FINAL PROJECT REPORT
FOOD DIARY

TEAM MEMBERS

ANG PAU HUANG, EDWIN
CHEOK MEI LI (MAVE)
PADMAPRIYA MATHIVANAN
PRIYANSH MISHRA

TABLE OF CONTENTS

1. INTRODUCTION

Many of us want to improve our health, their focus and energy level. One essential step towards achieving this is by monitoring and changing what we eat. A common recommendation on monitoring and developing new eating habits is to **keep a food diary**. A diary helps people gain insights into their eating habits, patterns, **identify and categorise food that is healthy (and not-so-healthy)** to eat every day.

Figure 1: Food Diary

Problem our project address

However, given the hustle and bustle of our everyday life, it is difficult to form the habit of manually entering the nutrition of our daily food intake into a food diary, physical or digital. This is when technology can help us. Food diary apps like MyFitnessPal enable us to easily access food entries that has been logged by past users, as well as enable us to scan bar codes to quickly log packaged food.

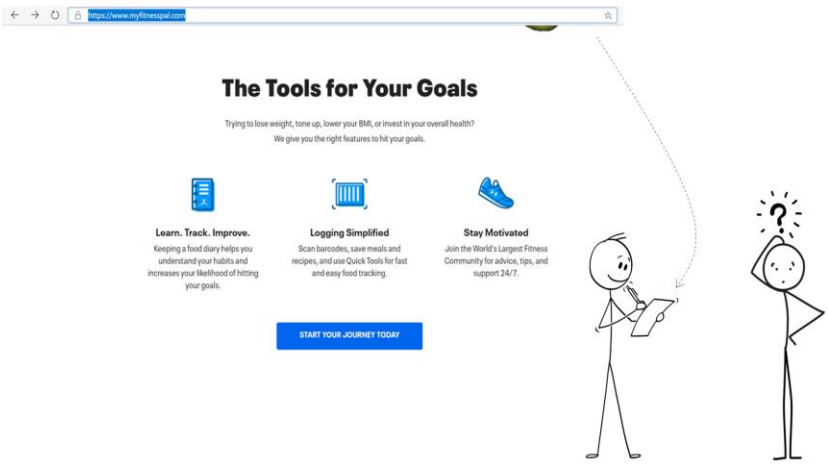


Figure 2: Page from MyFitnessPal website

However, there is the problem of giving a name to the food you just ate (for instance, what is the name of the fruit in Figure 3), and finding a past entry that matches it. The scanning of the barcode also only works if the barcodes exist in the database of the application.

Figure 3: Salak or snake fruit

Our solution

We as team of four members aims to provide a solution that enables consumer to log their food entries by simply taking photos of their food. With the prevalence of social media, many people take photos of their food anyway. This allows people to easily be aware of the nutrition quality of what they eat and track their intake from raw food (fruits eg.) and packaged food.

Our current MVP (or simple food diary, as we call it) can be of great help not just to keep track of calories but also to inform users on the nutrition grade of the food product that they intake. We hope this application paves way towards more healthier eating habits.

Our project experience

This COVID-19 pandemic has not made it easy for us to collaborate as we did not see each other in person during project development period. However, we had regular online meetings and helped each other through the challenges of working on this project. The pandemic also further motivates us to want to develop this application to help our loved ones keep healthy. Overall, we are very grateful for this project experience, and this report is a log of our learnings.

2. SYSTEM DESIGN

2.1 User Workflow

Firstly, we would like to provide an overview of our simple food diary app. This minimum viable product is created on the principles of user friendliness and simplicity, yet able to demonstrate the core selling point of our product which is that users do not have to enter much information into the system besides the image of their food to the food diary.

The users access the app through a web browser, and the images of the food can be sent in through the users' webcam or separately uploaded. With a click of a "Capture" button or a drag-and-drop of the image file, the image file will be captured in the food diary.

