on high-end tablets and not on smartphones at the moment.

With the trend of hardware improvements on smartphones, there is no doubt that the bottleneck for running CNN in smartphones will gradually disappear.

2.7.2 CNN compression

One of the ways to speed up CNN and reduce its memory overhead is to compress it. In a paper by Wu et al. [47], authors were able to quantize different CNN models and compress it by 15-20 times. As result, the computation speed of the network was sped up by 4-6 times while only sacrificing around 1% of its accuracy. However when a quantized AlexNet was tested on a *Huawei Mate 7* Android smartphone by the author, it still has an inference time of 0.95s, which was far from ideal for a real world application.

There are also many other techniques that had shown to be able to compress CNN e.g. using Hashing [5], Tensor decomposition [44] and packing CNN in frequency domain [45]. However, a lot of the compressed models are not publicly available.

2.7.3 Novel architecture

Besides compressing the CNN, interesting architecture has also been proposed to improve CNN memory overhead. SqueezeNet is one of the examples. It has a similar accuracy as AlexNet while having 50 times fewer parameters and being 510 times smaller in size (only 0.5mB) [16]. The idea behind SqueezeNet is to limit the use of `3×3` convolution layers and replace it with `1×1` convolution layers instead. Fully-connected layer in the network was also skipped out in order to reduce the memory overhead.

2.7.4 Software

In software side, using software acceleration to speed up CNN is also a possibility. DeepX is a software accelerating framework that allows user to make CNN more practical in smartphones [23]. Novel algorithms called Runtime Layer Compression and Deep Architecture Decomposition are used for inference phase compression and maximising resources utilisation by taking advantage of the repetitive nature of CNN models. However, I was unable to find the working version of DeepX online, therefore I was not able to test it out.

CNNdroid is another solution that can speed up CNN in a mobile setting [24]. It is an open source Android library that supports a variety of networks. According to the author, it can achieve up to 60 times speed up and 130 times energy saving on a mobile device by using a GPU-based acceleration in the mobile devices. Although the speed up is impressive and the library is available through github, CNNdroid does not support any of the Inception models that I was planning to implement into my project.

2.8 Conclusion

It has been proven time and time again through the Image-Net challenge that CNN is the state of the art image recognition technique. Looking at studies that used conventional methods for food recognition and ones that used CNN, it showed that the superiority of CNN holds true in the food recognition task.

Therefore CNN is the chosen technology to carry out the image recognition task for my project.

With so many different CNN models available, choosing the perfect CNN model to use for my project can be quite a challenge. From my research above, GoogleLeNet looked to be the model which is most suitable for my task due to its computational efficiency. In section 3.2, I will discuss further on the reasonings behind how I choose a suitable CNN model for my project.

3 Proposed Methodology for Food Recognition

3.1 Introduction

As discussed throughout section 2, it was clear that CNN is the state of the art image recognition technology. Therefore, I will be using CNN in this project for the food image recognition part of the application. In this section, I will be talking about the thought process behind how I chose my CNN model and how I implemented it. Since part of this was already discussed in section 2.5, some part of the explanation below will be fairly brief.

3.2 Choosing the CNN model

There are a number of decisions that have to be made in order to determine which CNN model is the most suitable for the application.

3.2.1 Online vs Offline

First, I have to decide whether to host the CNN model in a server or locally in user's smartphone.

If the CNN is run on a server, the computational cost of the CNN model will be less of a concern, as one can assume that a server will be able to handle more computationally intensive tasks. This means that I will be able to choose the state of the art CNN in terms of accuracy without sacrificing a lot of speed. However, this also means internet connection from the user is necessary at all times if they want to use the application.

On the other hand, if the CNN model is run offline, user will be able to use the application at all times regardless of their internet connection. The down side is that CNN models that are more computationally intensive are not viable under this environment. This is because the computational power of smartphone is scarce. A single image passing through the CNN may take a few seconds, or even causes the phone to crash. Moreover, size of the CNN model can get very big and will uses a lot of the user's phone memory. This is something that may deters people from using the application.

After some considerations, I decided to run the CNN offline. This is because there are workarounds for the downsides for running the CNN offline but the cons for running the CNN online is impossible to avoid. As discussed in section 2.5, there are CNN models available that has a high accuracy that have a relatively low number of parameters in the model. The memory footprint of the CNN can also be mitigated by quantizing and stripping layers of the model, as discussed in section 2.7. Moreover, having a CNN that runs offline will set my application apart from existing food logging applications that uses CNN, as all of them requires internet connection to function properly [17, 39].

3.2.2 Pre-trained vs Fine-tune vs Train from scratch

As discussed in section 2.4.4, training a CNN is essential for it to function properly. A lot of computational power, data and time is needed to train a CNN from scratch. Although it would be a good experience to train the CNN from scratch, the time and resources for doing this project is limited and therefore it is not realistic for me to do that in this project.

Fine-tuning a pre-trained CNN for my task is also another option that I considered. Although this is a much more realistic option than training the network from scratch, I am still not 100% sure that I will have the time to do it. Moreover, the sources for new training data is also a concern. Therefore, I decided to leave this as one of the future works to be done for the project.

At the end, I went for the safest option. I decided to go for a simple pre-trained CNN model.

3.2.3 Trained data set

There are a lot of pre-trained CNN models available that are trained with different data sets. One of the most popular one is the Image-Net challenge data set [36]. There are 1,281,167 images across 1000 different labels for this data set, ranging from Egyptian cat to suspension bridge. Another data set that looked to be relevant to this project is called 'Food-101' [3]. This data set contains 101,000 images across 100 labels of different food. Judging by the name of the data set, it looked like CNN models that are pre-trained with Food-101 data set is what I am looking for. However, upon further inspection on the labels for both data sets, the Image-Net challenge data set contained significantly more labels for fruits and vegetables comparing to the Food-101 data set. Food-101 data set mostly contained dishes and not individual food item.

On top of looking at the labels of the Image-Net challenge dataset, I also looked into the number of images for the type of labels that I am interested in. This is because if the number of images for the fruits and vegetables labels are significantly less than the average number of images per label within the dataset, it may lead to a worse than average accuracy during the CNN classification. On average, there is around 1,281 images per label in the Image-Net challenge dataset. Among the 22 labels that I am interested in, there are on average 1192 images per label. Although the labels that I am interested in has slightly less number of images per label, the difference should not be significant enough to cause a noticeable difference in accuracy on classification of fruits and vegetables and other non-related labels in the CNN.

In the end, I decided to go for CNN models that are pre-trained in the Image-Net challenge data set. The fact that there are more models that are trained with Image-Net data set available comparing to the FOOD-101 data set also helped me made that decision.

3.2.4 Final Decision

To summarise, the CNN model I was looking for has to be able to run offline and was pre-trained using the Image-Net challenge data set. After searching around, I found the *Inception 5h* model, which fits all of my needs described above.

Inception 5h is a version of GoogleLeNet that was described in section 2.5.5 and 2.6. There are Android application demos in github using this particular model in an offline environment [43]. Using my personal smartphone for testing (*Xiaomi Mi5*), the CNN in the demo has an inference time of around 200-400ms. The reasonable speed from the demo is the biggest reason why I decided to settle for this model. The fact that the file for this CNN model is only around 50MB is also a bonus.

3.3 Implementation

As mentioned in the previous paragraph, there are demos of Android application which implemented the *Inception 5h* model available in github [43]. I chose to implement my CNN model the same way the demos did, using the TensorFlow Android Inference Interface. This decision lowered the risk of my project significantly.

First step of implementing the CNN model required me to import the TensorFlow Java API into my Android project. As the TensorFlow Java API is available in the JCenter remote repository, all I had to do was to put a single line in my project's *build.gradle* file to import it. Since there is no official documentation on the Inference Interface other than the demo application itself, the next step for me was to study the source code of the demo to understand which functions in the API was relevant for implementing the CNN and what parameters they take in. After studying the source code, I put the relevant code in the demo into my own project. These code can be seen in *Classifier.java* interface and the *TensorflowImageClassifier.java* class. As the CNN takes in images with the size of `224×224`, I had to modify my code so the size of the image output from the user's phone camera matches it. The last step is the interpret the results outputted by the CNN in a meaningful way. From the code I imported from the demo, the result outputted by the CNN is a list of items. The number of items on the list is determined by the following equation.

$$numberofitems = max(n, 3)$$

with n being the number of labels that has a higher confidence level (probability) than the threshold, which is 0.1 in this case. The list is also ordered by confidence level in a descending order.

To interpret the list in a meaningful way, I followed the flowchart as shown on figure 12. To determine 'Is it food?' in the flowchart, I had to check if the name of the item matches any 'Food Name' in the 'Conversion' table (table 7 in the database. I will talk more about the database in section 5.2.

3.4 Challenges

Originally, the CNN implemented into my application had an inference time of around 1000-1500ms. Comparing this to the demo which used the exact same CNN model, the demo only had an inference time of 200-400ms. After some investigation, I found out that the significantly longer inference time was because I was reloading my CNN model every time an image was taken by the user. To decrease the inference time, I modified my code so that the CNN model do not have to be reloaded again after it has been loaded once.

3.5 Conclusion

At this point, after the CNN is implemented into the application, the user is able to register their fruits and vegetables consumption solely by photos taken by them. However, there are several limitations that is preventing this to be useful in real life situation. The most obvious one being that the application, in its current state, will only be able to identify one object from the user's camera input. In the next section, I will be talking about how I developed different strategies to overcome this limitation.

