# The Campus Food Classifier

Liam Donnelly

April 2020

Final Year Project Report

1799798

Supervisor: Harish Tayyar Madabushi


UNIVERSITY OF
BIRMINGHAM

Bachelor of Science Degree in Computer Science
School of Computer Science, College of Engineering and Physical Sciences

## Abstract

This report will cover the task of classifying food items from campus eateries, using Machine Learning on a mobile device. Putting Convolutional Neural Networks into practice, we will explore their feasibility for use in everyday life, with background variation and different lighting conditions. The report will discuss the considerations and challenges involved in collecting the data, training a model and implementing the system.



Key Words: *Machine Learning, Convolutional Neural Netwoks, Android App Development, Food Recognition*


## Project Location

The code can be found in a git repository at the following address:
https://git.cs.bham.ac.uk/ldd798/final-year-project.git

A video demonstration can be found here:
https://photos.app.goo.gl/QbwdXVLvDYu1qZfEA

# Contents

# List of Figures

# 1 Introduction

This report looks at the process of creating a food recognition system to run on a mobile device. The aim of the project is to implement a classification model in a mobile app so it can be used to easily record food items from university campus eateries. The project has the wider aim of helping users to record their food intake so as to provide a better idea of nutritional intake and spending. This is important because it can be very difficult for an individual to keep track of their food intake, especially for young adults, such as students, who are likely new to taking on full responsibility for their diet. With contactless technology, making purchases is faster and easier than ever so it seems more difficult to have a grasp on spending. Although one simple solution would be to record food intake manually, this is time consuming and doesn't link to spending and nutrition information. Therefore there would be great benefit in developing a system to easily capture food purchases and to provide related information.

The proposed solution should prioritise speed and accuracy. This means the users shouldn't face the inconvenience of waiting for the system to classify the item or having to classify the item over and over until it's correct. The system should also be robust enough to deal with variation and show the correct classification in different environments and contexts. For example, the system should be capable of handling different backgrounds and lighting conditions.

## 1.1 Motivation

Our society has a problem with our diet. Around 90% of diabetes cases are manageable by diet. It's predicted by Public Health England that the UK will have another 1 million people with diabetes by the next decade which is an increase of about 25%. In the UK over a quarter of adults are classed as obese. The NHS spends £6.1 billion each year treating diabetes and obesity related ill health. Adults aged 19-64 are consuming 2.4 times the World Health Organization's recommended daily sugar intake. Tooth decay, which is largely preventable in children, was present in 23% of 5 year-olds in 2017. (Health profile for England: 2018, 2018)

With the speed and convenience of contactless payment technology, it's easier than ever to lose track of small purchases such as lunch items. Since adopting contactless, Master Card recorded a 30% increase in spending in the first 12 months (MasterCard Advisors Study on Contactless Payments, 2020). UK finance realised that 46% of debit card transactions in November 2019 were made using contactless (Card spending — UK Finance, 2020). This isn't surprising given the increasing ubiquity of NFC enabled smart phones (especially amongst individuals at university).

Given (i) the requirement for the UK to improve our eating habits, (ii) the ease of contactless spending supported on campus and (iii) the prevalence of powerful smart phones, it seems that an app running on the very device used to make these payments would have a significant effect. With over 30, 000 students at the University of Birmingham, there are many people to benefit.

## 1.2 Background on Image Recognition

There are a variety of image recognition techniques available. A spatial colour descriptor technique can be used for feature extraction (Ho Young Lee, 2003) which calculates image similarities by comparing their histogram vectors. Another example uses a Random Forest for feature extraction (Bossard, Guillaumin and Van Gool, 2014). An alternative solution would be to scan barcodes however this method lacks the fun and sense of enjoyment of image recognition that encourages users to record their food. It also wouldn't capture

items like fruit and coffees which don't have barcodes. So this method lacks generalisation while increasing use effort.

CNNs (Convolutional Neural Networks) have shown a strong presence in this area, popularised by LeCun et al. (1998). Since then, the technique has been applied to many impressive problems with incredible accuracy such as recognising whales from aerial imaging (Guirado et al., 2019). The method has proven robust to 2D image variation and noise so it could be used to recognise food items even with poor quality images. It has eliminated the need for handcrafted feature extraction techniques.

## 1.3 Goals

- Create an app to allow users to quickly and easily record campus food items so as to give users a better understanding of their nutrition intake and spending.
- Test the suitability of ML and Computer Vision techniques to achieve this with a custom dataset of campus food items with adequate accuracy.
- Collect a dataset to fulfil the requirements of a working model for training and optimising.
- Produce a generalized classification model, working in a variety of environments and settings.
- Perform a detailed analysis of the model.

## 1.4 Tasks

This will involve three areas of development:

- Understanding the requirements of the model to prepare a suitable representative dataset of images.
- Through analysis, gain insight into how the model uses the data to train and optimise the classification of unseen items with high accuracy.
- Implementing the model effectively in a mobile app to compliment the goal of improving the user's understanding of nutrition and spending.

## 1.5 Software Goals

The app should allow the user to easily and conveniently record an item and the corresponding information should be presented in an easy to read way.

## 1.6 Wider Goals

- Make the process of recording food easier for people.
- Increase people's awareness for their spending and food intake.
- Increase nutritional understanding of food items around campus.
- The system should demonstrate a tool to potentially improve the lives of users.

### 1.7 Report Outline

We will first run through the background and theory required to understand the techniques and terminology used in the report. Next we will summarise the related academic literature surrounding CNNs and Mobile Recognition systems. The methodology will firstly describe the final solution and then cover our approach to experimenting and taking on the engineering challenges. We will then give an empirical evaluation of our implementation. The discussion section will focus on a qualitative evaluation of the system before we conclude with how the project has met the initial goals.

## 2 Background

### 2.1 Neural Networks

A neural network is a collection of neurons. The first Artificial Neural Network (ANN) was invented by Rosenblatt (1958) although first described by Turing (1948) in his paper titled Intelligent Machinery. A neuron can be thought of as a unit that takes inputs and has a capacity to fire outputs. Modern ANNs consist of banks or layers of neurons. The layers are fully connected meaning each neuron in a layer is connected to every neuron in the layers immediately preceding and succeeding as shown in Figure 1 (NN Visualiser, 2020).

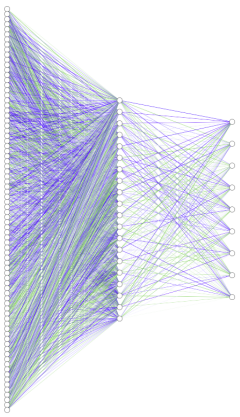

Figure 1: Fully Connected Layers

The input to the network is a vector of numbers which is propagated through the network via these fully connected banks of neurons. For a classification neural network the final layer consists of the same amount of neurons as class labels. For example to classify songs into happy, sad and neutral we need a neuron for each so the final layer would have 3 neurons. The neuron with the strongest signal represents the label with the strongest prediction.

When a signal travels from one neuron to the next it is multiplied by a weight value. The edges in the diagram represent the connections and each have an associated weight. Each neuron takes in all the weighted inputs from the previous layer. If the sum of these weighted inputs is greater than a certain threshold, the neuron will fire a signal through to the next layer at a certain strength which, in turn, will be weighted for each of the receiving neurons. Each neuron also has a learnable bias value added to the weighted inputs before being fed through an activation function to determine the strength of the output signal.

The goal of training the neural network is to adjust these weights and biases according to the error from the training data given by the label. A well trained network will have found a combination of weights and biases which has maximised the accuracy for the training data. However, if the NN is evaluated on the training data it may be ineffective on unseen data. This is to say the model has overfitted the training data and will generalise poorly to unseen data. To avoid overfitting, a validation set of unseen data is used to test the model. Achieving accuracy on the validation set is a better measure of generalisation.

To train the model the data is divided up into training and validation sets. Each iteration of training is called an epoch which can be thought of as a version of the model with a specific set of weights. At each epoch we can evaluate the model on the validation set. Each epoch tries to build upon the experience from the previous epoch, so as the number of epochs increases we should see an increase in the validation accuracy. We should stop training after the training and validation accuracy diverge and the learning plateaus. During

an epoch the data is arranged into batches. The size of the batch determines how many data items are run through the model before the weights are updated.

The model with the best validation accuracy is chosen as the strongest. However, since the model is chosen based on the validation set, the validation set is therefore indirectly influencing the model. To reduce the effect of this potential bias, a third set is used to independently test the model. We'll call this the test set.

## 2.2 Back Propagation

The process of updating the learnable parameters (i.e. the weights and biases) of a Neural Network is achieved through the backpropagation algorithm invented by Paul Werbos (1974). Since the method to learn the biases is the same for weights, we will just be referring to weights. As mentioned, the goal is to find the most optimal set of weights which results in the training set of images being classified with the highest accuracy. Each item in the training set will require a different set of adjustments so these changes to the weights are accumulated over a batch before the model is updated. As the name suggests the adjustments to weights are calculated from the back to the front of the model.

Backpropagation works by taking the final output from the network and comparing each neuron to the labelled answer to calculate a loss. Different types of data will use different loss functions but the idea is to produce a quantitative error measure for each output. For a classification model the loss function may be to simply square the difference between the neuron's expected output with its actual output.

With the error function the adjustments required to all weights can be calculated through Gradient Descent to minimise the loss/error. The error function can be thought of as describing a high dimensional terrain that is navigated by the weights. Gradient Descent produces a direction for the weights to be adjusted so as to find a minimum of the error terrain. In other words, Gradient Descent works to differentiate the loss function to find the vector of adjustments for each weight given the input data.

The Backpropagation algorithm iterates through each neuron and calculates the best weight changes required for the associated inputs to that neuron. These adjustments are calculated by finding the partial derivative of the loss function with respect to the weights.

## 2.3 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a neural network designed for image data since it better captures the spatial information between pixel clusters which is key in images. This was introduced by LeCun et al. (1998). Instead of a 1D vector as an input, a set of 2D matrices are used. For a RGB image a set of three 2D matrices are used as input: one for each colour.