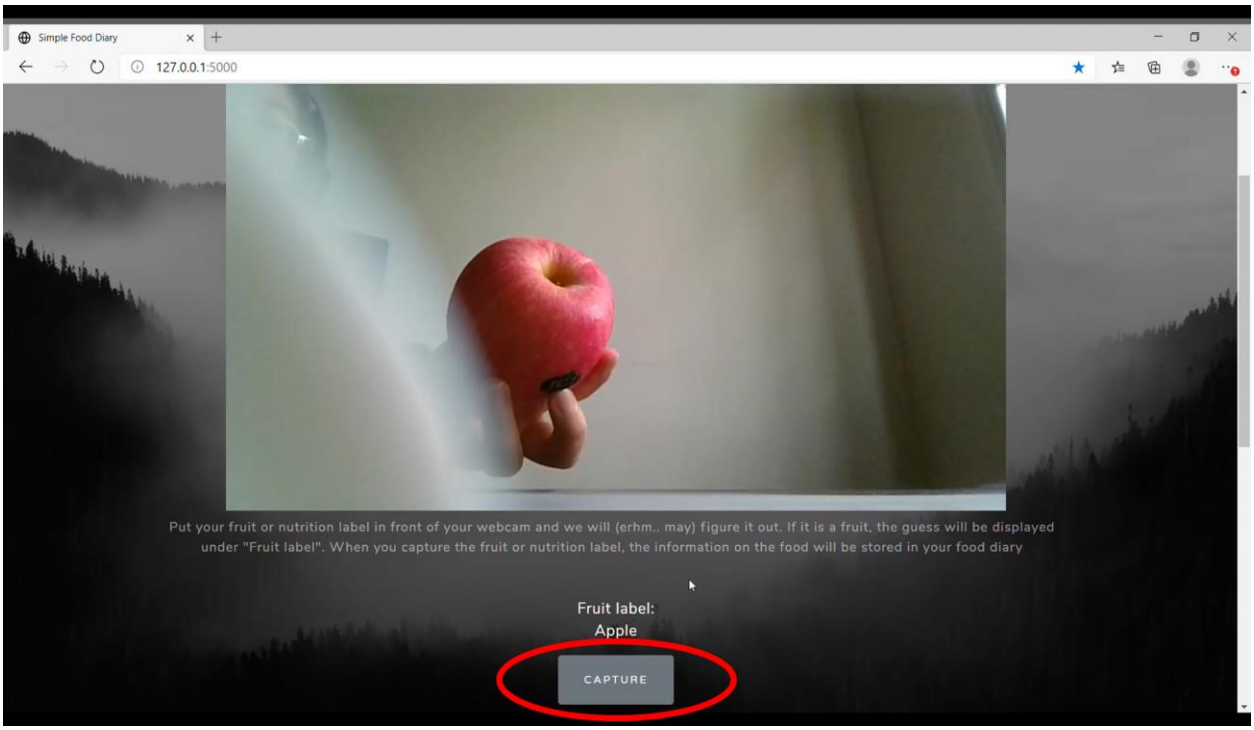


Figure 4: Capture button

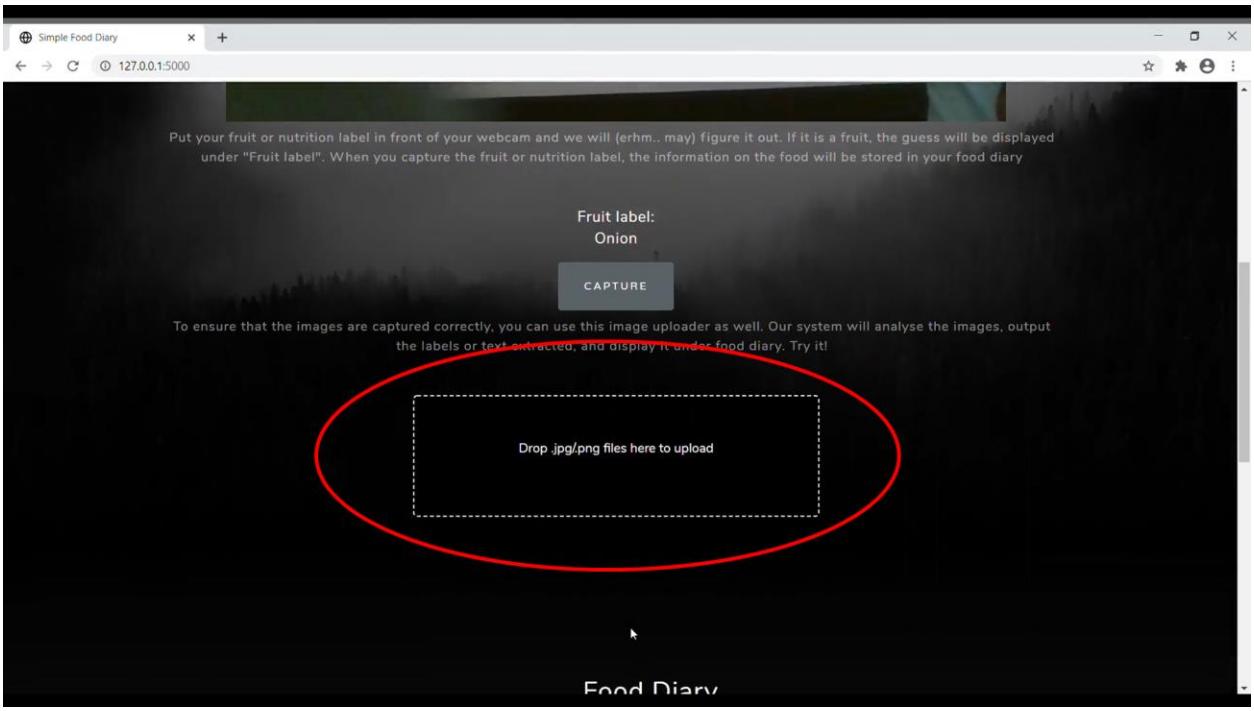


Figure 5: Upload image

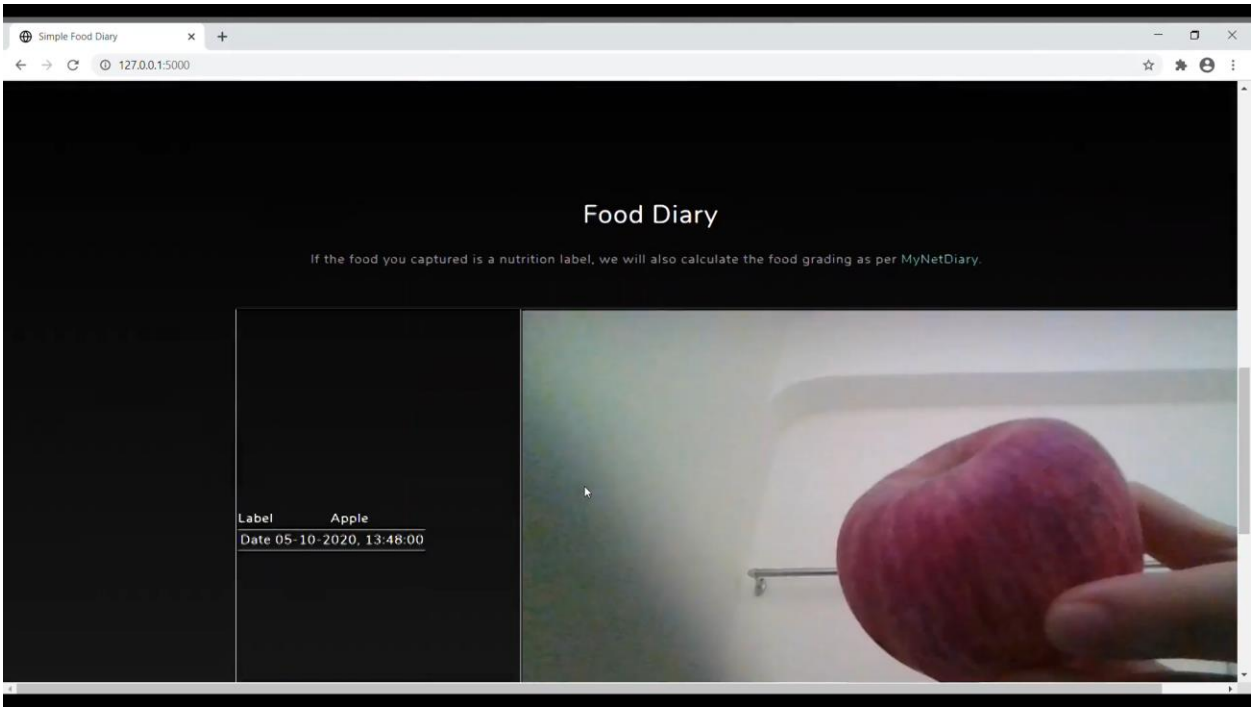


Figure 6: Food diary (Apple)

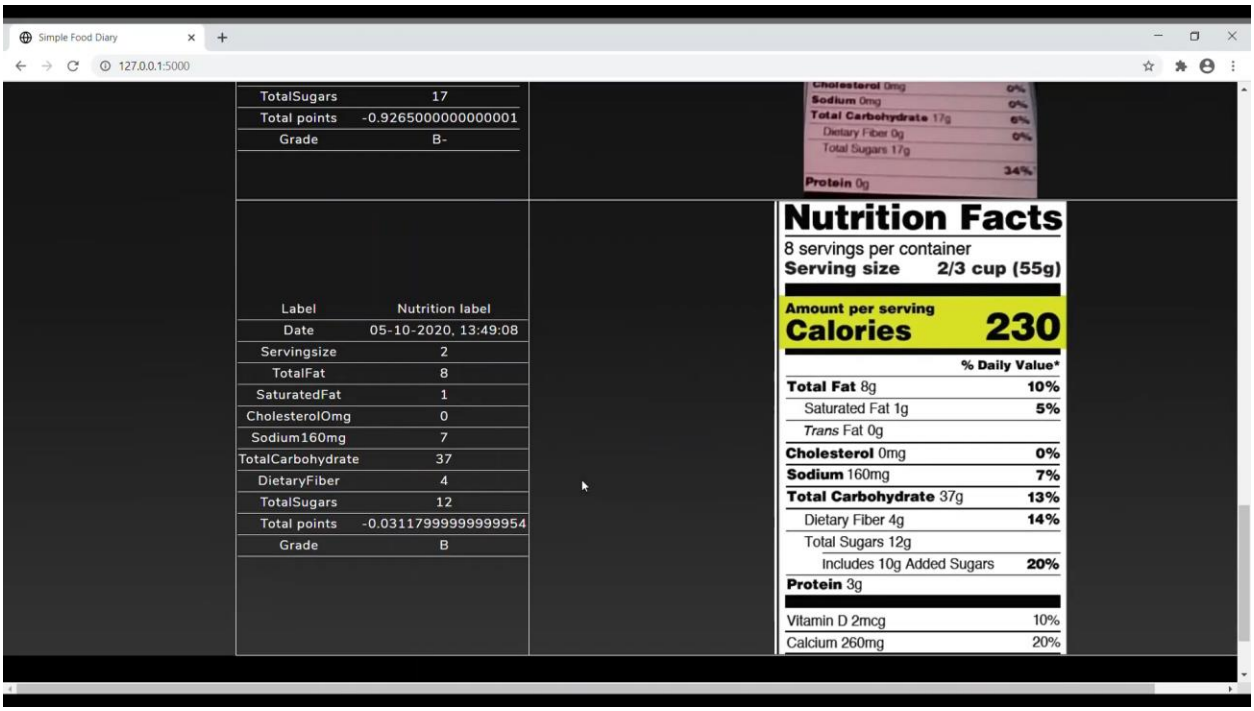


Figure 7: Food diary (nutrition label)

The following diagram illustrates the overall user workflow for our application:

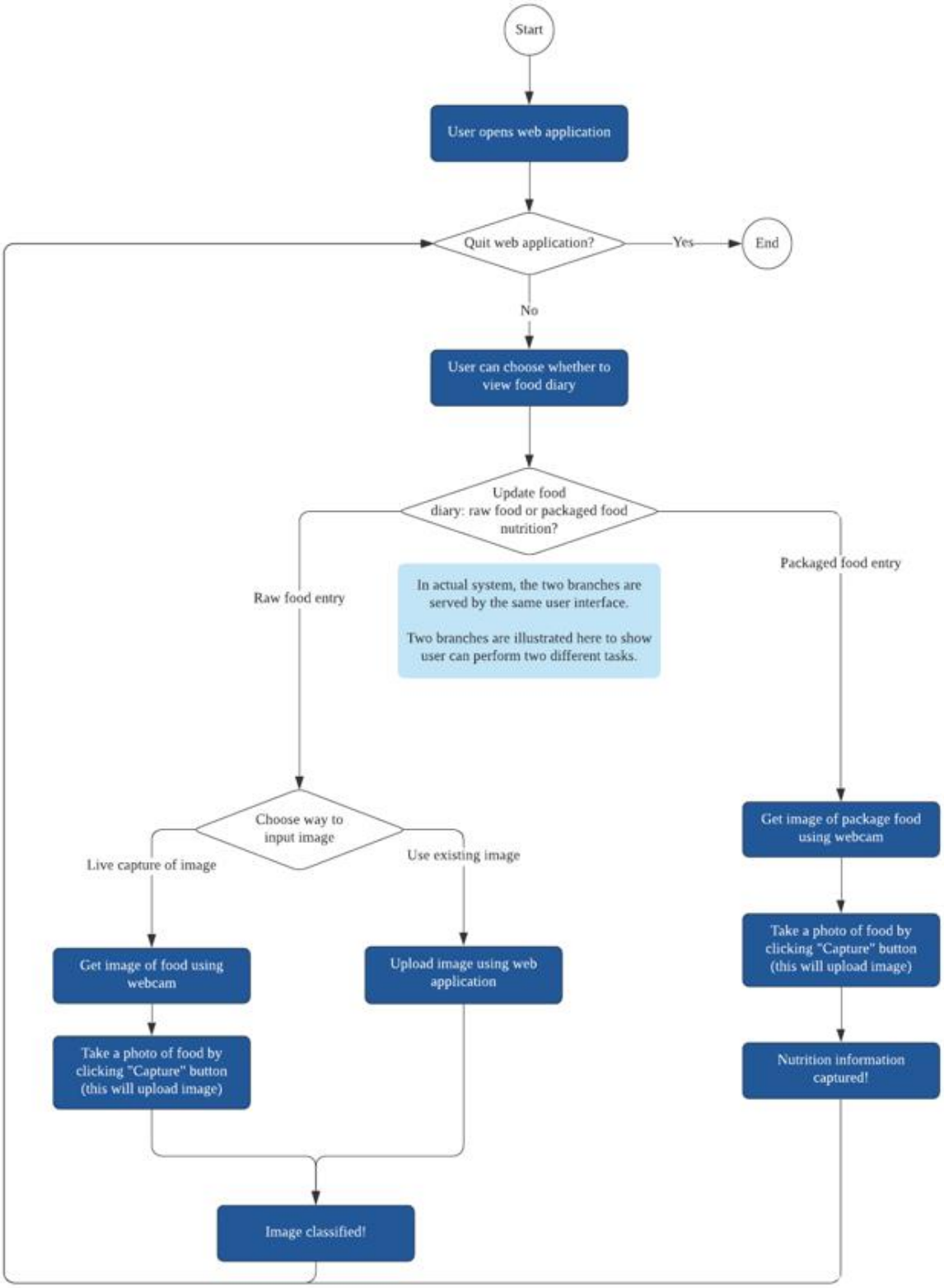


Figure 8: User workflow

2.1 System Architecture

Now, we will go into the intricacies of the system. The key components of the system are:

1. **Python/Flask server:** The backend application that exposes a REST API for the front-end application to retrieve/send information. It hosts most of the components within the architecture and interacts with them, directing data flow among the components.

