CookWhatAh

ISS PRSPM TEAM 18 PROJECT REPORT

CHENG KOK CHEONG A0038791W

DANIEL TAN HOONG XIANG A0074608B

ONN WEI CHENG A0092201X

YANG JIESHEN A0003901Y

TABLE OF CONTENTS

APPENDICES

1. PROBLEM STATEMENT

Electrolux Food Survey 2014 survey reports that although 33% of respondents are more likely to eat out than to eat at home, 80% of respondents prefers a home-cooked meal.[1] Given that many residing in Singapore prefers a home-cooked meal, what is stopping us from cooking at home? Among others, reasons suggested range from the possibility of leftover ingredients from making a meal causing food wastage to their inexperience in the kitchen.

Other than just being a mere preference, home-cooked meals also provide other benefits. According to an article written by Mount Elizabeth Hospital, in light of healthier options found in the hawker centre in the last few years, home-cooked food still proves to be the healthier option as it is packed with better nutrition and enables better control of fat, sugar and salt.[2]

1.1. OUR PROPOSAL

To help people living in Singapore enjoy more home-cooked food, our team proposes CookWhatAh, an Integrated Cooking Application, that can help users solve many of their problems they find when cooking at home. CookWhatAh targets both the uninitiated and experienced cooks with the following functionalities:

Identifying ingredients with photos

In cases where users might not know the name of the ingredient, CookWhatAh will be able to detect the ingredients using image recognition. This will be perfect for users who does not do the grocery shopping themselves but still want to whip up a quick meal.

Recipe recommendation based on selected ingredients

With a smaller family size, sometimes cooking a meal will come with many leftover ingredients. CookWhatAh can search for recipes that contains selected ingredients. This will be great for users that wish to find recipes that uses leftover ingredients and reduce food waste.

Easy to follow instructions and nutrition information

Being on a digital platform enables CookWhatAh to provide videos and/or step-by-step instructions for recommended recipes. This is a drastic improvement from the traditional cookbook and can guide inexperience cooks to replicate the dish more accurately. Nutrition and caloric information are also readily available for recipes provided by CookWhatAh to help with making better food choices.

[1] https://www.asiaone.com/singapore/cook-home-its-hassle-many
[2] https://www.mountelizabeth.com.sg/healthplus/article/food-delivery-dining-out

Additional ingredients recommendation

Not forgetting more experienced cooks, CookWhatAh can also recommend additional ingredient(s) that will work well with the dish. This can help with experimenting with additional and more complex flavours, therefore spicing up the dish.

1.2. PROJECT OBJECTIVES

The goal of the project will be to deliver a Minimum Viable Product of our Integrated Cooking Application, CookWhatAh with the following objectives:

Image Recognition system

To give a good overview of the application, the image recognition system should offer a good classification accuracy of different ingredients over a variety of categories.

Recipe Recommendation

The recipe recommender should be able to offer recipe recommendations based on selected ingredients. It is preferred that the recommender will be able to select recipes that will make use of all ingredients, if not, a subset of the selected ingredients. The selected recipes should also provide easy to follow instructions and other information relevant for recipe selection.

Additional ingredients recommendation

The additional ingredient recommender should be able to offer sensible ingredient(s) recommendations based on ingredients in the current recipe.

Convenient Deployment

To ensure that the app can be conveniently used by users, the application should be deployed on a mobile device to provide easy access to a camera. The UI of the application should also offer alternative input possibilities other than requiring for a photo of the ingredient to be uploaded.

2. SYSTEM OVERVIEW

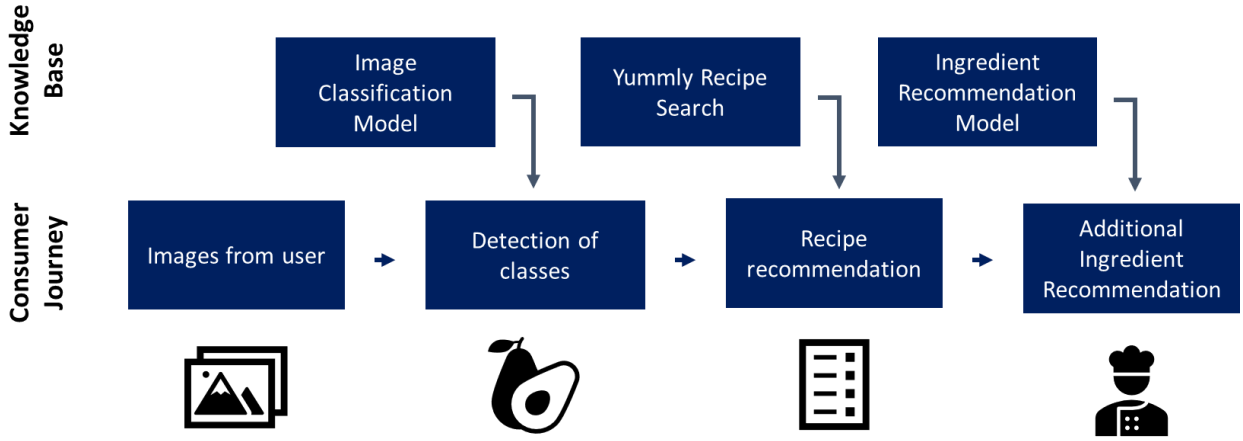


Figure 1: System Overview

An overview of the system in given in **Figure 1**. The following sub-sections summarise the 3 major pipeline of the application. The application has been also been developed as a telegram application for easy usage.

2.1. IMAGE CLASSIFICATION SYSTEM

The image classification step of the application will be responsible for the classifying the images into one of the 25 ingredients shown in **Table 1**.

Type	Ingredients
Meat	Beef, Chicken & Salmon
Vegetables	Bean Sprout, Broccoli, Cabbage, Carrot, Celery, Corn, Cucumber, Eggplant, Green Bean, Olive, Onion, Potato, Spinach & Tomato
Fruit	Apple, Avocado, Banana & Lemon
Others	Bread, Cheese, Mushroom & Egg

Table 1: Classifiable Ingredients

Image classification will automatically be triggered after an image is uploaded to the telegram application by the user. The identified ingredients will be added to the list of selected ingredients for recipe recommendation. This will be the primary method of adding ingredients to the recipe list. Alternatively, we also allow users to key in the ingredients manually.

In the event where the image is unclear or is unable to be detected with a high level of accuracy, the application will offer a list of guidelines for the users and request for a better photo to be uploaded. More information on the guidelines can be found in **Appendix III**. Users are also able to refine the ingredients identified through the edit and delete functions provided.

2.2. RECIPE RECOMMENDATION SYSTEM

After all the ingredients has been added to the ingredient list, the user will be able to proceed with recipe recommendation step based on the selected ingredients. Our recipe recommendation system leverages the search engine of a reputable recipe website, “Yummly.com”, with more than 2 million recipes and huge community base who will rate and upload recipes. It contains a variety of information such as cooking instructions, videos and even caloric information, which makes it a strong contender as compared to other websites.

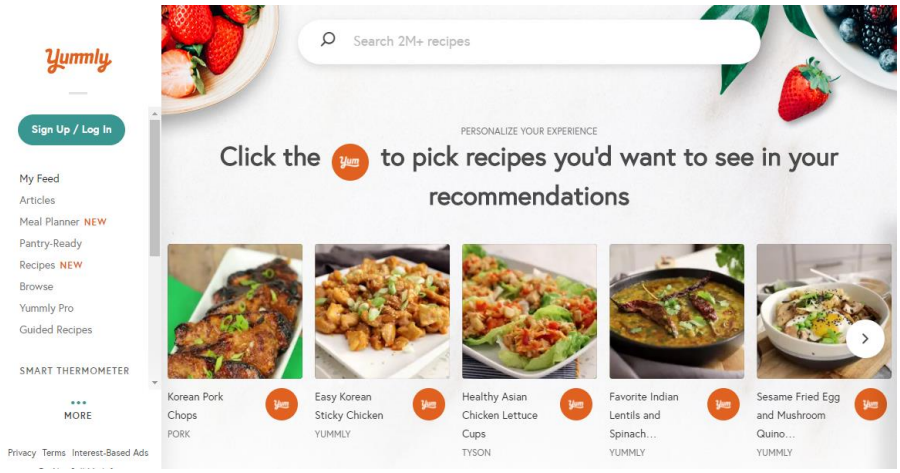


Figure 2: Yummly.com Website

Utilising the search engine of the “Yummly.com”, the top 10 recipes containing selected ingredients will be recommended to the users. However, in the case where there are no recipes available that contains all the selected ingredients, our system will exclude the ingredients one by one starting from last added ingredient.