The more fundamental NN, as described in section 2.1, is the final module of the network for a classifying CNN. First, the decomposed image is fed though a number of blocks. Each block comprises 2 components, a Convolutional layer and a Max-Pooling layer. We can think of the image, starting off as 3 2D matrices, being crushed and scaled down in exchange for more dimensions before reaching the final module which consists of dense fully connected layers.

First we must understand what a convolution is before understanding a convolutional layer. This is shown in Figure 2 (Convolution, 2020). A convolution is a mathematical operation applied to a matrix whereby a filter matrix (also called a kernel) is passed over each element of the matrix being convolved. This produces a feature map which is the sum of the filter values multiplied by the original matrix values (similar to how

a neuron takes input). Depending on the filter, convolutions on images have the effect of reducing noise (Gaussian filter) and finding edges (Sobel and Canny filter).

At each block, the convolutional layer convolves the input matrices with a number of filters to produce feature maps. This stage can be thought of as breaking the image down into its distinguishing features. As mentioned, the first 2 dimensions must be scaled down. Since the image input must be a square, the first 2 dimensions are of equal size and the data passing through the blocks can be imagined as a collection of square images.

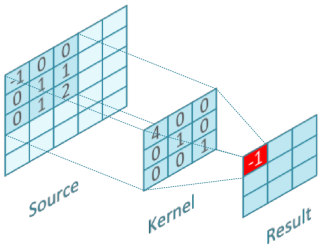


Figure 2: Convolution Operation

As we pass through the network the 2D square feature maps are down sampled, but the number of them increases. This is achieved through the Max-Pool operation. Max-Pooling reduces the resolution of each square by dividing it up and replacing the values of each section with the largest value as shown in Figure 3 (Max-Pooling, 2020).

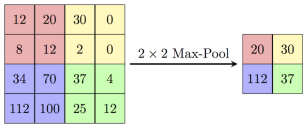


Figure 3: Max-Pooling

After the image is decomposed into many feature maps, it is flattened into a 1D vector and passed to a typical NN usually consisting of 2 layers to be classified.

In order for the model to learn how to decompose and represent the features of the image it must learn the best values for each filter. In this sense a filter can be thought of as a set of weights which are updated through a modified version of Backpropagation.

Transfer learning is the process of using a pre-trained model and applying the knowledge to a different dataset.

### 2.4 Bayesian Inference

Bayesian inference is a probability technique using Bayes theorem to update the prior probability distribution of a hypothesis with more evidence. The formula below is used in the solution to present the probability of the most likely food item given the user's location. As more data is collected on what the user classifies in each distinct location, the app can provide a better prediction of the most probable item in each location. This works in the app by maintaining a frequency and distribution table where each cell represents an item and a location.

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)} \tag{1}$$

## 3 Related Work

There are many examples of work in food classification both in the academic world and in industry. We will first cover the different techniques used and then look at more specific implementations as well as example apps. Through exploring and evaluating these examples we will pay attention to the aspects most relevant to this project such as the dataset used and the practicalities of mobile implementation.

### 3.1 Random Forests

Although a CNN is a popular technique there are many ways of approaching the task of food recognition. Bossard, Guillaumin and Van Gool (2014) used Random Forests to classify images of food dishes. The Random Forest is used to mine discriminative components of the images such as a portion of food on the plate. A SVM (Support Vector Machine) component model then clusters the components to classify the image with a dish description.

They used a novel dataset with 101,000 images. Notably, these images were taken in real world conditions with noise and variation as they were taken from a website which allowed users to upload images of their meals. This makes the dataset much more challenging and is a better test of the technique.

The dataset contained 1000 512x512 image of the most popular dishes and was split 75% and 25% for training and testing. The method showed the best accuracy with around 30 trees with a max depth of 5.

Compared to other approaches such as IFV (Improved Fisher Vectors) and BOW (Bag of Words), Random Forest classification shows the best results at 50.76%. However, a CNN showed a 5.64% increase although it took 6 days to train.

### 3.2 Segmentation

Myers et al. (2015) also applied a CNN to images of meals. The CNN was used in a mobile app and the nutritional information was retrieved from the internet. The group use a dataset collected from restaurant menus as well as the popular Food-101 dataset. They then performs segmentation and volume estimation to estimate calories in food outside of restaurants.

The motivation for the paper is similar to this project in that they recognise the tedious and time consuming ways of current mobile food recoding apps. To make progress they focus on restricting the setting and use case.

The paper's model first only processes images that are a suitable distance from the camera. On the Food-101 dataset the CNN achieved 79% accuracy.

They found that the model has a high top-1 error rate due to very visually similar items. A non-generic method of dealing with this is to cluster the items in the dataset so a model can be trained on a cluster.

To improve the performance of the classifier, the paper employs segmentation to find the individual food items in an image with a highlighted amorphous region. This is done through a CNN with a fully connected Conditional Random Field (CRF) on the end (Chen et al., 2014). Impressively a CNN was also used to predict the depth information to better estimate the portion sizes.

Finally the system was implemented in an android application. The app could classify in less than a second but without using depth prediction and segmentation. The app also makes use of prior probabilities to better classify the food.

The paper shows CNNs are capable of fine grain recognition to detect different types of a specific food such as burgers.

### 3.3 Simpler CNN

Kagaya, Aizawa and Ogawa (2014) experimented with training small CNNs with 2-4 layers, each with either 32 or 64 filters. They built a dataset of 170,000 64x64 images containing 10 categories of food. Using the custom dataset, they found an accuracy of 73.7% using just 2 layers of 32 filters. They compared this to other conventional SVM techniques using SPM (Spatial Pyramid Matching) with image colour histograms, GIST

descriptors (a convolutional technique) and SIFT (Scale Invariant Feature Transform). The CNN showed to be over 10% more accurate. The most notable insight from the paper is how significant colour is to food recognition. Compared to general image recognition the filters for food are much more colour specific. The paper also put the CNN to the binary food detection task and found an accuracy of 93.8%, an improvement on the baseline SVM method.

### 3.4 Transfer Learning and Fine-tuning

Yanai and Kawano (2015) used a pre-trained AlexNet model on the ImageNet database of 2000 categories. They then fine-tuned the model by freezing the weights and changing the size of the softmax output layer to 100. The model was tested along with other feature extraction SVM methods. For training and testing the paper created a dataset of 100 categories of Japanese food. Each category contained around 1000 images mined from twitter. Looking at the images there's lots of variation and many meals have similar ingredients and colours. This results in a very challenging dataset.

They showed an accuracy of 78.77% and only a small increase in accuracy by pre-training on food related images. The paper also applied the model to twitter images and found it was very suitable for large amounts of data as it only took 0.03 seconds for a classification.

### 3.5 Mobile Implementation

Bonis et al. (2018) used CNNs with the goal of making it practical and accurate enough for a mobile app. They used 2 datasets with 101 classes each with 790-1000 images. The dataset contained noised and specific images. By noised they mean images where the food item is less obvious since the datasets were collected from search engines. This meant many of the images were from phone cameras as they were sourced from social media. Before using the dataset the images were pre-processed by subtracting the mean image to emphasise the most important areas.

The paper tested the two famous CNN architectures, AlexNet and VGG-Net, as well as more complex models such as ResNet and GoogLeNet which is an Inception model. They trained the CNNs from scratch using an optimiser which adapted the learning rate with a step-down policy. At around 60 epochs they found the best accuracy with GoogLeNet at 56% and 43%. It also showed a reasonable model size of 40Mb.

With fine tuning the GoogLeNet achieved best accuracy again at 73% and 63%. The model was fined tuned using a Learning Rate Multiplier parameter instead of freezing the weights at the start of the model.

The paper also experimented with using a threshold to express the confidence of the model.

The model was implemented in an android app. There was an offline CNN embedded in the app and an online CNN for phones to query from a Java Web Application. In order to make use of different phones' processing capabilities the app used the RenderScript framework.

### 3.6 Fruit

Zhang et al. (2014) showed an FNN (FeedForward NN) could be used to classify 18 different types of fruit. They say fruit is a particularly challenging task. The paper applies colour discretization to reduce the number of colours which has a reduction in processing cost because there are only 64 colour possibilities. However the images are larger in scale at 256x256. Interestingly they use an artificial bee colony technique to optimise the model and achieve an impressive 89.1% accuracy although the training is very intensive.

### 3.7 App Examples on Market

PictureThis is a plant recognition app that uses advanced image recognition techniques. The app works by the user scanning a plant or they have the option of choosing an image from the phone's storage. The app requires the user to be connected to the internet. This probably helps the app to take up less storage since it only uses around 90MB. Hosting the model online would also allow it to be updated remotely and it could improve from user generated data. However it would impact to the classification speed. Each plant seems to take a few seconds which is disguised by an animation.

The app always presents users with the top 3 matches. The associated information is presented in a vertical scroll view. Users are able to confirm the match and save it to their online account which can be linked to Facebook.

