Singapore Food Image Classification

Agenda

- Introduction to Team and Project Topic
- Problem Statement
- Persona, Pain Point and HMW Statement
- Big Ideas
- AI Design Essentials
- Project Planning/Timeline
- Technical Planning & Consideration
- Implementation Overview
- Technical Architecture
- Live Demo
- Project Learning

Introduction

Project Objective:

As lifestyle improve, people are getting more health conscious especially in their food consumption. With the wide range of local food that is always available, consumers are spoiled for choice, however, they do not have any ways to get 1st hand information on food nutritional value (ex. calories, fats, etc..)

This result in general ignorance of what they are consuming and their daily recommended calorie intake

We have developed an AI solution to give consumer first hand food nutritional information so that they can make an informed decision on what they are consuming.

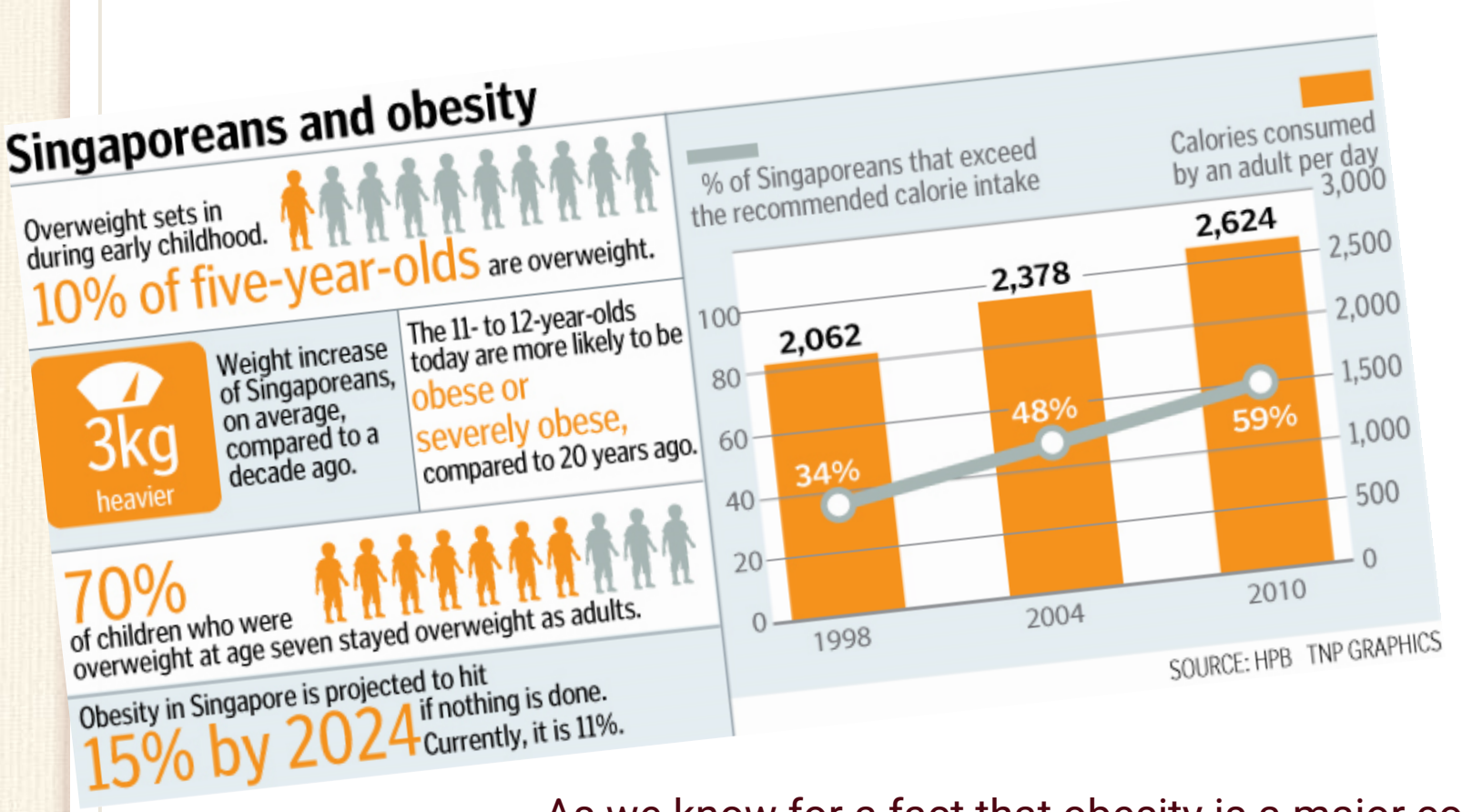
With our SnapFood app, consumers will be able to obtain this information at the tip of their fingers, anytime and anywhere

Project Topic:

How our AI solution is able to provide nutritional information of popular local foods

HOW? simply taking photos of local foods

ENTERPRISE DESIGN THINKING



Problem Statement

As we know for a fact that obesity is a major contributor to diseases such as diabetes, congestive heart failures, restrictive mobility. In recent local studies of Singapore and obesity, it shows

1. Singaporean's average weight is increasing as compared to a decade ago
2. Obesity in Singapore is projected to hit 15% by 2024 if nothing is done (Currently, it is 11%)
3. An adult Singaporean average calories per day is increasing

Problem Statement (continued)

Currently the available weight managements mobile apps in the market needs to input the food manually in order for the inbuilt calories calculator to work. These leads to several problems such as

1. Not entering the right food
2. Pick the lowest calorie option every time to “cheat”
3. Wait too long to log and ends up forgotten about it

Persona

Name Samantha

Profile

Age	34 Millenial
Location	Southern Singapore
Education	University degree
Job	Social Influencer
Family	Married no kids
Work experience	12 years professional experience
Technical literacy	Tech savvy

Motivations

- Lead healthy lifestyle
- Active on social media, wants to increase followers
- Health conscious

Goals

- Keep healthy and stay slim and attractive
- Stay connected to her followers
- To monitor her daily calories intake

Needs

- Wants to understand her food options
- Want to make sure she stays within her daily caloric intake
- Need to have an app to give her instant information on the food she is eating
- Need to be on top on all new fashion & food trends

Pain Points

HMW statement

How might we *provide accurate nutritional value of local food* for *millennials* so that *they know their calories intake and can prevent diet-related diseases*

Big Idea

User can earn rewards points or vouchers if they choose healthier choice's food

User can have a personal buddy to buy the food based on individual's profile and bring over to user

User can store their activities such as daily steps counts onto the database and able to calculate the net daily calories

User can get personalized info (based on previous search, food intake, or visited stalls, etc.)

User can use the IoT API (eg. smart watch) to link up automatically and store the activities such as swimming, jogging etc

User can receive recommendation of good yet healthy food within their proximity

User just needs to take photo of the food to know what nutrients & other substances are in the food

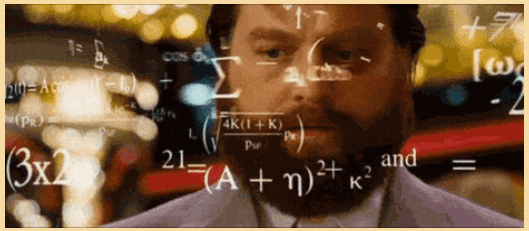
User can get the list of food stalls of their favourite food

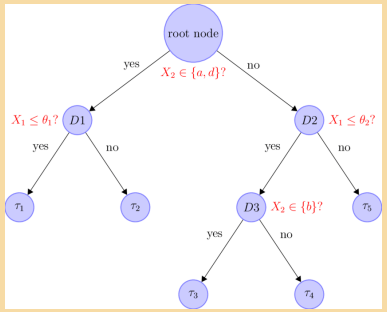
User can obtain nutritional information of food at their fingertip

What should we use for our solution?

MACHINE LEARNING

Input → Feature Extraction → Classification → Output

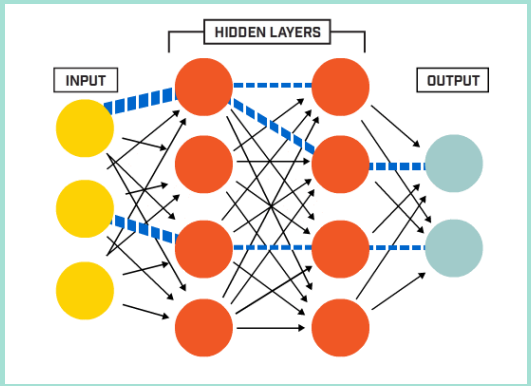




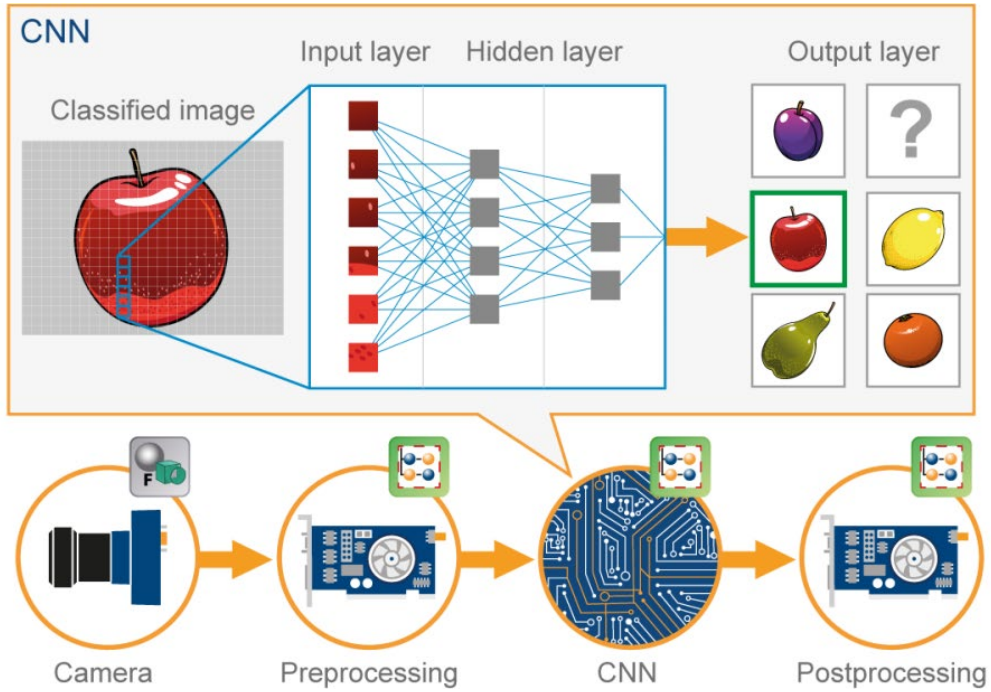
- Traditional machine learning use hand-crafted features, which is tedious and costly to develop
- More suited for structured data (e.g. data in tabular format such as excel sheets or SQL database)

DEEP LEARNING

Input → Neural Networks → Output



- Deep Learning learns hierarchical representation from the data itself and scales with more data
- More suited for unstructured data such as images, videos and sound recording



Proposed Solution

We decide to use Convolution Neural Network (CNN) as our solution to the problem

Automatic recognition of dishes would not only help users effortlessly organize their extensive photo collections but would also help online photo repositories make their content more accessible

Additionally, mobile food photography can be used to help patients estimate and track their daily calorie intake on Singapore local foods efficiently.

AI DESIGN ESSENTIALS

Intent

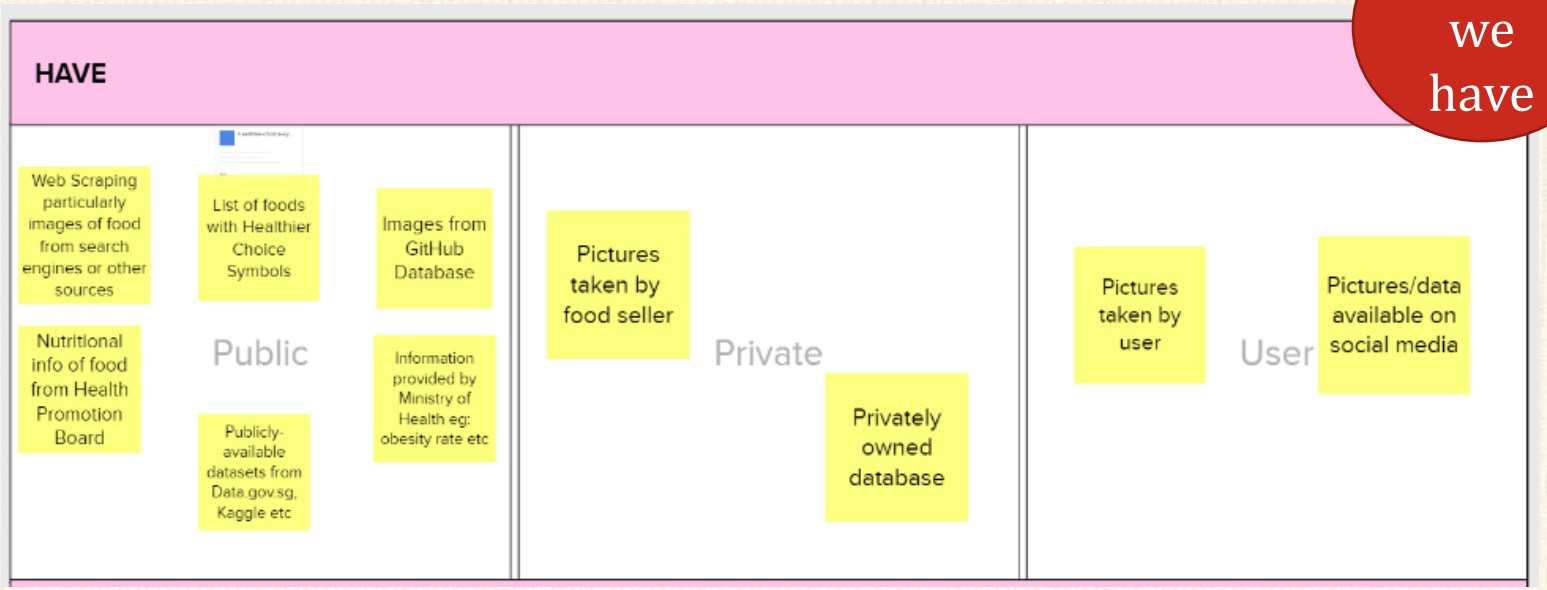
ENRICH USER INTERACTIONS

- Obtain nutritional information of food at their fingertips
- Provide a platform for user to give feedback or to share experiences for a particular food
- Show what other users are eating
- Detect sudden change in health vital signs
- Alert healthcare personnel In case of emergency

RECOMMEND WITH CONFIDENCE

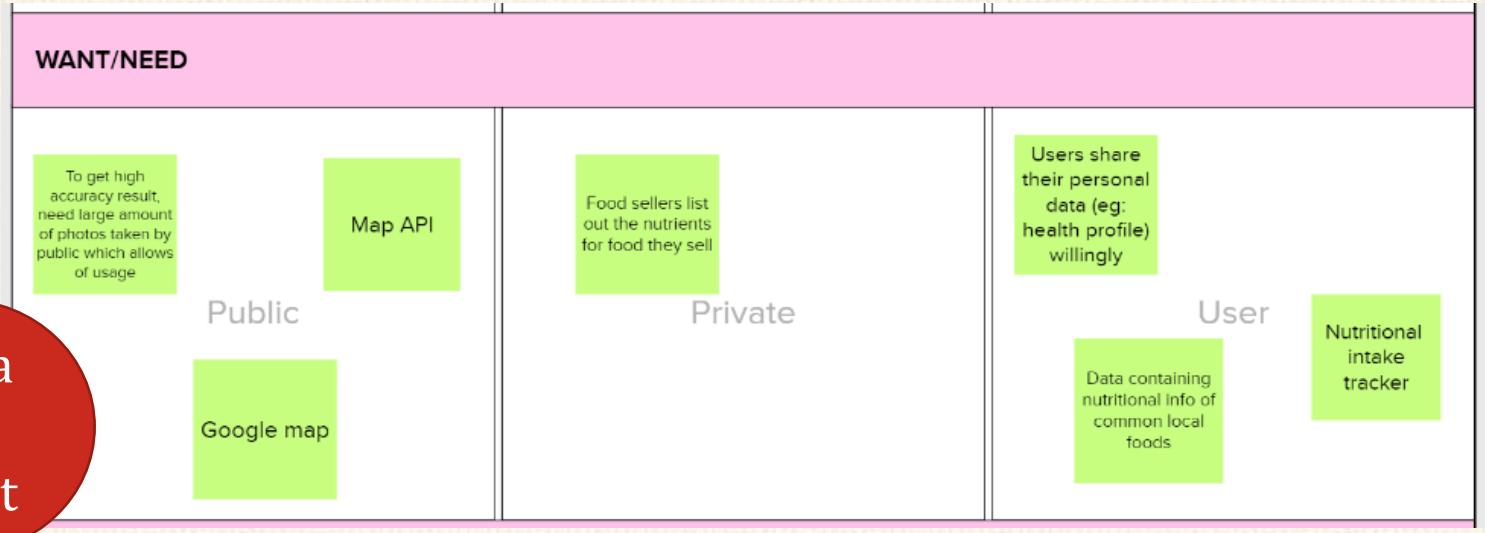
- Personalised nutrition recommendation based on user's health profile & history
- Provide food rating based on nutritional value (1 to 10)
- Nutrition intake tracker

Data



Data we have

- Choosing data sources is a fundamental first step in designing our AI.
- Focus on the HAVE and WANT data.
- Having the correct data goes a long way in a project.