2. **Front-end:** For the HTML pages to interact with flask server, it needs to have Jinja templating. The diagram below also shows the system flowchart of how the user interacts with the interface.

3. **Amazon Rekognition:** This is an external service we send our images to, to have the text extracted.

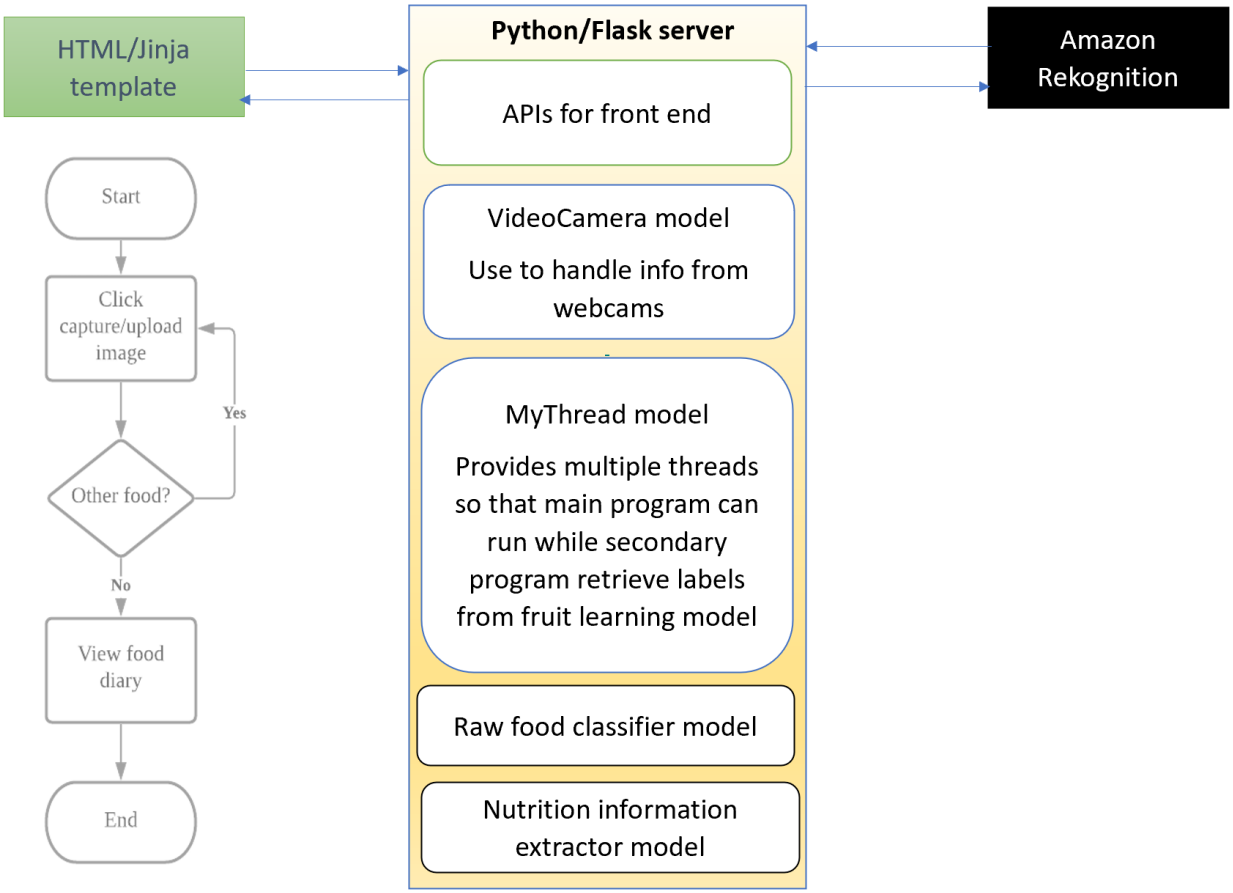


Figure 9: Key System Components

The system architecture diagram below shows the relationship between the different components of the system. The systems within the "Internal server" swim lane is built by us, while we leveraged on Amazon Web Services' Rekognition to extract text from the nutrition labels that were recognized by our models.

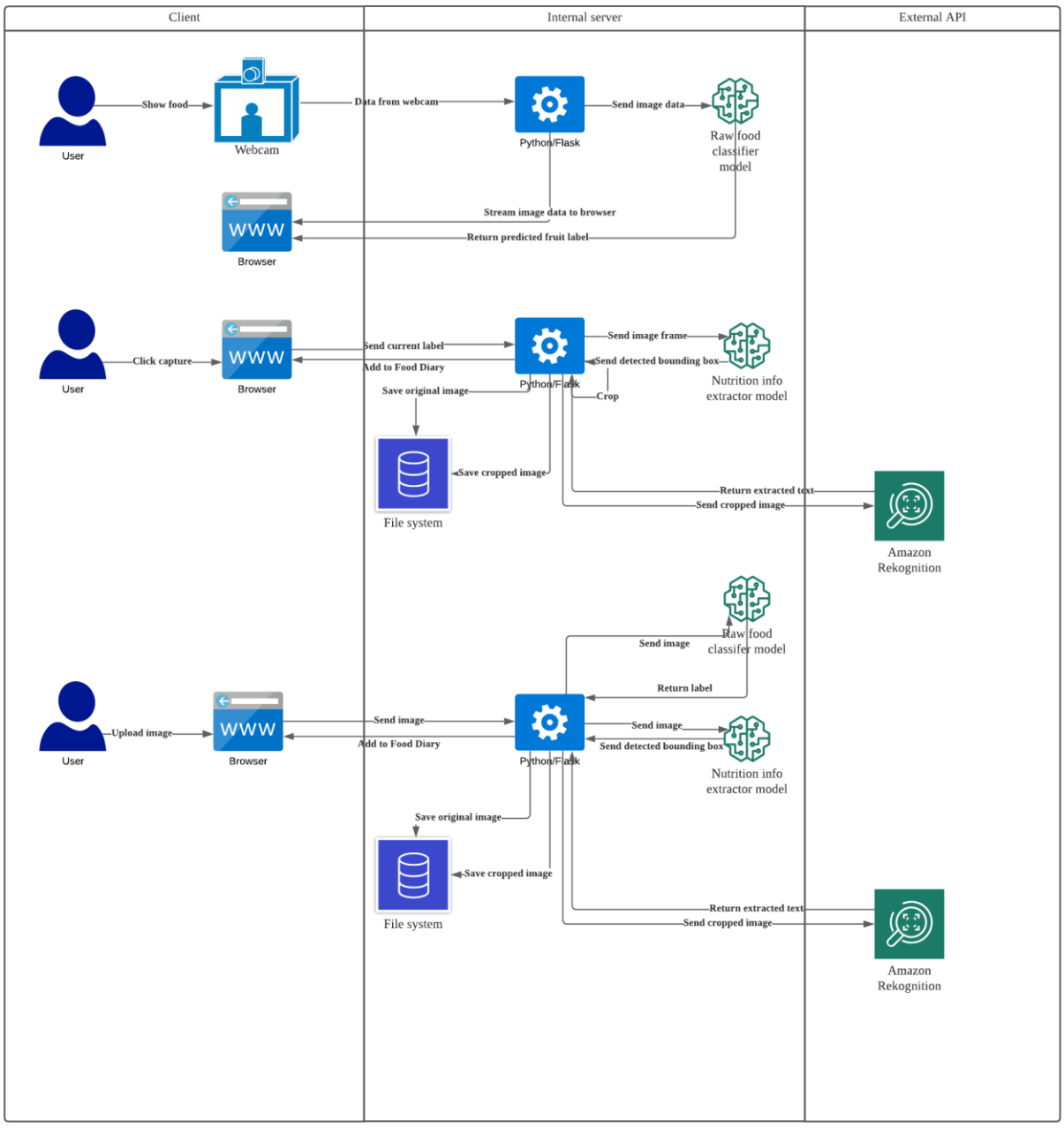


Figure 10: System Architecture

3. MODELS AND TECHNIQUES

We will now go into the details of two deep learning models that resides in the "Internal Systems" swim lane.