For camera controls the app has chosen to let the users zoom, focus and use the front camera. There's also a transparent template to guide how the flower should be positioned in frame as show in Figure 4. Location information is incorporated into the app to improve the classification and show your plants to a community of users through the internet. For the user's benefit the app provides help and guidance instructions and a beginners video. (PictureThis - Plant Identifier, 2015)

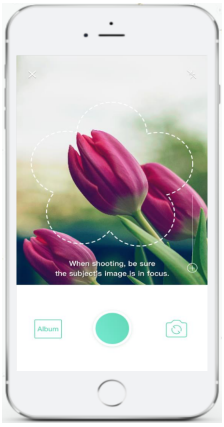


Figure 4: PictureThis App

Calorie Mama AI is a food recognition app that can scan natural foods such as salmon and fruit. It queries a model through the internet to obtain an array of most likely classifications and associated nutritional information. The user selects the item and provides the quantity for the app to record. To incentivise the user to use the app every day, a streak of number of consecutive days is recorded. The view finder and capturing process is similar to PictureThis with a framing template and ability to use images from the phone's storage. The camera flash is also an option. The accumulative totals are shown on the home screen. The app also encourages the user to group their food by meal. As well as food intake, the app also facilitates recording water intake and exercise. (Calorie MAMA, 2017)

### 3.8 Summary of Related Work

In summary, from researching related work it seems a CNN is the most prevalent and robust method compared to feature extraction through hand crafted techniques. It can achieve high accuracies even with custom datasets of 100 classes with around 1000 images per class. CNNs can also operate on mobile phones with mobile camera quality images. Small architectures have proven to be adequately successful although pre-trained and fine-tuned models can be used to further increase accuracy, particularly with more than 10 classes. Fruit seems to be the hardest category to tackle. This is where more complex methods such as segmentation and nature inspired optimised Inception models prove most successful. With regards to mobile implementation it's important to consider model size and speed.

## 4 Methodology

### 4.1 Final Solution Specification

The final solution, chosen based on the results of experimenting, is an android app that presents the most likely predictions using a CNN. The custom dataset consists of 9 classes of over 138,000 images. After training and optimising a CNN from scratch, we use transfer learning by pre-training and fine-tuning the

VGG16 model architecture. The final solution implemented this transfer learned model. The app records the accumulative totals of sugar, calories and spending of recognised items saved by the user. With the location of the user, the app predicts the item they're most likely classifying based on previous activity.

## 4.2 Approach

The project development was completed with a fast prototyping ethos similar to the principles of agile. This meant developing working code quickly, frequent evaluations and gradually introducing complexity to consequently reduce the time spent debugging.

In preparation for developing a food classification model from scratch, it was first necessary to experiment with a CNN to classify other datasets such as cats and dogs, MNIST and flowers. This was implemented in Google Colab, an online Jupyter Notebook. The cloud service was chosen as it allowed the use of online GPUs as well as a convenient way to store annotated cells of python code between computers. After experimenting, a working architecture had been established.

The initial goal was to make a binary classification on a custom dataset as a proof of concept. The first step was to create a dataset. This involved collecting 2 videos of a coffee and an apple separately to be converted into frames. A coffee and an apple were chosen since they were distinct in colour and shape. Initially a plain wooden table was used as the background because it had less pattern variation compared to the datasets used in related works.

To be consistent, the same background was used for each item under the same lighting conditions as there was no natural light. The videos were converted to 150x150 images using VLC then regularised manually so as not to create a biased model. The images weren't uniformly scaled and were in the incorrect orientation (Figure 5) however the initial goal was to classify items, so this will be addressed later on.

Figure 5: Coffee and Apple frames

To load the items into the colab notebook as fast as possible, they were hosted in a zip file by a server. Alternatively a Google Drive can be mounted or the files can be uploaded from the computer's local storage but both of these methods proved slower. Hosting the images allows for the dataset to be backed up and accessible from any computer.

Once the images were loaded to the notebook they were split into training and validation sets. The training and validation sets were split 75% and 25% respectively because many related works use this split such as Myers et al. (2015). The Keras API running on the Tensorflow backend was chosen as it contains libraries for data pre-processing, setting out and training neural networks. The most popular alternative is PyTorch but there are a variety of other APIs for neural networks. However Keras was chosen as it focuses on 'friendliness, modularity, and extensibility' which fits the development ethos of this project. It is also more lightweight than PyTorch which is important for the needs of this project. (TensorFlow, 2020)

The images in the training and validation sets are scaled between 0-1 before being used for training. It was important to shuffle the images too so the model doesn't learn the pattern of the data. The CNN architecture was set out to contain 4 convolutional blocks with a Max-pooling after each. Before the final output neuron layer a dense layer of 512 neurons was used. This architecture was based on a combination of the simple CNN with 2 layers and the larger VGG16 from the paper mentioned above by Kagaya, Aizawa and Ogawa (2014).

| Layer Type | Shape |
|---|---|
| Conv2D | (148, 148, 32) |
| MaxPool | (74, 74, 32) |
| Conv2D | (72, 72, 64) |
| MaxPool | (36, 36, 64) |
| Conv2D | (34, 34, 128) |
| MaxPool | (17, 17, 128) |
| Conv2D | (15, 15, 128) |
| MaxPool | (7, 7, 128) |
| Flatten | 6272 |
| Dense | 512 |
| Dense | 2 |

Table 1: Initial Model Architecture

The batch size was initially set to 5 as it allowed the model to train reasonably quickly. The model is then compiled and trained using the Adam optimiser and a binary loss function. Adam (Suresh, Gnanaprakash and Santhiya, 2019) was used as it seemed to be a popular comprise between optimisers in related work and is supported by Keras. After just 4 epochs the model showed full accuracy on both the training and validation sets. However due to the size of the dataset and the similarities of many of the images, we can't infer that the model would perform as well in practice. However this gave confidence that the model was functional and that collecting mobile phone camera quality data in this way was sufficient.

To visualise the training, the python library Matplotlib was used to show accuracy and loss over epochs for both the training and validation sets. By visualising the training it's easier to recognise trends and see where the model starts to plateau. It's also useful in spotting overfitting from a divergence of performance between the training and validation sets.

Next we decided to classify bananas from oranges using the same process and parameters. Two fruits were selected because, from the related work (Zhang et al., 2014), this is recognised as a more challenging task. The size of the dataset was also increased to over 6000 images to see the effect on the process of collecting, converting, loading and training. This was important since the related works used over 100,000 images so the process used needed to be scalable. It was evident from this stage that VLC wasn't fast enough, otherwise there was little reduction in efficiency. Like before, after a few epochs the model showed high accuracy.

Augmentation was then applied to make the dataset more challenging and robust. This step was applied early to potentially save time. With small more challenging datasets it's faster to experiment with the hyperparameters and architecture of the model. Keras' image pre-processing tools were used to artificially rotate, zoom, scale and shift the object horizontally and vertically. The images were also reflected horizontally. This would have the effect of reducing the overfitting of a model and making it more invariant to such changes. The model showed good capability even with the augmented dataset. To further avoid overfitting, dropout was implemented before and after the first dense layer. This also had little effect on the model performance.

The next goal was to have a working multiclass classification model on a custom dataset. The process was repeated using 5 classes: apple, coffee, crisps, juice and orange. Using this dataset threw up problems. The model was showing no errors but it was giving an extremely small training accuracy. Debugging the model was simpler due to the prototyping approach. Since the only changes were the dataset and the optimiser, the next step was to test the optimiser on another dataset. After testing the 'Sparse Categorical Cross Entropy' optimiser on another multiclass dataset it was still unclear why the model wasn't working with 5 classes. To solve the issue it was necessary to gain a better understanding of the theory. This led to systematically

changing the hyperparameters and removing augmentation as well as dropout to make it easier for the model to learn. After working through the operations applied in the CNN, the error was obvious: The number of neurons in the last layer of the network didn't match the number of classes in the dataset.

With a working model the next goal was to expand the dataset to 9 classes: Apple, Banana, CoffeeMedium, CoffeeSmall, CoffeeUob, CrispsBlue, CrispsGreen, Orange and Juice (Figure 6). The number of classes were chosen because it is enough to best represent each type of food consumed by users and sold on campus. It is also close to the number of items classified by Kagaya, Aizawa and Ogawa (2014) with only two convolutional blocks.

When capturing the videos there were many important considerations in order to be consistent and systematic so the model doesn't learn the pattern of the conditions instead of the visual pattern of the item. A forward-thinking systematic approach would also save the time needed since this was an arduous task.


Figure 6: Examples from the full dataset

**Image Background** - It is important to include a variety of backgrounds with different textures and patterns so as to create a robust model. We will define a scene as a tuple of background and lighting. With more scene exposure, models are more robust to environment variables.

Initially 4 scenes were captured including 2 wooden surfaces under both natural and electric lighting, 1 pale wooden surface under natural light around midday and 1 white background with no daylight and harsh shadows.

**Lighting** - When filming in locations with natural light all the data must be collected within a narrow time period so the lighting conditions don't change. Rooms without windows provide consistent lighting but there may be multiple light sources to consider.

**Video Length and Rotation Pace** - Deciding the length of each video was influenced by many factors such as memory, video quality and number of scenes. The camera recorded at 30 FPS (frames per second) so the number of frames could be calculated by multiplying by the length of the video in seconds. Since we intended on using multiple scenes and just 10 seconds would produce 300 images FPS wasn't a limiting factor.

It was more important for the video to have a fair distribution of all the angles of the item to give the model the best possible representation. This relied on keeping a consistent rotation pace when filming. One lap of steadily filming the item in focus took around 11 seconds. However, we wanted the model to classify items both near and far so we decided on filming at a distance of 0.5m and 1m. This led to each item being recorded for at least 22 seconds.

**Blur** - Moving too quickly around the item or shaking the camera could lead to some images in the video to blur. This makes it harder for the model to learn the angles of the item where the camera was out of focus. Therefore it's important to keep the item in focus for the entire video. The camera and its settings were kept consistent throughout the data collection as they have an effect on image quality.

**Filming Angle** - The filming angle was decided to be around 30 – 45 degrees. A higher angle means there will be more surface in the background. This leads to more consistency but a less accurate view of the object. Taller items, such as the juice and coffees, required a slightly lower filming angle for the best view of the item. This increased the depth of the background and introduced more variation. The chosen filming angle was a compromise as the shorter items like crisps required a higher filming angle. It was also considered a natural filming angle with a phone.

**Camera rotation, filming direction and framing** - When filming an item whilst moving, it's natural for the camera to rotate to point in the direction of travel and therefore moving the item to one side of the frame. To counter this effect all images where reflected horizontally in pre-processing therefore there was no need to be consistent will filming direction but we were still conscious of camera rotation and framing.