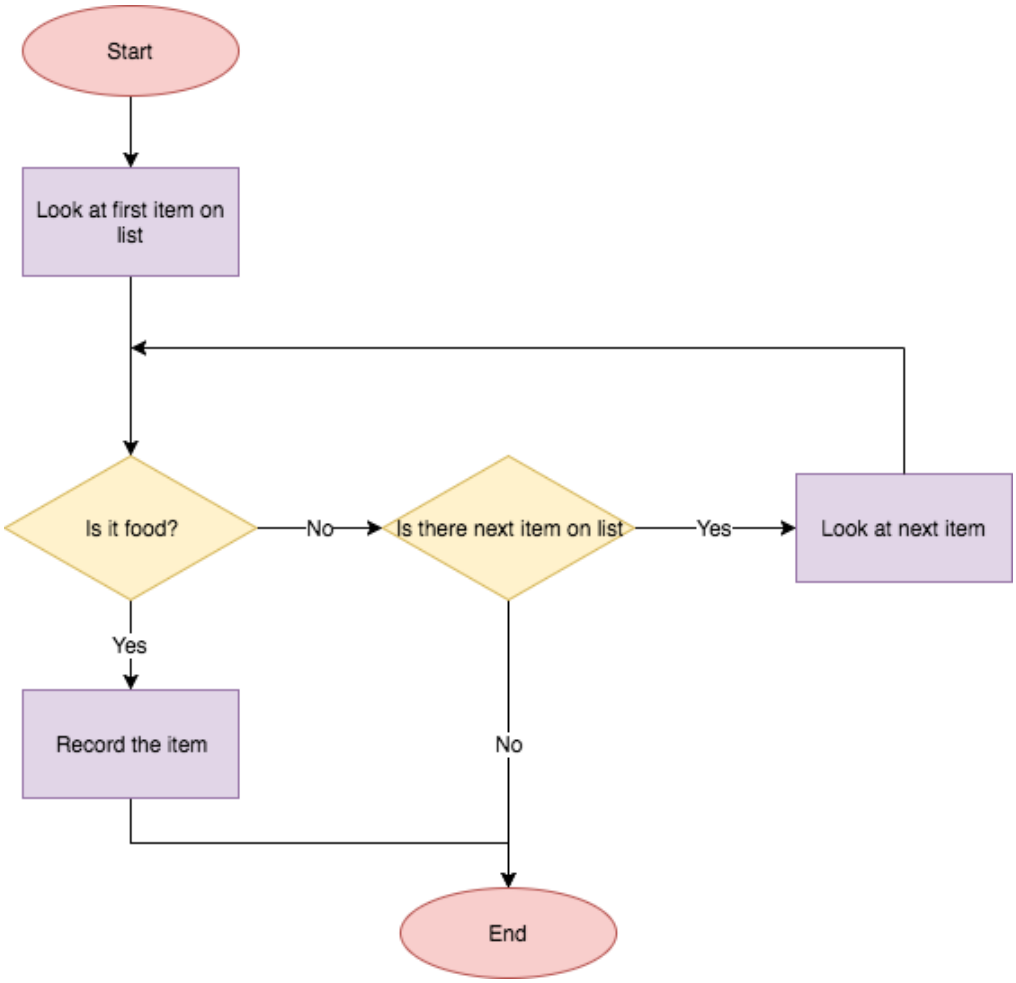


Figure 12: Flow chart for interpreting the result set of CNN.

4 Proposed Methodology for Identifying Multiple Objects in a Single Image

4.1 Introduction

In the following sections, I will describe different strategies that I developed in attempt to identify multiple objects in a single image. The result of the different strategies will be detailed in section 4.8.3 and the evaluation of the strategies will be detailed in section 4.8

Since computational power is scarce in smartphones, all the strategies developed has a large window size and a big step size. This ensures that the total number of windows will not be too high, therefore keeping the total inference time of the CNN to a minimum. Moreover, since the camera resolution varies for different smartphones, to make the result and the computational speed more consistent, all the strategies will result in constant number of windows regardless of the original image size.

It should be noted that the images used below to demonstrate different strategies are not a typical scene expected from the user's input. The images are just there to illustrate the strategies.

4.2 Quarter Crop

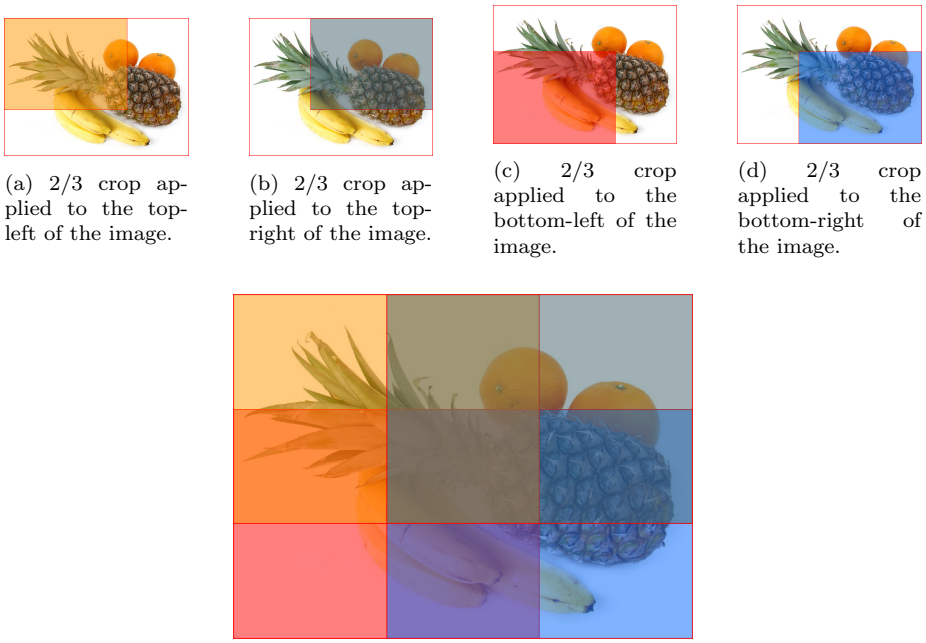
The idea of this strategy is to divide the original image into 4 smaller non-overlapping images (top-left, top-right, bottom-left and bottom-right). As illustrated in figure 13, the window size of this strategy is `width/2`×`height/2`.

Figure 13: Quarter Crop applied to an image. The 4 different windows are marked by different colours.

Since most phones will capture a rectangular image by default, the windows will also capture a rectangular image. As the CNN is only able to take in images with the size of `224`×`224`, it is necessary to resize the image before passing it into the CNN. This means that the image ratio will be changed during the resizing process, and the image that will be passed into the CNN will be slightly stretched. The wider the user's image is, the more stretched the processed image will become.

4.3 2/3 Crop

Similar to the strategy in 4.2, this strategy also divides the original image into 4 smaller ones. The only difference is that the window size is `width*2/3`×`height*2/3`



(a) 2/3 crop applied to the top-left of the image.

(b) 2/3 crop applied to the top-right of the image.

(c) 2/3 crop applied to the bottom-left of the image.

(d) 2/3 crop applied to the bottom-right of the image.

(e) 2/3 crop applied to the whole image. Overlapped regions appeared darker.

Figure 14: Visualisation of 2/3 Crop applied to an image.

instead. This is illustrated in figure 14.

As shown on figure 14e, the increased window size led to a few overlapping regions by different windows in the center of the image.

Same as strategy in 4.2, the image also has to be re-sized before passing into the CNN, therefore the image being passed into the CNN will also be slightly stretched.

4.4 Quarter Crop 2

Similar to strategies in 4.2 and 4.3, this method also produce 4 smaller images. What sets this strategy apart is that it uses a 1:1 window. This means that the ratio of the window image do not need to be altered before passing into the CNN. Comparing the result of 4.2 and 4.3 with this strategy may provide interesting insight on if slightly stretching the image will have a significant affect on the CNN result.

Unlike in 4.3, this method will not produce a center region where all 4 windows overlap. This is because the window size used in this method is determined by the longest edge of the image. For example, if the image has a longer width than the height, then the window size is `width/2`×`width/2`. On the other hand, if the image has a longer height than width, then the window

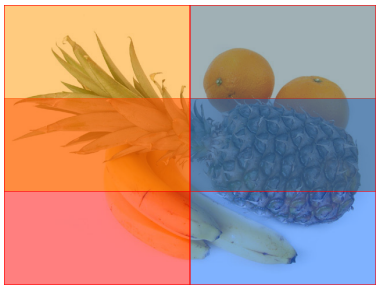


Figure 15: Quarter Crop 2 applied to an image. Overlapped regions appeared darker.

Figure 16: 12 Crop applied to the whole image. 12 different windows are indicated by 12 different colours.

size is `height/2`×`height/2`.

Depending on the original image ratio, there will be some overlapping from 2 of the windows. This is illustrated in figure 15, where this strategy was applied to a 4:3 image.

4.5 12 Crop

In this strategy, images taken by the user is first re-sized into `896`×`672`. There are 2 main reasons behind why the image is re-sized into this specific size. Firstly, from my personal testing using my own smartphone (*Xiaomi Mi5*) and through the emulator (*Nexus 5X*) in android studio, the image captured through the application has a 4:3 ratio. In order to avoid stretching the image, a decision was made so that the re-sized image also has the a 4:3 ratio. The second reason why the specific size was chosen is because the CNN takes in image of size `224`×`224`. To avoid rounding error and complications, it makes sense intuitively to use a windows size of `224`×`224` and re-size the width and height of the image to a factor of `224`. As shown on figure 16, this method will produce 12 non-overlapping images.