3.1 Raw Food Classification Model

A hybrid self-tuning classifier is set up using components: feature extractor and classifier:

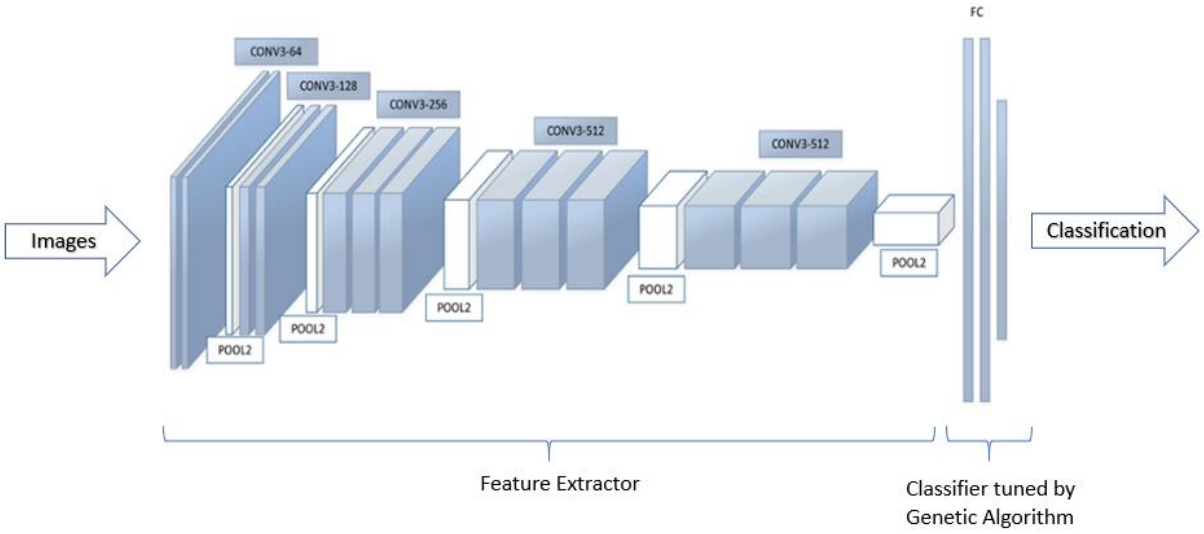


Figure 11: Raw Food Classification Model

Combining feature extractor and classifier:

The two components of feature extractor and classifier is combined using Keras functional API. After input images of raw food are passed into the feature extractor, the extracted features are then passed into the classifier. The classifier uses the features to classify input images into types of raw food.

The benefit of this setup is that the classifier is self-tuning by using GA. The design is elaborated in sections 3.1.1 and 3.1.2.

The benefit is that if feature extractor is changed to a different one (ResNet eg.), the hybrid model will tune itself to optimise the classifier's accuracy according to the new feature extractor.

3.1.1 Feature extractor

Configuration:

- Following VGG 16 architecture but excluding the classifier (of VGG16 architecture), the configuration listed in Table 1 is set up layer by layer using Keras Functional API.
- Pre-trained weights[1] based on ImageNet was used in feature extractor.
- Input images into the feature extractor are resized to 256 x 256 pixels.

Feature Extractor Configuration				
		Kernel size	**Stride**	**Number of nodes**
Convolutional Block 1 *Activation function = ReLU*	Convolutional	3 X 3	1, 1	64
	Convolutional	3 X 3	1, 1	64
	Max Poolling	2 X 2	2, 2	-
Convolutional Block 2 *Activation function = ReLU*	Convolutional	3 X 3	1, 1	128
	Convolutional	3 X 3	1, 1	128
	Max Poolling	2 X 2	2, 2	-
Convolutional Block 3 *Activation function = ReLU*	Convolutional	3 X 3	1, 1	256
	Convolutional	3 X 3	1, 1	256
	Convolutional	3 X 3	1, 1	256
	Max Poolling	2 X 2	2, 2	-
Convolutional Block 4 *Activation function = ReLU*	Convolutional	3 X 3	1, 1	512
	Convolutional	3 X 3	1, 1	512
	Convolutional	3 X 3	1, 1	512
	Max Poolling	2 X 2	2, 2	-
Convolutional Block 5 *Activation function = ReLU*	Convolutional	3 X 3	1, 1	512
	Convolutional	3 X 3	1, 1	512
	Convolutional	3 X 3	1, 1	512
	Max Poolling	2 X 2	2, 2	-

Table 1: Feature Extractor Configuration

[1] https://github.com/fchollet/deep-learning-models

3.1.2 Self-tuning classifier

Configuration:

- Fully connected and dropout layers are used in the classifier.
- Given a range of possible neural network hyper-parameters shown in Table 3, genetic algorithm is used to tune the classifier for highest possible accuracy (within a finite number of training epochs for classifier).
- No pre-trained weights are used for the classifier.
- Figure 12 shows that classifier configuration is set up using neural network parameters selected by genetic algorithm (GA).

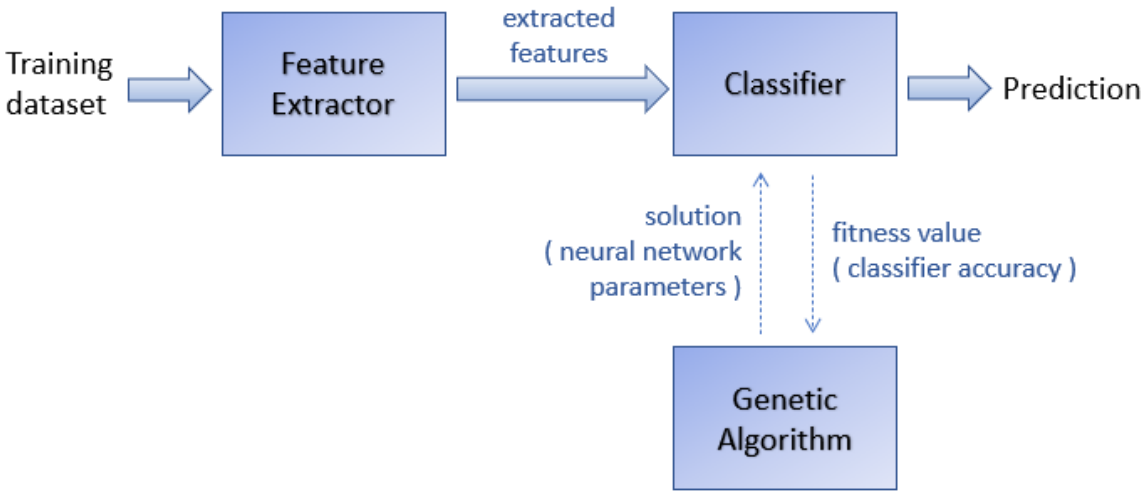


Figure 12: Classifier tuned by GA during training

- Activation function used in each dense layer of classifier is ReLU.
- The classifier is performing prediction for 5 classes of raw food (ie. fruits and vegetables), hence the last layer has 5 output nodes. For the purpose of multi-class classification, the activation type at the last layer is SoftMax.
- Loss function used is categorical cross-entropy loss.
- Optimizer used is Adam.
- GA model is not used after the hybrid model inclusive of classifier is trained. Figure 13 shows classifier being used in production setting.

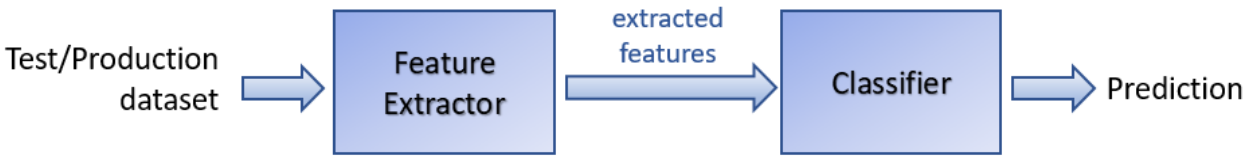


Figure 13: Tuned classifier performing predictions

- Table 2 shows the final classifier configuration that is tuned by GA in this project.

Classifier Configuration	
	Number of nodes
Flatten	-
Fully connected dense	512
Dropout (0.25)	-
Fully connected dense	512
Dropout (0.25)	-
Fully connected dense	512
Dropout (0.25)	-
Fully connected dense (output layer)	5

Table 2: Classifier Configuration

The following describes the design of genetic algorithm used to tune the classifier.

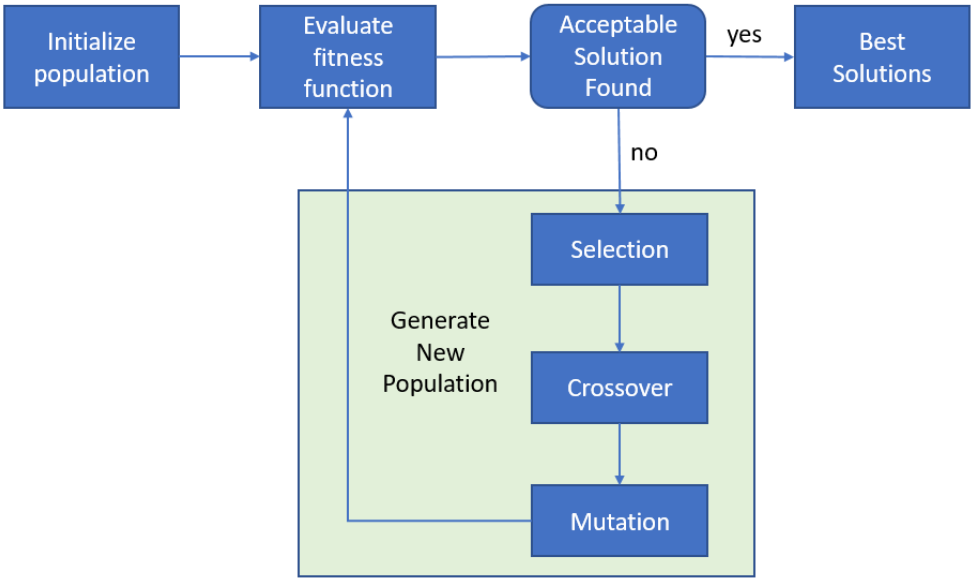


Figure 14: Genetic Algorithm

1. Chromosome design:
   - The genes of the chromosome are made up of neural network hyper-parameters. The range of possible values for hyper-parameters is shown in Table 3:

Hyper-parameters range	
Neural network parameters	**Range of values**
Number of neurons	64, 128, 256, 512, 1024
Number of layers	1,2,3,4
Activation type	ReLU, tanh, sigmoid
Optimizer type	RMSprop, Adam, SGD

Table 3: Value Range for Chromosome Genes

   - The chromosome is represented using Python's dictionary data structure. One example of a chromosome represented as a dictionary is shown in Table 4.