Data we want

Ethical implication of using this data

Accountability

- To build trust with users, every team member is accountable for making sure sensitive data collected is used as what it is intended for

- Adopting & implementing data protection policies
- Putting written Personal Information Consent Form in place
- Maintaining documentation of our processing activities
- Recording and, where necessary, reporting sensitive data breaches

Data Privacy

- The proper handling of sensitive data (eg: personal data, financial data) to meet regulatory requirements as well as protecting the confidentiality of the data.
- Have to make it clear to users so that they will know the types of data we are collecting, why we are collecting, how data will be used and how users can contact us with questions or concerns

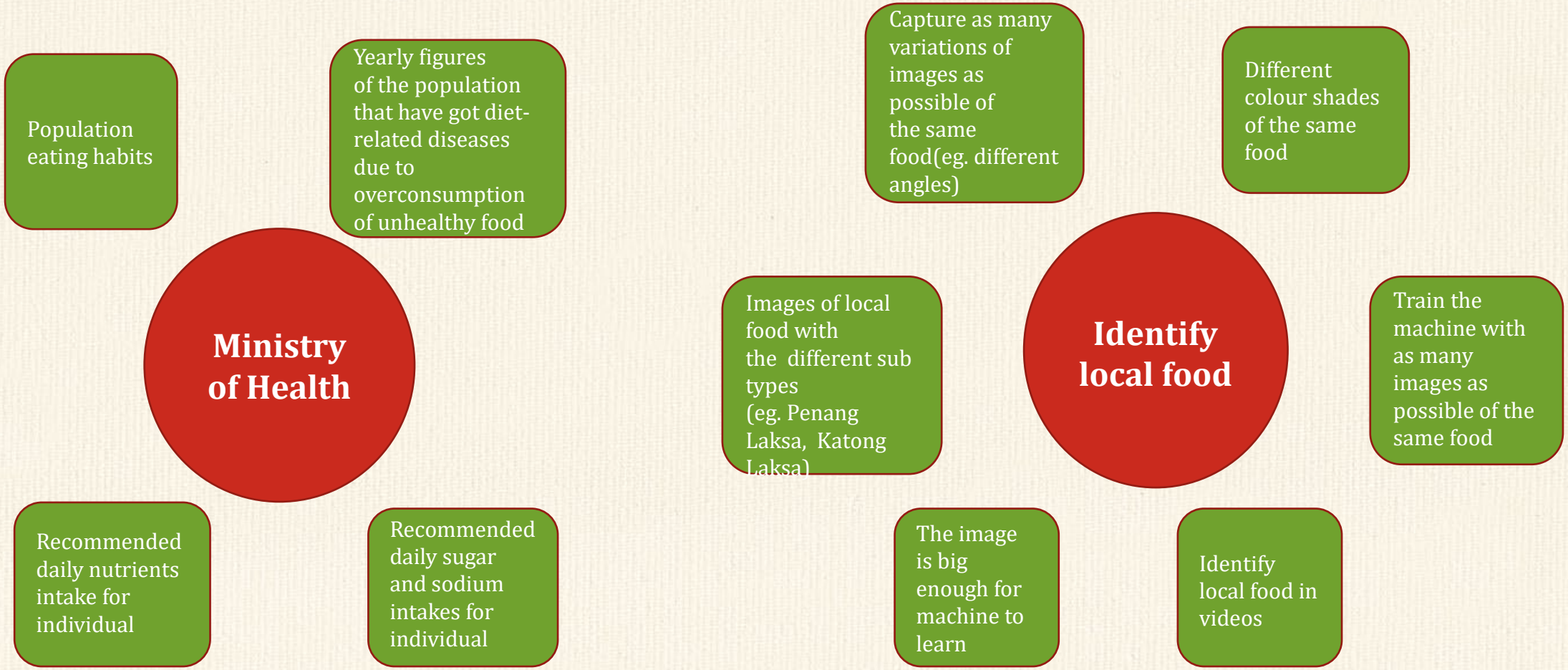
- Encrypt sensitives files
- Manage data access
- Manage data acquisition: acquire sensitive data that is actually needed
- Manage data utilization: Confidentiality risk can be reduced by using sensitive data only as approved and as necessary

Fairness

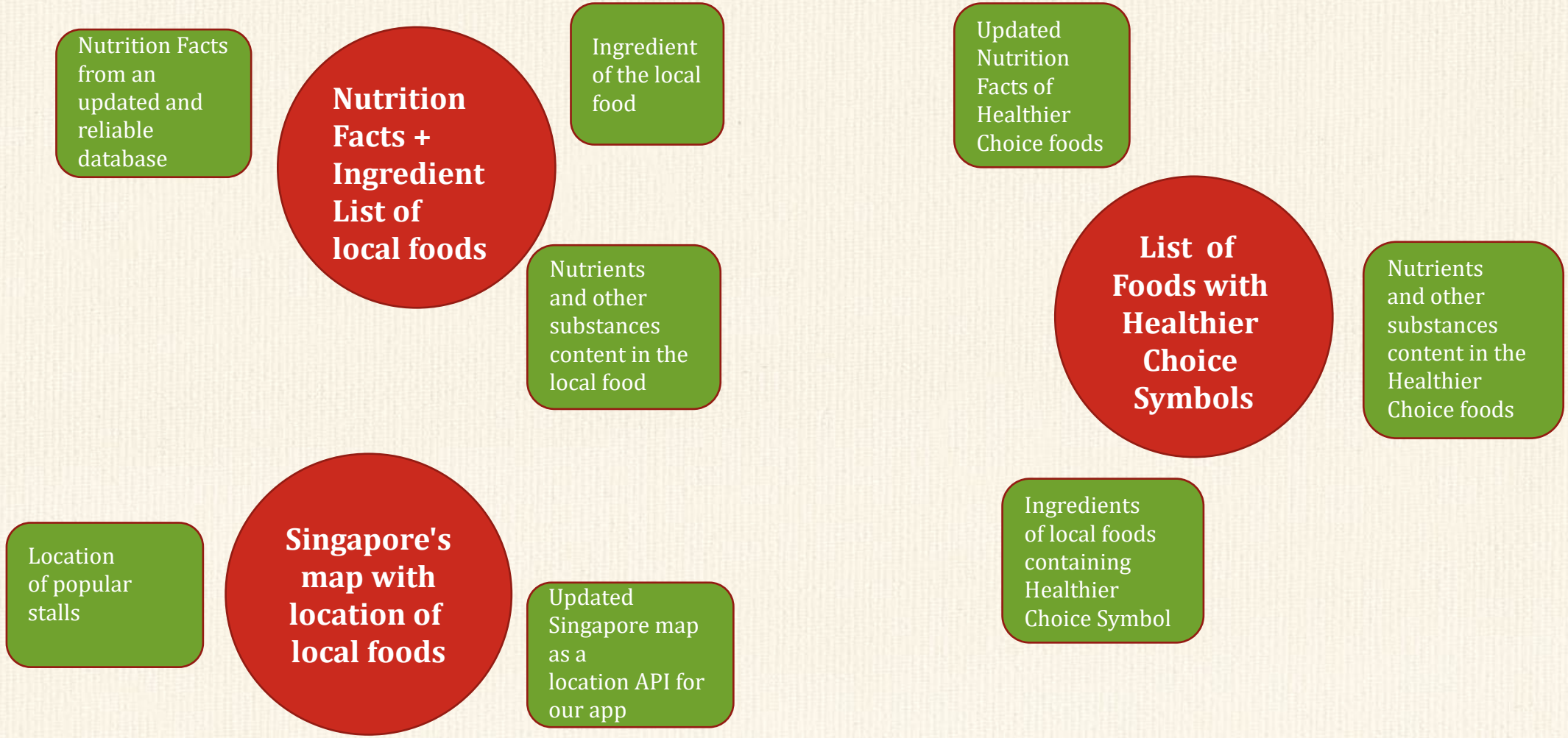
- To minimize bias and to promote inclusive representation

- Our AI solution must be able to recognize and identify images of any types of food including Chinese, Malay, Indian food etc
- Images can be uploaded by anyone regardless of gender, race, religion, income, etc
- Data is sourced from reliable and established organization or government body

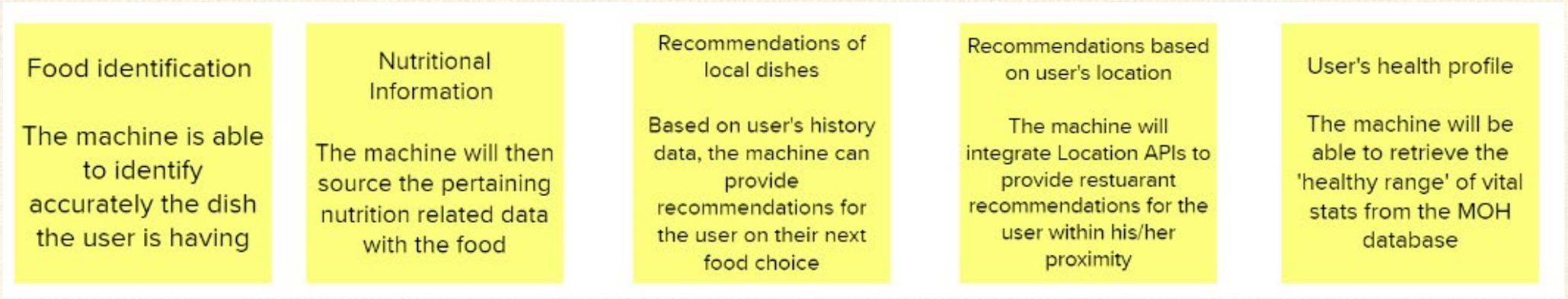
Understanding



Understanding (continued)



Reasoning Statement



SnapFood can **offer users with food nutritional information at their fingertips** by **having an accurate & updated Nutrition Facts** based on **food photos taken or images uploaded by users with their mobile phone**

Knowledge (Primary effect) - The intended value of our AI to users

Understands the nutrition of local food

- Health-conscious users know what nutrients (eg: protein, fat, carbohydrate) and other substances are in the local food
- Health-conscious users can tailor their food options to fit their needs and desires

Knows what local food is healthy

- Health-conscious users are more knowledgeable, and they are more aware of what's healthy and what they should avoid
- Allow health-conscious users to switch consumption away from 'unhealthy' to 'healthy' local food

Aids users in making informed decision

- Health-conscious users can make educated choices on their own health

Knowledge (Secondary effect) - The unintended value of our AI to business

Understands the nutrition of local food served

Food sellers (eg: Food centre, restaurant etc) know what nutrients (eg: Proteins, Fats etc) and other substances are in the food served

Knows what local food is healthy for sale

Food sellers know what healthy food they need to provide

Food sellers know what food is good for consumers

Knows the nutritional require-ments

Health Promotion Board provides nutritional requirements for different groups of people eg: adults, children, elderly etc

Knowledge (Tertiary effect) - Unintended consequences of our AI that could negatively impact business and users

Negative feedback

- User relies too much on the AI solution and forget the whole idea of achieving a balanced diet, to eat food in moderation
- User may not be aware of that AI solution only provides the 'Standard' nutritional informati on of the food
- Food may look healthy but prepared with unhealthy ingredients, unhealthy cooking method
- Could step into others privacy area when taking the pictures of the food
- User's eating habit may change
- The business of food sellers selling 'unheal thy' food may be impacted

PROJECT PLANNING AND TIMELINE

MVP PLANNING

MVP 1

- Gather at least **2 Food Classes** of local food (1000 images each class) using Image Scraping

- Build a Binary Classification model by using the images and able to achieve at least **80%** accuracy

MVP 2

- Expand our image database to at least **10 Food Classes** of local food (1000 images each) using Image Scraping

- Build Multi-Class Classification Model with 10 food classes and able to achieve at least **80%** accuracy

MVP 3

- Deploy the trained model to **localhost** through web app

- To be able to **upload images**

- Based on the food image, categorize the food type and provide a general **nutritional value**

MVP 4

- Build a **Transfer Learning Model** by using existing models and methods

- Experiment with using with only 10% of the original dataset to see if it can achieve **at least 80% accuracy**

- Expand our image database to fruit classes (**total 20 classes**)

MVP 5

- Deploy the web app onto a public web server

- Expand the image database to even more images of local food (**total 35 classes**)