This strategy was chosen to see if a smaller window size will be able to extract more useful information from the original image.

4.6 12 Crop Overlap

Similar to strategy described in 4.5, images taken by the user is also re-sized to `896`×`672` and a window size of `224`×`224` is also used.

However, the step-size used in this strategy is different. This method used a step-size of `112`, which is half the window size. As result, this strategy will produce 35 different images. The reason behind this strategy is to compare with 12 Crop strategy in 4.5 to see if overlapping windows will result in a higher accuracy.

4.7 Summary Table

Strategy Name	Window Size	Windows Overlap	Total Windows	Change Ratio
2/3 Crop	597×448	yes	4	yes
Quarter Crop	448×336	no	4	yes
Quarter Crop 2	448×448	yes	4	no
12 Crop	224×224	no	12	no
12 Crop Overlap	224×224	yes	35	no

Table 1: Summary of all the strategies, assuming that the original image has a size of `896×672`.

4.8 Evaluating different strategies

4.8.1 Images tested

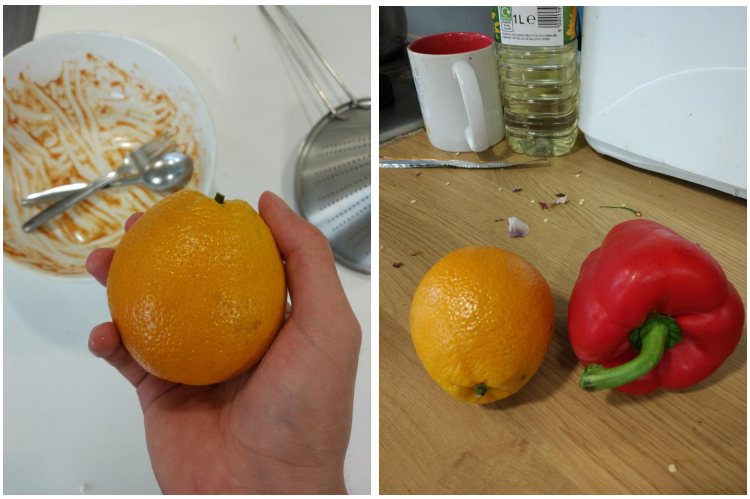
To find out the accuracy of different strategies, I first have to find some images to test it on. Although 'Google images' contains a wide range of images, most of them are very 'stock photo' like. As it is unlikely that the user of the application will take photo of such high quality on their own, I decided to take my own photos in order to get some test images that will more accurately present what I expect the photos taken by the user will look like. Since it is impractical for me to get all 22 food items that can be recognised by the application, I decided to focus on testing only a few different food but in a wide range of environments. Orange, bell pepper and banana were the 3 types of food I chose to test the accuracy of the strategies on.

In total 54 images were tested. These images can be split into 2 categories. 39 of these images are in Category 1, with 13 images for each type of food. The remaining 15 images are in Category 2. Category 1 consists of images that only contains one type of food. Category 2 consists of images that contains multiple types food items. In figure 17 you can see an example of image for each of the category. As you can see, I tried to make the images more realistic by giving them a 'noisy' background.

4.8.2 Defining a way to interpret the result

To evaluate the result of different strategies, I first have to define a systematic way to interpret the results. To keep things simple and consistent, I decided to use a similar way of interpreting the result as mentioned in section 3.3. The flowchart of this can be seen on figure 12. Each window from the strategy will go through the flowchart and if the item has already been identified in previous windows, then it will be ignored.

One change I made from the previous CNN implementation in section 3.3 was that I changed the confidence level threshold from *0.1* to *0.3*. The reason behind why I chose a higher threshold is because the window size is smaller and therefore the object of interest should take up more space of the window and



(a) Example of image for category 1 (b) Example of image for category 2

Figure 17: Example of images for each category

more likely to be the 'center piece' of the image. A higher threshold will reduce the likelihood of the wrong/uncertain label showing up as part of the results. This might also mean that strategies that produce a smaller window size may perform better when it uses a confidence threshold that is higher than *0.3*. To see if that is true, I tested all the images twice, once using a *0.3* confidence threshold and once using a *0.5* confidence threshold.

To determine if the result is accurate or not, I looked at the items recorded through the flowchart (figure 12) and compared it to the actual items in the image. In cases where there are extra items recorded that doesn't exist in the image e.g. a picture of orange is identified as orange & lemon, it will still be considered as inaccurate.

4.8.3 Result of all the strategies

Strategy Name	Accuracy of Images in Category	
	1	2
No Strategy	25.6%	0.0%
2/3 Crop	74.4%	46.7%
Quarter Crop	48.7%	33.3%
Quarter Crop 2	51.3%	46.7%
12 Crop	56.4%	26.7%
12 Overlap	66.7%	60.0%

Table 2: Table showing the highest accuracy of different strategies on images of different categories using a threshold of 0.3 or 0.5

Strategy Name	Accuracy of Images in Category	
	1	2
No Strategy	25.6%	0.0%
2/3 Crop	74.4%	46.7%
Quarter Crop	48.7%	33.3%
Quarter Crop 2	51.3%	46.7%
12 Crop	53.8%	26.7%
12 Crop Overlap	43.6%	40.0%

Table 3: Table showing accuracy of different strategies on images of different categories using a threshold of 0.3

Strategy Name	Accuracy of Images in Category	
	1	2
No Strategy	17.9%	0.0%
2/3 Crop	61.5%	20.0%
Quarter Crop	33.3%	13.3%
Quarter Crop 2	41.0%	20.0%
12 Crop	56.4%	20.0%
12 Crop Overlap	66.7%	60.0%

Table 4: Table showing accuracy of different strategies on images of different categories using a threshold of 0.5

4.8.4 Evaluating the results

Judging by the results in table 2, it looked like that 2/3 Crop and 12 Crop Overlap strategy are two of the better performing ones. Below, I will be comparing the results of different strategies by looking at different parameters and attempt to identify which factors contributed to the difference in accuracy among different strategies .

Comparing window number Unfortunately, there is simply no easy way to determine if window number had a significant effect on the accuracy. This is because window number is the uncontrollable outcome of changing the window size and the step-size (overlap) of the windows. Therefore rather than investigating if the number of windows had an impact on the accuracy, one should be looking at the window size and the step-size (overlap) instead.

Comparing strategies with different window size Although one may expect that a smaller window size will be more accurate at identifying multiple objects within the image, result showed this was not the case. By simply comparing the accuracy of strategies with different window size in table 2, there was no obvious relationship between window size and accuracy. Even though 12 Crop Overlap strategy performed significantly better than other strategies on

category 2 images, 12 Crop strategy did not. This suggested that the window size may not be the reason why 12 Crop Overlap has a high accuracy, but rather the overlapping between the windows. I will look into this in a later section.

Despite the lack of relationship between window size and accuracy in table 2, comparison of the results in table 3 and 4 showed that there is some relationship between accuracy and confidence threshold for strategies with different window size. I will be discussing this below.

Correlation between window size and optimal confidence level In section 4.8.2 I suggested that the strategies with a smaller window size may perform better when the confidence threshold is higher. Result from section 4.8.3 suggested that there may be some truth to this assumption. As you see by comparing table 3 and 4, the accuracy of the 12 Crop Overlap strategy is significantly higher when the confidence threshold is increased from *0.3* to *0.5*.

The opposite also seems to be true as all the strategies with a bigger window size has a significant drop in accuracy when the confidence threshold increases from *0.3* to *0.5*. One of the possible reasons is that the bigger window size means that it is more likely the window that contains the food also has more noise in it, therefore lowering the CNN's confidence level for the classification.

However, it is interesting to note that 12 Crop strategy did not follow this trend. Although the accuracy for category 1 images slightly improved when the confidence level is increased, the accuracy for category 2 images did not. Moreover, due to the low number of images tested, the relationship between window size and optimal confidence level may not be as clear as it seems.

Comparing overlap Results from 2 showed that strategies with overlapping windows generally has a higher accuracy. Quarter Crop 2 and 12 Crop Overlap both had a higher accuracy than its non-overlapping counterpart, Quarter Crop and 12 Crop. The trend is even more obvious when the strategies are tested on images from category 2. However, one could argue that the comparison between Quarter Crop and Quarter Crop 2 may not be a good indicator if overlapping improves the accuracy or not, as the window size of the 2 strategies are slightly different.

Accuracy of the 2/3 Crop strategy from table 2 showed that it is one of the best performing strategies, with the highest accuracy for category 1 images and the joint second highest for category 2 images. This strategy also happened to be the only strategy to have all of its windows overlapping. This further indicated the importance of overlapping windows for the accuracy of the strategy.

To further understand and prove the significance of overlapping of windows on accuracy, more strategies with different step-size has to be developed and tested. This is one of the future works for this project.

Comparing change in image ratio When developing my strategies, I made a point to develop strategies that do not change the aspect ratio of the input image before passing it into the CNN. Looking at result, this variable did not

seem to have an impact on the overall accuracy of the strategy. However, similar to the number of windows of a strategy, the change in image ratio is also dependent on the window size and the overlap of different windows. Therefore, it is difficult to say for certain if the change in image ratio really had an impact on the accuracy or not.