Dictionary key	Dictionary value
Number of neurons	128
Number of layers	2
Activation type	ReLU
Optimizer type	Adam

Table 4: Chromosome Example

2. Fitness value:
   - A classifier (neural network) model was set up using the chromosome values, which are neural network hyper-parameters. The fitness value used here is classifier's accuracy over training data.

3. Constraint:
   - The hyper-parameters need to be within the range of values specified. This constraint is guaranteed to be satisfied due to chromosome design.

4. Selection:
   - In this project, GA is retaining the top 50% (ie. fittest) of the population of chromosomes. This parameter value which determines the retention proportion is a variable that can be changed.
   - Crossover and mutation functions will then be performed on remaining less fit population. The selection of two chromosomes from this pool is random.

5. Crossover:
   - Two parents of chromosomes are used to form an offspring chromosome. For each neural network parameter (number of neurons, number of layers eg.) required by the offspring chromosome, the offspring inherits randomly from either one of the two parents.

6. Mutation:
   - One of the neural network parameters of the offspring chromosome is randomly selected for mutation. The mutation is performed by randomly picking one value from the range of possible neural network hyper-parameter values.

3.1.3 Raw Food Image Dataset

The dataset was progressively curated. It is made up of raw food images belonging to 5 categories, namely apple, long bean, onion, pineapple and potato. 436 Images was selectively downloaded from the internet one by one. This approach was taken so that diversity in images can be created, such that there is a range of images capturing single food, multiple food and food of different colour types.

	Number of training images	Number of validation images	Number of test images
Apple	40	27	5
Long Bean	76	39	5
Onion	67	36	5
Pineapple	32	17	5
Potato	46	31	5

Table 5: Raw Food Image Dataset

The following are the images used for testing.

Figure 15: Images used for Testing

3.2 Packaged Food Nutrition Information Model

Packaged food safety has gained increasing attention worldwide. In our solution, we tackled the problem on how to access the quality of a packaged food without disturbing the packaging, by just looking at the nutrition label table that is printed on the product.

After our food diary system captures the nutrition label, our solution can determine the food grade of the packaged food. This works in a three step process (I) to detect where the nutrition label is present (II) extract the information and quantity of each nutrition listed in the label (III) calculate the food score which gives the food grade as per standard universal methods in MyNetDiary. Our food diary system also allows user to keep track of their data.

3.2.1 Nutrition Information Detection and Extraction

Data Collection:

The dataset is a self-sourced (ie. look for each image one by one) set of images. They show packaged food with their nutrition labels. Data was collected with a diverse variety of backgrounds and with different types of nutrition labels. Also, real-time product images were collected by going to the super market.

Diversity in dataset helps in making our model learning to be more generalised during training, where it can identify the nutrition label on diverse kinds of food product. It also helps improve model accuracy in the detection. Approximately 500 images are collected.

Collection of Images with diversity

Figure 16: Nutrition Labels

Annotations for Training:

Images were annotated/labelled manually for the training. We used a tool called 'LabelImg' to annotate the labels for training. Images for testing are a collection of 10% of all available images. The Nutrition label is labelled in each image to train the model. Each image contains one or more products with labels hence multiple annotations are done on the images.

Figure 17: Using LabelImg for Annotation

3.2.2 Object Detection

Using TensorFlow Object Detection:

For the nutrition label detection, we used TensorFlow Object detection API to train our model on dataset after the dataset was annotated. The TensorFlow Object Detection API is an open source framework built on top of TensorFlow that makes it easy to construct, train and deploy object detection models.

After data split is done, we moved the 90% of data (for training) and 10% of data (for testing) to respective folders for training and testing folders. For training images, ".xml" files containing annotations of the images are also added to training folder.

- We generated our annotations and split our dataset into the desired training and testing subsets. After this, we converted our annotations into the so called (. record) format.
- Python script was used to convert the images both from train and test folders into tensorflow records for training the model. To avoid loss of any files, the script will not delete any image after converting them to tensorflow records.

Images → converted to CSV file → Converted to tensorflow records for training

Creating label Map:

TensorFlow requires a label map, which maps each of the used labels to an integer value. This label map is used both by the training and detection processes.

Training and configuration:

We used one of the pre-trained models provided by TensorFlow. The model we used is SSD Mobilenet model, because it provides a relatively good trade-off between performance and speed. We downloaded the latest pre-trained network for the model for the training.

Configuration of training pipeline:

The training pipeline contains all the possible configurations that we have tried for the training and hyper parameter tuning of the model. It is in this file where the hyper parameter tuning like batch size, learning rate, L1/L2 Regularizations, dropout, image argumentation is handled. Here, one of the most important configurations that we make is to define the number of classes. In this project it is set as one (Nutrition label).

```
# SSD with Mobilenet v2 configuration for MSCOCO Dataset.
# Users should configure the fine_tune_checkpoint field in the train config as
# well as the label_map_path and input_path fields in the train_input_reader and
# eval_input_reader. Search for "PATH_TO_BE_CONFIGURED" to find the fields that
# should be configured.

model {
  ssd {
    num_classes: 1
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
        unmatched_threshold: 0.5
        ignore_thresholds: false
        negatives_lower_than_unmatched: true
        force_match_for_each_row: true
      }
    }
```

Figure 18: Configuration of Training Pipeline

Training of the model:

The training is commenced by a python script. A very nice feature of TensorFlow, is that it allows you to continuously monitor and visualize a number of different training/evaluation metrics, while your model is being trained. Tensorboard helps with this monitoring. The training was carried out for 10,000 steps. The metrics from the Tensorboard are documented in Figure 17.

```
I0904 06:11:49.286639 139783200995200 basic_session_run_hooks.py:262] loss = 12.684037, step = 0
INFO:tensorflow:global_step/sec: 2.25213
I0904 06:12:33.688338 139783200995200 basic_session_run_hooks.py:692] global_step/sec: 2.25213
INFO:tensorflow:loss = 7.5453215, step = 100 (44.403 sec)
I0904 06:12:33.689689 139783200995200 basic_session_run_hooks.py:260] loss = 7.5453215, step = 100 (44.403 sec)
INFO:tensorflow:global_step/sec: 2.24184
I0904 06:13:18.294498 139783200995200 basic_session_run_hooks.py:692] global_step/sec: 2.24184
INFO:tensorflow:loss = 7.1984987, step = 200 (44.607 sec)
I0904 06:13:18.296532 139783200995200 basic_session_run_hooks.py:260] loss = 7.1984987, step = 200 (44.607 sec)
INFO:tensorflow:global_step/sec: 2.54277
I0904 06:13:57.621723 139783200995200 basic_session_run_hooks.py:692] global_step/sec: 2.54277
INFO:tensorflow:loss = 7.894999, step = 300 (39.479 sec)
I0904 06:13:57.775713 139783200995200 basic_session_run_hooks.py:260] loss = 7.894999, step = 300 (39.479 sec)
INFO:tensorflow:global_step/sec: 2.30486
I0904 06:14:41.008176 139783200995200 basic_session_run_hooks.py:692] global_step/sec: 2.30486
INFO:tensorflow:loss = 6.7467656, step = 400 (43.234 sec)
I0904 06:14:41.009751 139783200995200 basic_session_run_hooks.py:260] loss = 6.7467656, step = 400 (43.234 sec)
INFO:tensorflow:global_step/sec: 2.35127
I0904 06:15:23.538424 139783200995200 basic_session_run_hooks.py:692] global_step/sec: 2.35127
```

Figure 19: Model Training

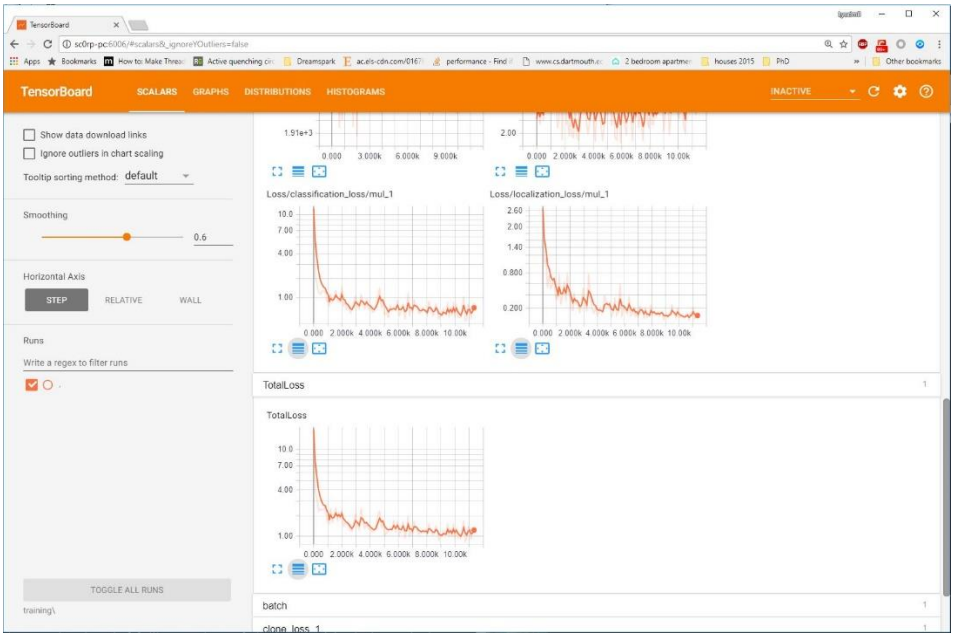


Figure 20: Tensorboard showing Model Training

After the training job was completed, we wrote another python script, to extract the trained model as an inference graph which is then used to perform object detection. Object detection can then be done on images. Live detection can also be done on pre-recorded and live video.