- Making app able to work on phone

- Implement a user feedback system to point out any inaccurate prediction

PROJECT TIMELINE

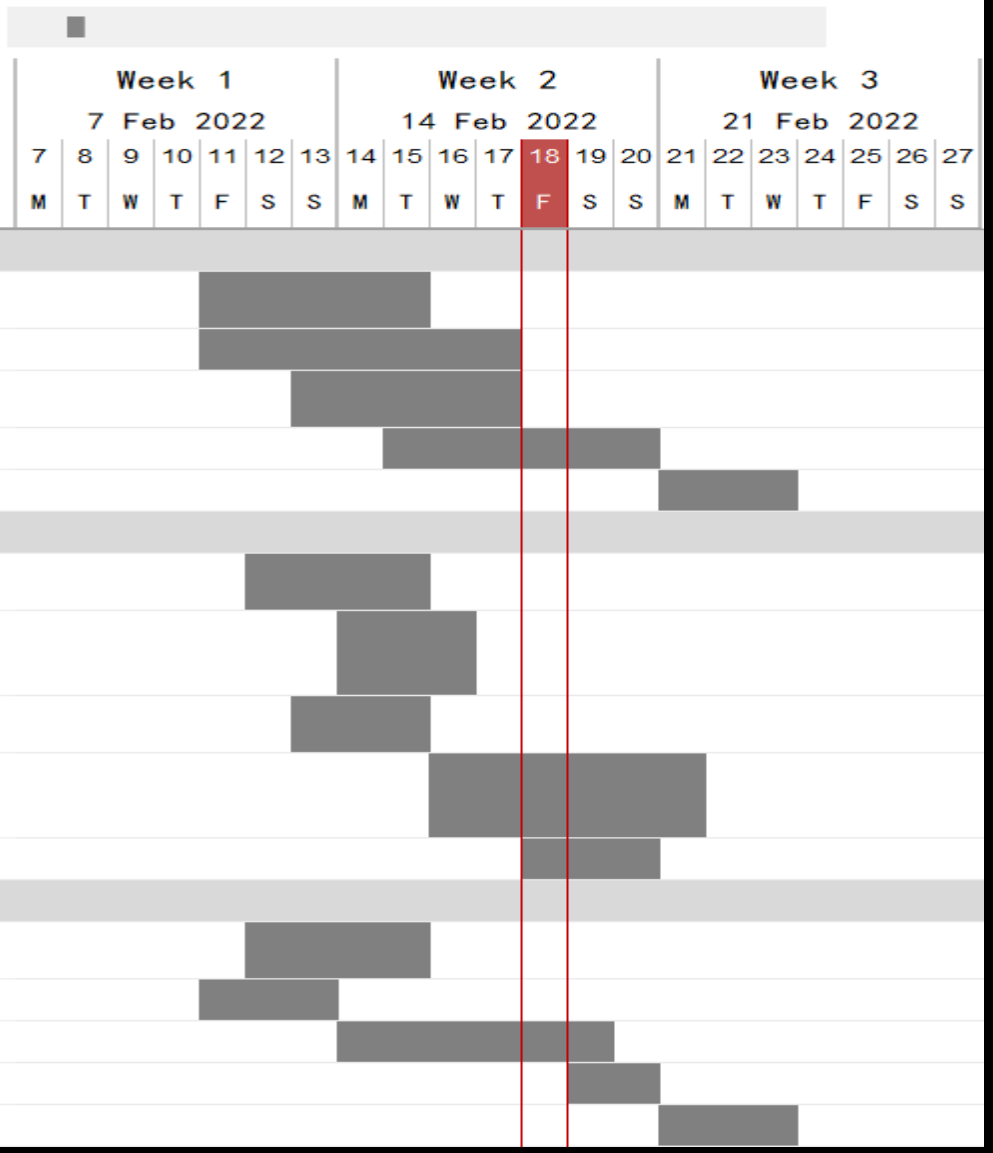
AIP Project Schedule

SnapFood

Project Start Date 2/11/2022 (Friday) Display Week 1

Project Lead

WBS	TASK	LEAD	START	END	DAYS	% DONE	WORK DAYS
1	**Slide Deck**			–			–
1.1	Revenue Model Information	Felicia	Fri 2/11/22	Tue 2/15/22	5	100%	3
1.2	AI Essentials and EDT	Chengwee	Fri 2/11/22	Thu 2/17/22	7	100%	5
1.3	Implementation Planning	Zhengtao	Sun 2/13/22	Thu 2/17/22	5	100%	4
1.4	Project Learning	Yilong	Tue 2/15/22	Sun 2/20/22	6	100%	4
1.5	Dry Run Rehearsal		Mon 2/21/22	Wed 2/23/22	3	100%	3
2	**Research and Development**			–			–
2.1	Image Gathering of foods	Zhengtao	Sat 2/12/22	Tue 2/15/22	4	100%	2
2.2	Web Application Research and Development	Steven	Mon 2/14/22	Wed 2/16/22	3	100%	3
2.3	Nutrition Information gathering	Yilong	Sun 2/13/22	Tue 2/15/22	3	100%	2
2.4	CNN Research and Development (Google Collab)	Zhengtao	Wed 2/16/22	Mon 2/21/22	6	100%	4
2.5	Citations	Felicia	Fri 2/18/22	Sun 2/20/22	3	100%	1
3	**AI Application**			–			–
3.1	Google and Bing Image Scraping	James	Sat 2/12/22	Tue 2/15/22	4	100%	2
3.2	Web Application	Steven	Fri 2/11/22	Sun 2/13/22	3	100%	1
3.3	Transfer Learning	Zhengtao	Mon 2/14/22	Sat 2/19/22	6	100%	5
3.4	Mobile Application	Steven	Sat 2/19/22	Sun 2/20/22	2	100%	0
3.5	Live Demo Rehearsal		Mon 2/21/22	Wed 2/23/22	3	100%	3



TECHNICAL PLANNING AND CONSIDERATION

WHY PYTHON?

- Easy Structure and Syntax
- Succinct and readable code
- Extensible and portable programming language
- Provides various libraries which play a crucial role in data science and program like Tensorflow (A key to our project)

WHY TENSORFLOW?

- Open-Source Artificial Intelligence Library released by Google
- Allows developers to create large scale neural networks with many layers
- Based on python language that invokes C++ to construct and execute dataflow graphs
- Mainly use for Deep Learning Projects such as:
  - Classification
  - Perception
  - Understanding
  - Discovering
  - Prediction
  - Creation

WHY STREAMLIT?

- Free open-source Python framework
- Allows engineers to quickly build interactive web applications
- Does not require to have any prior knowledge of web development
- Seamless to work on interactive loop of coding and viewing results in the web apps

DataSpell

The IDE for Professional Data Scientists

- Intelligent Coding Assistance
- Built-in error debugger
- Version Control tool that allows to manage Git Projects
- Built in database tool that helps to access and query databases
- A terminal function which facilitates working with command console
- Highest performance depends on the specification of the local computer
- An IDE dedicated more to data science and less to development

Pros

V

S

Google colab

- Runs on Google server. Can be accessed from anywhere without hard drive since it is stored in google drive
- Free GPU and TPU provided at no charges
- Can be shared with others without downloading
- No need to install anything, can be used through browser.
- Partial Free, subscription for better services

DataSpell

The IDE for Professional Data Scientists

- It is not free, monthly cost of 20 USD subscription. But comes with a free 1 month trial use for every new user
- Takes up a huge chunk of the computation memory on each startup
- Fairly slow to load, heavily dependent on the specs of the local CPU
- Installation and handling of packages can be a pain for new programmers

Cons

VS

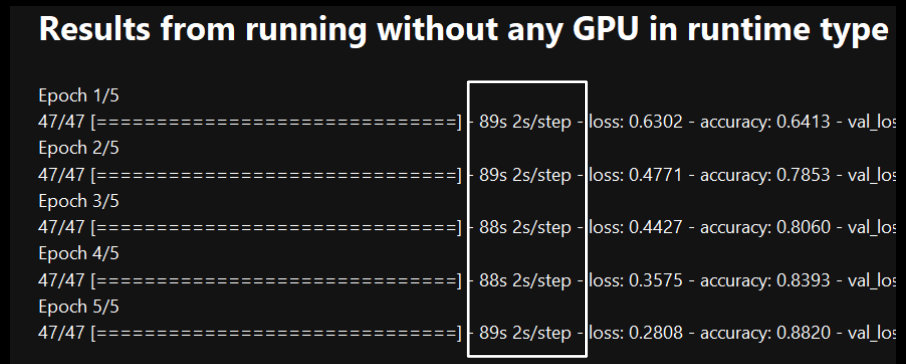
Google colab

- Can only provide the common tools like pre-added packages but it is not suitable for specialisation
- Does not provide a persistent storage facility and the uploaded files will be removed when the session restarted
- Having to repeat the same execution cells due to session restarted and limited idle time which will consume a lot of time to re-run everything especially for large deep learning projects
- Although GPU is free, but it has a very limited number even for 10 dollars per month subscription

Winner

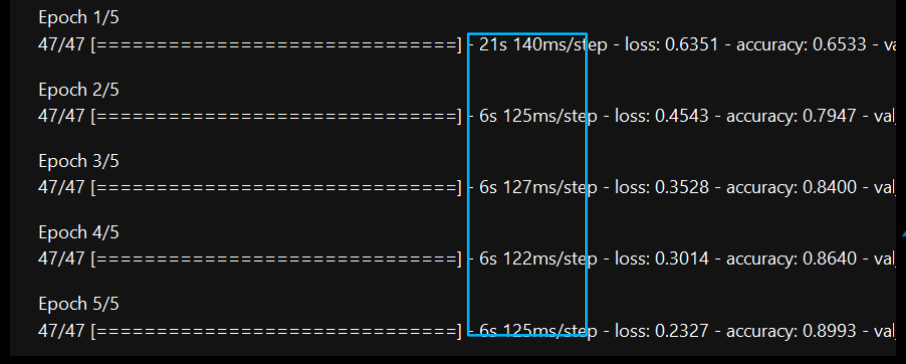
DataSpell
The IDE for Professional Data Scientists

Results from running without any GPU in runtime type



Total time taken: 443 seconds

Results from running GPU in runtime type



Total time taken: 45 seconds

Main Reason for choosing DataSpell

- **GPU is a major factor in CNN modelling building.**
- **Without presence of GPU, the execution of per epoch can takes nearly 10 times longer to execute.**
- **Executing 700k trainable params model can takes over 1 hour to complete with GPU. Imagine executing without a GPU presence**
- **And the restriction of GPU services with Google Colab has left us with no choice since Google Colab subscription model does not apply to Singapore**

Alternatives with different scraper

As we want to improve our model accuracy on classifying images, we switched to different Image Scraper and be able to get better image quality and thus improve our model.

★ Do take note that due to Bing smaller Image Database, we make use of the other techniques to allow us to train our model with smaller image quantity.

Bing Image Scraper

- Smaller Image Database
- More relevant image results
- Repetition Images are less likely to occur
- Able to gather large image size (1080x1080 pixels)

Google Image Scraper

- Huge Image Database
- 30% of the total gathered images are irrelevant
- Repetition Images appear more frequently
- Most of the image gathered are small in size. Mostly 250x250 pixels

Tools

- Mural
- Google Chrome
- Jupyter Notebook – web application that allows collaboration that integrate live code
- Dataspell
- Python - Programming Language
- Google Colboratory
- Github Cloud Storage
- Google Cloud Storage
- Anaconda – simplify package management
- PyCharm
- Spyder

LIBRARIES

- **Pandas** – Powerful/flexible library for Data Analysis, Manipulation, Filtering
- **Matplotlib** – Comprehensive library for creating visualizations
- **Numpy** – Used to perform mathematical and logical operations on arrays
- **Glob** – Filename pattern matching module
- **Keras**
- **Tensorflow**
- **Tensorflow Hub**
- **Selenium**
- **Streamlit**

IMPLEMENTATION OVERVIEW

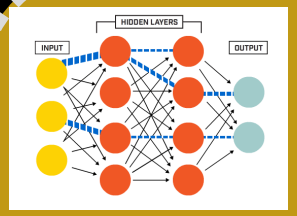
Image Scraping

PyCharm
A popular choice IDE software provides a wide range of tools dedicated to python users

Selenium
A webdriver that automatically detect tests on pre-selected browser

Google Chrome
Selected browser for automated image scraping on Google Image Search Engine

Google Cloud
Cloud repository that can store larger file size that can't be stored on Github



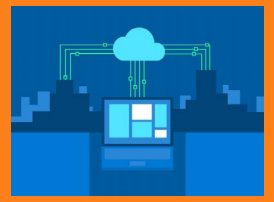
Neural Networks Modelling

Dataspell
An IDE for data science with intelligent Jupyter Notebooks, and other built-in tools

Tensorflow
An end to end open source platform for large scale machine learning tool

Keras
High level deep learning API by for implementing neural networks and interface for Tensorflow

TensorFlow Hub
A repository of trained machine learning model for transfer learning and fine-tuning



Web Deployment

Spyder
A lighter version of IDE software as running localhost operation gets very memory intensive and Pycharm has too many plugins

Streamlit
Open source python framework for building web apps for Machine Learning and Data Science

Anaconda
A Python distribution that simplify package management and deployment which is important for our web app

Github Repository
Streamlit cloud allows to deploy directly from Github repository onto the web

TECHNICAL ARCHITECHTURE

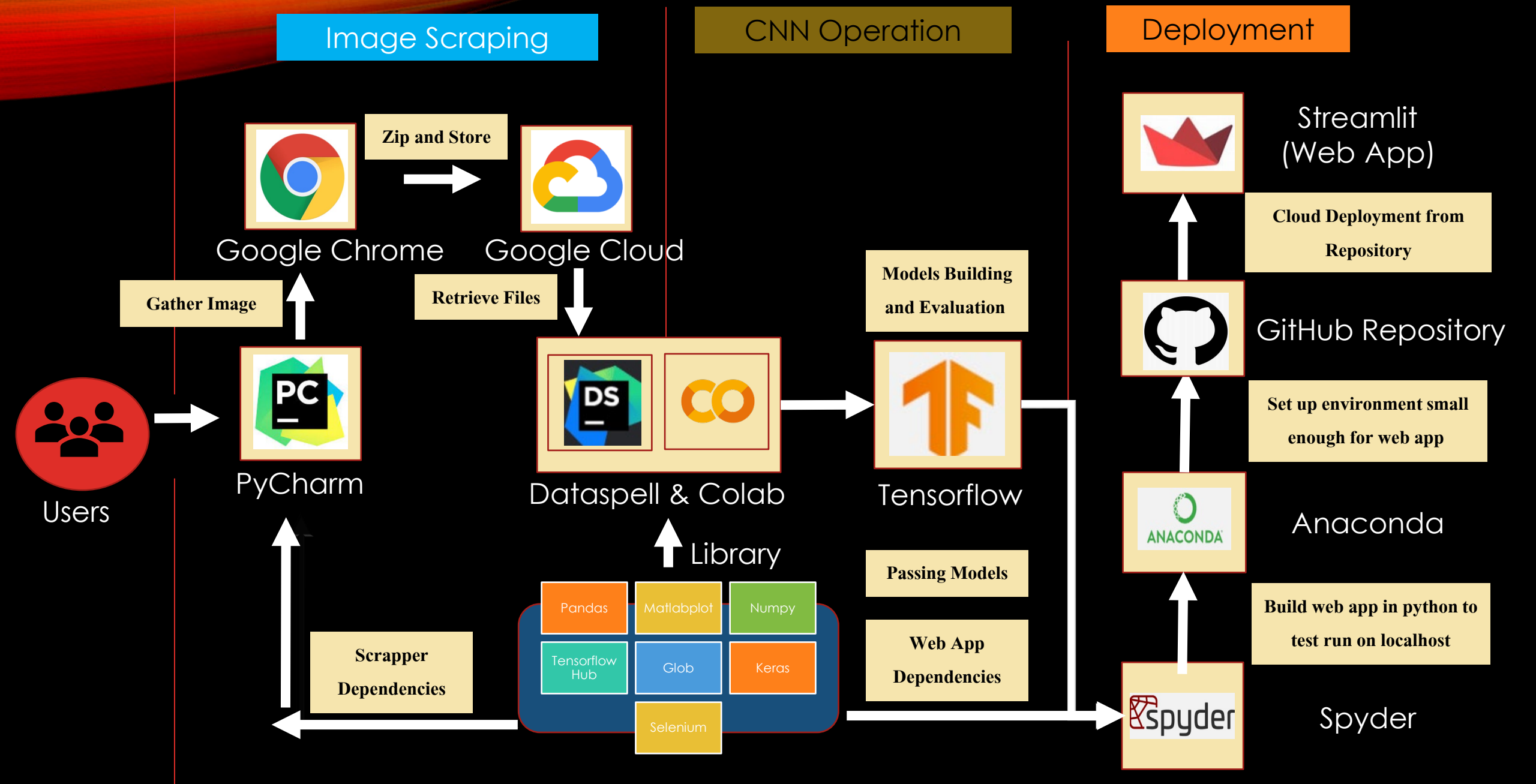


Image Scraping

What is Image Scraping?