Overall successfulness of the strategies All the strategies are originally developed in mind to allow the CNN to classify multiple objects in a single image. However, looking at the results of category 2 images from section 4.8.3, the accuracy leaves a lot to be desire. A lot more work still has to be done before the CNN can reliably classify multiple objects in a single image.

On the bright side, the strategies significantly improved CNN classification on images that has a noisy background. Result in section 4.8.3 showed that all the strategies showed a significant improvement in accuracy comparing to no strategies at all.

Looking deeper into the result By looking at the CNN result for each image individually, I made some interesting observations.

I found out that it is quite common that orange in the image is mis-classified as 'lemon'. As result, this negatively affected the general accuracy of all the strategies. Thinking about this intuitively, it made a lot of sense why that is the case. The skin texture of orange and lemon are very similar, and the lighting may cause some part of the orange appear more yellow than it is orange. Figure 18 showed two example of images that are classified as 'lemon' by most of the strategies.

Figure 18: Example of images that are mis-classified as 'lemon'

Another interesting observation I made is that strategies that has a smaller window size like 12 Crop and 12 Crop Overlap often identified food item that

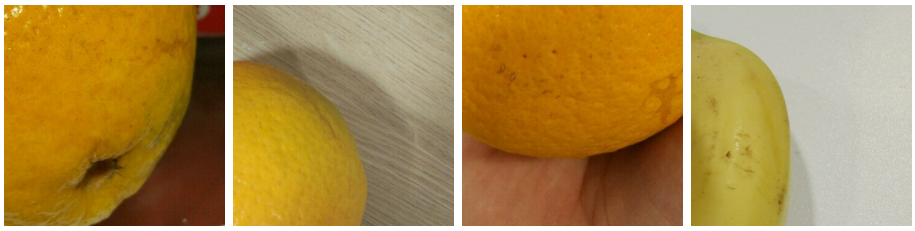


Figure 19: Example of windows that are mis-classified as 'spaghetti-squash'

does not exists in the image. This food item is usually 'spaghetti squash'. Figure 19 shows some of the windows that are labelled as 'spaghetti squash'. Looking at these images, it looked like the edge of a banana and orange are the prime 'suspect' that is causing the mis-classification. The fact that no images of bell-pepper was mis-classified as 'spaghetti squash' may suggests that the combination of the colour and the round edge is causing this issue. It is also interesting to note that while there are an average of 1,281 images per label in the ImageNet challenge dataset, there are only 758 images under the 'spaghetti squash' label. The lack of training images under 'spaghetti squash' label may be one of the reasons why 'spaghetti squash' is more frequently mis-classified in some windows than other labels.

It is also interesting to note that images with bell pepper performed very poorly for Quarter Crop and Quarter Crop 2 strategy. Table 5 below shows the accuracy of all the strategies on images that contains a bell pepper. As you can see the difference in accuracy is very significant. This negatively affected the overall accuracy of these 2 strategies as shown in section 4.8.3 significantly. Unfortunately, I was unable to figure out the reason behind this.

Strategy Name	Accuracy of Images in Threshold	
	0.3	0.5
No Strategy	23.1%	15.4%
2/3 Crop	92.3%	69.2%
Quarter Crop	30.8%	7.7%
Quarter Crop 2	38.5%	23.1%
12 Crop	69.2%	61.5%
12 Crop Overlap	61.5%	92.3%

Table 5: Table showing accuracy of different strategies on images that contained bell pepper.

Caveat

1. As mentioned in section 4.8.4, strategies with different window size have different optimal confidence level threshold. As I have only tested 2 different confidence threshold across all strategies, it is likely that the threshold that I chose is not the optimal one for each strategy. Therefore the accuracy showed in table 2 may not fully reflect how accurate each strategy

can be if the confidence threshold is optimised.

2. Only orange, banana and bell pepper are used in my test images. It is uncertain that if photos of other fruits and vegetables will achieve the same accuracy. As mentioned in previous section, there was a dramatic accuracy difference on Quarter Crop and Quarter Crop 2 strategies for images on bell pepper comparing to other food images that I tested. Moreover, spaghetti squash was also commonly classified in windows that did not contained any spaghetti squash. Therefore it is possible that the accuracy difference may also exist among other types of food.

3. Many of the test images I took are intentionally 'noiser' than what the user's photo may look. Figure 20 showed a few examples of these noisy images. Therefore, it is possible that when the user uses the application, their accuracy experienced may be slightly higher than the results shown on table 2.

4. None of the testing images are represented as a 'dish'. Therefore, it is unclear on how accurate the strategies are on identifying the fruits and vegetables that are part of the ingredients of a dish. However, one can assume that the accuracy on identifying 'dish-like' images will be lower than the accuracy on identifying category 2 images.

5. Although I was able to find some relationship between accuracy and different parameteres in different strategies, the low number of images tested means that the relationship may not be significant. More images, ideally taken by the user, are needed in order to determine if these relationships are significant or not.

Figure 20: Example of noisy images. All these images can be accurately classified by at least one of the strategies.

4.9 Final Decision

After evaluating the results of all the different strategies using 2 different confidence threshold, I decided to implement the 2/3 Crop strategy with a *0.3* confidence level threshold in my final application. Looking at the results from table 2, it showed that this strategy with this particular confidence threshold is the best performing strategy for classifying single type of food (Category 1 images) and is the joint second on classifying images that consists of multiple types of food (Category 2 images). Although one could argue that the 12 Crop Overlap strategy generally has a superior accuracy when taking account into both category 1 and category 2 images, the number of windows 12 Crop Overlap strategy produces is too high. 12 Crop Overlap produces 35 windows in total while 2/3 Crop produces only 4. Using my personal smartphone (*Xiaomi Mi5*) for testing, 2/3 Crop strategy has around 1 second of inference time while the 12 Crop Overlap strategy has around 8-9 seconds of inference time. As it is important for the application to be responsive, 2/3 Crop strategy is clearly the better strategy to implement into my application.

4.10 Conclusion

In conclusion, I developed 5 different slidding window strategies and tested them on images taken by myself. My result and evaluation showed that overlapping of windows is the biggest factor on the accuracy of the strategies. Result also showed a general correlation that strategies with smaller window size benefits with a higher confidence threshold and vice versa. Even though the strategies did not perform particularly well on images that consisted of multiple types of food, all of them showed a significant improvement on identifying single type of food in an image with a noisy background. In the end, I chose to implement the 2/3 Crop method with a *0.3* confidence level threshold into my application as it has the best balance between accuracy and inference time among all the strategies that I developed and tested.

5 Design & Implementation of User Interface and Database

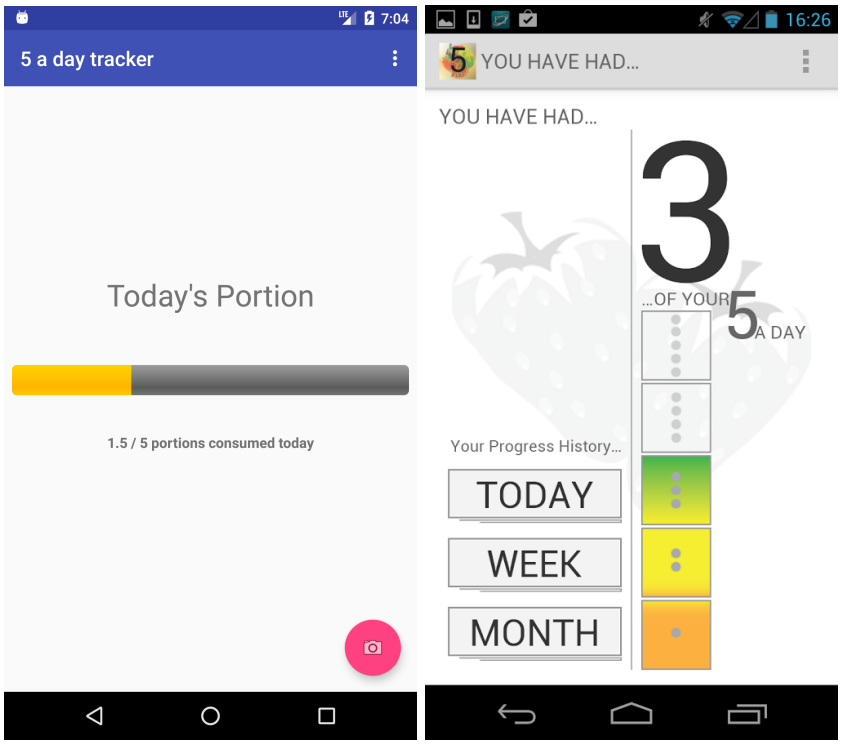
Before making my application, I had to decide on the Android API my application will support. The higher the API level, the more features the application can support. The downside is that older phones may not be able to run it. In the end, I decided to make my application to support any phone that has 23 or higher API. This means only smartphones that has Android 6.0 or newer platform version will be able to run the application. This decision was made because I want to explore on using newer features when developing the application.

In the sections below, I will first talk about the design decisions I made when developing my user interface. Then I will describe the details on how I designed a database to tie everything together.

5.1 User Interface

Before I start designing and developing my user interface, I decided to look into similar smartphone applications that are available in the market and take some ideas from them. One application called *5 A Day* by *Sharp Telecommunication of Europe Ltd* [9] particularly caught my eye due to its simple and visual design.