From the list of recipes, the user will then be able to select a recipe that they are interested in. After which, more details including the ingredients required and a link to a step-by-step recipe will be provided all through the convenience of the application.

2.3. ADDITIONAL INGREDIENT RECOMMENDATION SYSTEM

When a recipe is selected, the additional ingredient recommendation step will also be activated to recommend one additional ingredient on top of the ingredients given by recipe from “Yummly.com”. This recommendation can be continued iteratively until the user stops asking for any ingredient, or when the system can no longer generate any new ingredients. This will be useful for experienced cooks who want to go beyond standard recipes and experiment with new ingredient(s) that might enhance the flavour and complexity of the dish.

The first recommendation is generated by using the ingredients in the selected recipe to make the first prediction. After which, if the user asks for another ingredient, the preceding recommendation(s) together with the original ingredients will be used to make the next prediction.

3. SYSTEM IMPLEMENTATION

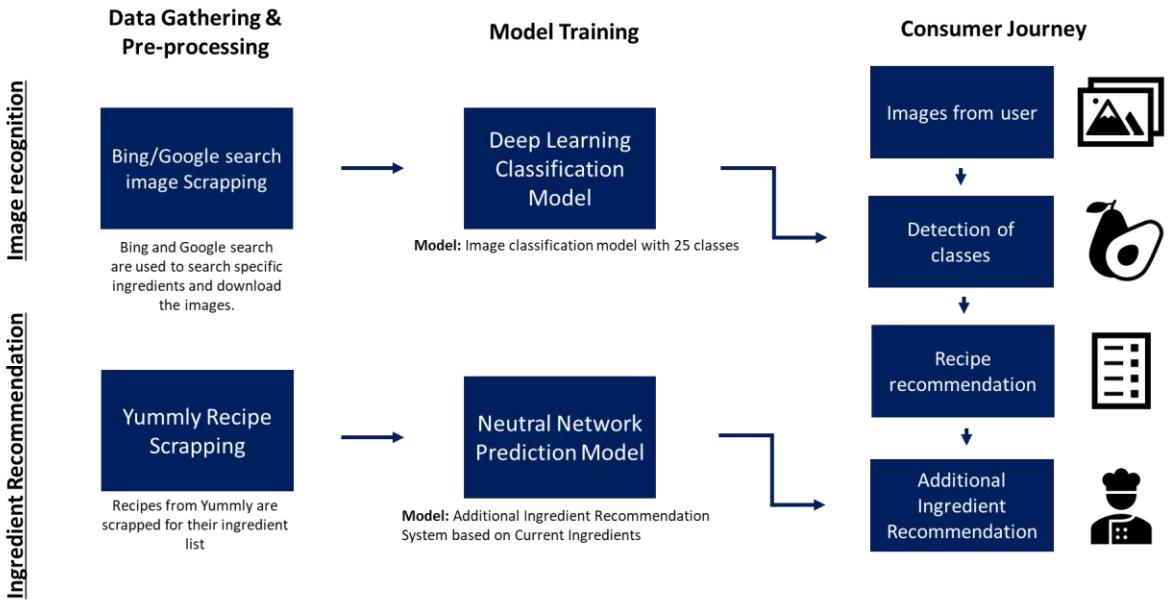


Figure 3: System Implementation Flow

Figure 3 summarises the implementation flow of the image classification and ingredient recommendation model. It also explains the integration of the models with the consumer journey (experience) of the application. The following subsections will illustrate in greater details, the implementation of the models, recipe recommendation system and system integration.

3.1. DATA GATHERING & PRE-PROCESSING

3.1.1. BING/GOOGLE SEARCH IMAGE SCRAPPING

To facilitate the training of our model which requires large number of datasets, two different scripts with different performance were developed for data mining.

Bing Image Scrapping

The first script leveraged the Bing search API of Microsoft azure, which allowed us to search and download images using the search engine. However, we realised that the classification of the images was not ideal. There were duplicated images and numerous misclassifications which required us to do a lot manual cleaning.

Google Image Scrapping

Taking the learnings from the first script, and through our experiences with using Google Image search, we realised that Google related image search was a better alternative to Bing search API which yielded more accurate findings and less misclassification. However, the development of Google Image Scrapping was much more involved as we did not have access to the Google search API, at least for free.

The script activates the google search related image search by using the chrome browser and simulated input commands using “selenium” python library. After which, the script reads the HTML tags from google image related search results and uses the “beautifulsoup” python library to parse them into web addresses to be downloaded. For efficiency, “concurrent future” python library was also used to run concurrently searches.

This method resulted in images with less duplications and misclassification. A hash comparison script was also adapted to compare images and remove duplicates. Manual cleaning, albeit much lesser than the previous method, was still required as some images contain multiple ingredients and should be removed for training of a single target ingredient classification model. Manual cleaning also reduced the number of similar images (resized or augmented) that cannot be removed with hash comparison.

3.1.2. YUMMLY RECIPE SCRAPPING

Recipes and ingredients derived from Yummly recipe scrapping was used in both the recipe recommendation system and additional ingredient recommendation model. Yummly was chosen as the website of choice because of the following reasons:

In-build Search Engine

“Yummly.com” has an in-build search engine that allows for the search for recipes that only contains our selected ingredients, which makes it a perfect search engine for our use case. Furthermore, the usage of the in-build search engine from “Yummly.com” reduces our workload of designing a search algorithm and a database.

Website organisation

“Yummly.com” has very organised HTML tags that allows us to easily retrieve ingredients off the recipes without cleaning and additional processing

Information Availability

Other than ingredients required for recipes, “Yummly.com” also contains a variety of other information such as cooking instructions, videos and even caloric information. Tapping on this resource, users of our application is also able to get access such resources and improve their cooking experience.

3.1.2.1. SCRAPPING FOR RECIPE RECOMMENDATION

To use “Yummly.com for recipe recommendations, the in-build search engine is first used to obtain recipe recommendations based on the selected ingredients using the python “request” library. After which, the python library, “BeautifulSoup”, is used to search for HTML tags in the source code to extract useful information. An iterative process runs through the links of the recommended recipes to extract information such as ingredients, recipe links, prep time, etc. This information is stored in short-term memory and parsed when the user selects a recipe.

3.1.2.2. SCRAPPING FOR ADDITIONAL INGREDIENT RECOMMENDATION

The same methodology was also used to scrape data for the additional ingredient prediction system but with some modifications. The final dataset consists of 6700 recipes with approximately 600 ingredient groupings.

Scoping of recipes to be scrapped

As “Yummly.com” consist of more than 2 million recipes, the search was scoped using a combination of the 25 pre-defined ingredient classes. This ensures that recipes scrapped will include the pre-defined ingredients and useful for our usage.

Filtering of recipes

To ensure a credible training dataset for our additional ingredient recommendation system, we tapped on the rating results of the huge community on “Yummly.com”. Only recipes with more than 3.5 out of 5 stars were used for training.

Processing of information scrapped

A total of 6700 recipes with more than 5000 unique ingredients were obtained even after scoping and filtering. This was due to the inconsistent naming conventions, causing repeated ingredients. Therefore, the dataset was categorized and labelled manually to get rid of plurality, brands and other undesirable noise. The dataset was further processed through a one-hot encoder to be used for training of the additional ingredient recommendation model.