It's also important to be aware of the framing of the item because the image will be cropped, so the vertical framing must be consistent. To account for this a template was used in the camera app whilst recording. An item framed lower can allow more of the top of the image to be cropped, resulting in less background variation when filming at an angle.

**Number of videos per item** - Some items such as coffees and the juice always look the same. Since the crisps and fruit items have variation in size, colour and shape, it was necessary for multiple items to be filmed in each scene. For example some apples have different colouring, bananas have dark patches, fruit may have stickers and crisps can be crumpled differently. Unlike the other items, crisps look different upside down and aren't always captured in the same orientation. We made the decision to record crisps with the front of the packet facing upwards because they were more distinctive with the picture on the front and more coloured surface area was visible. Users are more likely to capture crisps with the front facing upwards as well. Since 2 of each fruit item was filmed it was necessary to film two videos of each item in each scene.

**Number of Scenes** - The number of scenes was decided by considering the number of distinctive backgrounds around campus and the number of lighting conditions. This led to a goal of 17 scenes. With 2 videos per scene and 9 items this equated to 306 videos. This was also justified by the size of the dataset since this amounts to over 100,000 images which is a similar size used in related work.

## 4.3 Development Stages

With a large amount of data, a more efficient method of converting the videos to frames was required. The CV2 library in python was used to automate the process of clipping the videos, extracting the frames, cropping, scaling, rotating and then storing in organised folders. To process the scenes of videos for a single item took several hours using the python script but this was massively more effective than VLC as it saved time regularising the classes and it pre-processed them offline ready to be used by the model. This involved many hours of testing to ensure the data was properly pre-processed and organised. Proper testing was necessary to save time especially when working with storage and processing restrictions.

The dataset with 4 scenes showed good training accuracy so the next goal was to test the model and implement a working prototype in a mobile application. To do this, we used a template application from Tensorflow's online repository (tensorflow/examples, 2020). We decided to use an android app since it's the most used operating system for mobile phones and was used on similar projects like Bonis et al. (2018)

mentioned in the related work. The phone used to create the dataset was also running android. First, the model was converted from a Keras model to a tflite model which is a format designed for mobile devices. The tflite model is quantized to reduce the its size and increase computational efficiency. However there is a small reduction in quality but since the model was showing high accuracy this was a sensible trade-off. The tflite file is loaded into the app along with a label text file. The Android Studio SDK is used to build the app for testing on a phone as well as emulated devices.

The app continuously classifies items, presenting the top 3 predictions and accuracies at the bottom of the screen. The speed of each inference was very impressive. However the predictions were very incorrect. The app only occasionally predicted the correct answer and not for long. To find the problem we had to narrow down the issue to the model or the app's implementation of the model. Since the model was showing good performance by the validation set accuracy, the problem seemed most likely to be the app so we analysed the code.

The initial hypothesis was a fault with the method used to get the camera image fed to the model. However, after a better understanding of the app's use of the Tensorflow Lite API, there was no obvious problem. It was noticeable after using the app extensively that it was often misclassifying items as oranges or apples. These two classes had a few more images in the dataset due to the conversion method. This led us to believe that the problem was the model so we took to analysing the model in depth. In parallel we researched further into related work and studied the code of other Tensorflow Lite apps such as a image style transfer, image segmentation and a hand writing digit recogniser to better understand the use of the API. (TensorFlow Lite Examples — Machine Learning Mobile Apps, 2020)

To create a completely even distribution of images for each class the sampling rate was fixed and the frames were re-processed. There was concern this would leave out the final angles in the video. With no improvement, we experimented with systematic hyperparameter optimisation (such as optimiser, learning rate, batch size, model architecture, augmentation and dropout) and generating different datasets. This still only resulted in a small increase in performance when using the app but from looking through the data it gave rise to a new hypothesis.

The images from frame to frame were highly similar. Since all images were shuffled and then split into the training and validation set, this meant highly similar images a few frames apart featured in both sets. Therefore, the validation accuracy wasn't representative of the true unseen accuracy as the model was exposed to a very similar image. To solve this problem, we first introduced a step parameter to remove a set number of consecutive frames. This was also an effort to reduce overfitting so that the training set would include less collections of similar frames. However, this reduced the amount of training data. We also switched to only using frames from the same scene/video in each set. This means the validation set contains videos in a completely different setting to any images in the training set.

Furthermore, we implemented a test set to give an even fairer accuracy on unseen data. A test set was used for reasons explained in the background section and the related work. By increasing the dataset size and having 17 different scenes, the model is also less likely to overfit. The dataset was distributed in the following way: 10 scenes for training, 4 scenes for validation and 3 scenes for testing. This gave the model exposure to an even number of backgrounds and lighting conditions. To reduce the risk of making a mistake when calculating the accuracy, we manually implemented the testing loop. The test set showed positive results. The model showed high accuracy evenly distributed across classes.

As a last test of the model we decided to feed it a single image and then displayed the image with the corresponding predicted label. This way we could validate the image's ground truth label with our own eyes.

With this came errors and a new hypothesis.

This model predicted Blue Crisps incorrectly however the image fed to the model was distorted in colour as shown by the blue orange in Figure 7. To solve this the colour model needed to be changed to be compatible. This formed the hypothesis that the image fed to the model in the app may also be incompatible.

In response to this hypothesis we implemented a different template app from an online repository by Shekhar (2020). Before testing we cropped the bitmap fed to the model from the camera to have the same dimensions used for the training data. After loading the the model with a large dataset of over 138,000 images and optimised hyperparameters, the new template app was very successful. It worked very often with different scales and rotations of items - a major milestone!

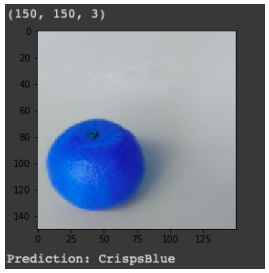


Figure 7: Blue Orange

Next, we decided to apply transfer learning by fine-tuning the VGG16 model shown in Figure 8 (VGG16 Diagram, 2020). This model was chosen as it's a popular model available in the Keras library and showed great results from the realted work section above (Bonis et al., (2018). It contains 5 blocks, each containing 2 convolutional layers and a Max-Pooling. To train the model it needed to first be adapted to take a 150x150 input shape and the final number of output neurons needed to match our number of classes. We froze all weights up to the final dense layer. This is done to retain the knowledge learned from the model being trained on the ImageNet database. The final dense layers use the advanced feature representation of the convolutional layers to learn the output class. Testing took slightly longer but the model achieved high accuracy.

## 4.4 App Development

When developing the app there were key decisions to be made. The decision was made to adapt the working template app and extend its functionality because the app was basic anyway and this would save re-implementing tflite inference again. The app also used Java in the Android Studio SDK for development. The alternative language is Kotlin or using Swift for iOS but there's less community support online so Java was the best option. See the link in the Project Location section for a demonstration.

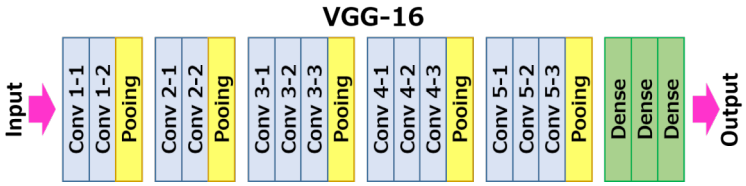


Figure 8: VGG16 Architecture

The template app consisted of a view finder, a button to take a picture and a TextView showing the most likely predictions. Detecting by pressing a button seems to give the user more confidence in the app. Once the prediction is made, a thumbnail of the image is shown at the bottom of the screen to make the user aware they have captured an image. We extended this to show corresponding nutrition and pricing information. The calories, sugar and price for the predicted label is instantly displayed at the bottom of the screen next to the thumbnail. We chose only those 3 metrics as too much information can be confusing, and the goal of the app is to allow quick easy understanding of food content.

The user is also able to confirm the prediction and add to their accumulative nutrition and spending values. This allows the app to be used for querying items as well. The food item is confirmed by tapping the TextView of the prediction. If the prediction is incorrect, the user can press the detect button again

to re-detect the item from a better perspective. It was decided to keep the functionality to present the top 3 predictions to give the user an understanding of how well the model works in different conditions. The user can respond by working with the model to achieve better accuracy. This is a 'Seamful' design approach (Chalmers and MacColl 2003).

The camera view finder was placed at the top of the screen so the associated item information could take up more space. The view finder was rectangular so as to encourage the user to centre the item in the frame. By tapping the view finder the camera switches to using the front camera. This is useful if the user is holding the item.

Next we implemented a history page. In Android Studio this is achieved by creating a second activity which is activated when the user presses the 'History' button. The history page displays the accumulative total values for calories, sugar and spending as well as displaying the last item saved. To go back to the camera activity, the user simply swipes to the right. This gives the app more of an intuitive feel and allows faster use as opposed to clicking the back button, which is also available.

The orange white and grey theme is consistent throughout. A third activity was implemented to predict the food item given the user's location. These two variables are related since different places on campus sell different food and users have habits based on location . This feature was included to increase the speed of recording items. When the user confirms an item, the app updates a frequency table and updates the probability distribution of each item given the location using Bayesian Inference. This distibution is stored in table of probabilities and used to obtain the most likely item. This is presented on the location page for the user to confirm and add to the accumulative totals. With this 3rd activity, the app is complete as shown in Figure 9.

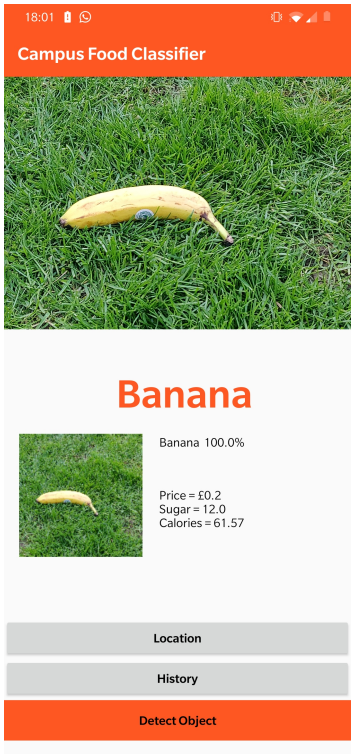


Figure 9: App Screenshot

# 5 Empirical Evaluation

To evaluate the system we will first evaluate the model in isolation and then in the context of the app use. This will include both the transfer trained VGG16 model and the model fully trained from scratch.