Figure 21: Detecting Nutrition Labels

Hyper parameter tuning:

The initial training was done with limited images that was gathered from the internet. The initial model could not have a better accuracy for object detection then after several training and learnings. Hyper parameter tuning had been done in the configuration file to reach a better accuracy and lower loss for our final model. The following was done:

- Trained the model using more images (ie. improved dataset)
- Applied low learning rate and tried the learning scheduler
- Different trial trainings performed with various batch sizes
- Data augmentation like flipping images, cropping, RGB to grayscale, rotation of 90 degrees. This adds more data and are handled from the configuration file.

3.2.3 Text Extraction

There are many OCR features offered by Google and AWS. We chose 'Amazon Rekognition' feature to extract the detected text from the Nutrition label, because the unstructured format of the extraction from AWS and Google OCR offerings is not 100% reliable. We experimented and we found that we can make the fullest use out of 'Amazon Rekognition'.

The detection bounding box of the nutrition label by the model is then cropped to perform this text extraction.

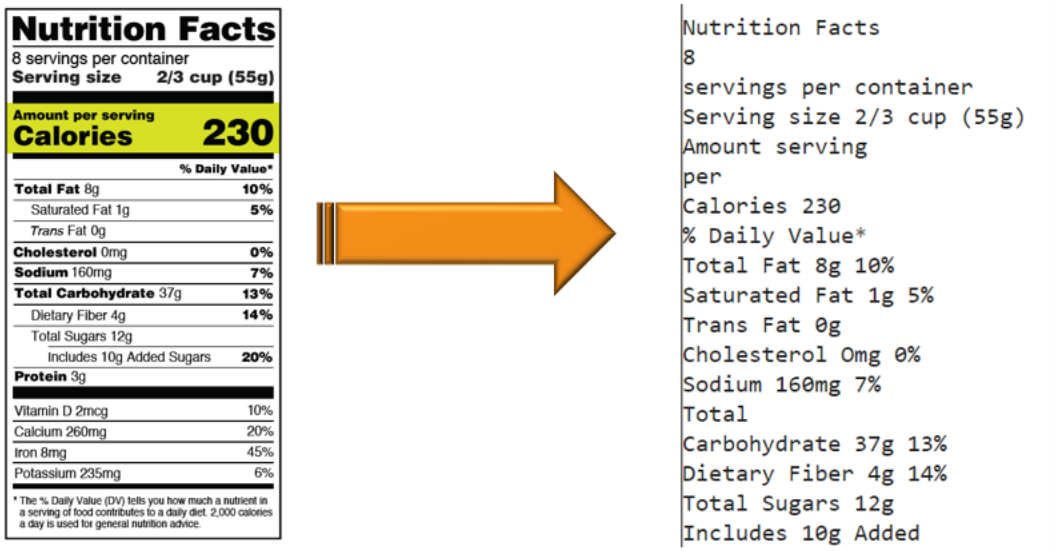


Figure 22: Extracting Nutrition Information

3.2.4 Food Grading

Food Grade is a letter (A, B, C or D) corresponding to Food Score number. MyNetDiary has such a scoring system - it is called "Food Grade."

Food Score is calculated using an equation derived from food ratings of nutrition experts using information found on the Nutrition Facts panel. That is, Food Score mimics how a nutrition expert would score the healthfulness of a food based upon its nutrition label.

The equation uses the content of twelve required nutrients listed on the 'Nutrition Facts' panel: *total fat, saturated fat, cholesterol, sodium, total carb, fibre, sugar, protein, vitamins A and C, calcium, and iron*.

Some nutrients have a stronger effect on Food Score as compared to others.

- For instance, fibre can affect Food Score more strongly and positively than any other nutrient. That means foods higher in fibre will score higher than foods that are lower in fibre.

- Protein, vitamins A and C, calcium, and iron can also have a positive effect on the score.
- The reverse is true for saturated fat, where food higher in saturated fat will have a lower Food Score.
- Total fat, cholesterol, sodium, total carbohydrates, and sugars will also have a negative effect on the score.

Figure 23: Grading of Packaged Food

Most Food Scores will vary from -5 (less healthy) to +5 (healthier) for serving sizes listed on the Nutrition Facts label. Some people have a good feel for numbers whereas others prefer grades. The relationship between the Food Score and Food Grade is a simple conversion developed by MyNetDiary. Higher positive numbers are considered healthier by this food scoring system so their Food Grade is higher. Negative numbers are considered less healthy so their Food Grade is lower.

3.2.5 Overall process flow

The following diagram summaries and shows the overall process flow for packaged food nutrition information capture and extraction.

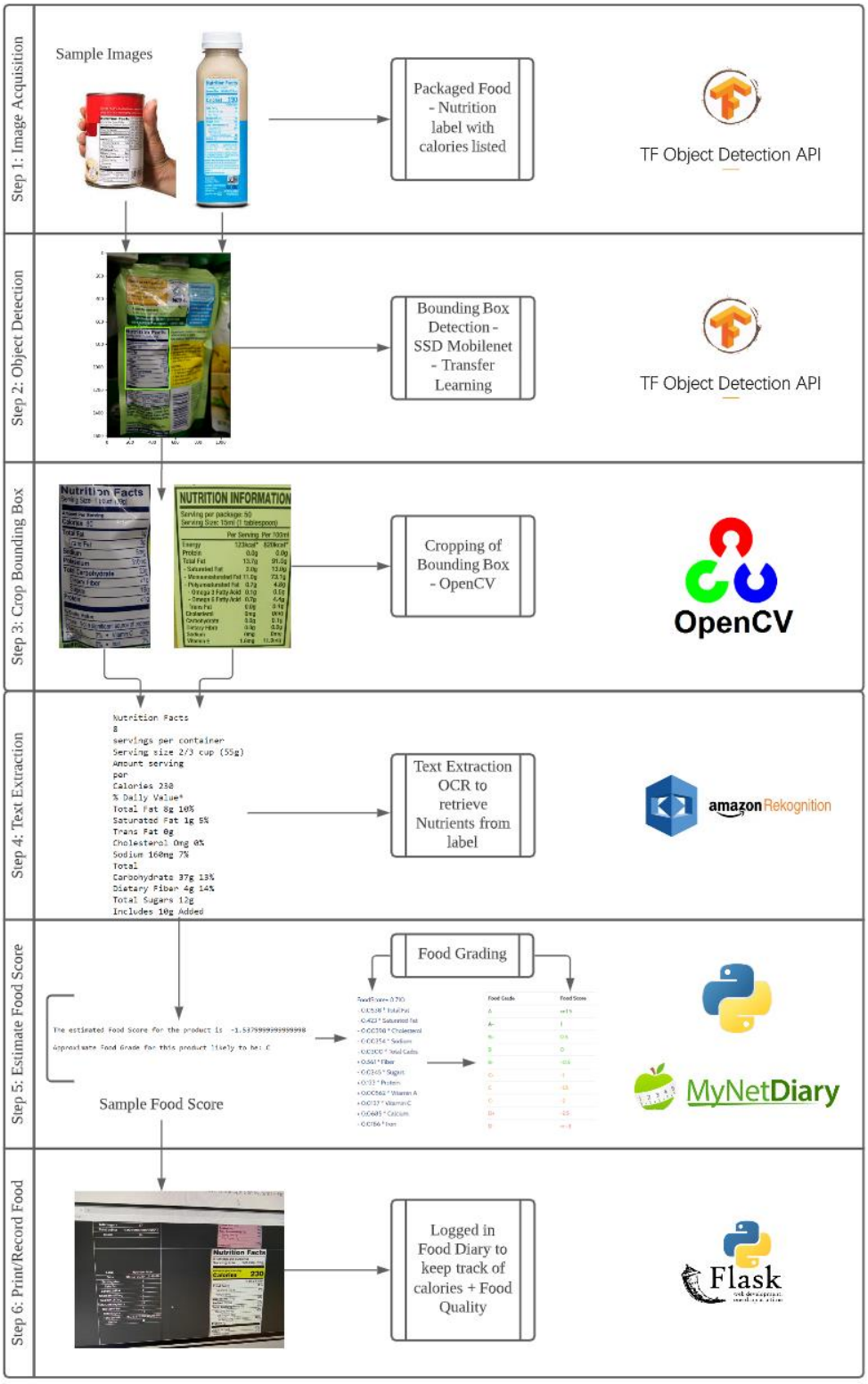


Figure 24: Nutrition Information Capture and Extraction Process

4. SYSTEM PERFORMANCES

4.1 Raw Food Classifier Model

The following shows the test results over unseen new test data for the hybrid raw food classifier model. The final accuracy over test dataset for the model is 96%.