3.2. MODEL DEVELOPMENT AND TRAINING

3.2.1. IMAGE CLASSIFICATION MODEL

A convolution deep neural network model was used to build the image classification model. The model takes in coloured images with 3 channels and gives a prediction of one of the 25 classes.

The input images are first rescaled from [0,255] to a [0,1] and passed through 5 2D convolution layers with kernel size (3,3) for features extraction. The output of the convolution layers is flattened and passed through a single fully connected layer with a SoftMax activation function for classification. The output node with the highest "probability" will be chosen as the predicted ingredient. The model has a total of 648,729 parameters.

A total of approximately 25,000 ingredient images over 25 classes were used for the training of the model. **Table 2** shows the number of images per class. The ingredients were divided into a training and testing dataset with an 80/20% split.

No.	Category	Quantity	No.	Category	Quantity
1	Potato	1574	14	Cabbage	1012
2	Tomato	1353	15	Eggplant	619
3	Bean Sprout	953	16	Mushroom	793
4	Salmon	1019	17	Egg	1035
5	Chicken	1040	18	Lemon	711
6	Apple	930	19	Celery	959
7	Broccoli	1026	20	Beef	1001
8	Avocado	1133	21	Spinach	1301
9	Banana	1093	22	Cucumber	505
10	Green Bean	1141	23	Corn	744
11	Bread	1007	24	Cheese	1083
12	Olive	1172	25	Onion	1085
13	Carrot	583			

Table 2: No. of ingredients per class

Various techniques were being used to improve accuracy of the model and to facilitate training. Batch normalization was used to keep the distribution of the inputs from layer to layer relatively stable to ensure that the contours (features) instead of the pixel value of the image were being learnt, while L2 Regularisation was used to prevent overfitting. Geometrical images augmentation including shifts, rotation and flips were used during training to increase the generalisability of the model. Dropout and variable learning rates were also used to improve training efficiency. More design information of the model can be found in **Appendix X**.

With the various techniques implemented, an accuracy of 88.16% was obtained, with precision of individual items ranging from 70% to 95%. Further interpretation of the results including other experimentation will be discussed in **Section 4**.

3.2.2. ADDITIONAL INGREDIENT RECOMMENDATION MODEL

The main objective of the ingredient recommendation model is to recommend or predict an additional ingredient, which is not present in the selected recipe, but can possibly enhance the recipe. Two methods, clustering and artificial neural network, were explored, with the latter being eventually chosen for our model.

3.2.2.1. CLUSTERING OF INGREDIENTS

There are 583 unique ingredients groupings found in our recipe database. Clustering was performed to uncover any potentially interesting relationships between the ingredients. The intuition was that ingredients that are naturally compatible with each other in creating pleasurable taste perception would be found together in a recipe and therefore form clusters among themselves.

Due to the categorical nature of the dataset, instead of using KMeans, which works based on Euclidean distances, KModes was employed instead. KModes uses the concept of dissimilarity to cluster data. A new centroid is determined by the most common category per feature (mode). The analysis uses “KModes” python library.

Two methods of clustering are explored. In the first method clustering was done based on similar recipes. After which, the centroids of the clusters can be used to determined common ingredients. In the second method, clustering was done based on ingredients and the clusters themselves can be used to determine common ingredients. **Figure 4** and **Figure 5** shows a plot of the silhouette score, a combine measurement of inter and intra-cluster distances, and cost, a measurement of intra-cluster distance for method 1 and 2 respectively.

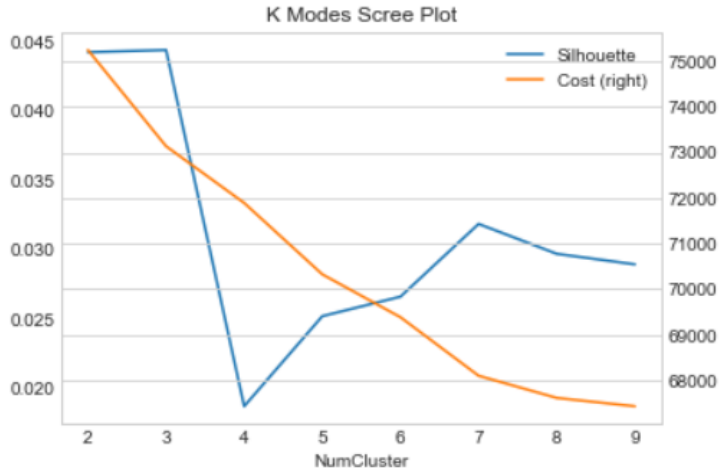


Figure 4: K Modes Scree Plot (Cluster by Recipes)

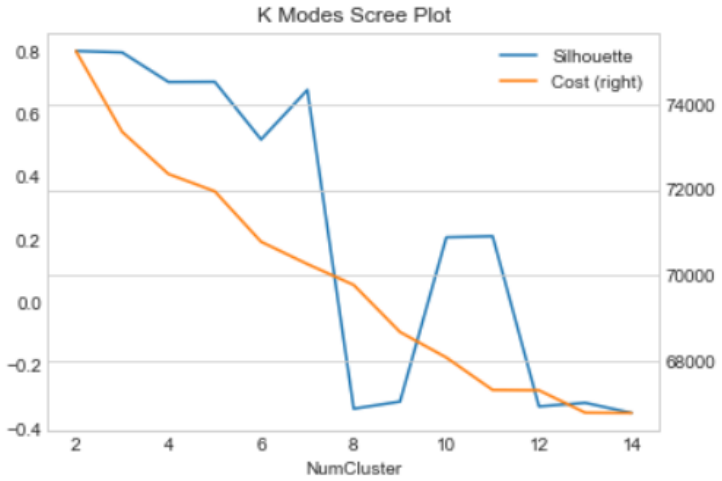


Figure 5: K Modes Scree Plot (Cluster by Ingredients)

The first method gave rise to extremely low silhouette values which was not ideal. Each cluster centroid also had overlapping ingredients and gave no meaningful results. The second method on the other hand had a better balance between silhouette scores and cost. Therefore, further analysis was conducted on the latter.

The clusters formed from the dataset, using the second method, typically consist of a big cluster (due to the sparsity of the dataset) consisting of most of the ingredients, with approximately 500 ingredients out the total of 583, and a few smaller clusters, with one or a few ingredients.

Other than relying on the silhouette scores and cost to decide on the number of clusters, the clusters chosen also had to make sense. Balancing the sensibility of the clusters with a high silhouette score and low cost, 11 clusters were chosen. The smaller clusters were found to be more meaningful and we were able to label 7 of the 11 clusters. **Table 3** shows our finding.

Cluster #	**1**	**2**	**5**	**6**	**7**	**8**	**9**
Ingredients	Bay Leaf	Egg	Olive Oil	Avocado	Banana	Garlic	Ginger
	Carrot	Flour	Pepper	Bell Pepper	Cream	Onion	Sesame Oil
	Celery	Oil		Cheese	Milk	Salt	Soy Sauce
	Stock	Sugar		Chicken	Pudding		Vinegar
	Thyme			Chili			
				Cilantro			
				Cumin			
				Sour Cream			
# of Ingredients	**5**	**3**	**6**	**3**	**11**	**7**	**5**
Label	**Western**	**Bakes**	**Seasoning**	**Mexican**	**Dessert**	**Garlic, Onion & Salt**	**Asian**

Table 3: K Means Cluster Details

To make use of the clustering results for ingredients recommendation, the system will have to recommend the missing ingredients from a cluster, for which some of the ingredients are present in the recipe.

However, based on our clustering results there are several disadvantages in using this method. Firstly, by dropping the main cluster of approximately 500 ingredients, a lot of information will be lost, and we will have to rely on the remaining clusters and ingredients for recommendation. In other words, recommendations cannot be made for recipes with ingredients that are not within any of the remaining clusters.

Secondly, as recipes can contain ingredients that does not only belongs in one cluster, inter-cluster effects cannot be account for. For example, a recipe containing carrots and corn, two typical ingredients found together, might work better with soy sauce than paprika, or vice versa, depending if the dish is more Chinese or Mexican.