Similar to [9], I have also decided to implement a 'Daily View', 'Weekly View' and 'Monthly View' for my application. In figure 21 - 23, you can see a side by side comparison of some screens of my application to [9]. I will be talking briefly about the design decisions I made on each 'View' below and how I implemented them. I will also talk about other functions of the application like 'manual input', 'camera input' and notifications.



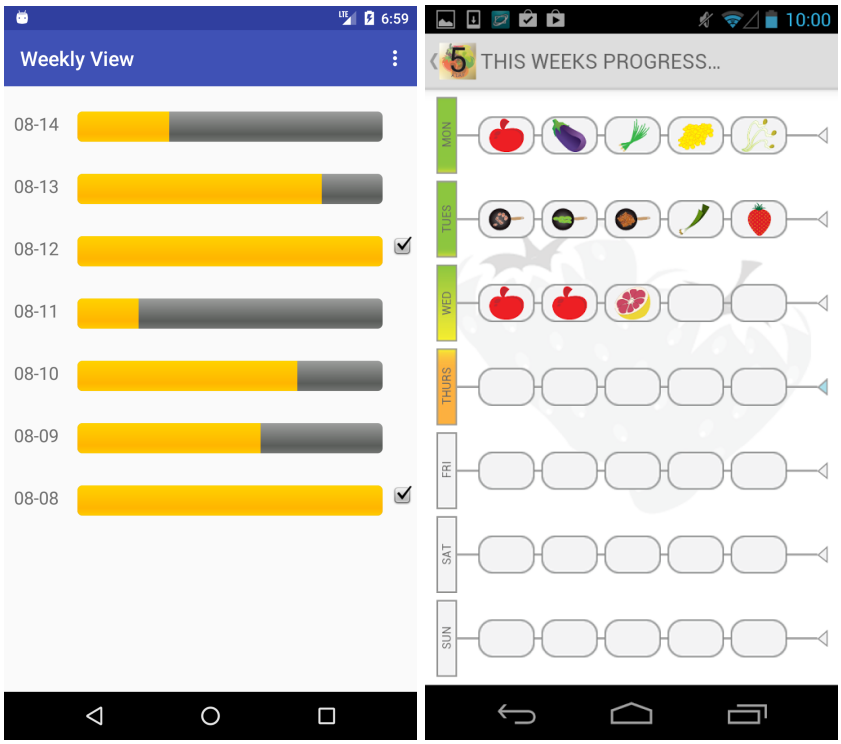
(a) Daily View in my application (b) Main screen from [9]

Figure 21: Day view/main screen of my application and [9]

5.1.1 Daily View

'Daily View' is the main screen the user see when they open the application. This can be seen on figure 21a. The purpose of this screen is to show the progress of the user's consumption for the day and give them an easy access to input new food. The main design decision I made for this view was to use a progress bar to represent how many portions of fruits/vegetables are consumed by the user. This allowed users to be able to have an idea on their '5 a day' progress with just a glance on the screen.

Implementing the progress bar was very straight forward as *ProgressBar* was one of the default widgets that Android supports. The only issue was that the number set for the progress had to be an integer. This means that if user consumed 0.5 portion of fruits/vegetables, nothing will be shown on the progress bar. However, this was easily solved by setting the max value for the *ProgressBar* as 50 instead of 5 and multiplying everything by 10 before setting the value of the *ProgressBar* to get rid of the decimal points.



(a) Weekly view in my application (b) Weekly view from [9]

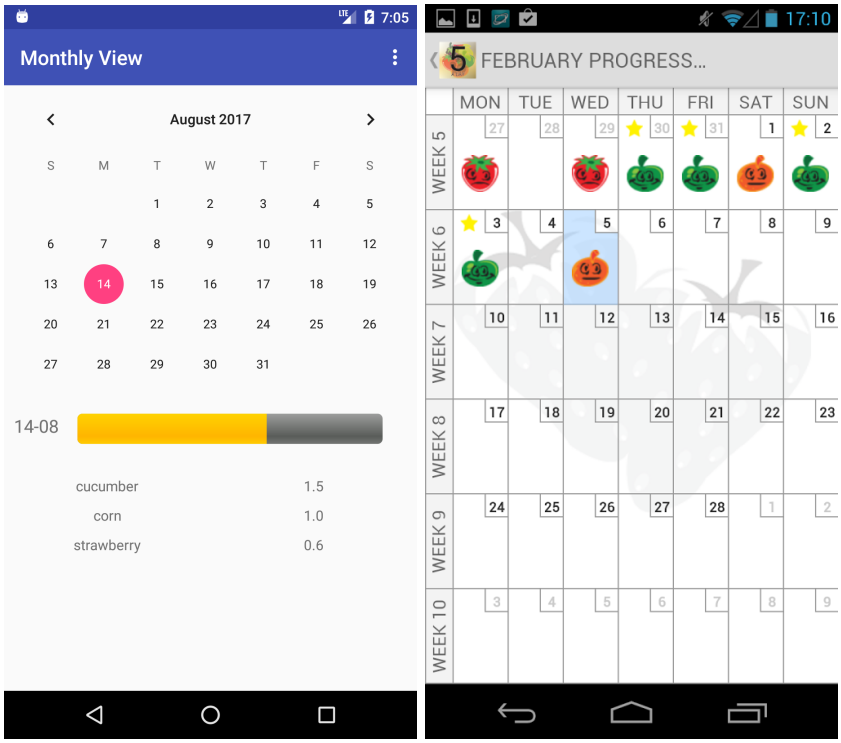
Figure 22: Weekly view screen of my application and [9]

5.1.2 Weekly View

'Weekly View' is a screen that shows the user's consumption record for the past 7 days. This can be seen on figure 22a. Although it would be very visually appealing to implement a similar looking 'Weekly View' interface as shown on figure 22b, there were a few things that prevented me from doing so. Firstly, in the design on figure 22b, the interface did not consider fruits and vegetables that only account for 0.5 portions. And as you can see on figure 22b, it is impossible to represent only 0.5 portions using this interface design. Secondly, implementing such 'graphically intensive' interface will require a lot of artwork. Each fruits/vegetables in the database will require its own artwork. As the focus of this project is to implement a CNN into an application for real world usage, using an extensive amount of time to make the application look more appealing was not on top of my list. As result, I resorted to using a list of *ProgressBar* widgets to represent the total consumption for the past week. I also made it more visual by adding an indicator on the right of it if the user meets their target for that day.

Implementing the list of *ProgressBar* with date and a graphical indicator next to it required the use of a *RecyclerView*. Using a *RecyclerView* allowed me

to show different contents as a list while being able to customise it as well. To implement a *RecyclerView* I had to write my own adapter class that overrides a few methods from the *RecyclerView.Adapter* superclass. I also had to write an inner class that extends *RecyclewView.ViewHolder* for the *RecyclewView* to display the customised list.



(a) Monthly view in my application (b) Monthly view from [9]

Figure 23: Monthly view screen of my application and [9]

5.1.3 Monthly View

'Monthly View' is the view that shows user a more detailed record of their previous fruits and vegetables consumption habit. The adapter for *RecyclerView* that was used in 5.1.2 was also used in this view to display the record for any selected day. This was a design decision just to make the layout of the application more consistent. Figure 23a gives a screenshot of this view.

CalendarView, which is one of the default Android widgets, was used to implement the calendar in the view. I had to set a listener for the *CalendarView* so when the user clicks on a different date, the application will display the record for a different day. One issue that I ran into was that the 'month' that was retrieved through the listener started with 0, meaning that January was 0, February was 1 etc. This was fixed by adding 1 to the month when I

am displaying it. Upon further research, it turned out that it was one of the 'tradition' of Java Date/Calendar API to start counting month from 0 [32], [31].

5.1.4 Manual Input

The screen for manual input can be seen on figure 24. As the name suggests, this screen allows users to enter their fruits/vegetables manually instead of using the CNN. Even though the idea of the application is for the user to take a photo and the application will automatically register their input, I decided that there should be a more 'fool-proof' method as the CNN will not be able to identify every single food. This limitation is further discussed in section 6.2.1. One design decision I made in this screen was to use text input for the user's manual input instead of allowing user to choose from a long list of available options. The reason behind is that this makes inputting much quicker. The downside of this is that if user's input has a spelling error or is not recognised in the database, then their input will not be registered. To prevent this, I enabled auto complete for the user's input.

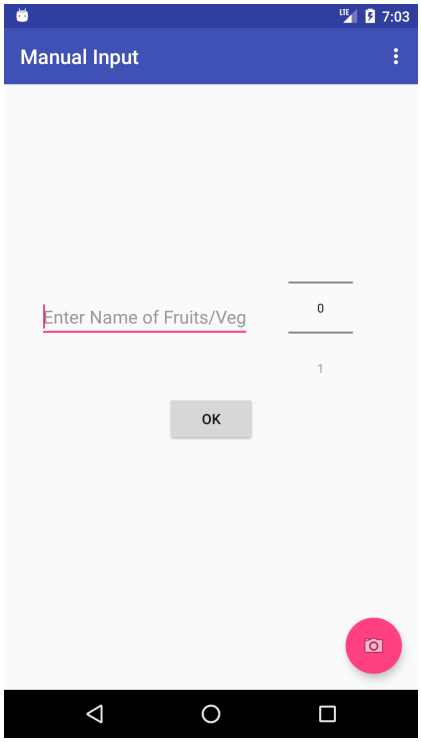


Figure 24: Screenshot of 'Manual Input' screen in my application.