- **Image scraping is a subset of the web scraping technology**. It is like pulling an image URL out of a website and save the image onto the local computer automatically.
- Upon executing the Python program after inputting the search key (example laksa) and the number of images required, the program will capture the images and save on the destination storage
- Initial Estimation Number of Images
  - 750 Images for Training
  - 250 Images for Testing
- We will be using
  - -PyCharm to execute the codes
    -Google Chrome as a default browser
    -Selenium as webdriver
    -Google Image website as target search engine

Google Image Scraper Program

Selenium Webdriver

PyCharm

Python codes to sequentially jump to next image frame until loop completed

Loop for 1000 times

Storing Image of Food one at time

Google Chrome

Problems with Google Image Scraping

A) 30% of the total images gathered are unwanted and ended up as Noise during training.
Such as:

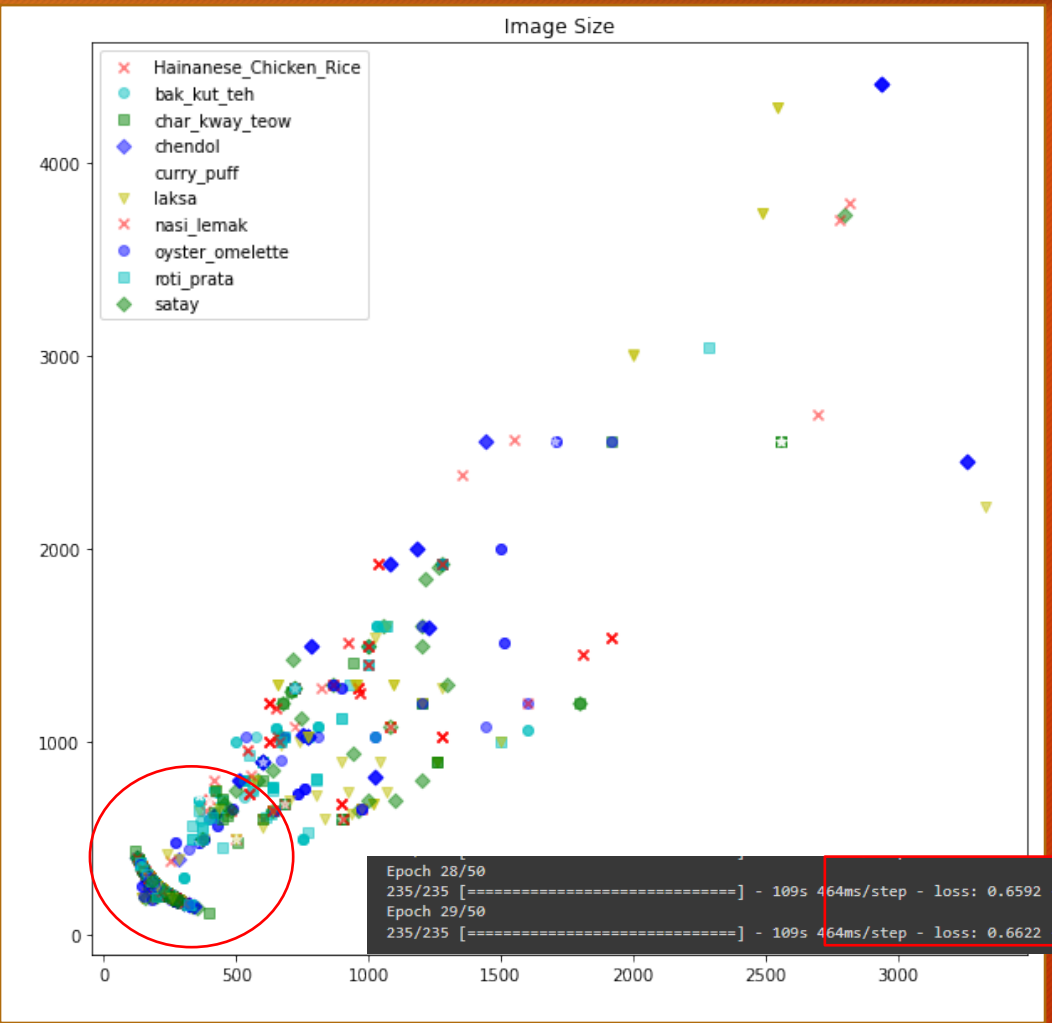
1) Food still in preparation

2) Food stall images

3) Incorrect Food Type

4) Irrelevant Images

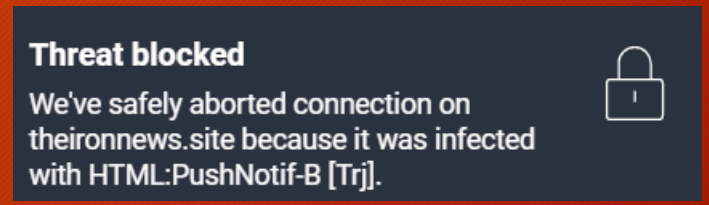
Problems with Google Image Scraping

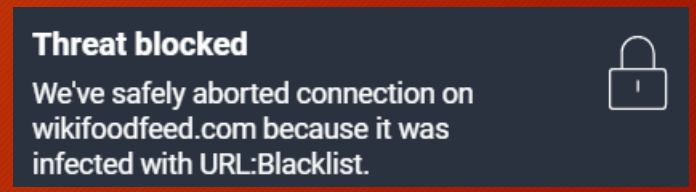


(Image from Multiclass classification after 50 epochs)

B) 80% of the 10k images falls within (200x200) to (500x500) pixel range.

Because of the small in size, the models of Binary/Multi-Class Classification trained from these have rather high loss and low accuracy rates



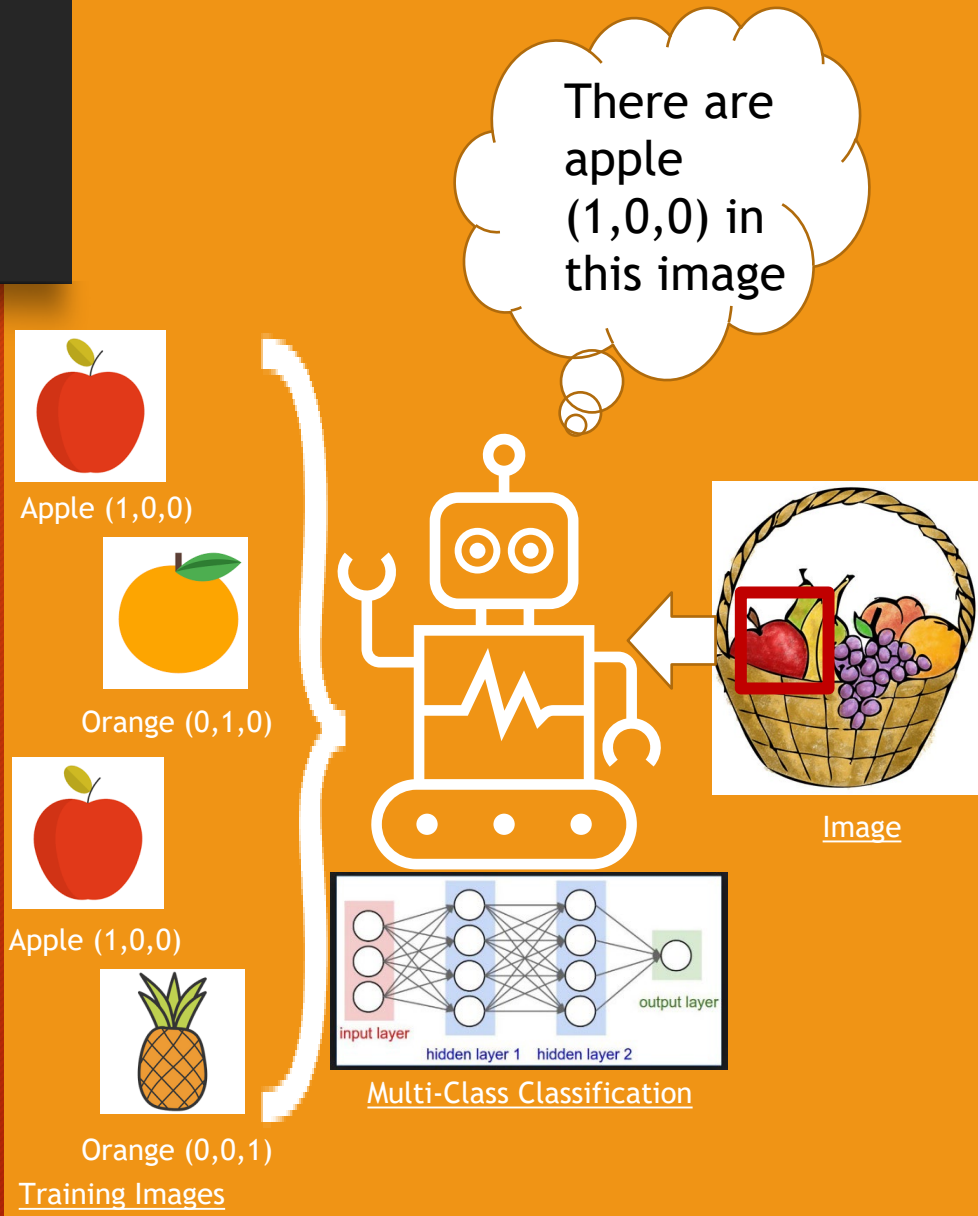


C) During the Image Scraping process, we encountered quite a number of viruses/malwares that are attached to the images. We are lucky that we have active anti-virus software to prevent any further damage it might have done


UNDERSTANDING
(CONVOLUTIONAL NEURAL NETWORK

What is Convolutional Neural Networks?

- Convolutional Neural Networks (CNN) is a Deep Learning algorithm which can take in an input image , assign importance (learnable weights and bias) to various aspects/objects in the image and be able to differentiate one from another.
- Most commonly applied to analyze visual imagery.
- CNN use relatively little pre-processing compared to other image classification algorithms.
- CNN layers mainly consists of
  - Input Layer
  - Hidden Layers
  - Output Layer



Source:
Online lecture notes from https://academy.zerotomastery.io/courses/1240775/lectures/

Steps in modelling with TensorFlow

=

```
<tf.Tensor: shape=(1, 224, 224, 3), dtype=float32, numpy=
array([[[[0.47462985, 0.27462983, 0.19787915],
         [0.5018007 , 0.29003602, 0.19591837],
         [0.5140056 , 0.3022409 , 0.20812324],
         ...,
         [0.6666667 , 0.4916363 , 0.39976063],
         [0.65918356, 0.48663455, 0.3807522 ],
         [0.64705884, 0.4745098 , 0.3764706 ]],
```

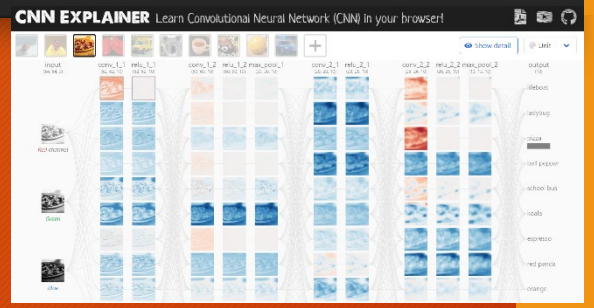
2. Turn all data into numbers (Neural Network cannot handle images)

1. Import the images

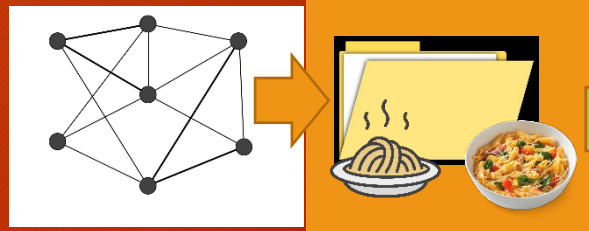
```
array([[[105,  91,  56],
        [110,  95,  62],
        [112,  97,  66],
```

```
array([[[0.41176471, 0.35686275, 0.21960784],
        [0.43137255, 0.37254902, 0.24313725],
        [0.43921569, 0.38039216, 0.25882353],
```

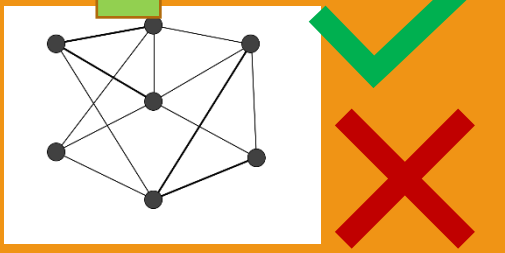
3. Normalize the tensors to fit between 0 to 1 (Neural Network perform better under normalized data)



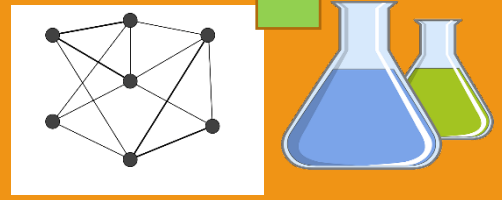
4. Build a Neural Networks model with layers



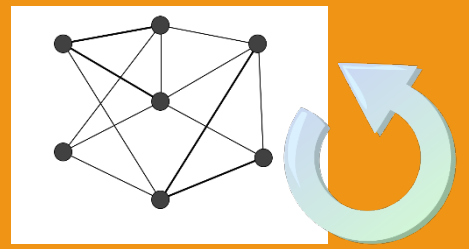
5. Fit the model to the data and make a prediction



6. Evaluate the model



7. Improve through Experimentation

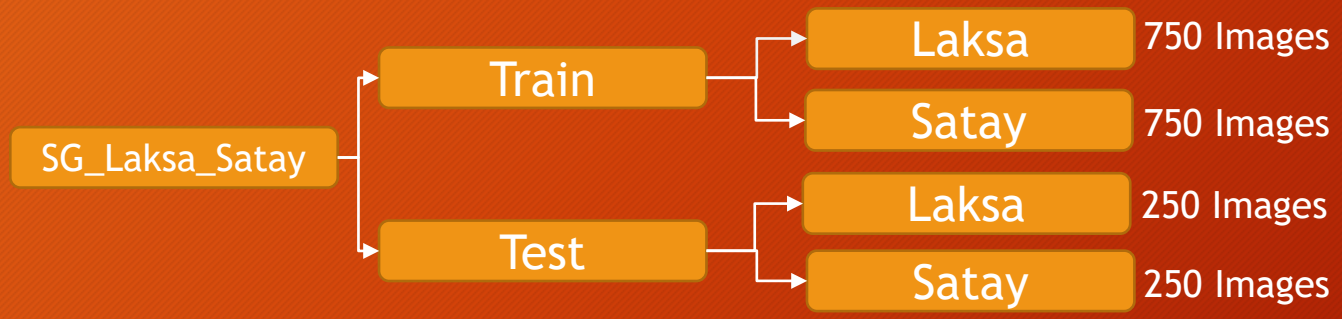


8. Save and reload the model

Binary Classification CNN (Setting up to import images)

Why start with binary classification first (as MvP 1)?
1.Because we want to follow the motto of "**FAIL FAST, FAIL CHEAP**". If it fails at this experimental stage, at least we can divert our focus onto some other projects instead.
2.We want to test if the number of images of the training data and the test data is sufficient to train the model. If it works, we can expand further to more food classes (Scaling up)

Steps to import images
1) Open up a new Project in **DataSpell**
2) Import the dependences (Important: Tensorflow v2.7)
3) Import zip file (**SG_Laska_Satay**) from **Google Cloud Storage**
Note: These are the images we took with Google Image Scraping and stored in Google Cloud Storage
4) Check the file structure and total number of images



1)

2)
```
import zipfile
import os
import pathlib
import numpy as np
import pandas as pd
import glob
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from IPython.display import Image
import random
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
```

3)
```
## Get the data
!wget https://storage.googleapis.com/sgfood/SG_Laksa_satay.zip

#Unzip the download file
zip_ref = zipfile.ZipFile("SG_Laksa_satay.zip")
zip_ref.extractall()
zip_ref.close()
```

4)
```
[ ] #Walk through pizza_steak directory and list number of files
    for dirpath, dirnames, filenames in os.walk("SG_Laksa_satay"):
      print(f"there are {len(dirnames)} directories and {len(filenames)} images in {dirpath}. ")

    there are 2 directories and 0 images in SG_Laksa_satay.
    there are 2 directories and 0 images in SG_Laksa_satay/test.
    there are 0 directories and 250 images in SG_Laksa_satay/test/laksa.
    there are 0 directories and 250 images in SG_Laksa_satay/test/satay.
    there are 2 directories and 0 images in SG_Laksa_satay/train.
    there are 0 directories and 750 images in SG_Laksa_satay/train/laksa.
    there are 0 directories and 750 images in SG_Laksa_satay/train/satay.
```

There's a total of 1500 train images (750 each) and a total of 500 test images (250 each) for this dataset

Binary Classification CNN (Exploratory Data Analysis)

5) Produce some random images on every execution of code section to see if able to display and give us a general idea on the quality and dimensions of the images

Note: The scaling at the side refers to the pixel of the height and width of each picture.