3.2.2.2. ARTIFICIAL NEURAL NETWORK

In general, the compatibility of an ingredient relies on how well it enhances the sensory response of the consumers. It is inherently difficult to lay down rules that dictate whether a collection of ingredients could go well with each other. It was hypothesized that neural network approach could overcome this limitation that comes from the very nature of the issue, by making prediction based on a dataset of recipes with combination of ingredients generally well-accepted by the population.

The neural network model was designed based on the idea that each credible recipe consists of ingredients that work well with each other. If one ingredient was removed and the rest of the ingredients were parsed through the model, it should predict the missing ingredient.

Each recipe is able to generate datapoints equivalent to the number of ingredients that make up the recipe. For example, a recipe A, that uses ingredients 1, 2, 3, 4 & 5, will be able to generate 5 training datapoints as shown in **Table 4**.

	Input Ingredients	Output Ingredient
Recipe A – 1	1, 2, 3, 4	5
Recipe A – 2	1, 2, 3, 5	4
Recipe A – 3	1, 2, 4, 5	3
Recipe A – 4	1, 3, 4, 5	2
Recipe A – 5	2, 3, 4, 5	1

Table 4: NN Dataset Generation

The above data augmentation was applied to each recipe which resulted in 84,000 datapoints from the original 6700 recipes. Each datapoint is made up by a combination of the 583 unique ingredients groups. By treating each ingredient as both an input feature and output target, the prediction problem can be viewed as a single target multi-class classification problem (583 training features, 583 possible classes and 84,000 data samples).

The dataset was first split into training and unseen datasets in the ratio of 4:1, and then the training set was further split into actual training and validation sets in the same ratio. After multiple trial and errors, an architecture of 4 fully connected hidden layers was adopted as it produced the best validation accuracy of 60%.

In particular, the following is incorporated in the architecture:

a. He Normal scheme was used to initialize the weights of all the hidden layers for better convergence between the training and validation performance metrics
b. L2-regularizer was included in the loss computation of all the hidden layers to reduce overtraining.
c. Dropout layers were introduced between successive hidden layers.

3.3. APP DESIGN

3.3.1. TELEGRAM INTEGRATION

Telegram was chosen to be the user interface for our system as it allows for easy integration of the application with smart phones cameras, while also providing us a ready-made chatting interface that many are familiar with. This allowed us to create an application that is convenient to use but relatively easy to deploy.

The python “telegram-bot” library was used to gain access to the functionality of telegram for the detection of user inputs in the form of photos and text messages. Depending on the prompts from the user and the flow of the system, the application will send the request to our different models and reply with the appropriate response, which include the identified ingredient, recommended recipes or suggested ingredients. Exception handling was used to ensure that unintended replies from the users will be handled smoothly without crashing the system.

3.3.2. HEROKU INTEGRATION

The application was also deployed on Heroku so that we do not need to host the system on a personal server nor run any code before the usage of the application. Anyone who is interested to use the application will be able to access it on telegram at any time without any knowledge of coding. The application can be accessed by searching CookWhatAh on Telegram or using the QR code in **Appendix I**.

However, using Heroku as a free account does not come without its complications. Being a free account, only 450 hours are only available per month and the server come with limited memory and storage space of 500 MB. We face several issues initially, including R14 memory errors and limited space to install libraries. We had to optimise the system by running garbage collection to reduce the memory space used at any one time and install partial libraries such as “tensorflow-cpu” instead of the usual “tensorflow” to work around space constraints.

4. FINDINGS AND DISCUSSIONS

4.1. IMAGE CLASSIFICATION MODEL

4.1.1. MODEL EVALUATION

The image classification model achieved a highest accuracy of 88.16% over 25 classes, with many classes having precision over 90%. The details of the individual classes are shown in **Figure 6.**

Best accuracy (on testing dataset): 88.16%

	precision	recall	f1-score	support
Apple	0.9553	0.9194	0.9370	186
Avocado	0.9675	0.6593	0.7842	226
Banana	0.9457	0.9587	0.9522	218
BeanSprout	0.9476	0.9526	0.9501	190
Beef	0.8545	0.9100	0.8814	200
Bread	0.8986	0.9254	0.9118	201
Broccoli	0.8230	0.9756	0.8929	205
Cabbage	0.8601	0.8137	0.8363	204
Carrot	0.9145	0.9224	0.9185	116
Celery	0.9056	0.8534	0.8787	191
Cheese	0.8982	0.9398	0.9186	216
Chicken	0.8396	0.8558	0.8476	208
Corn	0.9724	0.9527	0.9625	148
Cucumber	0.7952	0.6535	0.7174	101
Egg	0.9679	0.8744	0.9188	207
Eggplant	0.8729	0.8374	0.8548	123
GreenBean	0.9761	0.8947	0.9336	228
Lemon	0.9310	0.9507	0.9408	142
Mushroom	0.6923	0.6835	0.6879	158
Olive	0.9125	0.9359	0.9241	234
Onion	0.7860	0.7788	0.7824	217
Potato	0.7368	0.8917	0.8069	314
Salmon	0.9333	0.8276	0.8773	203
Spinach	0.8125	0.9500	0.8759	260
Tomato	0.9776	0.9704	0.9740	270
accuracy			0.8816	4966
macro avg	0.8871	0.8755	0.8786	4966
weighted avg	0.8865	0.8816	0.8811	4966

Figure 6: Model Evaluation

An accuracy of over 88% for a classification model of 25 classes is 20 times more effective then random classification which makes it a very reliable model.

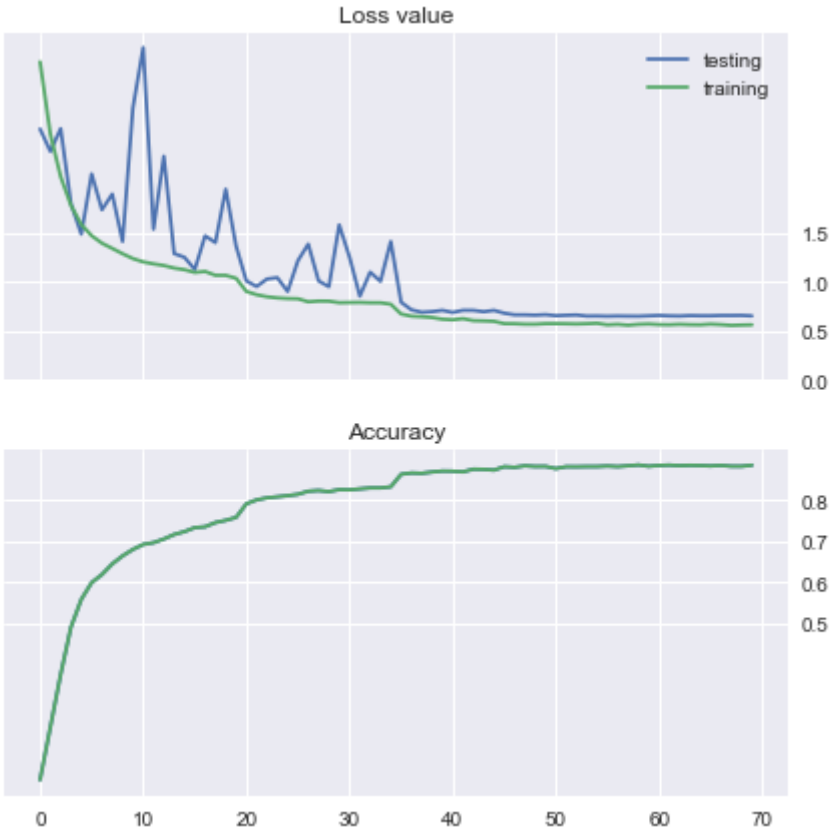


Figure 7: Loss value and accuracy across epoch

Epoch	Learning Rate
0	0.0005
20	0.00025
35	0.00005
45	0.000005
55	0.0000025

Table 5: Stepwise learning rates changes

Figure 7 shows the effectiveness of the stepwise learnings rates implemented during training. Even though the loss value of the training data fluctuated at lower epoch, it stabilised at higher epoch where learning rates were lower. A step improvement in accuracy was seen at epochs 20 and 35, where the learning rates were reduced.