## 5.1 Fully Trained CNN from Scratch

We started off with the model architecture in Figure 10. It consists of 3 convolutional blocks. Each block is made up of a convolutional layer with a Rectified Linear activation and a Max-Pooling layer. The input shape is 150x150. After the third block the data is flattened to a 1D vector of 10368 and a dropout is applied. This is fed into a fully connected layer of 512 neurons. Dropout is applied again before the data is fed into the final fully connected layer of 9 neurons. At the last layer the Softmax function is applied to produce a probability distribution.

To optimise the model we experimented with the hyperparameters. As we've mentioned, we introduced a parameter called step to adjust the amount of data the model trains on and the variation of the data. We

will start by showing the model's performance by comparing 4 optimisers. The following hyperparameters are set for this test:

Step: 10, Batch Size: 5, Epochs: 25, Learning Rate: 0.001, Dropout: 0.3, Augmentation: Horizontal Flip

To compare we will look at the plots for accuracy and loss over the training as well as the distribution of accuracy over the test set.

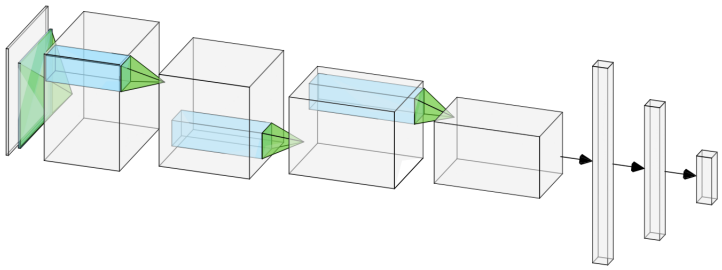

Figure 10: Fully Trained Model Architecture

**SGD**

Figure 11 is the result of the SGD (Stochastic Gradient Descent) optimiser which is the default for Keras. From the training plots we can see a swooping curve as both the training and validation set improve. However it only reaches a validation accuracy of 86.55%. From the distribution graph, this model shows the worst accuracy for apples. It achieves a high accuracy of 94.36% for the separate test set.

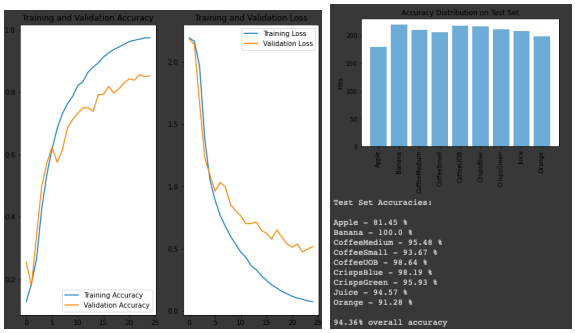


Figure 11: SGD Training Graph and Test Set Accuracy

**Adam**

The Adam optimiser seemed to be a lot more aggressive as it achieved above 95% accuracy on training and validation after just 2 epochs as shown in Figure 12. As a result, the training plots show a jagged line as the model struggles to achieve higher accuracy. The training accuracy is higher than the validation which suggests overfitting to the training data.

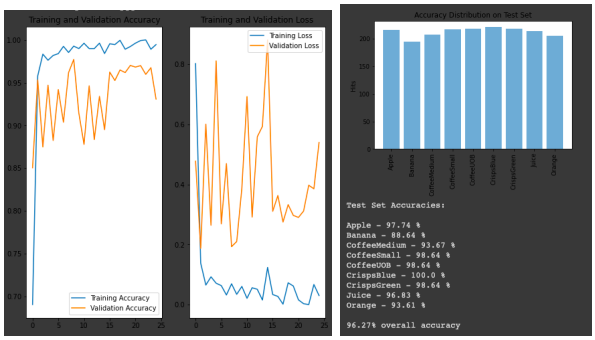


Figure 12: Adam Training Graph and Test Set Accuracy

The test set shows good distribution and an overall accuracy of 96.27% which is slightly better than SGD.

**Adagrad**

Adagrad starts off like SGD with a low accuracy showing a smoother learning curve. It showed a validation accuracy of 86.57% and 80.35% on the test set. This is lowest of the 4 optimisers. The bar chart shows a poor distribution of accuracy (Figure 13).

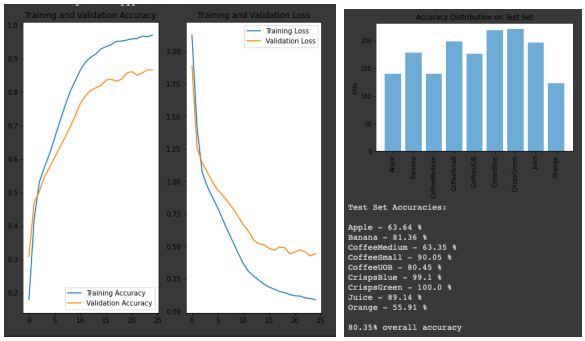


Figure 13: Adagrad Training Graph and Test Set Accuracy

**RMSProp**

Like Adam, RMSProp is also aggressive in achieving a high accuracy quickly (Figure 14). After just a few epochs there was less than 1% increase in both validation and training accuracy at 95%. This gives the highest test set accuracy of 97.98%. The accuracy distribution is very even across items as expected.

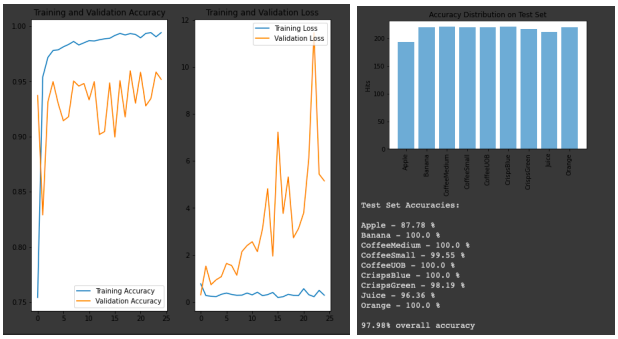


Figure 14: RMSProp Training Graph and Test Set Accuracy

Since Adagrad showed the best results, we then experimented with the step hyperparameter. With a step of 5 the test set accuracy improved slightly to 98.04%. With a step of 15 the accuracy further improved to 98.49% but this took much longer.

**Augmentation**

With the best optimiser and step choices, we augmented the data by rotation, width shift, height shift, zoom and horizontal flip. This resulted in a slight increase in test accuracy to 98.54% (Figure 15).

## 5.2 Transfer Learning Model

The model trained using the VGG16 model showed a test accuracy of 96.02% and a similarly even accuracy distribution after hyperparameter experimenting. Due such high accuracies shown in Figure 15 on the test set it wasn't worth further optimising the models.

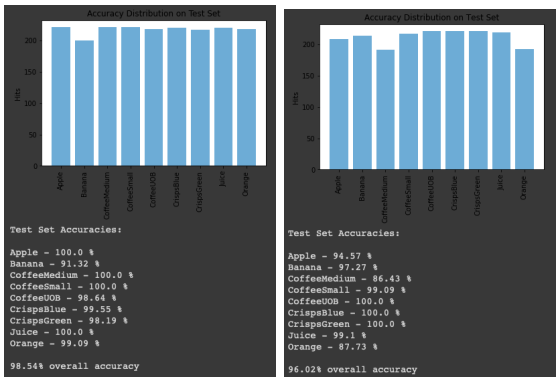


Figure 15: Test Set Accuracies of Fully Trained Model With Augmneted Data (left). Transfer Learned Model (right)

### 5.3 Transfer vs Fully Trained

We will now evaluate the best version of the fully trained model compared to the model trained using VGG16 through in app testing with challenging backgrounds and lighting. To do this we tested each model 6 times on 6 items in 5 different scenes.

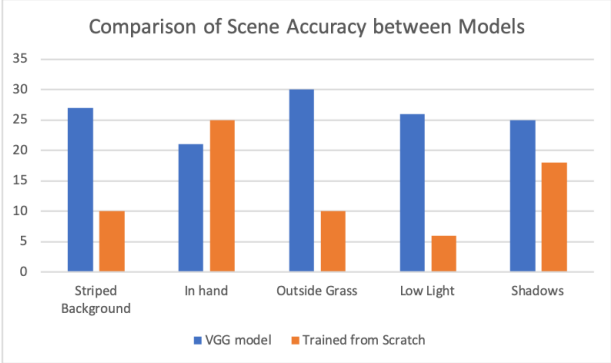


Figure 16: In App Testing Comparison

In Figure 16 we can see the VGG16 model performed very well overall. The model trained from scratch performed better with the in-hand tests but significantly worse overall and in particular in poor lighting. The overall accuracy in this test was 71.67% for the VGG16 model and 38.33% for the model trained from scratch.

## 6 Discussion

This section will analyse the system from a qualitative perspective gained from testing the app, experimenting with the model on the Jupyter notebook and from general use. We'll start by analysing the positive aspects before being more critical.

## 6.1 Positives

**Colour** - From looking at the dataset there are many structural similarities that the model does well to distinguish between. For example the colour of the green crisps packet and the juice bottle is very similar. This suggests the model is paying attention to the shape and spatial information in the images.

**Coffee Cups** - The three types of coffee cups are also very similar in both colour and shape. This gives the impression that the model may be finding fine distinctive features such as the size and colour on the side of the cups.