6) Ensure the shape of the input image is right

```
<tf.Tensor: shape=(194, 259, 3),
```

-194 refers to the height
-259 refers to the width
- 3 refers to the colour channels

```
array([[[227, 241, 241],
        [227, 241, 241],
        [228, 242, 242],
```

The numbers in the array refers to the colour representation in the image

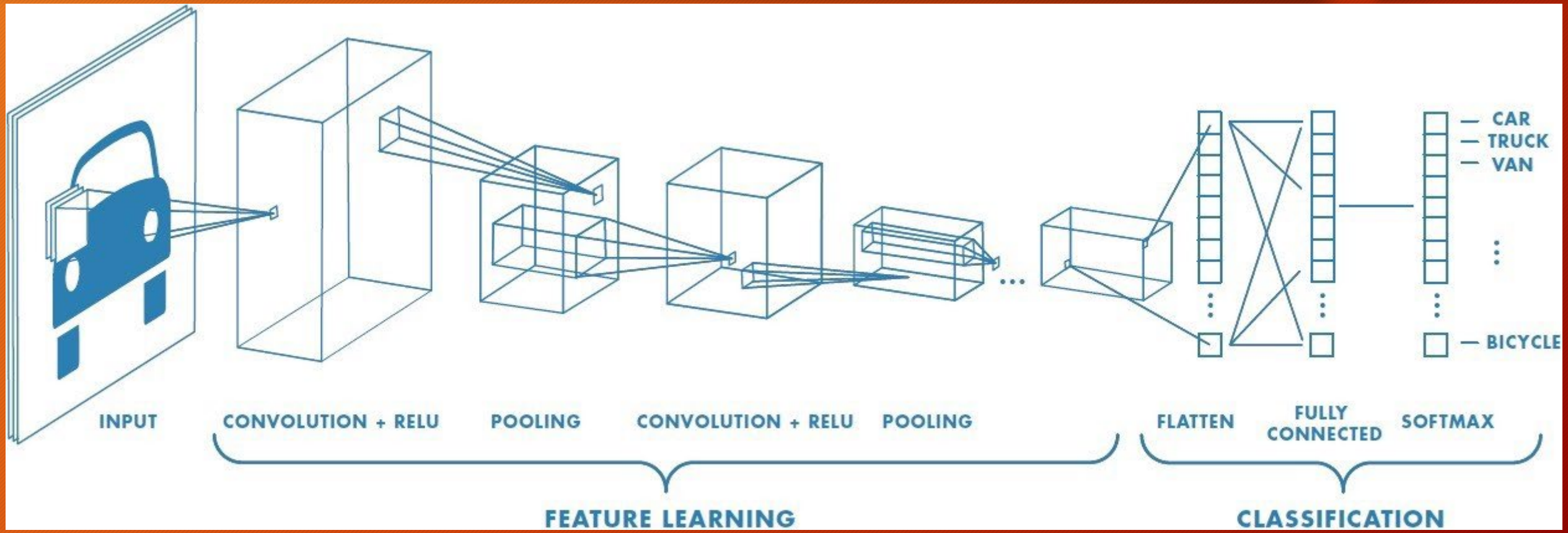
#FF0000	RGB (255, 0, 0)
#FF7F00	RGB (255, 127, 0)
#FFFF00	RGB (255, 255, 0)
#FFFF00	RGB (0, 255, 0)
#0000FF	RGB (0, 0, 255)
#4B0082	RGB (75, 0, 130)
#8F00FF	RGB (143, 0, 255)

```
[ ]  # converting into into Tensor
     tf.constant(img)

<tf.Tensor: shape=(194, 259, 3), dtype=uint8, numpy=
array([[[227, 241, 241],
        [227, 241, 241],
        [228, 242, 242],
        ...,
        [135, 132,  53],
        [131, 128,  49],
        [129, 126,  47]],

       [[227, 241, 241],
        [227, 241, 241],
        [228, 242, 242],
        ...,
        [122, 123,  30],
        [115, 116,  23],
        [111, 112,  19]],

       [[228, 242, 242],
        [228, 242, 242],
        [228, 242, 242],
        ...,
        [124, 131,  19],
        [113, 120,   8],
        [105, 112,   0]],
```

7) Checking on the dimensions of all the images in the train directory

Note: We noticed the dimensions of all images are rather small as suspected. We will do a trial run to see the performance.

Architecture of Classification Model (Diagram Explanation)



Source: A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way

Architecture of Binary Classification Model (4 Section of Codes for complete modelling)

1) Preprocess data

- Setup Directories for Train and Test Data
- Define Batch Size (size 32 is often used and recommended)
- Resize and Define Image Target Size (Ensure the image from Image Generator are equal size)
- Define Class Mode (Binary)

```
# Import data from directories and turn it into batches
train_data = train_datagen.flow_from_directory(directory=train_dir,
                                               batch_size=32,
                                               target_size=(224, 224),
                                               class_mode="binary",
                                               )

test_data = test_datagen.flow_from_directory(directory=test_dir,
                                             batch_size=32,
                                             target_size=(224, 224),
                                             class_mode="binary",
                                             )
```

2) The main body of CNN

- Setting up of the Hyperparameters for the core

(More in-depth explanation will be shown later)

```
# Create a CNN model
model_1 = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(filters=10,
                         kernel_size=3, # can also be (
                         activation="relu",
                         input_shape=(224, 224, 3)), #
  tf.keras.layers.Conv2D(10, 3, activation="relu"),
  tf.keras.layers.MaxPool2D(pool_size=2, # pool_size ca
                            padding="valid"), # padding
  tf.keras.layers.Conv2D(10, 3, activation="relu"),
  tf.keras.layers.Conv2D(10, 3, activation="relu"), # a
  tf.keras.layers.MaxPool2D(2),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(1, activation="sigmoid") # bina
])
```

3) Compile CNN Model

- Setting up of the Hyperparameters for the compiler

(More in-depth explanation will be shown later)

```
[ ]  # Compile the model
     model_1.compile(loss="binary_crossentropy",
                     optimizer=tf.keras.optimizers.Adam(),
                     metrics=["accuracy"])
```

4) Fit the model

- Define the directory path for the train data
- Define the number of epochs (Epoch is the hyperparameter of gradient descent that control number of complete passes through data set)
- Define the number of steps per epoch
- Define the directory path for validation data
- Define the number of steps of validation per epoch

```
# Fit the model
history_1 = model_1.fit(train_data,
                        epochs=5,
                        steps_per_epoch=len(train_data),
                        validation_data=test_data,
                        validation_steps=len(test_data))
```

Architecture of Binary Classification Model (The Main Body)

Hyperparameter	Binary Classification
Input Layer Shape	Number of features (example: (224,224,3))
Hidden Layer(s)	Problem specific, minimum = 1, maximum = unlimited
Neurons per hidden Layer	Problem specific, general 10 to 100
Output Layer shape	1 (Y/N)
Hidden activation	Usually ReLU (rectified linear unit)
Output activation	Sigmoid
Loss function	Binary Cross entropy
Optimizer	SGD (stochastic gradient descent), Adam

Source: Adapted from page 295 - Hands on Machine Learning with Scikit Learn, Keras & TensorFlow book from Aurelien Geron

```
# Create a CNN model
model_1 = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(filters=10,
                         kernel_size=3, # can also be
                         activation="relu",
                         input_shape=(224, 224, 3)),
  tf.keras.layers.Conv2D(10, 3, activation="relu"),
  tf.keras.layers.MaxPool2D(pool_size=2, # pool_size
                            padding="valid"), # paddi
  tf.keras.layers.Conv2D(10, 3, activation="relu"),
  tf.keras.layers.Conv2D(10, 3, activation="relu"),
  tf.keras.layers.MaxPool2D(2),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(1, activation="sigmoid") # bi
])
```

Codes from the Main body of the CNN model

Architecture of Binary Classification Model (The Compiling section)

Hyperparameter	Binary Classification
Input Layer Shape	Number of features (example: (224,224,3))
Hidden Layer(s)	Problem specific, minimum = 1, maximum = unlimited
Neurons per hidden Layer	Problem specific, general 10 to 100
Output Layer shape	1 (Y/N)
Hidden activation	Usually ReLU (rectified linear unit)
Output activation	Sigmoid
Loss function	Binary Cross entropy
Optimizer	SGD (stochastic gradient descent), Adam

Source: Adapted from page 295 - Hands on Machine Learning with Scikit Learn, Keras & TensorFlow book from Aurelien Geron

```
[ ]  # Compile the model
     model_1.compile(loss="binary_crossentropy",
                     optimizer=tf.keras.optimizers.Adam(),
                     metrics=["accuracy"])
```

Codes from the compiling of the CNN

Key Point to Note

ReLu - A non-linear activation function that defined if input is zero, the function output zero. Else the function output its input

Sigmoid - Output activation function specifically for binary classification

Adam - Optimizer of stochastic gradient descent method that is based on adaptive estimation of first order and second order

Loss Function - A method to calculate the prediction of error in Neural Network. In simpler words, the Loss is used to calculate the gradient while the gradient are used to update the weights

Model Summary

```
model_1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 10)      280

 conv2d_1 (Conv2D)           (None, 220, 220, 10)      910

 max_pooling2d (MaxPooling2D  (None, 110, 110, 10)     0
 )

 conv2d_2 (Conv2D)           (None, 108, 108, 10)      910

 conv2d_3 (Conv2D)           (None, 106, 106, 10)      910

 max_pooling2d_1 (MaxPooling  (None, 53, 53, 10)       0
 2D)

 flatten (Flatten)           (None, 28090)             0

 dense (Dense)               (None, 1)                 28091

=================================================================
Total params: 31,101
Trainable params: 31,101
Non-trainable params: 0
```

Note: The model's layers are based on CNN Explainer Website. As we have no idea how to start, we used that as an example to construct our first model

Keys points to note

Conv2D - Layers that are represented for 2 dimensional inputs such as images

MaxPool2D - Downsamples the input along its spatial dimensions by taking the maximum value over an input window

Flatten - Covert a 3-dimensional layer into one dimensional vector to fit into input of Dense layer. Example 5x5x2 tensor converted into a vector size of 50

Dense - A fully-connected neural network layer

Fit the model (Execution code)

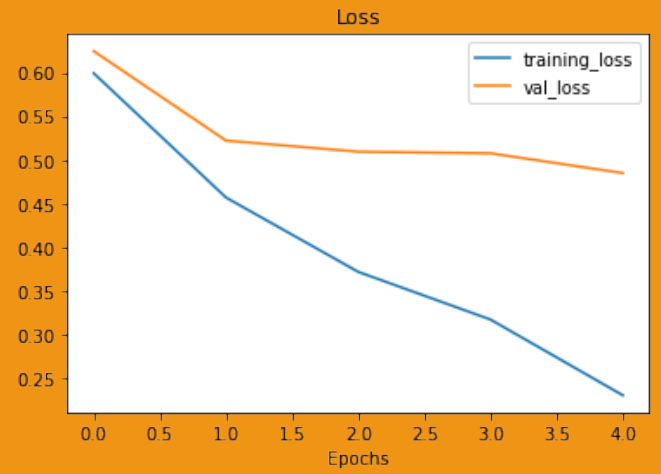
```
[ ]  # Fit the model
     history_1 = model_1.fit(train_data,
                             epochs=5,
                             steps_per_epoch=len(train_data),
                             validation_data=test_data,
                             validation_steps=len(test_data))
```

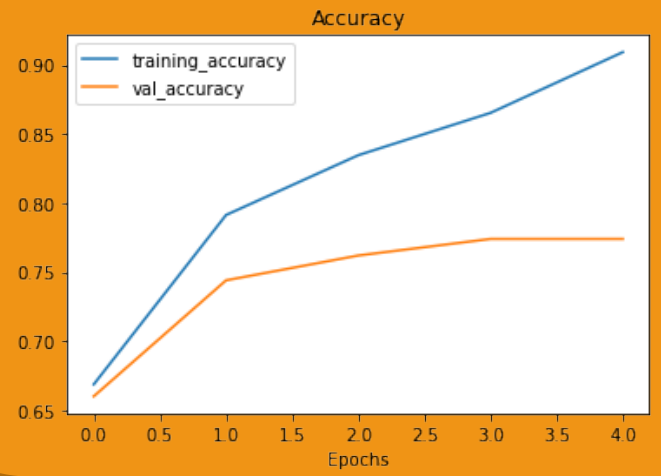
Upon execution of the Fit the model to data, we will have a summary of per epoch
- The estimated time taken to train the model
- Accuracy and Loss for the training data
- Accuracy and Loss for the validation data

```
Epoch 1/5
47/47 [==============================] - 16s 148ms/step - loss: 0.5991 - accuracy: 0.6687 - val_loss: 0.6244 - val_accuracy: 0.6600
Epoch 2/5
47/47 [==============================] - 9s 200ms/step - loss: 0.4570 - accuracy: 0.7913 - val_loss: 0.5222 - val_accuracy: 0.7440
Epoch 3/5
47/47 [==============================] - 6s 136ms/step - loss: 0.3721 - accuracy: 0.8347 - val_loss: 0.5094 - val_accuracy: 0.7620
Epoch 4/5
47/47 [==============================] - 6s 136ms/step - loss: 0.3176 - accuracy: 0.8653 - val_loss: 0.5077 - val_accuracy: 0.7740
Epoch 5/5
47/47 [==============================] - 6s 136ms/step - loss: 0.2310 - accuracy: 0.9093 - val_loss: 0.4851 - val_accuracy: 0.7740
```

Why 47 ?
Because total number of image for training is 1500 (750 each class)
1500 divided by 32(Batch size) = 46.875
So 47 is enough to fit the training data per batch

Evaluating models





Judging by our loss curves, it looks like our model is overfitting the training dataset.

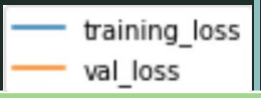
Note: When a model's validation loss starts to increase, it's likely that it's overfitting the training dataset. This means, it's learning the patterns in the training dataset too well and thus its ability to generalize to unseen data will be diminished

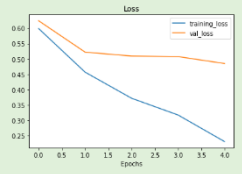
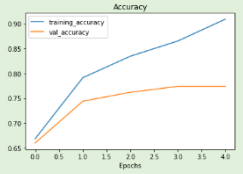
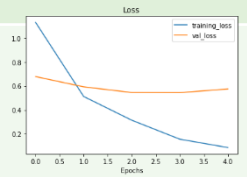
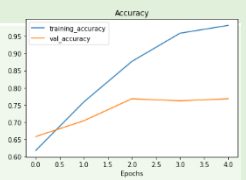
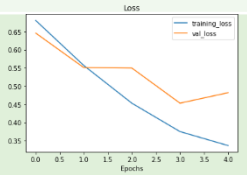
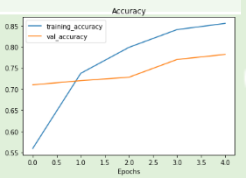
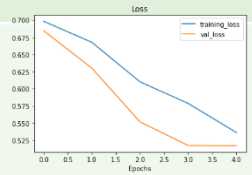
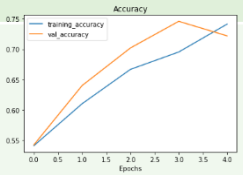
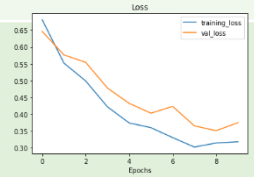
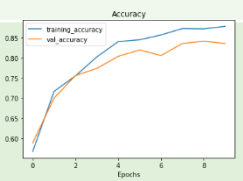
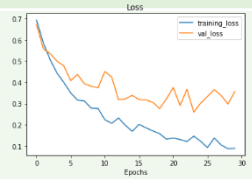
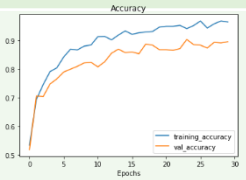
Methods to reduce Overfitting (also known as Regularization)

- Add Data Augmentation
- Add regularization model (Such as Max2D pool)
- Allows the model to train on more data
- Deploy Early Stopping (Explain in detail in later section)

IMPROVEMENT AND SOLUTIONS THROUGH EXPERIMENTS

Improving with Experiments (Binary Classifications)



Model	Epoch	Time (sec)	Total Params	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy	Action Taken	Loss Graph	Accuracy Graph	Remarks
Model 1	5	43	31,101	0.2310	0.9093	0.4851	0.7740	(Baseline Model) -Conv2D+Relu, input() -Conv2D+Relu -MaxPool2D -Conv2D+Relu -Conv2D+Relu -MaxPool2D -Output fully connected			Our base model is based on the CNN explainer website **https://poloclub.github.io/cnn-explainer/**
Model 2	5	31	477,341	0.0867	0.9813	0.5751	0.7680	Removed Max2DPool to see the results			For model 1 to 3, the validation loss and accuracy does not follow the training loss and accuracy well. Clearly it is overfitting
Model 3	5	71	8,861	0.3367	0.8553	0.4819	0.7820	Reinstall back Max2Dpool, Add in another Conv2D+Max2Dpool layer			
Model 4	5	108	8,861	0.5360	0.7413	0.5167	0.720	Data Augmentation with - Rotate, Shear, Zoom Range -horizontal flip=True			Significant improvement as the validation line starting to follow the training line
Model 5	10	211	115,873	0.3184	0.8787	0.3752	0.8360	Incremental increase of 32 filters per layer and increase to 10 epochs, and Data Augmentation			Improvement from model 4
Model 6	30	631	399,009	0.0885	0.9653	0.3559	0.8960	Adding another Conv2D layer with incremental increase of 32 filters per layers, increase to 30 epochs and Data Augmentation			Improved validation accuracy from model 5 and able to fulfil MvP 1 requirement