The accuracy of the testing and training data also converged which signified a good model learning without overfitting to training data. This gave confidence that the various methods discussed in **Section** Error! Reference source not found. were effective.

4.1.2. FINDINGS AND LEARNINGS

Although the overall accuracy of the model was relatively high, there are some classes with lower than desirable precision and recall. The lowest precision and recall were found for “mushroom” at 69% and 68% respectively.

After evaluation of different factors, one of the probable reasons found was that the dataset that we used for training consist of undesirable images. Our images were scrapped from google using search terms such as fresh mushroom and raw egg to ensure that only uncooked ingredients are being downloaded. After which, the dataset was also manually filtered for mixed and incorrectly classified images.

Figure 8: Example of sliced vs whole ingredients

Figure 9: Example of ingredient in large quantities vs small quantities

However, despite our efforts to construct a good training dataset, images containing ingredients that were chopped and sliced as shown in **Figure 8** remains. Images also include ingredients that were present in large quantity as shown in **Figure 9**. It is suspected that these imperfections in the dataset increased the difficulty of deriving correct features from the images during training, causing lower accuracy.

Moreover, as ingredients are typically only processed before cooking and usually used in relatively small quantities, restricting our dataset to contain only images of uncooked, whole ingredients in small quantities will improve the accuracy of the model while being in line with the objective of our image classification model.

We targeted for the number of images per class used for training were kept balance within the 900-1200 range per class, but due to resource constraints, it was not achieved for all classes. Although down sampling or super-sampling techniques could have been used no additional steps were taken because the affected classes had relatively good accuracy levels.

4.1.3. REAL LIFE IMAGES ANALYSIS

To further validate our image recognition model, real life images were used for analysis. The result of the analysis proves to be promising with some exceptions.

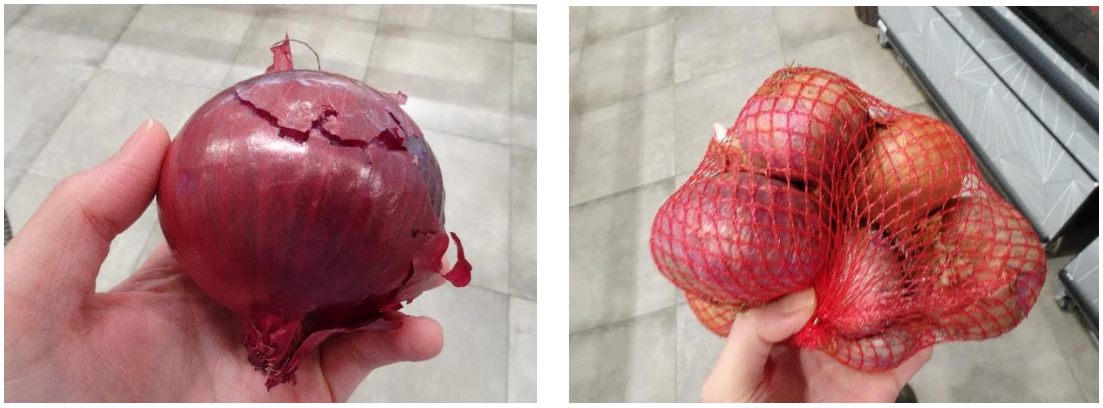


Figure 10: Example of bagged ingredients vs individual ingredient

As the image scrapped online typically contain ingredients that are not bagged, the model does not perform well on ingredients that are bagged or wrapped. **Figure 10** shows an example of such images. The model correctly predicts the left image as “onion” with 71% probability, while incorrectly predicting the right image as “beef” with a probability of 99%.

Through the analysis we also found that images with prediction probability under 50% are typically predicted incorrectly. Therefore, a threshold of 50% was placed for the model. The users will be asked to re-upload a better photo using suggestions given by the application on how the photo can be taken. More information of the suggestions can be found in **Appendix II**.

4.1.4. OTHER MODEL EXPERIMENTATIONS

To improve accuracy of the model, we also tested other model architectures. We noticed that as the number of classes increased, the accuracy of the model drops slightly. Therefore, one of the other model architectures that was explored involves first classifying the images into different shapes – Round, Elongated, Others – before further classifying the ingredients into its individual classification using separate models for each initial classification. However, for the model to be comparable to the current model, each of the model must at least have the same or higher accuracy level as compared to the current model. This was not achieved with the experimental model and therefore the current model was used.

4.2. ADDITIONAL INGREDIENT RECOMMENDATION MODEL

4.2.1. PRINCIPAL COMPONENT ANALYSIS

As the feature dimension is rather huge, Principal Component Analysis (PCA) was attempted to reduce the dimension of the dataset, such that important information of the dataset can still be captured with lower number of independent representations. Conceptually, this would help with faster training of any downstream model.

Statistical significance was achieved when Bartlett and KMO tests were performed on the feature dataset for factor analysis. **Figure 11** shows the Pareto Chart which displays the cumulative variance explained by the principal components (PC). 80% of the variance in the data can be explained by the first 350 – 400 PC's (about 30 – 40% reduction in the dimension). However, since the benefit of PCA is not particularly significant for the dataset and we were not very much limited by memory capacity, PCA was not further pursued and the full dataset was be used for model training.

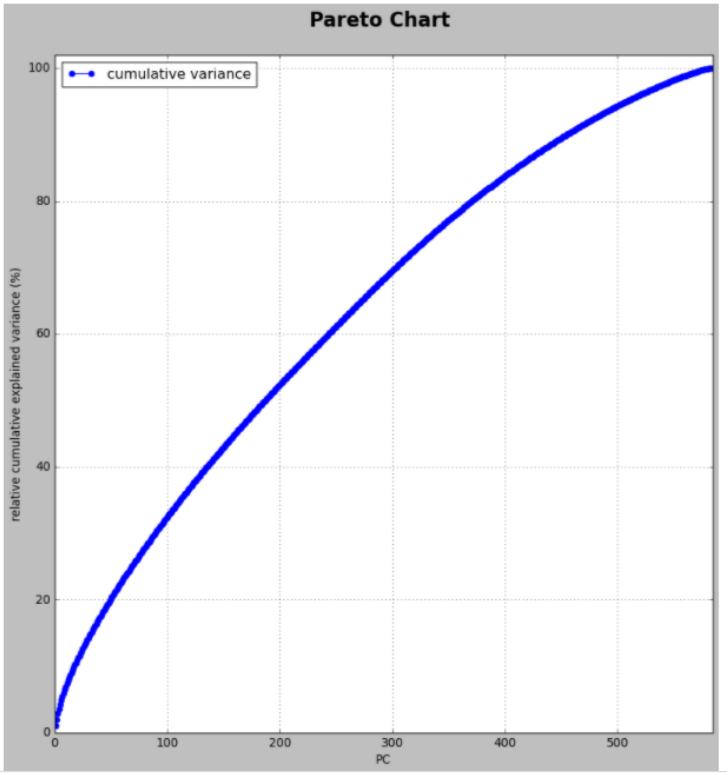


Figure 11: Pareto chart of cumulative variance explained by PCs

4.2.2. FINDINGS AND LEARNINGS

The model produced a prediction accuracy of around 61% when applied on the unseen dataset, which was consistent with the best validation accuracy achieved. One concern while training the model was that a huge class imbalance was observed in the dataset.