**Item Variation** - The model seems to be very proficient in dealing with the item's variations. For example, it can classify the crisp packets when crumpled or even upside down. It also goes further by being able to classify an orange juice bottle of a different brand with different dimensions, overall size and colour as shown in Figure 17. We've found the model is also resistant to bruises in bananas and stickers on the fruit as well.

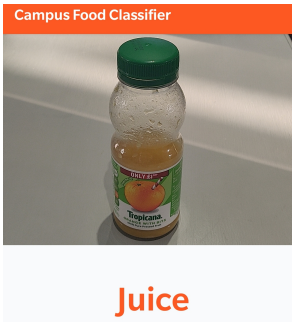


Figure 17: Item not in dataset

**Image Quality Variation** - From using the app it's clear the model is robust enough to classify items even from poor quality images containing blur and rotation etc. This is probably attributed to the method used to collect the images because walking round the items whilst filming produces many imperfect frames. Conversely, a tidy dataset or using a camera on a smooth track wouldn't produce natural variations such as angle, rotation, blur/focus, position of reflections, framing and the distribution of images around the object. Overcoming this natural variation challenges the model and increases generalisation. The high performance on images of different quality suggests the app could be implemented for phone's with cameras of different qualities.

**Items out of Frame** - To extend the previous point, we've found from using the app that the model seems to have the capability of classifying items correctly even with only part of the item in frame. Again we believe this is down to the dataset. Often after pre-processing, the items are partly out of frame. Therefore the model has learned to overcome this obstacle. This seems to work well for crisps in particular. We believe this is down to the overall outline of the image not changing as the frame cuts off some of the packet. (Figure 18)

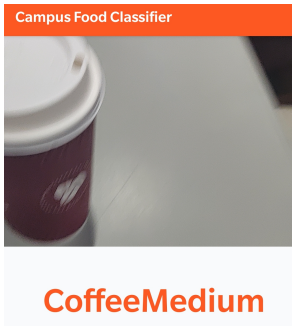


Figure 18: Blurry and out of frame image

**Taller Items** - It may be expected for the model to perform worse on drink items since they are taller so the filming angle is required to be lower. Therefore there is slightly more variation at the top of image. However the accuracy distribution across items was shown to be even in the evaluation and seems to hold true when used in the app in the real world. This shows the model is good at separating the background from the item.

**Multiple Items** - The model shows surprising competency when faced with an image with multiple items. It mostly classifies the nearest item in the middle of the screen. It seems the model has learned to weight the pixels in the middle of the image more heavily despite the ability to classify items in the corner of the frame as well.

**Inference speed** - It's worth discussing the feel of the implementation of the android app here too. The speed of the tflite model in the android app is well within reasonable bounds at under 500ms. The corresponding nutritional and spending information is presented immediately after the prediction.

**Capability on a Small Dataset** - The final positive observation is how capable the model is on a

reduced dataset. During testing it was noticed that performance (by measure of the training, validation and test set accuracy) wasn't dramatically reduced by increasing the step value - the number of frames to skip in the dataset. This shows the model was learning the patterns of the items from as a few as 104 images per item. However this may be attributed to the test set not being challenging enough.

## 6.2 Possible Improvements

**Poor Lighting** - The model performs incredibly well under good environmental conditions such as good diffuse lighting with a plain untextured background but in poor conditions there are confusions between items. These confusions seem to be related to colour since the three coffees are often confused. However, it's worth mentioning that when the model is confused the confusion is between a medium and a small coffee which are far more visually similar than the UOB Coffee.

**Green Banana** - Interestingly when testing with a banana with an unripened patch of green in low light, the model predicted juice or green crisps. This is evidence of the model focusing on the colour to classify since both items are heavily green.

**Apple Orange Confusion** - In terms of the model's overall ability to classify in difficult conditions, the weakest item seems to be the apple. It's most commonly misclassified as an orange which makes sense as the colour and shape are highly similar. However better accuracy is achieved in good lighting and when the item fills the frame. This may be due to the model learning the difference in texture and colour as well as the difference in shape which in particular is more obvious up close.

**Similar Image Consistency** - After taking a series of very similar images, occasionally the model will show a different answer. As a user this gives a slight feel of unease and a lack of confidence in the system. If the system could incorporate prior experience and continue to learn, maybe it would be more resistant to these infrequent irregularities.

**Blue Crisp Preference** - When classifying the crisp packets, the model seems to have a slight preference for classifying blue crisps over green. This could be down to the blue crisp images in the dataset being clearer. This could also be explained by the blue crisps being slightly less distinctive in colour so when faced with a difficult image, the model goes for the more general option. Maybe this could be fixed with a higher resolution being fed into the model. The horizontal reflection augmentation step may also be leading to the crisps being less distinguishable since the image on the crisps is one of the main defining features other than the colour.

**Striped Background** - When testing the coffee on the striped background we noticed that if more of the light blue stripe was present in the frame, the coffee was more likely to be classified as a small coffee. This would make sense because the small coffee has a portion of blue on the outside so the model may be responded to that colour which it associates with the small coffee.

## 6.3 Feature Map Analysis

Now we will cover some observations about the model based on a sample of the feature maps created by feeding in images of a coffee, crisps and an orange. Figure 19 shows some of the 3x3 filters used to convolve a matrix. The bottom left two filters have a resemblance to a Gaussian smoothing filter as the pixel in the centre is weighted more heavily and the outer corners have less influence. The middle filter resembled a Sobel filter which has the effect of picking up horizontal left edges.

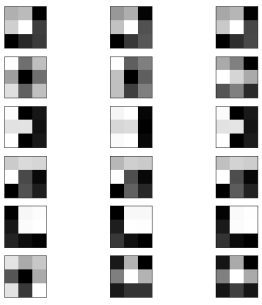

Figure 19: Examples of the Filters

Figure 20 shows the result of an evenly distributed random matrix being fed through the model. Each 3x3 square shows the grey random noise image after a convolutional block.

From this we may infer that the top left has been convolved by a filter that finds edges running diagonally left to right since we can just observe small wiggles oriented in this direction. Similarly, the middle right square may be looking for edges running in the opposite direction. The bottom left feature has no highlighted areas, this maybe because a filter applying a smoothing effect. The green image in the centre was clearly convolved by an easily excited filter with high weight values. After the next convolutional block, the lines indicating the direction of the edge detection are more prominent in the feature maps.

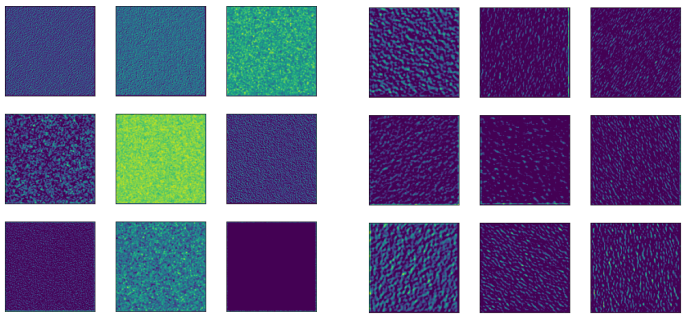

Figure 20: Feature Maps Produced with a random noise grey image

Now to observe the feature maps produced at different stages of the model when an image of a coffee cup is fed through. We can see after just one block that the model is showing focus for the coffee cup because in the top right image the background is removed completely (Figure 21). The model is showing the difference of intensity over the cup's curved surface indicated by the brighter region on the left of the cup and darker towards the right. From the top middle image we can observe it has also picked out the texture of the cardboard cup as you can see the vertical curves. The bottom centre picture shows the middle is paying a lot of attention to the lip of the cup. The middle image may represent a smoothing step to carry the main details that are required in the next block that would otherwise be missed without smoothing. The rest of the feature maps show the model's ability to detect the outlining edges of the coffee as well as the lip of the cup. This is visible from the first block since there's little background variation. So the boundaries of the greatest change in pixel intensity are the edges we would want the model to pick up.

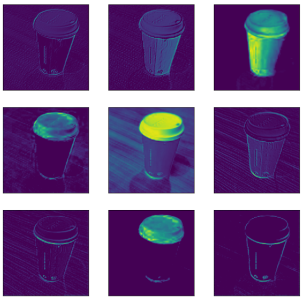

Figure 21:

In Figure 22, the next stage in the model shows the feature maps are mainly focused on the outlining edges of the coffee. However we can still see the background slightly in these feature maps. There's also a trace of the shadow made by the coffee in the bottom left image. As we can see, the edges are less crisp at this level which is a trend as we move further along.

At the next stage we see that that the model has extracted the key components to make up the outline of the image almost as though drawn in pencil (Figure 23). Now imagining a computer recognising the image becomes a lot more reasonable. At the deepest block we observe in the model produces feature maps almost unrecognisable from the input image

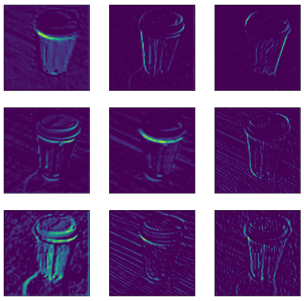

Figure 22:

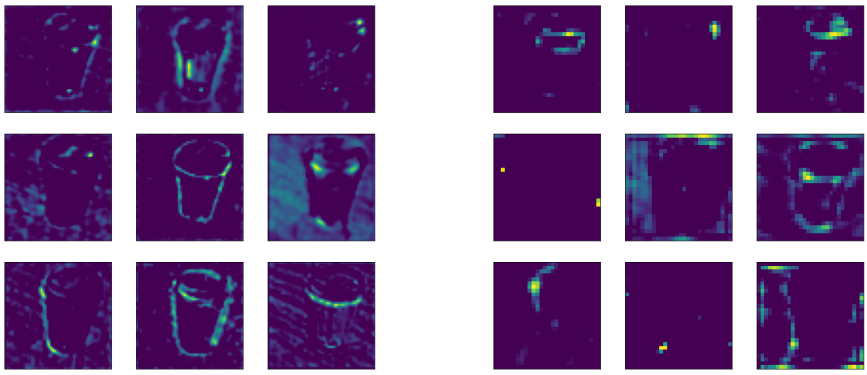

Figure 23: Features Maps of an image of a Medium Coffee at the later stage of the model.