Predicting Results

Prediction: laksa

Prediction: satay

```
model_6.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_25 (Conv2D)          (None, 222, 222, 16)      448

 max_pooling2d_17 (MaxPoolin  (None, 111, 111, 16)     0
 g2D)

 conv2d_26 (Conv2D)          (None, 109, 109, 32)      4640

 max_pooling2d_18 (MaxPoolin  (None, 54, 54, 32)       0
 g2D)

 conv2d_27 (Conv2D)          (None, 52, 52, 64)        18496

 max_pooling2d_19 (MaxPoolin  (None, 26, 26, 64)       0
 g2D)

 conv2d_28 (Conv2D)          (None, 24, 24, 128)       73856

 max_pooling2d_20 (MaxPoolin  (None, 12, 12, 128)      0
 g2D)

 conv2d_29 (Conv2D)          (None, 10, 10, 256)       295168

 max_pooling2d_21 (MaxPoolin  (None, 5, 5, 256)        0
 g2D)

 flatten_7 (Flatten)         (None, 6400)              0

 dense_7 (Dense)             (None, 1)                 6401

=================================================================
Total params: 399,009
Trainable params: 399,009
Non-trainable params: 0
```

1010
1010

MVP 1

- Gather at least **2 Food Classes** of local food (1000 images each class) using Image Scraping

- Build a Binary Classification model by using the images and able to achieve at least **80%** accuracy

MVP 1 Completed

With this, our binary classification model is able to predict correctly as per intended with a validation accuracy of 89.6% on Model 6. We will move on to MVP 2

Multi-Class Classification

Number of Classes : Increased to 10 Classes
Training Images per Class: 750
Testing Images per Class: 250

Total Number of Images: 10k images

No	Food
1	Hainanese Chicken Rice
2	bak kut teh
3	char kway teow
4	chendol
5	curry puff
6	laksa
7	nasi lemak
8	oyster omelette
9	roti prata
10	satay

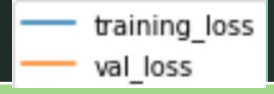
Since Binary Classification has proven to be successful, we decided to retain the same number training and testing images per class.

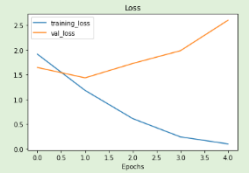
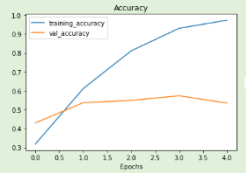
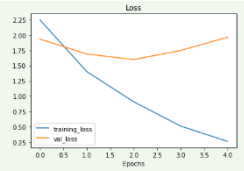
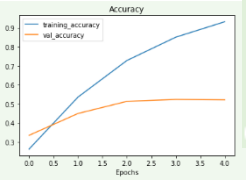
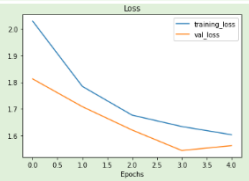
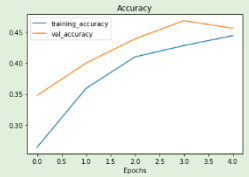
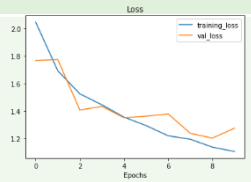
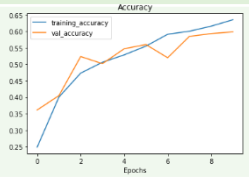
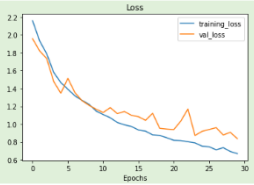
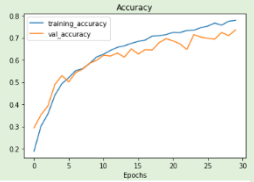
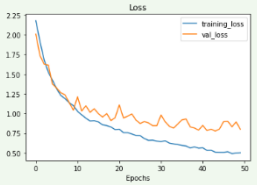
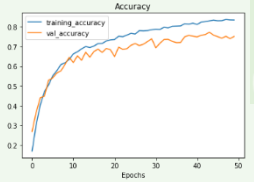
Note: The parameters stays the same as what we have used in Binary Classification, except

-> Output Activation
- ‘`softmax`’

->Loss Function
- ‘`categorical_crossentropy`’

Improving with Experiments (Multi-Class Classifications)



Model	Epoch	Time (sec)	Total Params	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy	Action Taken	Loss Graph	Accuracy Graph	Remarks
Model 1	5	176	283,920	0.0867	0.9793	2.0259	0.5736	(Baseline Model) -Conv2D+Relu, input() -Conv2D+Relu -MaxPool2D -Conv2D+Relu -Conv2D+Relu -MaxPool2D -Output fully connected			Our base model is based on the CNN explainer website **https://poloclub.github.io/cnn-explainer/**
Model 2	5	161	283,920	0.2174	0.9425	1.5935	0.5804	Removed 2 convolutional layer; from 4 layers to 2 layers			Our model 1 and 2 is overfitting by a huge margin.
Model 3	5	530	292,800	1.5927	0.4419	1.4698	0.4920	Data Augmentation -rotation-range=45 -shear & zoom=0.2 -horizontal-flip=True			Data augmentation clearly improves our validation curve
Model 4	10	~1080	74,730	1.1069	0.6352	1.2759	0.5984	-Increased filters to 32; -added 1 more Conv2D -added 1 MaxPool2D -Epochs increased to 10			
Model 5	30	~3450	392,860	0.6700	0.7787	0.8375	0.7356	Conv2D(200)+Relu,input MaxPool2D And reduce by 100 filter after every layer			Our validation line is doing fine for the most part but however, we are unable to achieve at least 80% validation accuracy even after 1 and half hours of training
Model 6	50	~5450	392,860	0.5021	0.8344	0.8008	0.7512	Same parameters as Model 5 but increased to 50 epochs			

Trying to reach 80% Accuracy

After we done Model 6 with 1.5 hours just to train the model and only 75% accuracy, we realised that the "brute force" method is not going to achieve the 80% accuracy objective that we have set in MVP 2

Revised Model 7

- Conv2D(32, stride=2, padding = same), input(224,224,3)
- Conv2D(32, stride=2, padding = same),
- MaxPool2D
- Dropout(0.2)
- Conv2D(64, stride=2, padding = same),
- Conv2D(64, stride=2, padding = same),
- MaxPool2D
- Dropout(0.2)
- Conv2D(128, stride=2, padding = same),
- Conv2D(128, stride=2, padding = same),
- MaxPool2D
- Dropout(0.2)
- Conv2D(256, stride=2, padding = same),
- Conv2D(256, stride=2, padding = same),
- GlobalAveragePooling2d
- Dense(512)
- Dropout(0.2)
- Dense(10)

1) Apply stride and padding on filter

2) Apply dropout layer which randomly sets inputs to 0 to prevent overfitting

3) Filters are doubled after MaxPool2D and Dropout

4) GlobalAveragePooling apply average pooling on spatial dimensions

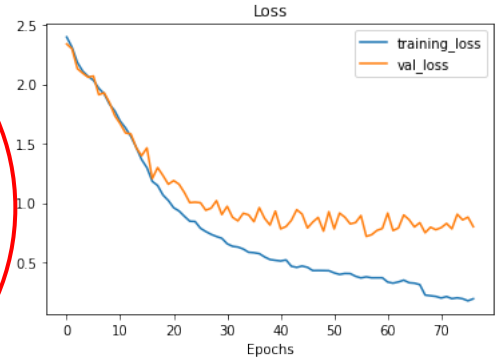
5) Callback function (Early stopping) which stop the training automatically when it cease to improve

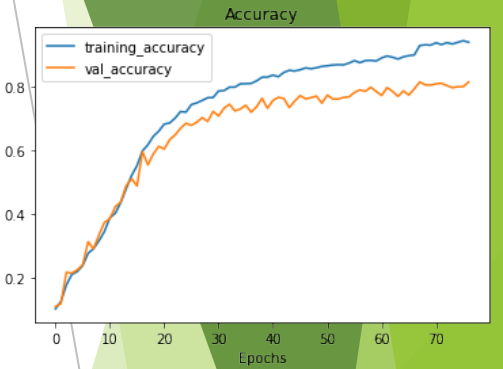
Source: https://stackoverflow.com/

Model	Epoch	Time(Sec)	Total Params
7	100(Early Stop at 77)	~6930	712,490

Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
0.1943	0.9397	0.8009	0.8148

Wrong Prediction
Should be Char Kway Teow





MVP 2

- Expand our image database to at least **10 Food Classes** of local food (1000 images each) using Image Scraping

- Build Multi-Class Classification Model with 10 food classes and able to achieve at least **80%** accuracy

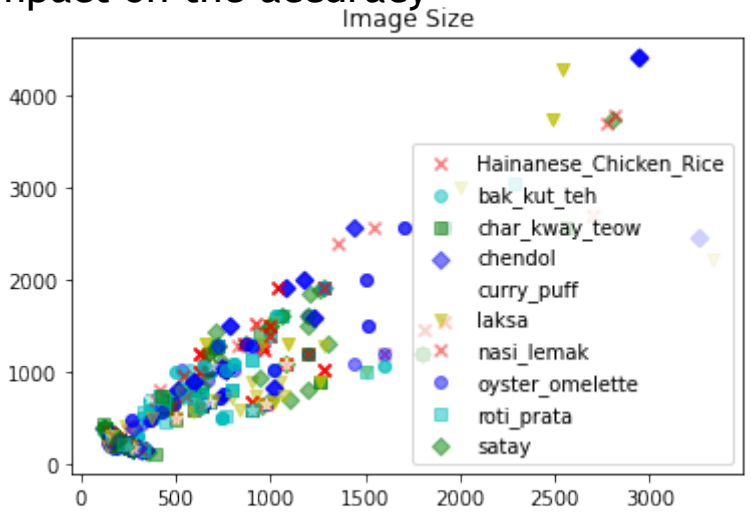
MVP 2 Completed

With this, our multi-class classification model is able to predict with a validation accuracy of 81.5% on Model 7. We will move on to MVP 3 and 4.

Problems.....

Even though we have successfully achieve our MVP 1 and 2 goals, however we can foreseen a lot of problems just to train another 10 more classes.

- Time and resources needed just to train our next model on multi-class classification will be huge
- The human intervention of collecting the images will be **time consuming**
- As more classes are added in, the **accuracy tends to drop** due to the correlation with foods that look similar.
- **Images are generally too small** with Google Image Scraping as mentioned as it will impact on the accuracy

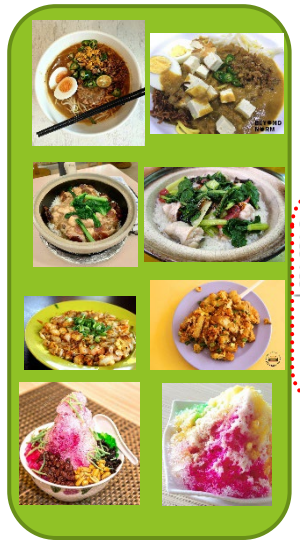


Solutions.....

We will use **TRANSFER LEARNING** from a pre-trained model in Tensorflow Hub (a repository where most of the **state of the art** models are kept)

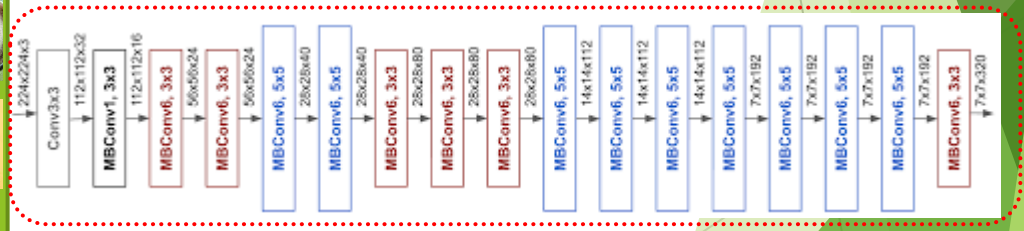
What is Transfer Learning?

Transfer Learning is a popular machine learning method where a model developed for a task is reused as the starting point for a model on second task



Input Data
(20 Classes of food)

Source: https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html



Remain the same
(Frozen, pre-trained on Imagenet)

Output(20)

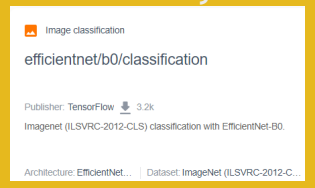
Dense Output Layer

Steps to ensure Transfer Learning work

Find the right Model → Find the right images → Run tests

Steps to find the model that suited us

1. Go to https://tfhub.dev
2. Select Images
3. Under Problem Domains – Clear everything except **"Image Classification"** and **"Image Feature Vector"**
4. A list of available models will appear
5. Browse the State of the Art website (https://paperswithcode.com/) to have a brief idea on which is the most popular model and how they perform
6. We picked **EfficientNetB0** and **ResNetV1** model to compare the efficiency.



With Transfer Learning, a solid model can be built with comparatively little training data because the model is already **pre-trained**

Binary/Multi-Class
- 750 training images per class
- 250 testing images per class

Transfer Learning
(Experiment with 10% of the 750)
- 75 training images per class
- 25 testing images per class
- 25 validation images per class

Using **Bing Image Scraper** instead of **Google Image Scraper**
Reason:
- Need only 75 images this time
- Need images that is large enough for feature extraction to run effectively

Running 20 classes of food (this time include 10 classes of fruits) on both **ResNetV1** and **EfficientNetB0**

Model	Epoch	Time(Sec)	Total Params
ResNet	5	~270	23,605,780
Training Loss	**Training Accuracy**	**Validation Loss**	**Validation Accuracy**
0.1253	0.9887	0.5683	0.8200

Model	Epoch	Time(Sec)	Total Params
EfficientNet	5	~326	4,075,184
Training Loss	**Training Accuracy**	**Validation Loss**	**Validation Accuracy**
0.1953	0.9807	0.4187	0.8940

Now we seen the power of transfer learning. With just 5 epochs, EfficientNet has beaten our multi-class by reaching nearly 90% acc. and only 4M params(MVP 4 accomplished)

Web app on localhost

WHY?