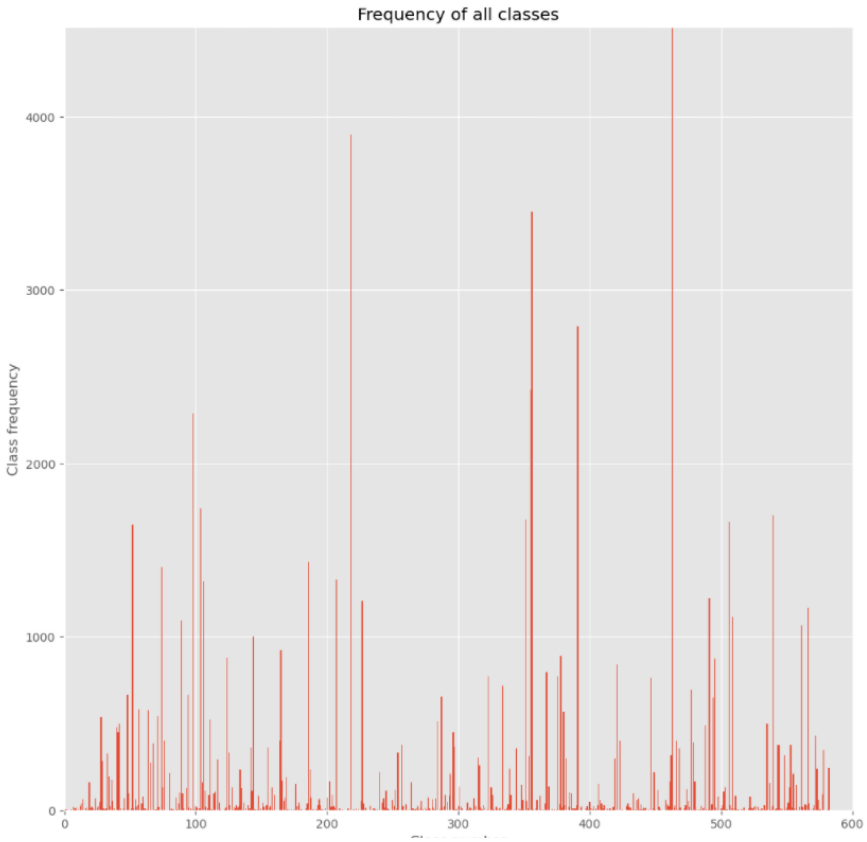


Figure 12: Frequency of Ingredients

Figure 12 shows the number of instances for each class in the dataset. While some class can have as few as only one training instance, there are other classes dominating with frequency in the order of the thousands. To further examine this, the true positives of all classes were plotted against the class supports. **Figure 13** shows the results.

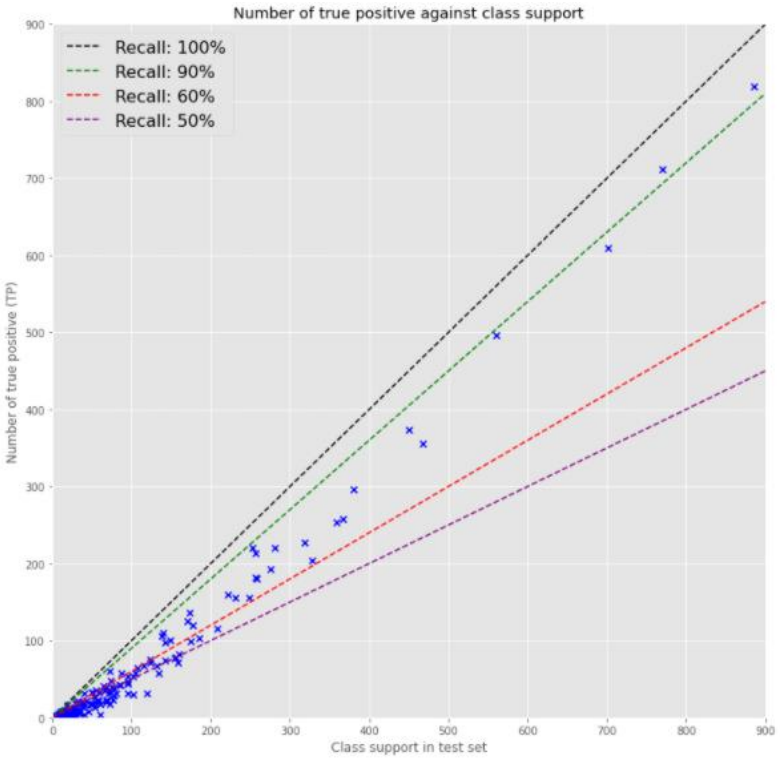


Figure 13: Plot of true positive against class support

The model performed better with the majority classes, with the test set being correctly predicted 90% of the time, while medium classes are being correctly predicted 60-90% of the time. There are two possibilities here, either:

a. The training was dominated by the majority class, such that any pattern in the minority class could not be effectively learned.
b. There was simply insufficient data variation in the minority class to make effective prediction.

To evaluate which of the two possibilities were more likely, two approaches were set out to verify if the model prediction was indeed biased by the majority classes (scenario a):

i. **Approach 1:** Both minority and majority classes of the training set were over-sampled and under-sampled respectively to have equal number of 1000 training samples.
ii. **Approach 2:** Samples of classes with total count less than 2 (first quartile of class frequency) were first removed before the train-validation-test split. The minority and majority classes of the training set were then over-sampled and under-sampled respectively to have equal number of 1000 training samples.

The model was then trained again with the resampled training set, which yielded a validation and unseen data accuracy of about 56% and 59% respectively. **Figure 14** shows the evolution of the training and validation accuracy against the training epochs.

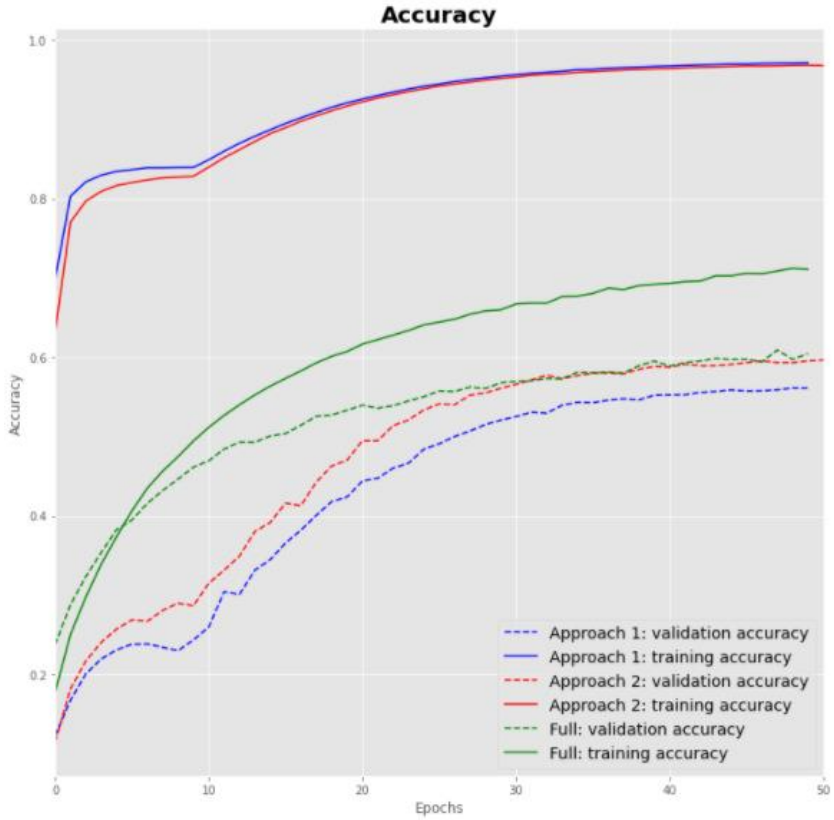


Figure 14: Accuracy of approaches 1 & 2

In all cases, the unseen data accuracy is comparable with the validation accuracy that the models converged towards. When training the model with resampled training data, there was generally a greater overfitting in the model, which was expected as the sample instances in the very extreme minority class have been over-sampled many times for the training dataset. Hence, the model weight might be biased toward these replicated datasets, which might not appear as frequently in the validation dataset as it was not over-sampled.