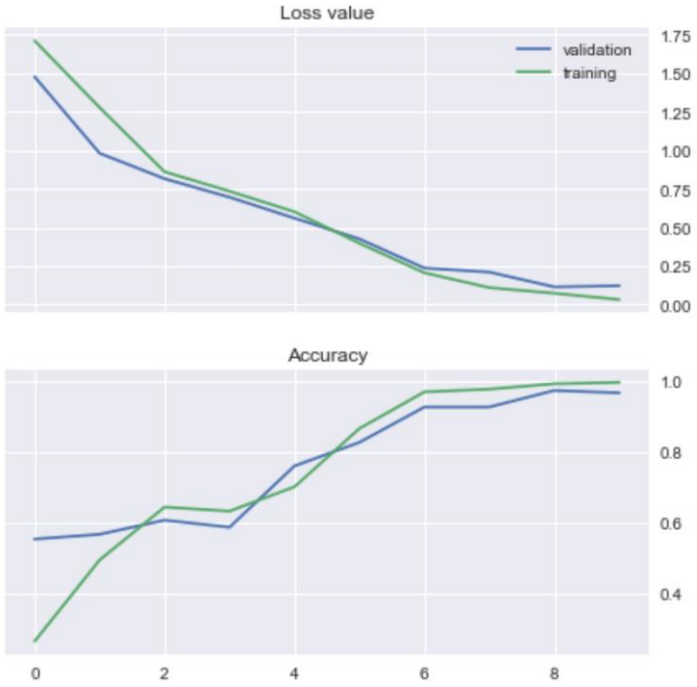


Figure 25: Raw Food Classifier Model Training

Confusion matrix for prediction for 5 classes

```
[[4 0 1 0 0]
 [0 5 0 0 0]
 [0 0 5 0 0]
 [0 0 0 5 0]
 [0 0 0 0 5]]
```

	precision	recall	f1-score	support
0	1.00	0.80	0.89	5
1	1.00	1.00	1.00	5
2	0.83	1.00	0.91	5
3	1.00	1.00	1.00	5
4	1.00	1.00	1.00	5
accuracy			0.96	25
macro avg	0.97	0.96	0.96	25
weighted avg	0.97	0.96	0.96	25

Figure 26: Raw Food Classifier Model Testing

4.2 Packaged Food Nutrition Model

Comparing the initial and the final executions of the training, the metrics for precision and recall curves shows that accuracy had increased after hyper parameter tuning. (Figure 25). And the classification/localization and overall loss were reduced after hyper parameter tuning (Figure 26).

The model training was carried out for 10,000 steps (epoch).

Metrics	*Training Initial*	*Training Final (After Model Tuning)*
Precision	~48.0	>65
Recall	~44.0	>44.5
Classification Loss	3.6	2.7
Localization Loss	0.76	0.3
Overall Loss	2.0	1.3

Figure 27: Nutrition Information Extractor Model Training 1

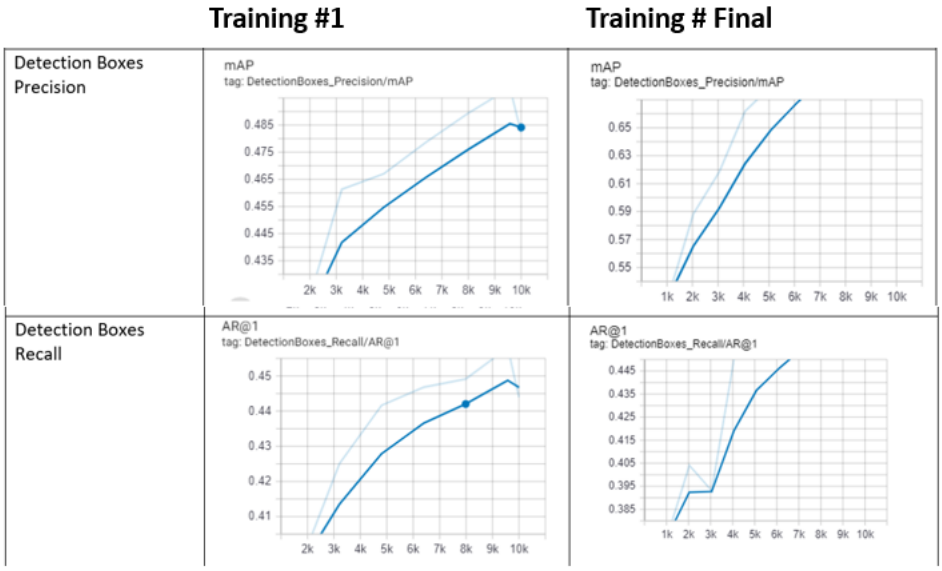


Figure 28: Nutrition Information Extractor Model Training 2

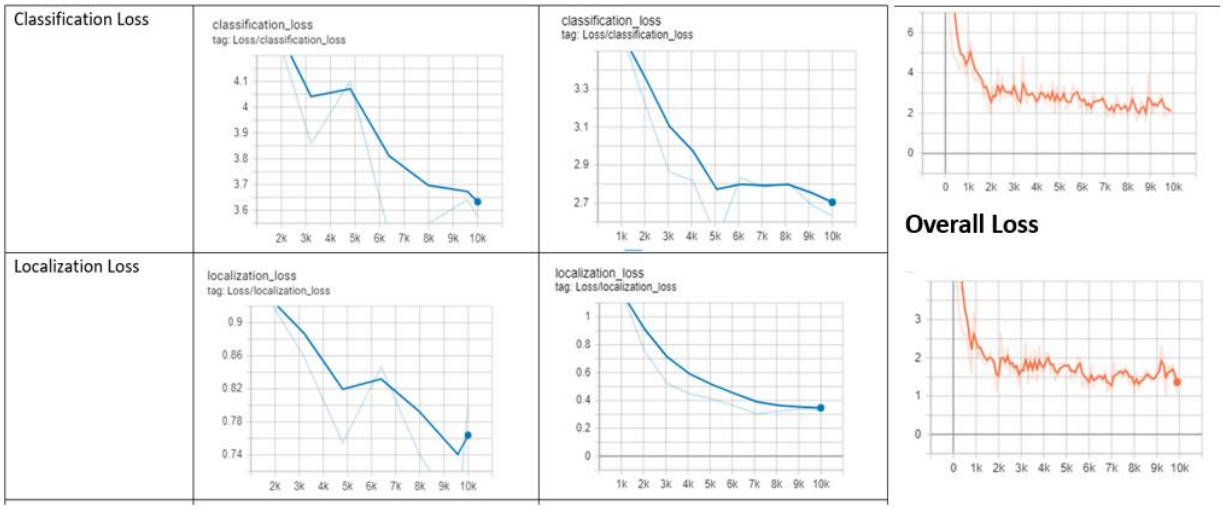


Figure 29: Nutrition Information Extractor Model Training 3

There are other metrics that have to be taken into consideration when it comes to object detection like Mean Average Precision accuracy with intersection over union as 50% and 75% (mAP with IOU = 0.50 and mAP with IOU = 0.75). Appendix 2 provides complete metrics given for the trainings.

5. FINDINGS

The learning models' accuracies are quite high when tested with various images in a laboratory environment.

However, when input images stream in from sensors (i.e., webcam in our project) in everyday life, the learning models may sometimes predict wrongly. That is because there are background 'noise' when the image is streamed by webcam. For instance, the image that is streamed in may be someone holding up a fruit against a city background, and the background "distracts" the raw food classifier model from correctly predicting the class of fruit. Another experience we had in this project was that poor webcam resolution and dim lighting can also affect the model's performance.

Figure 30: Fruit with 'Noisy' Background

Object detection model may work better as compared to a image classification model since it will find the object first instead of taking into consideration the entire image. However, the cons of this approach would be that the model uses more resources and/or time to find the object. Hence, in terms of user experience, if the server does not have sufficient resources, the user will have to wait longer for the prediction.

For Amazon Rekognition's text extraction, it strictly only works well with very clear images with big font size. After we developed this application, we noticed that real nutrition labels can be quite small. The labels come with small font or typically do not have flat surfaces. An example is shown below. Labels on such products can be difficult for text extraction.

Figure 31: Unclear Nutrition Labels

6. DISCUSSIONS AND FUTURE DEVELOPMENT

When we first decided on developing this food diary, the plan was to allow users to indicate what type of food it is (i.e., packaged food or fruits) before the models predict the exact type of food. Hence, we developed multiple models for each type of food. When we later removed the step of user indicating the type of food, the system have to take on more load consulting different machine learning models in order to reason finally what the exact type of food is. To improve the performance of the system, we can consider combining the models so that it can predict multiple types of food.

For the nutrition label learning model in future, we aim to have better text extraction, especially from webcam, by enhancing the sharpness of the image. We would also like to add other features to better analyse information from the nutrition table, for instance features that can benefit people with obesity or diabetes problem.