Need to test if it works locally before deploying to web server

Distributor of data-science packages and simplification

-Need to set up a separate environment for streamlit application
-Aim to keep it small enough to make it operational

(Not recommended to share the same environment with the image scraping/CNN modelling as porting to HTTP can be a challenge with all the dependencies)

streamlit	
tensorboard	Tensorflow's visualization toolkit
tensorboard-data-server	Data server for tensorboard
tensorboard-plugin-wit	
tensorflow	Tensorflow is a machine learning library.
tensorflow-hub	A library for transfer learning by reusing parts of tensorflow models.
tensorflow-io-gcs-filesystem	

Streamlit - Python framework for building web app

- Works with **Tensorflow, Keras, Matplotlib** etc
- Using **Spyder** as default IDE for this operation because of the lightweight compared with Pycharm
- run() function is the main body for running application
- Using streamlit widget easily by using **st.(component name).(variable name)**
- everything else can be done with external functions

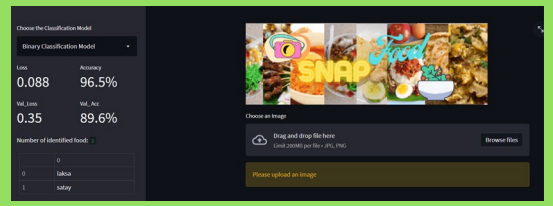
```
def run():
    st.image("https://github.com/DSstore/AIP/raw/main/snapfood.gif")
    app_mode = st.sidebar.selectbox("Choose the Classification Model",
        ["Binary Classification Model", "Multi-Class Classification Model", "EfficientNet Model"])


    if app_mode == "Binary Classification Model":
        col1, col2 = st.sidebar.columns(2)
        col1.metric("Loss", "0.088")
        col2.metric("Accuracy", "96.5%")
        col3, col4 = st.sidebar.columns(2)
        col3.metric("Val_Loss", "0.35")
        col4.metric("Val_ Acc", "89.6%")
```

HTTP: PORT 8501

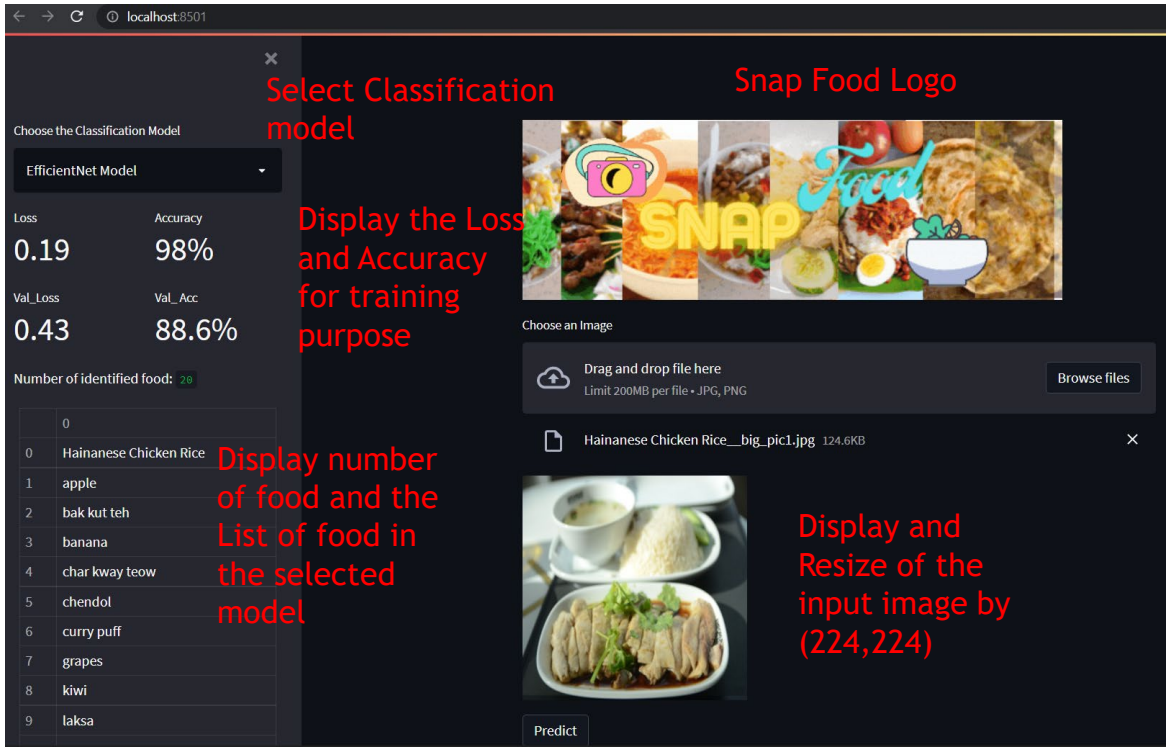
Default Port for Localhost

```
streamlit run your_script.py
```

- Run the above command in the default terminal
- A popup will appear in the default browser under localhost:8501
- For any amend of codes, just have to rerun this page



Introduction of Web app on localhost



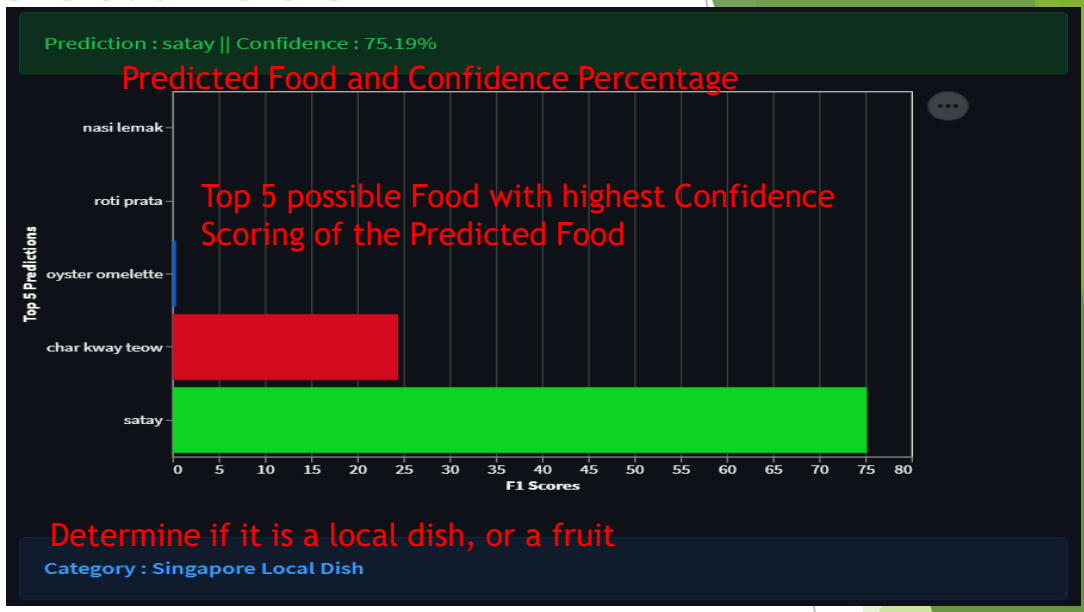
Select Classification model

Snap Food Logo

Display the Loss and Accuracy for training purpose

Display number of food and the List of food in the selected model

Display and Resize of the input image by (224,224)



Predicted Food and Confidence Percentage

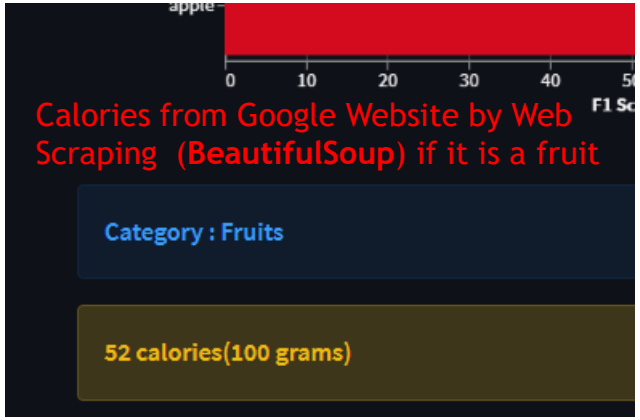
Top 5 possible Food with highest Confidence Scoring of the Predicted Food

Determine if it is a local dish, or a fruit

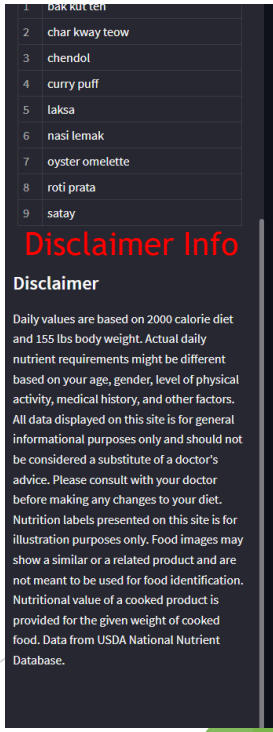
MVP 3

- Deploy the trained model to **localhost** through web app
- To be able to **upload images**
- Based on the food image, categorize the food type and provide a general **nutritional value**

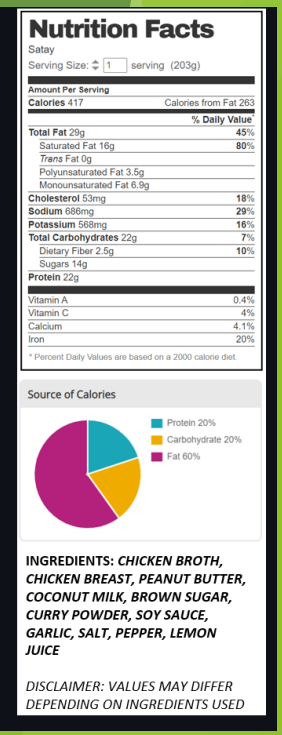
With this, we have met our goals for MvP 3, now to our last MvP 5



Calories from Google Website by Web Scraping (**BeautifulSoup**) if it is a fruit



Disclaimer Info



A general list Nutrition Facts

Breakdown of Calories Info

List of ingredients

Problems on deployment

Initial Plan for web deployment

Models and local host ok

Model pass to Google Cloud

Host Model on AI platform

Create service account to access model

Wrap app with docker container

Push and deploy container on Google Cloud Engine

Monitor app

Our initial plan is to pass the model to Google Cloud and make use of Google AI for Web deployment.

However, we were not able to link the service account to model despite trying for a few days.

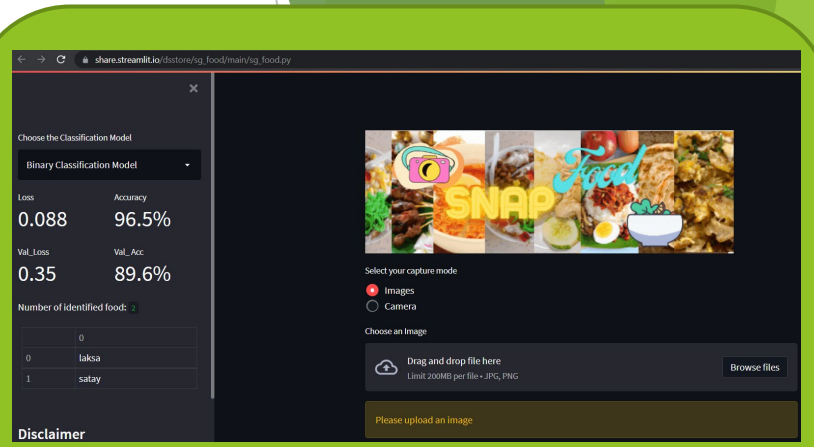
Alternative Solution

Streamlit actually has cloud for deployment

Place the main.py files and the models into GithubHub

```
altair==4.2.0
keras==2.7.0
Keras-Preprocessing==1.1.2
Pillow==8.4.0
protobuf==3.14.0
requests==2.25.1
scikit-image==0.18.3
streamlit
tables==3.6.1
tensorflow==2.7.0
tensorflow-io-gcs-filesystem==0.24.0
beautifulsoup4
tensorflow_hub
```

Trial and error to formulate on a smallest list of requirement into text file

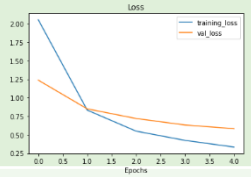
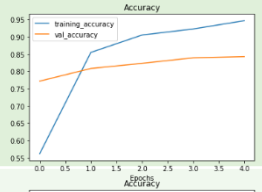
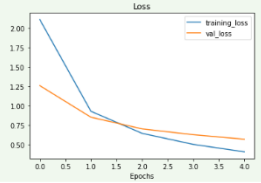
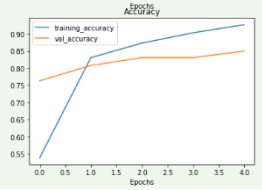
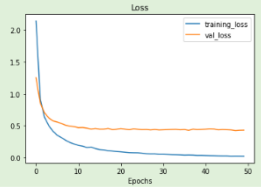
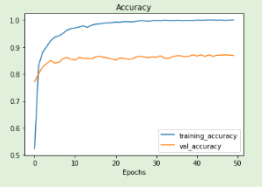


A web address will be given and all operations will be run remotely on streamlit server now

Once linking to Github repository done, streamlit will try to build the web app according to the requirements

35 classes of food

EfficientNet for 35 food classes

Model	Epoch	Time (sec)	Total Params	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy	Action Taken	Loss Graph	Accuracy Graph	Remarks
Model 1	5	510	4,049,564	0.3334	0.9463	0.5811	0.8423	- feature_extraction layer -Output_layer (Dense)			Model 1 already fulfil the requirement but we will to improve it
Model 2	5	590	4,049,564	0.4068	0.9272	0.5699	0.8503	Data Augmentation -rotation-range=45 -shear & zoom=0.2 -horizontal-flip=True			Adding data augmentation improves our validation as usual
Model 3	50	~5700	4,049,564	0.0224	1.00	0.4299	0.8686	50 epochs; -Added Callback for EarlyStopping, Checkpointer and reduceLR			We tried going 50 epochs and using EarlyStopping callbacks. We are able to achieve 86% accuracy.

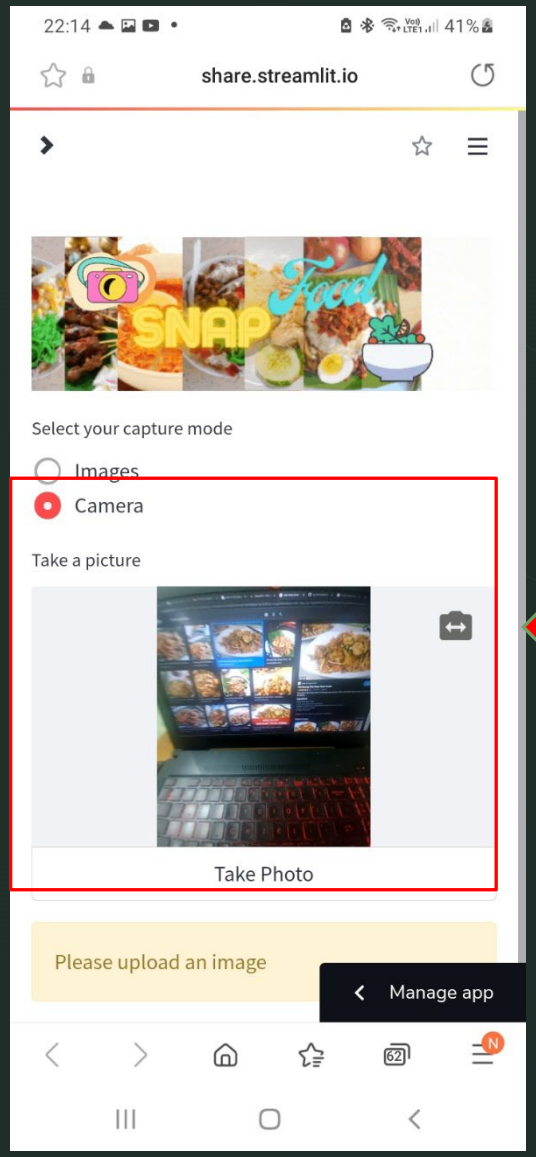
25 Local Food

hainanese chicken rice	claypot rice	mee rebus	roti john
bak chor mee	curry puff	mee siam	roti prata
bak kut teh	fish head curry	nasi briyani	satay
ban mian soup	fried carrot cake	nasi lemak	wanton mee
char kway teow	ice kacang	oyster omelette	
chendol	laksa	popiah	
chicken curry noodle	mee hoon kueh	prawn noodles soup	

10 Fruits

Apple	Banana
Grapes	Kiwi
Mango	Orange
Pear	Pineapple
Pomegranate	Watermelon

Applicable on Phone



Since we amended the size of the application through multiple tries, it is finally small enough to be fit onto the phone.