However, the overall validation and unseen data accuracy is comparable to those obtained without re-sampling (full model). This suggests that the full model is not heavily biased towards majority classes. Therefore, it was likely that the minority classes had insufficient data variation to be picked up by the model during the training process. Considering all the above and the fact that the full dataset resulted in the best generalizable accuracy with least amount of over-training, the full model was chosen as the final ingredient recommendation model.

Another interesting observation was that the model accuracy could be easily trapped in local minimum when the initial learning rate was relatively high or when L1-regularizer was used. While adding more neural nodes aggravated the overfitting, adding more hidden layers resulted in lower prediction accuracy.

5. CONCLUSION

5.1. SUMMARY OF ACHIEVEMENTS

Our integrated cooking application, CookWhatAh, delivers a convenient deployment on mobile application through the telegram application. It allows for easy access to a camera on our smart phone, while offering alternative inputs possibilities. The 3 smart systems, namely, the image recognition system, the recipe recommendation system and the additional ingredient recommendation system, despite some areas to improve, offers good classification and recommendation results. In conclusion, the project has achieved its objectives and delivered a minimum viable product.

5.2. FUTURE IMPROVEMENTS

Despite a successful implementation for a minimum viable product, there are areas for future improvements.

Image Classification Model

a. As discussed in **Section 4.1.2**, better testing accuracy can be improved by restricting our dataset to contain only images of uncooked, whole ingredients in small quantities.
b. As discussed in **Section 4.1.3**, better real-life accuracy can be improved by having greater variation of images for the local context, which includes the typical ways ingredients will be packaged in the supermarkets.

Recipe Recommendation System

a. The recipe recommendation system depends heavily on the HTML tags on "Yummly.com". Any changes in HTML code on their website will require updates to the application. This can be improved by working through a supported API (which Yummly does not have currently) to query information
b. The result of our recipe depends on "Yummly.com" in-built search algorithm, hence we are unable to tweak and improve search results. This can be improved by maintaining our own database of recipes to have control over the search results.

Additional Ingredients Recommender

a. While we are able to get an accuracy from the model by using the validation dataset, a better way to find out the real capability is to engage a domain expert like a chef to try out and feedback if the recommended ingredients actually boosts the original recipe.
b. The current ingredients mapping and one hot encoding is a snapshot of only ingredients included in the training dataset (5000 items from 6700 recipes). For majority of the recipes, there will be some ingredients which are not seen in the

training dataset and will be excluded from the analysis. This can be improved by scrapping more recipes data to cover more ingredients in the mapping with periodic updates. Alternatively, we can gain access to the mapping of ingredients through official API to get any underlying categorization done by the website, if available.

APPENDIX I: SYSTEMS MAPPED TO COURSE OBJECTIVES

System	Sub-System	Relevant Course Objectives
Image Classification System	Bing/Google Search Image Scrapping	- Web Scrapping - Intelligent Sense Making - Supervised Learning - Deep Learning Techniques - Hybrid Machine Learning
	Image Classification Model (Convoluted Neural Network)	
Recipe Recommendation System	Yummly Recipe Scrapping	- Web Scrapping
	Recipe Recommendation using Yummly.com	
Additional Ingredient Recommendation System	Yummly Recipe Scrapping	- Web Scrapping - Supervised Learning - Deep Learning Techniques - Hybrid Machine Learning
	Additional Ingredient Recommendation Model (Neural Network)	

APPENDIX II: COOKWHATAH ACCESS GUIDE

CookWhatAh rides on a Telegram application as the user interface to implement our system. Please install Telegram in your respective platforms to run the application.

Link to CookWhatAh_bot (Heroku Hosted):

https://t.me.CookWhatAh_bot

In the event where Heroku servers are down or the operating hours are maxed out, you will still be able to enjoy our Integrated Cooking Application, CookWhatAh by running it on your computer.

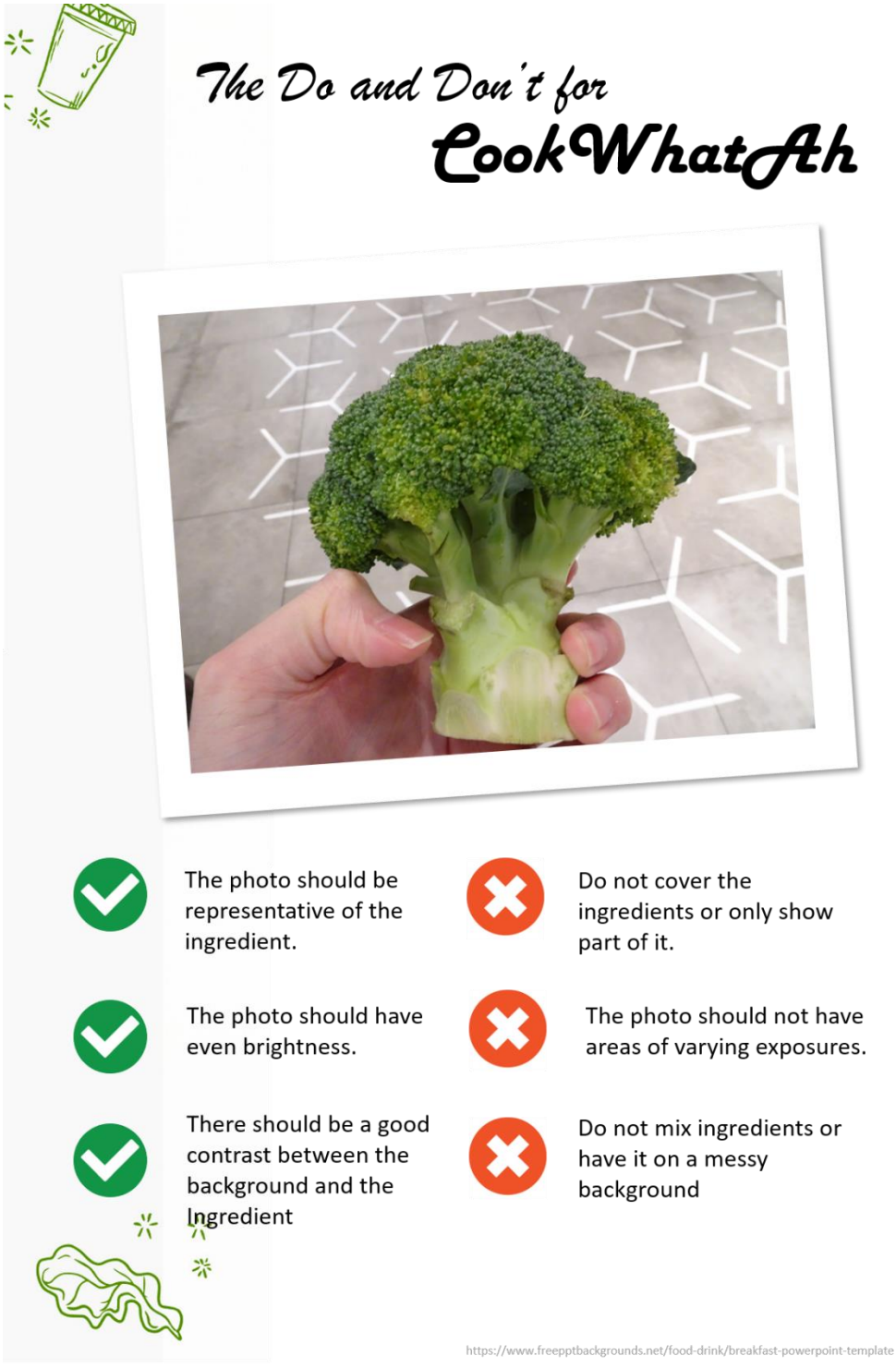
Installation instructions for CookWhatAh_2_bot (Local IDE):

1. Please download CookWhatAh_2_LocalIDE.zip from this link
2. Unzip and extract files into directory of choice
3. Install python dependencies using requirements.txt
4. Run app.py and keep it running while you use the telegram bot
5. In your telegram app in your smartphone/desktop, please search "CookWhatAh_2" or click https://t.me/CookWhatAh_2_bot

Sample Images:

1. Some sample images are located here
2. You can upload these images to try out our system

APPENDIX III: GUIDELINES FOR PHOTO UPLOADS



APPENDIX IV: USER GUIDE

1. You will be able to access CookWhatAh using the QR code in **APPENDIX II** or search for CookWhatAh in telegram.
2. Please start the bot and you see instructions for your first steps. You will be able use the **help** command to see a list of commands that be used for the application.
3. You will be able to start the first ingredient identification by just uploading a photo.
4. Alternatively. You can manually add ingredients using the **add [new ingredient]** command.
5. In the case where the ingredient uploaded is identified incorrectly, you will be able to use the **del [line no.]** and **edit [line no.], [new ingredient]** command to delete and edit the ingredients respectively.
6. After all ingredients are selected, you can use the **done** command to generate the top 10 recipes recommended. After which, you can reply the recipe number to select the recipe.
7. For the more adventurous, an additional ingredient will also be recommended by to spice up your cooking.
8. You can also use the **extend** to prompt the system to recommend you more ingredients based on the current ingredients recommended.
9. If you use **extend [new ingredient]** command, the system will automatically add in the ingredient and recommend you additional ingredient based on the current ingredients, including the one you just added.
10. Users can use **recipe** command to show back the recipe name suggested by the CookWhatAh system.

Starting Screen

Photo upload

Add command

Done command

Del command

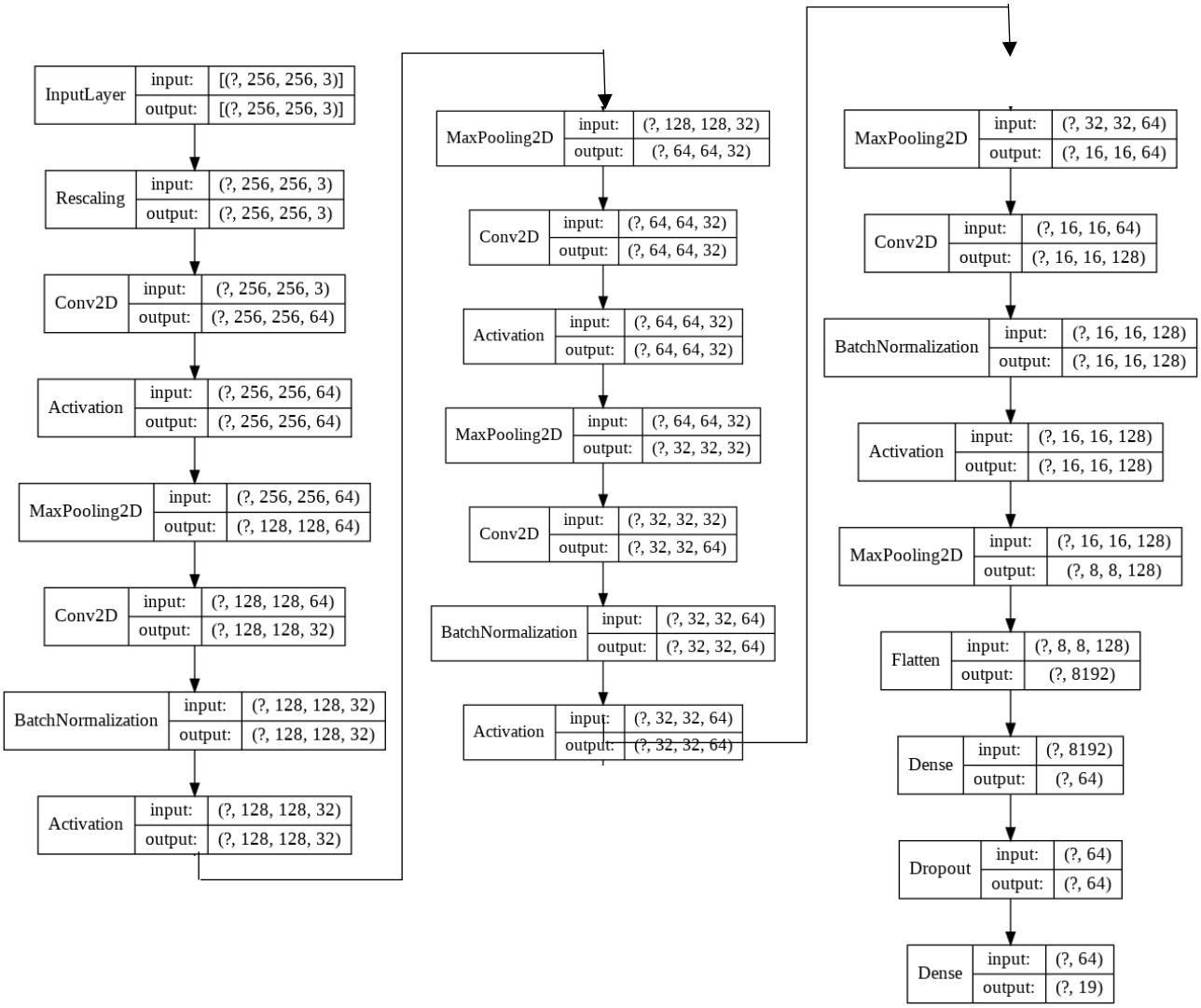
Edit command

Recipe command

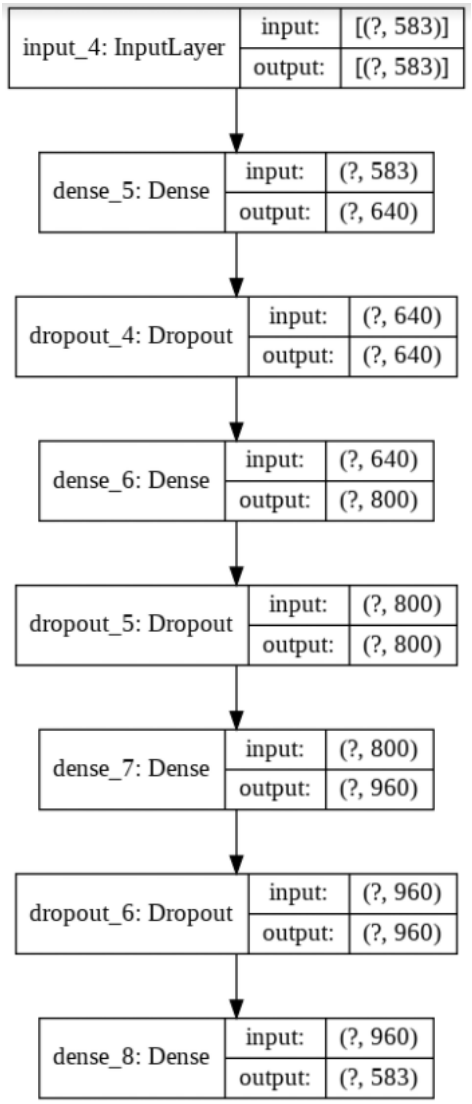
Extend command

Extend with ingredients command

APPENDIX X: MODEL DESIGN FOR IMAGE RECOGNITION SYSTEM



APPENDIX XI: MODEL DESIGN FOR INGREIDENT RECOMMENDATION SYSTEM




```
optmz = optimizers.Adam(learning_rate=0.001)

def createModel():
  input = Input(shape=(num_classes,))
  x = Dense(640, activation='relu', kernel_initializer='he_normal',
            kernel_regularizer=l2(1e-4))(input)
  x = Dropout(0.25)(x)
  x = Dense(800, activation='relu', kernel_initializer='he_normal',
            kernel_regularizer=l2(1e-4))(x)
  x = Dropout(0.25)(x)
  x = Dense(960, activation='relu', kernel_initializer='he_normal',
            kernel_regularizer=l2(1e-4))(x)
  x = Dropout(0.5)(x)
  output = Dense(num_classes, kernel_initializer='he_normal',
                 activation='softmax')(x)

  model = Model(inputs=input, outputs = output)
  model.compile(loss='categorical_crossentropy',
                optimizer = optmz,
                metrics=['accuracy'])

  return model
```