From looking at the feature maps of an image of crisps, particularly the bottom right image, it seems the model is capable of picking up the outline of the lower and top part of the item as well as the text and the image (Figure 24). Interestingly, the edge of the packet connecting the top and bottom regions of colour is not as prominent as expected. Like with the coffee, the model is able to remove the background very easily as shown in the top right image.

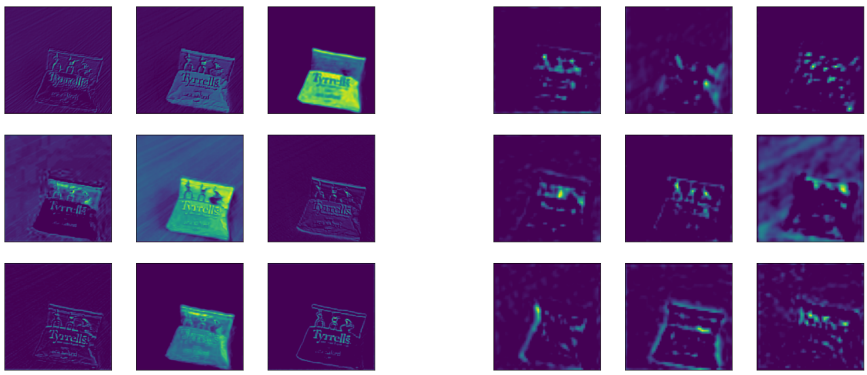

Figure 24: Feature map of Blue Crisps at earlier (left) and later (right) stages in the model.

Moving ahead 2 blocks, we can see from the bottom centre image that the full outline of the packet is picked out. More notably, the model is still carrying forward the information in the picture on the front of the packet. This may be because as well as the colour, the picture on the packet is a distinguishing feature between the 2 types of crisps.

The final item worth analysing through the feature maps is an image of an orange. Like before, the model can pick out the item from the background early on as shown in Figure 25. However, from looking at the feature maps from further along, it seems there are three other distinctive features other than the outline.

Firstly, we believe the green stork piece of the orange is carried through as a distinctive feature. This would make sense since the stork part of an orange is more obvious on an orange from wider angles than an apple. Secondly, the shine on the side of the orange is preserved through the network. This would be helpful as the reflection is affected by the 3D shape and the texture. Lastly, the top region of the orange was often visible. Both oranges and apples have a slight dip on the top however an apple's dip is much deeper and steep, leading to a different type of shadow. It could be the model is differentiating between an apple and an orange based on the upper shadow as shown in the bottom centre image below.

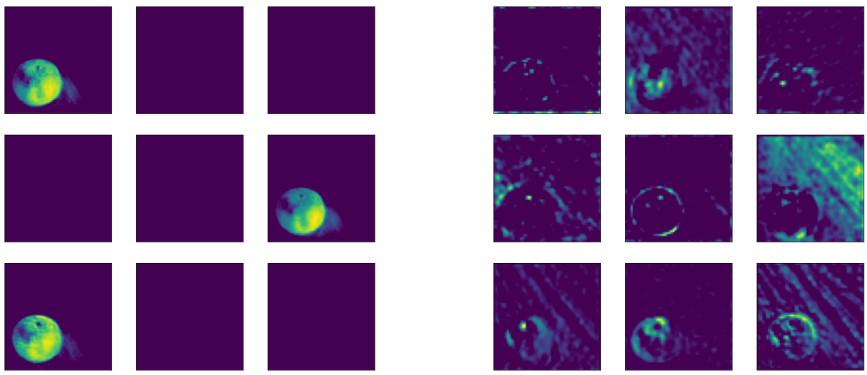

Figure 25: Feature Maps of an Orange at earlier (left) and later (right) stages in the model.

In summary, the model and the app seem to perform very well in normal conditions and there are some interesting observations to be made from digging deeper and experimenting with challenging images and conditions. The model does seem to rely on colour and can be confused by shadows and inconsistent backgrounds with great changes in pixel intensity. For a lightweight fast model it is very effective for general use. With regards to the dataset, it seemed to contain good variation in angle, scale and image quality with fair representation of all items in different conditions. However it could be extended to include more scenes so as to better challenge the model.

# 7 Conclusion and Future Work

To refer back to the goals, the project has produced a new way to record and classify food items from campus eateries. We have shown successful results with a pre-trained CNN and trained-from-scratch CNN developed with the machine learning task in mind. The app implementation meets the goal of allowing users to quickly and easily record food items. The associated nutrition and spending information is laid out in a presentable, easy to interpret manner. The app has shown good resilience to challenging environment conditions.

The project also had the wider goal to make the recording process easier. We believe the fun method of capturing the food, together with location based prediction, has met this goal. The presentation of the nutritional and the spending totals can provide the user a better awareness of their activity. This helps users to change unhealthy habits and improve their lives. In summary the project has met all of the goals initially set out.

There are many ways this project can be extended for future work. Aspects of the dataset, model and

app could be scaled to include more items. The dataset could also be made more challenging by introducing more background variation and shadows. The nutritional information on each item could be more detailed as long as this isn't a detriment to increasing the user's understanding.

An interesting extension would be to use an online model so the model can continuously improve over time with an expanding user created dataset. A more computationally complex approach using an inception model would then be possible. Using a genetic algorithm for hyperparameter optimization would be particularly interesting.

It would be interesting to extend the Bayesian prediction to take other measurements in to account such as sound levels from the microphone, time, weather and activity level throughout the day.

Finally, with regards to the app, it could be developed for iOS platforms as well. The user engagement could be increased by developing a community/social functionality. This could allow people to compete by sharing their sugar intake for example.

# References


Agarwal, C. and Sharma, A., 2011, November. Image understanding using decision tree based machine learning. In ICIMU 2011: Proceedings of the 5th international Conference on Information Technology Multimedia (pp. 1-8). IEEE.

Bossard, L., Guillaumin, M. and Van Gool, L., 2014, September. Food-101–mining discriminative components with random forests. In European conference on computer vision (pp. 446-461). Springer, Cham.

Chalmers, M. and MacColl, I., 2003, January. Seamful and seamless design in ubiquitous computing. In Workshop at the crossroads: The interaction of HCI and systems issues in UbiComp (Vol. 8).

Chen, L.C., Papandreou, G., Kokkinos, I., Murphy, K. and Yuille, A.L., 2014. Semantic image segmentation with deep convolutional nets and fully connected crfs. arXiv preprint arXiv:1412.7062.

2017. Calorie MAMA. Azumio Inc.

De Bonis, M., Amato, G., Falchi, F., Gennaro, C. and Manghi, P., 2018, September. Deep Learning Techniques for Visual Food Recognition on a Mobile App. In International Conference on Multimedia and Network Information System (pp. 303-312). Springer, Cham.

Guirado, E., Tabik, S., Rivas, M.L., Alcaraz-Segura, D. and Herrera, F., 2019. Whale counting in satellite and aerial images with deep learning. Scientific reports, 9(1), pp.1-12.

GOV.UK. 2018. Health Profile For England: 2018. [online] Available at: ¡https://www.gov.uk/government /publications/health-profile-for-england-2018¿ [Accessed 8 April 2020].

Kagaya, H., Aizawa, K. and Ogawa, M., 2014, November. Food detection and recognition using convolutional neural network. In Proceedings of the 22nd ACM international conference on Multimedia (pp. 1085-1088).

Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.

Lee, H.Y., Lee, H.K. and Ha, Y.H., 2003. Spatial colour descriptor for image retrieval and video segmentation. IEEE Transactions on Multimedia, 5(3), pp.358-367.

MasterCard Social Newsroom. 2020. Mastercard Advisors Study On Contactless Payments. [online] Available at: ¡https://newsroom.mastercard.com/press-releases/new-mastercard-advisors-study-on-contactless-payments-shows-almost-30-lift-in-total-spend-within-first-year-of-adoption/¿ [Accessed 8 April 2020].

Meyers, A., Johnston, N., Rathod, V., Korattikara, A., Gorban, A., Silberman, N., Guadarrama, S., Papandreou, G., Huang, J. and Murphy, K.P., 2015. Im2Calories: towards an automated mobile vision food

diary. In Proceedings of the IEEE International Conference on Computer Vision (pp. 1233-1241).

2015. Picturethis - Plant Identifier. PictureThisAI.

Rosenblatt, F., 1958. The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6), p.386.

Shekhar, A., 2020. Amitshekhariitbhu/Android-Tensorflow-Lite-Example. [online] GitHub. Available at: ¡https://github.com/amitshekhariitbhu/Android-TensorFlow-Lite-Example¿ [Accessed 8 April 2020].

Suresh, G., Gnanaprakash, V. and Santhiya, R., 2019, March. Performance Analysis of Different CNN Architecture with Different Optimisers for Plant Disease Classification. In 2019 5th International Conference on Advanced Computing  Communication Systems (ICACCS) (pp. 916-921). IEEE.

TensorFlow. 2020. Tensorflow. [online] Available at: ¡https://www.tensorflow.org/¿ [Accessed 8 April 2020].

TensorFlow. 2020. Tensorflow Lite Examples — Machine Learning Mobile Apps. [online] Available at: ¡https://www.tensorflow.org/lite/examples¿ [Accessed 8 April 2020].

GitHub. 2020. Tensorflow/Examples. [online] Available at: ¡https://github.com/tensorflow/examples¿ [Accessed 8 April 2020].

Turing, A., 1948. Intelligent machinery (1948). The Essential Turing, pp.395-432.

Ukfinance.org.uk. 2020. Card Spending — UK Finance. [online] Available at: ¡https://www.ukfinance.org.uk/data-and-research/data/cards/card-spending¿ [Accessed 8 April 2020].