Implementing this feature required the use of *AutoCompleteTextView* widget instead of the more commonly used *TextView*. I had to set an array adapter to the *AutoCompleteTextView*, so when the user types in 'a', the application will automatically display every entry in database that has a word that starts with an 'a' (i.e. it will show both 'apple' and 'green apple'). I also implemented a *NumberPicker* for the user to enter the quantity of the food. This is to prevent users from accidentally putting an unrealistic number and affecting their record.

5.1.5 Camera Input

'Camera Input' is the screen that allows user to take a picture of their food as a quicker way to input their food consumption. This screen is simply a basic Android camera screen.

To implement this, I used the *MediaStore.ACTION_IMAGE_CAPTURE* intent. This is the most basic way to implement a camera into the application.

Although it is possible to access the image taken directly without saving it, the quality of the image that can be accessed through that way is only as of thumbnail quality. As I wanted to do further processing of the image before passing it into the CNN, it is ideal to retrieve the image in a higher quality. To do this, I had to save the picture temporarily instead.

Although there are other ways to implement a camera interface in a more customised and sophisticated fashion using the new android camera 2 API [8], I did not feel the need for doing that as the basic camera is sufficient for the task.

5.1.6 Confirm Input

Since my application is not capable on detecting the quantity of food in the image, (this is discussed more in depth in section 6.2.2), I decided to add a separate screen to allow user to manually input the quantity of their fruits/vegetables after the CNN identifies it. This is shown on figure 25. Moreover, this screen also allows user to change the result of the CNN. This is necessary as it is possible that the CNN may misidentify food from time to time. Having user to confirm the CNN result before inputting it into the database will avoid users from having an incorrect food log unknowingly.

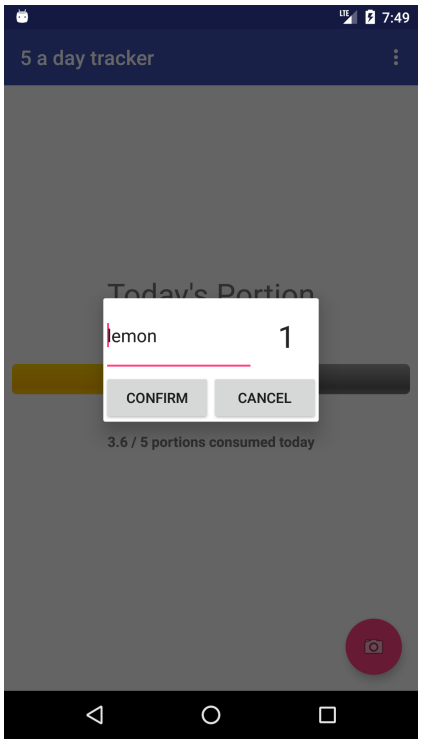


Figure 25: Screenshot showing the 'Confirm Input' popup in the application.

Implementing this pop up screen required the use of *DialogFragment* class. The list of identified food and the quantity was displayed by using *RecyclerView*. To keep track of the changes user made to the food name and quantity in the screen, I had to implement listeners for both the *AutoCompleteTextView* and *NumberPicker* in the *RecyclerView* adapter.

5.1.7 Notifications

Another design decision I made for the application was to enable notifications. I implemented the feature so that the application will notify the user at 12am what their total consumption for the previous day was. The user can then click on the notification, and that will bring them to the 'Month View' to show their detailed record for yesterday.

To implement notification for the application, I had to set up a *BroadcastReceiver* to keep track of the date changes in the user's smartphone. So when the date is changed in the user's phone, the *NotificationManager* class (which is a default Android class) is called and notification will be shown on user's phone.

5.1.8 Conclusion

Since the user interface is not the major part of the project, I did not spend a lot of time customising and polishing it. 'Clean' and 'Easy to use' was what I had in mind when I was designing the user interface, and I achieved that.

I also tried to overcome a few limitations that the application had by adding something extra in the user interface e.g. the 'Manual Input' and 'Confirm Input' screen. In the end, I used different classes and widgets that are available in Android to make the interface more user friendly. For example, I used *AutoCompleteTextView* to prevent spelling error, and used *NumberPicker* to avoid number entry error. I also used *ProgressBar* and *RecyclerView* to make the application more visually appealing.

5.2 Database

In this section, I will be talking about how I designed and implemented the database to make the application I designed above to store user's data. I will also talk about the challenges that I faced in the process.

5.2.1 Choosing the database

There are 2 main directions that I could have gone when choosing what database system to use for my application. I could have chosen a cloud based database or a database that store data locally. Since I aim to make all parts of my application fully functional without the use of internet, I went for the latter choice. Although one could argue that a cloud database could be used for backing up user's data whenever they have access to the internet, this is something that can be done as future work to improve the user's experience. In the end, I chose to use *SQLite* for the application. This is because Android fully supports it with the *SQLiteOpenHelper* class. The fact that Android does not support any *NoSQL* database natively also made my decision on choosing between relational or non-relational database easier.

5.2.2 Designing the database schema

As the database is just a tool to make the whole application functional, it made sense intuitively to design the database according to the functions I designed in the user interface. In the end, I came up with a database schema that only consisted of 2 tables. One is responsible for keeping user's entry, and the other is responsible for converting the quantity to portions for different food. These 2 tables can be seen on table 6 and 7. Using this schema, it is easy to find out the portions user consumed, as it can be done by joining two tables together using

'Food Name' and multiplying the 'Quantity' from 'Record Entry' table and the 'Portion' from the 'Conversion' table.

The 'Conversion' table (table 7) is also responsible for determining if the item is food or not when the application is trying to interpret the results from the CNN. This is done by checking if the item name from the CNN result matches any of the food name in the 'Conversion' table.

Record Entry	
ID	PK
Food Name	
Quantity	
Date	

Table 6: Record Entry table

Conversion	
Food Name	PK
Portion	

Table 7: Conversion table

5.2.3 Implementing the database

To implement the database into my application I had to make my own wrapper that extends the *SQLiteOpenHelper* class. I setup the database tables and insert the initial data through code that I written in a separate class. After the initial setup, I wrote some SQL queries in Java to retrieve and input information from the database. Retrieving information from the database was done by running the relevant query and extracting the information from the cursor that is returned by the database.

5.2.4 Challenges

One of the issues that I faced when I was implementing the database was with the date. Initially, I was using the current date, `date('now')`, that is derived from the *SQLite* as the time stamp. However, the code that I written in the application used the Java current date, `new Date()`, as an argument in the SQL queries to get the information for the current day. This led to cases where if the user input their food shortly after midnight, the 'Day View' will not be updated properly. After some testing, I found out that the Java current date updates at midnight but the SQLite current date updates at a later time. The discrepancy between the date around the midnight period led to bugs when user is using the application around midnight time. As result, I decided to not use the default SQLite time stamp when I am inputting data, and parse the current date from Java as string and use that as date instead.

5.2.5 Conclusion

I designed the database schema by looking at the user interface that I designed earlier to determine what information I have to store. Implementation of the database is a relatively straight forward process due to the support of *SQLite* database on Android environment.

6 Evaluation of the final application

In this section, I will evaluate my project in details. First I will look back at the aims & objectives that I set myself at the start and see if I have met them. Then I will look at the limitations of the application and discuss on possible solutions on how I can overcome it. Afterwards, I will look at the added value this project brings as a whole. Finally, a few different future works will be looked into.

6.1 Aims & objectives evaluation

Looking at the final application, I would say I have achieved all of the objectives I set myself for this project. Below I will go through the list of objectives that I talked about in section 1.3 and evaluate them individually.

6.1.1 Choose and implement a CNN

I implemented the *Inception5h* CNN model into my application got it working with an inference speed of around 200-400ms. Therefore I would say I have met the objective for this task.

What am I proud of Despite the lack of documentations for TensorFlow Android Inference Interface, I was able to understand it by looking through source code of the demo and implement the CNN in my application. I was also able to choose a suitable CNN for the task through the knowledge I gained from my research.

What could have been done better In my application, I only implemented a pre-trained CNN 'straight out of the box'. One thing that I could have done if given more time is to retrain the final layer of the CNN to make it more suited for classifying food. This way I could increase the amount of food that the CNN will be able to classify and also possibly improve its accuracy for classifying fruits and vegetables as well.

Another thing that I could have done is to test the inference speed and accuracy of multiple CNN by implementing them on my smartphone. Comparing different models in practical setting may give me a better perspective on the trade-off between speed and accuracy instead of just comparing the numbers on paper.

6.1.2 Develop and evaluate different strategies on identifying multiple objects in a single image

Five different sliding windows strategies were implemented and tested to determine which one has the best balance between accuracy and computational time for the CNN. The evaluation of the different strategies was detailed in section 4.8. From the results, I would say that I was able to meet the objective that I set.

What am I proud of I was able to come up with strategies with different parameters to allow me to understand why some strategies may perform better than others in some situations.

What could have done better Gathering more testing images is definitely something that I could have done better. As of now, due to the low number of test images, we could not be sure if the difference in accuracy between different strategies are significant or not.

Moreover, I also could have included more types of testing images, especially 'dish-like' ones.

6.1.3 Design a database

Using the native SQLite classes for Android, I was able to implement a database schema that I designed into the application. Therefore I have met the objective for this task.

What am I proud of I was able to utilise what I learnt from my database course and apply it into this task.

What could have been done better The current database is stored locally in the user's smartphone. One thing I could have added is to integrate a cloud database into my application, giving user the choice to backup their input to the cloud.