The food diary is also more suitable to be a mobile application than a web application. Users can then easily access their food intake history. A possible system architecture is that we can retain our current architecture in this project, and only replace the web browser with mobile apps so that the mobile application can send or retrieve data to/from the remote flask server.

7. REFERENCES

1. The Architecture and Implementation of VGG-16
   https://medium.com/towards-artificial-intelligence/the-architecture-and-implementation-of-vgg-16-b050e5a5920b

2. Object detection using TensorFlow
   https://www.tensorflow.org/hub/tutorials/object_detection

3. TensorFlow records from dataset
   https://www.tensorflow.org/api_docs/python/tf/data/TFRecordDataset

4. Understanding the latest food label as per FDA
   https://www.fda.gov/food/new-nutrition-facts-label/how-understand-and-use-nutrition-facts-label

5. Understanding nutrition labelling in Singapore
   https://www.hpb.gov.sg/docs/default-source/default-document-library/handbook-on-nutrition-labelling.pdf?sfvrsn=0

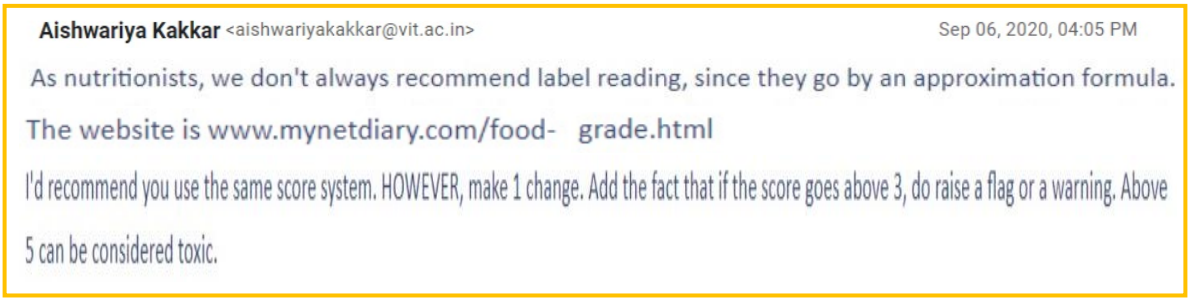
6. Related Machine Learning to label detection
   https://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=8204&context=etd

7. Food Scoring from MyNetDiary
   https://www.mynetdiary.com/food-grade.html

8. APPENDIX

1. Knowledge acquisition from Health professionals. It is essential in this project for user's benefit to be able to correctly derive the food grade of the product. To learn how to do this, we approached professional nutritionists to get a better insight about how food nutrition are classified. We obtained their advice on how to perform the calculation to identify the food grading. The following is one interview excerpt:

   Interview of the Nutritionist – Aishwariya Kakkar



Aishwariya Kakkar <aishwariyakakkar@vit.ac.in> Sep 06, 2020, 04:05 PM

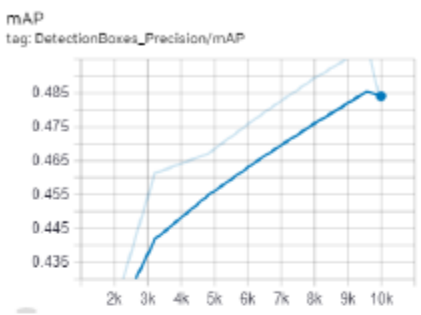
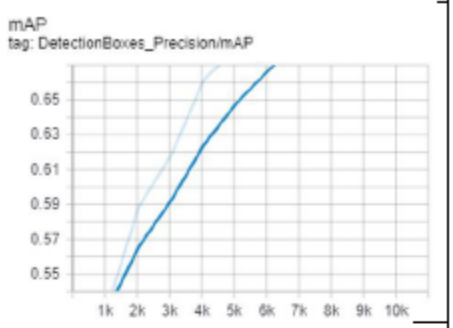
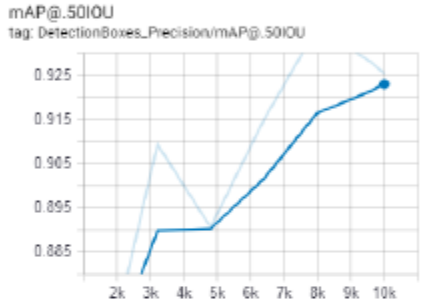
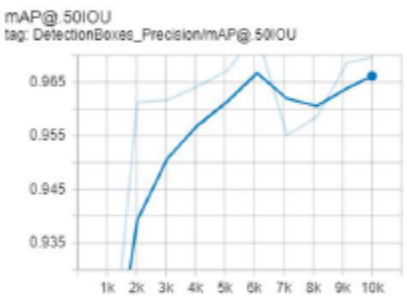
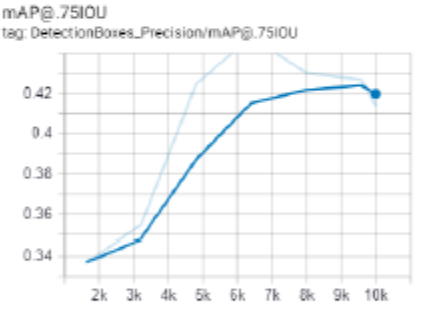
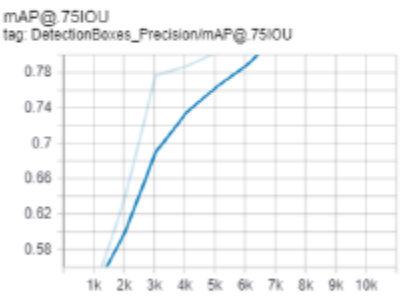
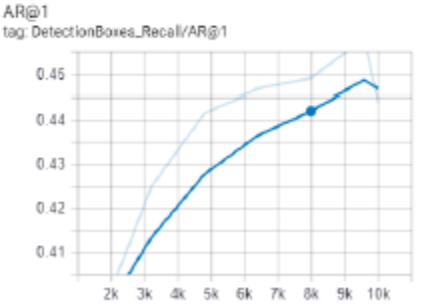
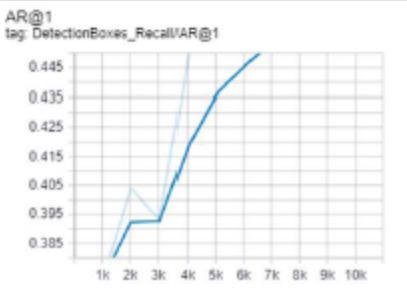
As nutritionists, we don't always recommend label reading, since they go by an approximation formula.

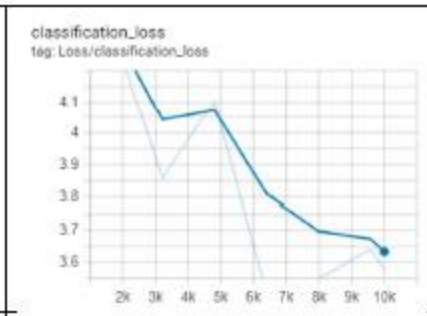
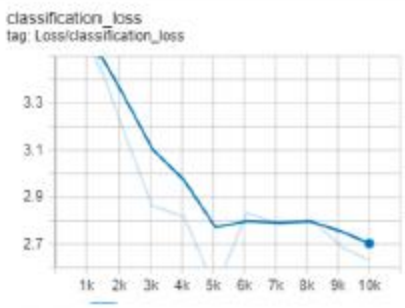
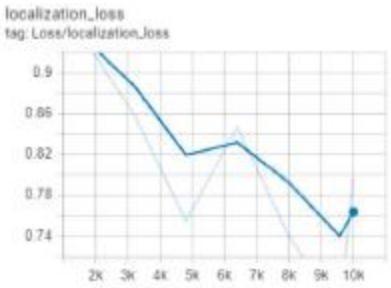
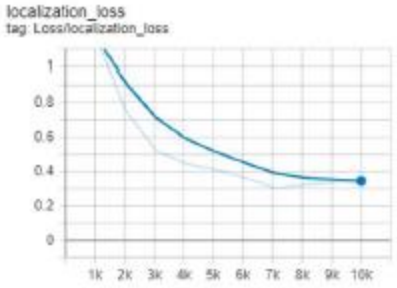
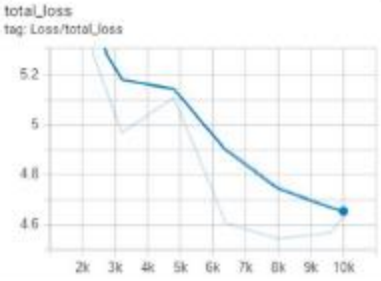
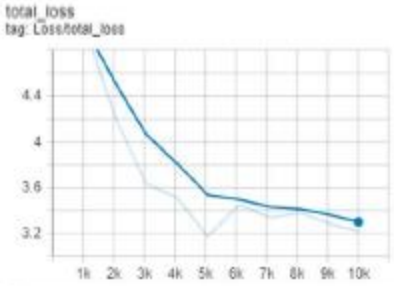
The website is www.mynetdiary.com/food- grade.html

I'd recommend you use the same score system. HOWEVER, make 1 change. Add the fact that if the score goes above 3, do raise a flag or a warning. Above 5 can be considered toxic.

2. Metrics for packaged food nutrition model training.

Nutrition label – Object Detection using TensorFlow API

Metrics	Training #1	Training #2
Detection Boxes Precision		
mAP with IOU = .50		
mAP with IOU = .75		
Detection Boxes Recall		

Classification Loss		
Localization Loss		
Total Loss		
Loss