Werbos, P.J., 1994. The roots of backpropagation: from ordered derivatives to neural networks and political forecasting (Vol. 1). John Wiley  Sons.

Yanai, K. and Kawano, Y., 2015, June. Food image recognition using deep convolutional network with pre-training and fine-tuning. In 2015 IEEE International Conference on Multimedia  Expo Workshops (ICMEW) (pp. 1-6). IEEE.

Zhang, Y., Wang, S., Ji, G. and Phillips, P., 2014. Fruit classification using computer vision and feedforward neural network. Journal of Food Engineering, 143, pp.167-177.

2020. Convolution. [image] Available at: ¡https://www.researchgate.net/figure/A-valid-convolution-of-a-5x5-image-with-a-3x3-kernel-The-kernel-will-be-applied-to_fig5_322505397¿ [Accessed 8 April 2020].

2020. Max-Pooling. [image] Available at: ¡https://www.semanticscholar.org/paper/Improvement-of-Decision-on-Coding-Unit-Split-Mode-Jiang/ba0d8bb137bbb257ff4fddfbf5ada6f3788efa8f¿ [Accessed 8 April 2020].

2020. NN Visualiser. [image] Available at: ¡http://alexlenail.me/NN-SVG/index.html¿ [Accessed 8 April 2020].

2020. VGG16 Diagram. [image] Available at: ¡https://neurohive.io/en/popular-networks/vgg16/¿ [Accessed 8 April 2020].

# 8 Appendicies

Libraries

- Keras
- Tensorflow
- Numpy
- Matplotlib
- CV2
- Tensorflow Lite Java

| Scene | CoffeeMedium | CrispsBlue | Apple | Banana | Orange | Juice | Total % |
|---|---|---|---|---|---|---|---|
| Striped | 4 | 3 | 4 | 6 | 4 | 6 | 75 |
| Background | 6 | 3 | 0 | 0 | 1 | 0 | 27.8 |
| In hand | 1 | 5 | 3 | 6 | 0 | 6 | 58.3 |
| | 6 | 3 | 1 | 4 | 5 | 6 | 69.4 |
| Outside | 6 | 6 | 1 | 6 | 5 | 6 | 83.3 |
| Grass | 0 | 3 | 0 | 0 | 1 | 6 | 27.8 |
| Low Light | 4 | 3 | 6 | 6 | 6 | 1 | 72.2 |
| | 0 | 0 | 1 | 3 | 2 | 0 | 16.7 |
| Shadows | 1 | 5 | 2 | 6 | 5 | 6 | 69.4 |
| | 1 | 4 | 0 | 5 | 5 | 3 | 50 |
| Totals % | 53.3 | 73.3 | 53.3 | 100 | 66.7 | 83.3 | 71.67 |
| | 43.3 | 43.3 | 6.6 | 40 | 46.7 | 50 | 38.33 |

Figure 26: Table of Testing the Transfer trained model(the top cells) and the fully trained model (the bottom cells)


Figure 27: Image of a packet of crisps being processed with the wrong colour model.

Figure 28: The Campus Food Classifier Logo

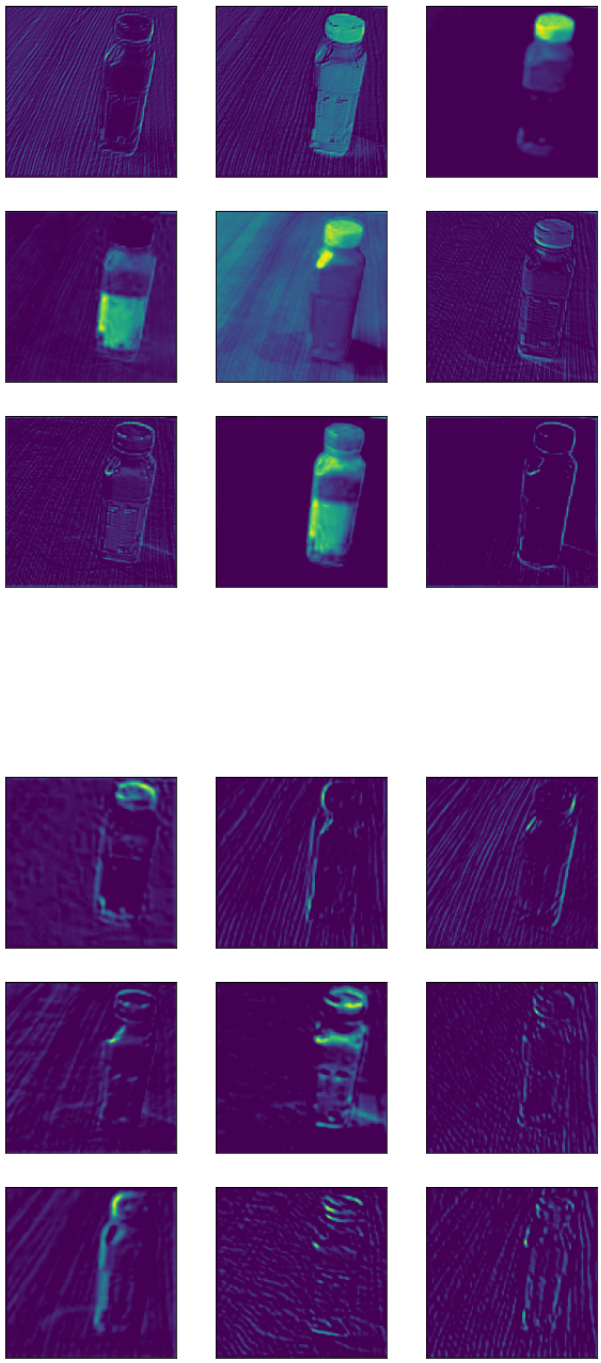

Figure 29: Feature Maps of a Juice bottle from the pre-trained VGG CNN

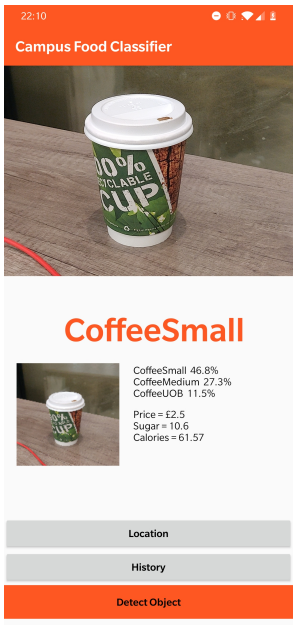


Figure 30: Screenshot of the app classifying an unseen coffee cup with different packaging.

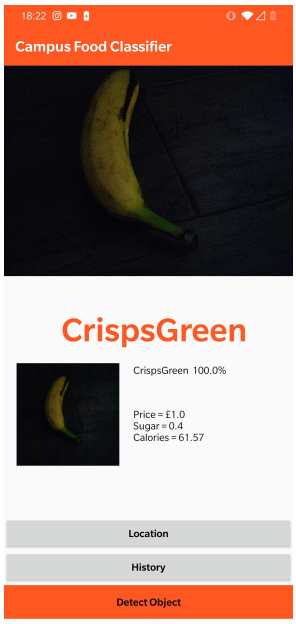


Figure 31: Testing in low light this banana with some slight amount of green was classified as Green crisps.

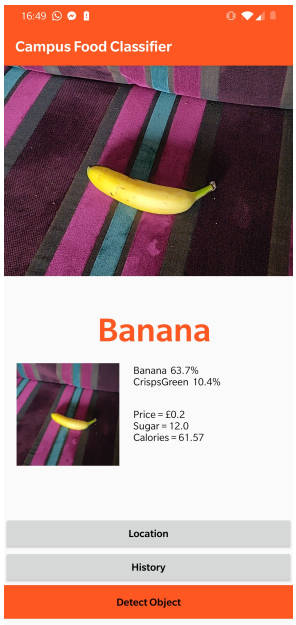


Figure 32: Screenshot of the app testing on a striped background

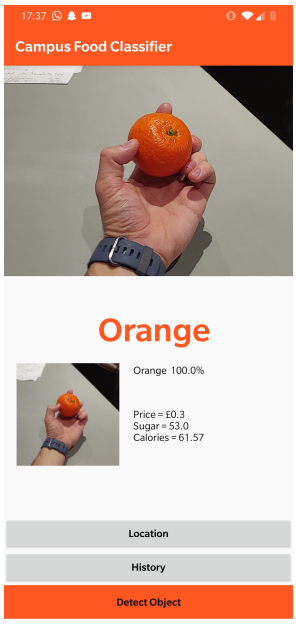


Figure 33: Screenshot of the app classifying an item in-hand.

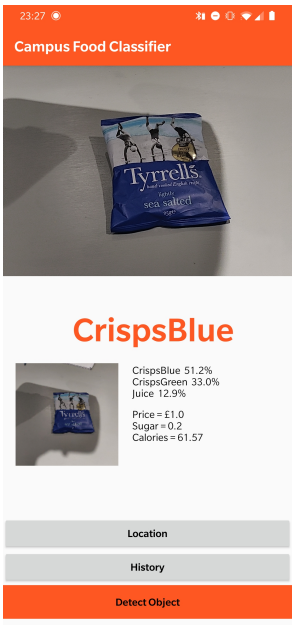


Figure 34: Testing the app with shadows.

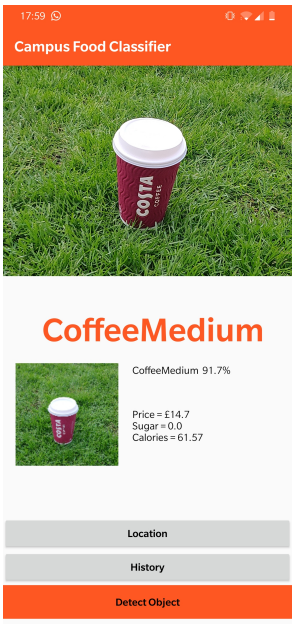


Figure 35: Testing the app on grass.