6.1.4 Design and create an user interface

In my final application, I had a usable user interface for the application that was created through Android Studio. Therefore I would say I have also met the target for this objective.

What am I proud of Before the start of the project, I had little experience on Android development. I was proud of myself that I was able to pick up on Android development in limited time and develop an application using a wide variety of widgets.

What could have been done better Although my final application was functional, it did not have the most eye catching layout. If given more time, I could have created a few custom widgets to make the application stand out more.

Particularly, I would create a custom *CalendarView* for my 'Monthly View' interface in my application, so that user can see if they met their record any day in the past month without having to click through all of the dates individually.

Another thing I could have added is to make a separate screen to show some interesting statistics of user's consumption e.g. their most consumed fruit is the past week, their fruits to vegetables consumption ratio etc.

6.2 Limitations of the application

Although the final application functioned as I hoped, it is obvious that there are several limitations. Below I will talk about some of the major limitations of the application and what could have been done to overcome it.

6.2.1 Unable to detect a wide range of food

The CNN I implemented for the application is only able to detect 1000 different types of objects, with only 22 of them in the 'Fruits and Vegetables' category. Therefore effectively, the application is only able to identify 22 different objects. Many common fruits and vegetables that are consumed everyday (e.g. grapes, carrot) are not included as part of the trained label in the CNN and therefore cannot be classified correctly.

Fortunately, a few of the fruits and vegetables that are not included as part of the trained labels are commonly misidentified as other similar looking food. For example, a bowl of salad is commonly mis-classified as 'guacamole', and apple is commonly mis-classified as 'pomogranate'. Therefore even though some food may be identified incorrectly due to the lack of labels, in some cases, the user's total portion record will still be unaffected by it.

Solution To overcome this limitation, I will have to train a CNN that is specifically trained to identify fruits and vegetables. This means either training a CNN from scratch or retraining a pre-trained CNN, like the current one, so that it is able to identify different types of fruits and vegetables instead of a wide range of objects that we are not interested in. However, within the time-frame for this project, I did not have enough time to carry out either of the tasks.

Among all the major limitations of the application, this one has the easiest solution.

6.2.2 Unable to detect the quantity of food through CNN

As I am using a coarse sliding window approach in attempt to identify multiple objects in a single image, it is impossible to estimate the quanitity of the same type of object in the image. Using figure 26a as an example, if I am using the 2/3 Crop strategy as mentioned in section 4.3 on this image, it is likely that the CNN output for all 4 windows will all be identified as orange. If I again apply the same strategy on an image that contains 2 oranges as shown in figure 26b, the result will be the same. The result output from the CNN will also show 4 windows that have a label of orange. This means that it is impossible, by just looking at the result output from the CNN, to determine the number of food that has the same label in the image.

Solution I came up with 2 solutions to overcome this limitation.

The first one is to ask the user to input the quantity of fruits and vegetables manually. In my current application, user will have to confirm their input and

(a) Image of a single orange (b) Image of 2 oranges

Figure 26: Output of the CNN for both images will be identical.

set the quanitity of fruits/vegetables in the 'Confirm Input' screen (as described in section 5.1.6) after taking a photo of their food. This ensures that despite the limitations as stated above, the user will still be able to keep track of their consumption correctly. Moreover, this confirmation screen will also allow user to correct some food that are wrongly labelled by the CNN. The advantage of this solution is that it is easy to implement and will allow user to have the correct input all the time. The disadvantage is that the whole idea of 'confirmation screen' goes against the idea of everything in the application being 'automatic'.

The second solution is to use image segmentation to segment different objects of interest in the image and identify each segmented area as a different object. There are many different image segmentation techniques and choosing the correct one require prior knowledge of different factors of the image. Although implementing this solution will make the application more 'automatic', there are simply too many unknown factors from the user's photo to make this solution reliable. It is also very hard to separate similar objects that overlaps each other in an image without perfect lighting. It is also possible that there will be over/under segmentation of the image, leading to an unreliable estimation of quantity of food and inconsistent inference time for the application. Moreover, my limited knowledge within the image-processing field also prevented me from implementing this solution.

6.2.3 Unable to detect the size of food

Although this sounds like a big limitation, in reality, the inability to detect the size of food may not have that much of an impact on keeping track of user's '5 a day' consumption. In my application, I assumed all the fruits and vegetables consumed are 'normal' sized. Therefore, unless the user is consuming an abnormally sized fruits and vegetables on a consistent basis, it is unlikely that there will be a massive difference between the actual total portions consumed and the portions consumed recorded in the application. However, in the long run, in order to accurately estimate the consumption of user, detecting the size of food is still necessary.

Solution To overcome this limitation, the easiest way to do this is to tell the user to take the photo with a reference object. This way all the size of the objects in the image can be estimated. However, asking user to carry around a reference object is impractical and makes it troublesome to use the application.

Another possible solution is to ask user to move the camera around the food instead of taking a single image. With the accelerometer in the smartphone, in theory it is possible to find out the distance the user moved around when capturing the food. Using the different positions recorded by the accelerometer, the size of the object captured in the image can be calculated. However, due to the limited time and my limited knowledge in this area I was not able to try to implement this solution within the time frame of the project.

Moreover, both of these solutions will also require the segmentation of the food in the image in order to estimate the volume of the food. As briefly discussed in section 6.2.2, it is very hard to do segmentation without prior knowledge of the image.

6.3 added value

There are several added values to this project. These are listed below:

1. Although there are some existing food logging applications in the market that is able to identify food in photos by the use of CNN [17, 39], as far as I am aware, none of them are able to carry out the food image recognition function offline as they use a remote server for the task. Therefore, the application from the project can be thought of as the first of its kind.

2. To my knowledge, there are no published researches that evaluated the effectiveness of different simple to implement methods (like slidding window strategies) to allow an image classification CNN to identify multiple objects in a single image. Therefore, the result in section 4.8.3 may be useful for individuals who have limited computer vision related knowledge but want to implement a CNN for various image recognition tasks for their own project.

3. The project and the thesis may provide some insight for individuals trying to implement a CNN into their own project but do not have the time and resources to train their own model.

6.4 Future work

There are a lot of further directions that the project can go. Unfortunately, due the limited time frame of the project, it is impossible to explore all of them. Below, I will talk about some of the possible works that could be done to further this project.

6.4.1 Retraining the CNN

One of the most obvious future work for this project is to retrain and fine-tune the current CNN to increase more food related labels. As result, the application should be able to identify more food and therefore making it more usable.

Moreover, as discussed in section 2.3.1, there is a fundamental difference in the filters of ImageNet trained CNN and food images trained CNN. Therefore it will be interesting as a future work to compare the accuracy difference between a ImageNet pre-trained CNN that is fine tuned for food recognition and one that is trained from scratch using food images only.

6.4.2 Integrate other image processing technique to improve performance of the application

As discussed in section 6.2.2 and 6.2.3, the application is unable to detect the quantity and the size of the object. Integrating different image processing techniques may allow the application to overcome this limitation. This is discussed more in depth in section 6.2.2 and 6.2.3.

Moreover, research from [27] have shown that using a bounding box to crop out the food from the image will significantly improve the classification accuracy of the CNN. Therefore pre-processing the image with a bounding box to crop out the food from the background is also one of the future works that can be looked into to improve the accuracy of the application.

6.4.3 Comparing different CNN models and existing image recognition API

Another direction that the project can go is to implement different CNN models offline and compare the inference speed and accuracy between them.

Moreover, there are many different existing image recognition API in the market e.g. Google's cloud based vision API [11], Microsoft's computer vision API [29] and Clarifai [7]. It will be very interesting to compare the inference speed and accuracy difference between the CNN that I used and the existing image recognition API in the market.

6.4.4 Further testing on different strategies and images

Since I only tested 3 of the 22 identifiable food by the CNN, there are still a lot of further testing that can be done. The best way to do this is to gather photos taken by different users and test it on all the strategies. This would allow me to have a better understanding on the different environment that the user will take their photo in and may allow me to optimise or develop a better strategy to improve the accuracy of the application.

Moreover, none of my test images contained 'dish-like' food. Therefore testing on these types of images are also one of the future works to do.

Lastly, to truly understand which strategy is the superior one, significantly more images has to be tested in order to determine if the difference in accuracy is significant or not.

6.4.5 Polishing and expanding the current application

There are multiple things that could be improved upon in order to enhance user's experience. For example, making custom widgets and different layouts to make the user interface less boring and using cloud based data storage to give user an option to back up their input data.

Another direction that the application can go is to expand it into a general food logging application. Using a different CNN that can identify more types of food, the application can be turned into a general food logging application. If the limitation of size estimation, as discussed in section 6.2.3, can be overcome, it can possibly even allow the application to carry out calorie estimation.

However, in reality, accurate food calorie estimation that is entirely automatic is unrealistic. This is because it is often impossible to tell exactly what are the ingredients of the food just by looking at the picture, even by a human.

6.4.6 Creating an API

Last but not least, another direction that the project can go is to write a wrapper for the essential parts of the application and re-code it as an API. This may be useful for someone who is looking to develop some sort of food logging application and want to introduce a bit of novel technologies into it without having to do much work.

6.5 Conclusion

In conclusion, the basic aims & objectives set for this project are met. Although there are some limitations that the application face, given the time frame for the project and my limited knowledge in computer vision, I was not able to overcome all of these limitations. I was only able to briefly look into these subject and have a very rough idea on how these limitation could be overcome.

Looking beyond, there are a wide range of directions this project can develop into. Some of them involve CNN training, some involve image processing and some are more trivial tasks like further testing and improving the user interface.

7 Conclusion

The aim of this project is to create an application that keeps track of user's fruits and vegetables consumption application through images taken by them.

I designed an user interface for my application by looking at similar applications in the market. Using Android Studio, I implemented the design by using different Android widgets. Then, using the skills I gained through the database course that I took, I designed and integrated a database into the application using SQLite. Afterwards, I implemented a convolutional neural network (CNN) into the application to enable image recognition for images taken by the user. To do this, I had to first look into different CNN models and evaluate each one of them to determine which one suits my needs the most. In the end, I chose the model *Inception 5h*, which is a variant of GoogleLeNet. I implemented it into my application by using the TensorFlow Android Inference Interface. The last part of building the application involved trying out and evaluating different strategies to allow the CNN to identify multiple objects inside a single image. The strategies I tried included different variations of sliding windows technique. After my evaluation, I decided to implement the 2/3 Crop strategy into my final application due to its superior accuracy and inference time.

After finishing the final application, I evaluated it by looking at the aims & objectives that I set at the start of the project. I concluded that I have met all the objectives that I set for myself. Although all basic objectives are met, the final application still has some limitations e.g. inability to detect number and size of food. As these limitations are mostly very challenging computer vision tasks, given my limited knowledge around this area, and the limited time-frame for this project, I was not able to overcome it.

References

[1] Morteza Akbari Fard, Hamed Hadadi, and Alireza Tavakoli Targhi. "Fruits and vegetables calorie counter using convolutional neural networks". In: *Proceedings of the 6th International Conference on Digital Health Conference* (2016), pp. 121–122.

[2] Christopher Alicea-Nieves and Camillo J. Taylor. *Caffe Framework on the Jetson TK1: Using Deep Learning for Real Time Object Detection.* [Accessed 10 August 2017]. 2015. URL: `https://www.seas.upenn.edu/sunfest/15-Alicea-Nieves.pdf`.

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

[4] Mei Chen et al. "PFID: Pittsburgh fast-food image dataset". In: *Image Processing (ICIP), 2009 16th IEEE International Conference on* (2009), pp. 289–292.

[5] Wenlin Chen et al. "Compressing neural networks with the hashing trick". In: *International Conference on Machine Learning* (2015), pp. 2285–2294.

[6] Stergios Christodoulidis, Marios Anthimopoulos, and Stavroula Mougiakakou. "Food recognition for dietary assessment using deep convolutional neural networks". In: *International Conference on Image Analysis and Processing* (2015), pp. 458–465.

[7] Clarifai. *Artificial Intelligence with a Vision.* [Accessed 10 August 2017]. 2017. URL: `https://www.clarifai.com/`.

[8] Android Developers. *android.hardware.camera2.* [Accessed 10 August 2017]. URL: `https://developer.android.com/reference/android/hardware/camera2/package-summary.html`.

[9] Sharp Telecommunications of Europe Ltd. *5 A Day.* Version 1.0. 2014. URL: `https://play.google.com/store/apps/details?id=com.sharp_eu.ste.fiveaday`.

[10] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (2011), pp. 315–323.

[11] Google. *Cloud Vision API.* [Accessed 10 August 2017]. 2017. URL: `https://cloud.google.com/vision/`.

[12] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778.

[13] Kaming He et al. *Deep Residual Learning.* [Accessed 10 August 2017]. 2015. URL: `http://image-net.org/challenges/talks/ilsvrc2015_deep_residual_learning_kaiminghe.pdf`.

[14] Department of Health. *Nutritional Aspects of the Development of Cancer*. London: The Stationery Office, 1998.

[15] Hajime Hoashi, Taichi Joutou, and Keiji Yanai. "Image recognition of 85 food categories by feature fusion". In: *Multimedia (ISM), 2010 IEEE International Symposium on* (2010), pp. 296–301.

[16] Forrest N Iandola et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and¡ 0.5 MB model size". In: *arXiv preprint arXiv:1602.07360* (2016).

[17] Azumio Inc. *Calorie Mama AI : Food Photo Recognition & Counter*. Version 5.36.3450. 2017. URL: `https://play.google.com/store/apps/details?id=com.azumio.android.caloriesbuddy&hl=en`.

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

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

[20] Yoshiyuki Kawano and Keiji Yanai. "Real-time mobile food recognition system". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops* (2013), pp. 1–7.

[21] Keigo Kitamura, Toshihiko Yamasaki, and Kiyoharu Aizawa. "Food log by analyzing food images". In: *Proceedings of the 16th ACM international conference on Multimedia* (2008), pp. 999–1000.

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

[23] Nicholas D Lane et al. "Deepx: A software accelerator for low-power deep learning inference on mobile devices". In: *Information Processing in Sensor Networks (IPSN), 2016 15th ACM/IEEE International Conference on* (2016), pp. 1–12.

[24] Seyyed Salar Latifi Oskouei et al. "Cnndroid: Gpu-accelerated execution of trained deep convolutional neural networks on android". In: *Proceedings of the 2016 ACM on Multimedia Conference* (2016), pp. 1201–1205.

[25] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[26] Min Lin, Qiang Chen, and Shuicheng Yan. "Network in network". In: *arXiv preprint arXiv:1312.4400* (2013).

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

[28] Austin Meyers et al. "Im2Calories: towards an automated mobile vision food diary". In: *Proceedings of the IEEE International Conference on Computer Vision* (2015), pp. 1233–1241.

[29] Microsoft. *Computer Vision API*. [Accessed 10 August 2017]. 2017. URL: https://azure.microsoft.com/en-gb/services/cognitive-services/computer-vision/.

[30] NHS. *5 A Day*. [Accessed 10 August 2017]. 2015. URL: http://www.nhs.uk/Livewell/5ADAY.

[31] Oracle. *Calendar (Java Platform SE 7)*. [Accessed 10 August 2017]. URL: https://docs.oracle.com/javase/7/docs/api/java/util/Calendar.html.

[32] Oracle. *Date (Java Platform SE 7)*. [Accessed 10 August 2017]. URL: https://docs.oracle.com/javase/7/docs/api/java/util/Date.html.

[33] Joceline Pomerleau, Karen Lock, and Martin McKee. "The burden of cardiovascular disease and cancer attributable to low fruit and vegetable intake in the European Union: differences between old and new Member States". In: *Public health nutrition* 9.5 (2006), pp. 575–583.

[34] QualComm. *TensorFlow machine learning now optimized for the Snapdragon 835 and Hexagon 682 DSP*. [Accessed 10 August 2017]. 2017. URL: https://www.qualcomm.com/news/onq/2017/01/09/tensorflow-machine-learning-now-optimized-snapdragon-835-and-hexagon-682-dsp.

[35] Caireen Roberts. *HSE 2013: Vol 1— Chapter 7: Fruit and Vegetables Consumption*. [Accessed 10 August 2017]. 2014. URL: http://content.digital.nhs.uk/catalogue/PUB16076/HSE2013-Ch7-fru-veg-com.pdf.

[36] Olga Russakovsky et al. "Imagenet large scale visual recognition challenge". In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252.

[37] Olga Russakovsky et al. "Imagenet large scale visual recognition challenge". In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252.

[38] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[39] National University of Singapore. *Food(lg)*. Version 1.0. 2017. URL: https://play.google.com/store/apps/details?id=com.nusidmi.foodlg&hl=en.

[40] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting." In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[41] Christian Szegedy et al. “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition* (2015), pp. 1–9.

[42] Christian Szegedy et al. “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning.” In: *AAAI* (2017), pp. 4278–4284.

[43] TensorFlow. *TensorFlow Android Camera Demo*. [Accessed 10 August 2017]. 2017. URL: `https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/android`.

[44] Peisong Wang and Jian Cheng. “Accelerating convolutional neural networks for mobile applications”. In: *Proceedings of the 2016 ACM on Multimedia Conference* (2016), pp. 541–545.

[45] Yunhe Wang et al. “CNNpack: packing convolutional neural networks in the frequency domain”. In: *Advances in Neural Information Processing Systems* (2016), pp. 253–261.

[46] Pete Warden. *How to run the Caffe deep learning vision library on Nvidia's Jetson mobile GPU board*. [Accessed 10 August 2017]. 2014. URL: `https://petewarden.com/2014/10/25/how-to-run-the-caffe-deep-learning-vision-library-on-nvidias-jetson-mobile-gpu-board/`.

[47] Jiaxiang Wu et al. “Quantized convolutional neural networks for mobile devices”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 4820–4828.

[48] Keiji Yanai, Ryosuke Tanno, and Koichi Okamoto. “Efficient mobile implementation of a cnn-based object recognition system”. In: *Proceedings of the 2016 ACM on Multimedia Conference* (2016), pp. 362–366.

[49] Shulin Yang et al. “Food recognition using statistics of pairwise local features”. In: *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on* (2010), pp. 2249–2256.

[50] Matthew D Zeiler and Rob Fergus. “Visualizing and understanding convolutional networks”. In: *European conference on computer vision* (2014), pp. 818–833.

[51] Matthew D Zeiler, Graham W Taylor, and Rob Fergus. “Adaptive deconvolutional networks for mid and high level feature learning”. In: *Computer Vision (ICCV), 2011 IEEE International Conference on* (2011), pp. 2018–2025.