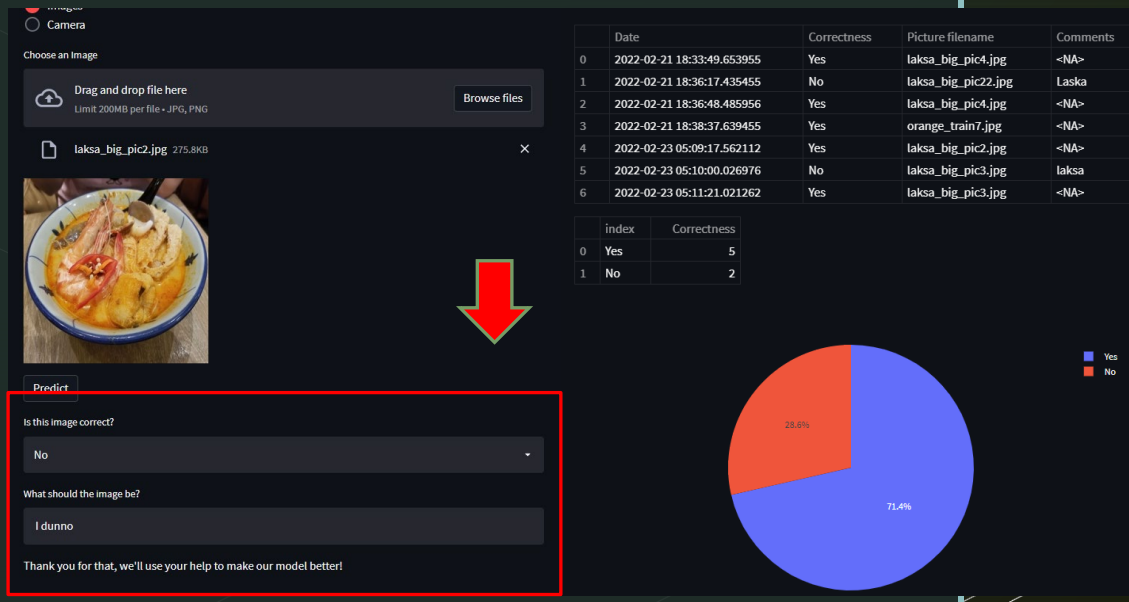
So we added a camera function that allows to work on a phone or a laptop.

With this, we finally completed most of our last MVP

MVP 5

- Deploy the web app onto a public web server
- Expand the image database to even more images of local food (**total 35 classes**)
- Making app able to work on phone
- Implement a user feedback system to point out any inaccurate prediction and improve the overall accuracy

The last part of implementing a user feedback is a **work in progress** which will be shown in live demo



LIVE DEMO

Web application
https://share.streamlit.io/dsstore/sg_food/main/sg_food.py
Backup application
https://share.streamlit.io/lightgamers/food-application/main/sg_food.py

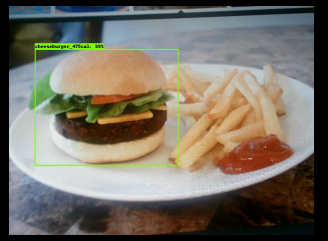
PROJECT LEARNING

CONSTRAINTS, CHALLENGES & MITIGATIONS

Constraints 	Challenges 	Mitigations 
• When the image contains more than 1 class of food, the output result will not be ideal. • Huge dataset will require higher processing power and will lead to slower output of end results. • Different lighting setup when taking the picture of the food. • The output result could not show the **ACTUAL** nutritional information of the image taken, as what we have is just a standard nutritional information.	• Collecting images of similar food but of different culture (E.g. Comparing laksa and Penang laksa) • Images collected from the different source are not of the correct file types. • Data collection of more food types for future updates and training of model.	• We will let the users understand that they can only upload the image of a single food item at one time. • Utilizing cloud based services for additional computing power in the future (as a paid service). • Sourcing for Nutritional Information of dishes with as many variations as possible in the future (E.g. Assam laksa and Penang laksa). • Ensure that we have the correct file types for images collected (.png/.jpg) in all future enhancements.

FUTURE ENHANCEMENTS

Current version 1.0: Identification of top 25 local dishes plus 10 common fruits from images



Version 2.0: Increase image identification to top 100 local dishes

Version 3.0: Identify multiple classes of food in a single image (Food Detection)

Version 4.0: Make the process of adding food classes to database through user feedback more streamlined. (Example: If enough images is collected from the users, we can push it to modelling and to the application with a few clicks)

Version 5.0: Personalized diet plan customized based on user's demography, activity level, medical condition, health and fitness objective

Version 6.0: Addition of a calorie count calculator API
(E.g. Https://caloriecontrol.Org/healthy-weight-tool-kit/get-moving-calculator/)

Version 7.0: Incorporate into mobile app (using A/B testing to gather feedback), so that consumer will have a better experience when taking photos of the food

REVENUE MODEL

Phase 1

- Focus on acquiring user, expanding market share and increase brand awareness, hence will not be seeking any meaningful revenue source.
- Seeking capital from Venture Capitalists and/or government agencies like Health Promotion Board(HPB)

Phase 2 (>10k active users/members)

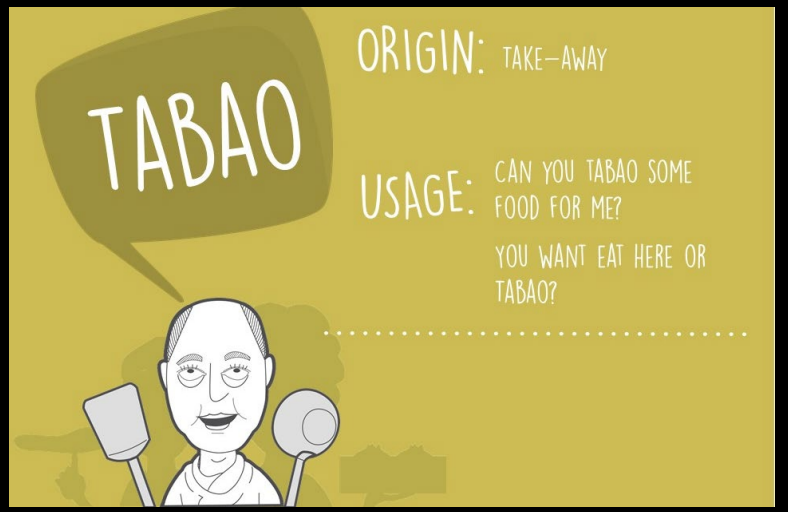
- Advertising revenue from lifestyle and fitness adjacent companies

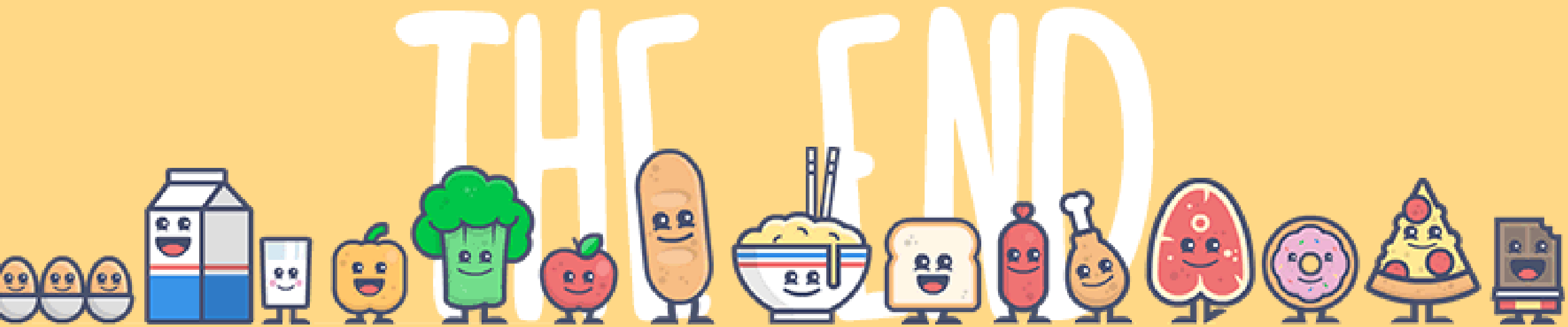
Phase 3 (>20k active users/members)

- Advertising revenue
- Subscription revenue from members for premium services

TAKEAWAYS

- Power of AI – AI makes it possible for machine to learn from experience and the data provided, to perform human-like tasks
- AI requires a large amount of data to learn from so as to give better result
- If AI is taught properly with enough training, it would have a lower error rate than humans
- Allow us to have a symbiotic relationship with AI while keeping in mind that the implementation of ethics is crucial for AI systems




THE END

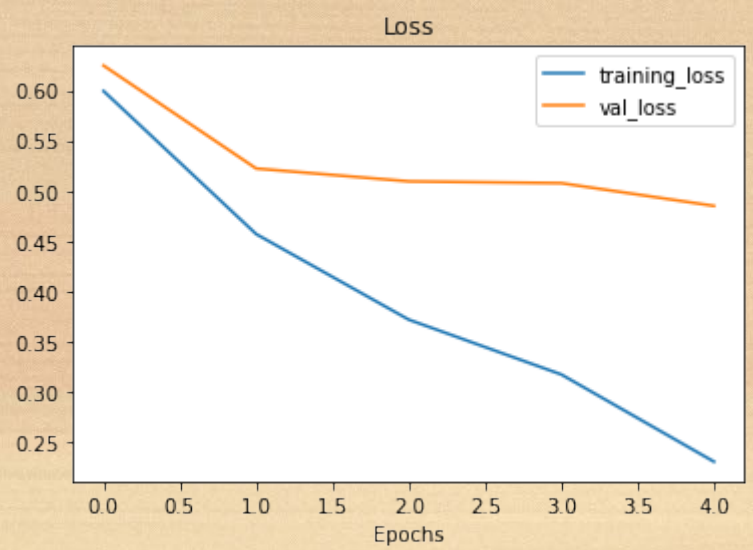
Appendix 1 – Binary Classification Test cases (1/6)

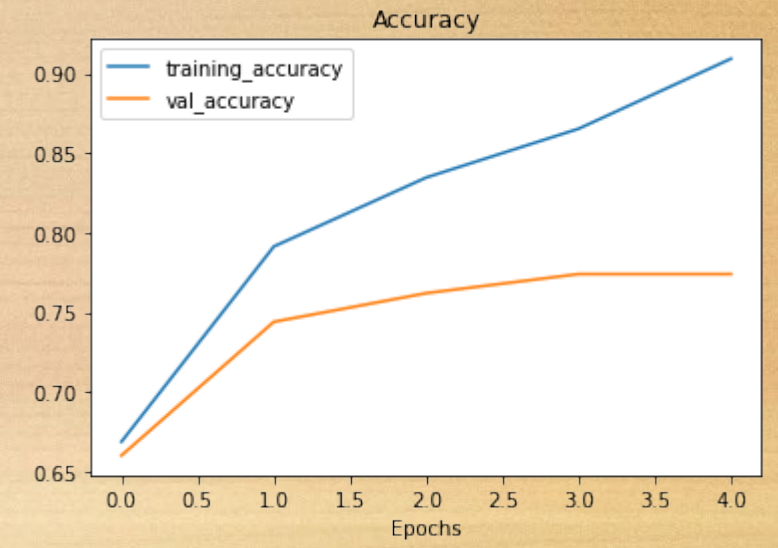
Model 1

```
Epoch 1/5
47/47 [==============================] - 16s 148ms/step - loss: 0.5991 - accuracy: 0.6687 - val_loss: 0.6244 - val_accuracy: 0.6600
Epoch 2/5
47/47 [==============================] - 9s 200ms/step - loss: 0.4570 - accuracy: 0.7913 - val_loss: 0.5222 - val_accuracy: 0.7440
Epoch 3/5
47/47 [==============================] - 6s 136ms/step - loss: 0.3721 - accuracy: 0.8347 - val_loss: 0.5094 - val_accuracy: 0.7620
Epoch 4/5
47/47 [==============================] - 6s 136ms/step - loss: 0.3176 - accuracy: 0.8653 - val_loss: 0.5077 - val_accuracy: 0.7740
Epoch 5/5
47/47 [==============================] - 6s 136ms/step - loss: 0.2310 - accuracy: 0.9093 - val_loss: 0.4851 - val_accuracy: 0.7740
```

```
model_1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 10)      280

 conv2d_1 (Conv2D)           (None, 220, 220, 10)      910

 max_pooling2d (MaxPooling2D  (None, 110, 110, 10)     0
 )

 conv2d_2 (Conv2D)           (None, 108, 108, 10)      910

 conv2d_3 (Conv2D)           (None, 106, 106, 10)      910

 max_pooling2d_1 (MaxPooling  (None, 53, 53, 10)       0
 2D)

 flatten (Flatten)           (None, 28090)             0

 dense (Dense)               (None, 1)                 28091

=================================================================
Total params: 31,101
Trainable params: 31,101
Non-trainable params: 0
_________________________________________________________________
```





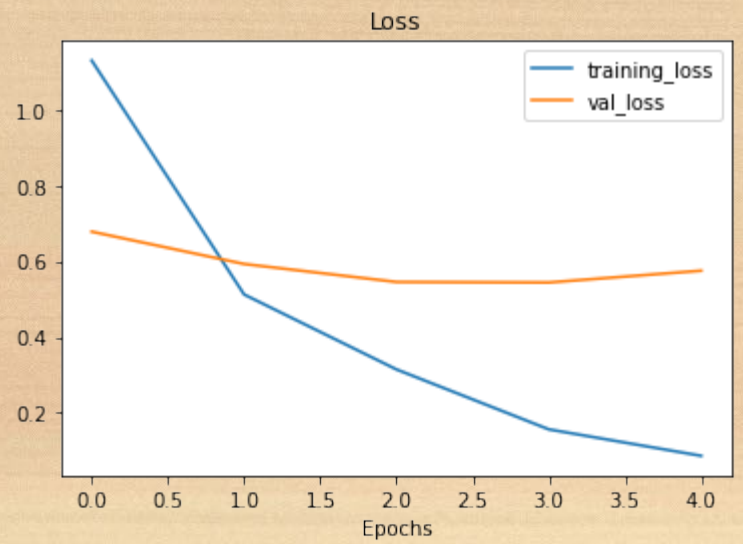
Appendix 1 – Binary Classification Test cases (2/6)

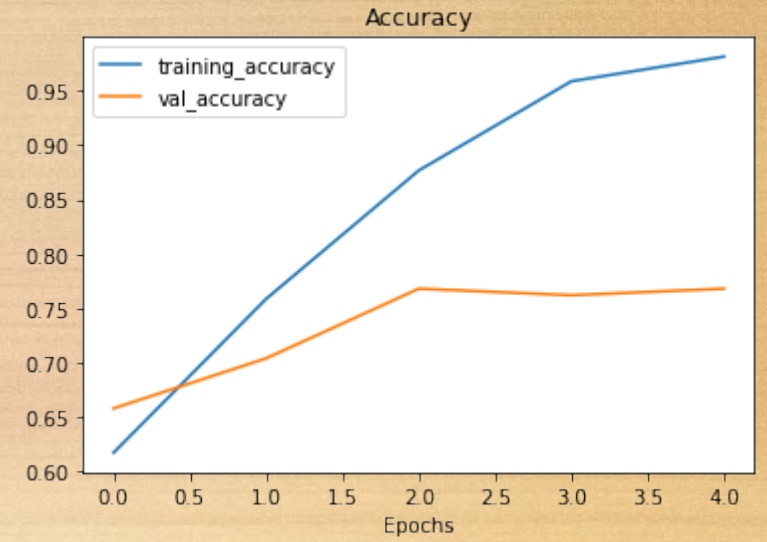
Model 2

```
Epoch 1/5
47/47 [==============================] - 7s 139ms/step - loss: 1.1297 - accuracy: 0.6173 - val_loss: 0.6782 - val_accuracy: 0.6580
Epoch 2/5
47/47 [==============================] - 6s 134ms/step - loss: 0.5124 - accuracy: 0.7587 - val_loss: 0.5929 - val_accuracy: 0.7040
Epoch 3/5
47/47 [==============================] - 6s 138ms/step - loss: 0.3147 - accuracy: 0.8767 - val_loss: 0.5454 - val_accuracy: 0.7680
Epoch 4/5
47/47 [==============================] - 6s 135ms/step - loss: 0.1565 - accuracy: 0.9587 - val_loss: 0.5441 - val_accuracy: 0.7620
Epoch 5/5
47/47 [==============================] - 6s 137ms/step - loss: 0.0867 - accuracy: 0.9813 - val_loss: 0.5751 - val_accuracy: 0.7680
```

```
model_2.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_7 (Conv2D)           (None, 222, 222, 10)      280

 conv2d_8 (Conv2D)           (None, 220, 220, 10)      910

 conv2d_9 (Conv2D)           (None, 218, 218, 10)      910

 flatten_2 (Flatten)         (None, 475240)            0

 dense_2 (Dense)             (None, 1)                 475241

=================================================================
Total params: 477,341
Trainable params: 477,341
Non-trainable params: 0
_________________________________________________________________
```





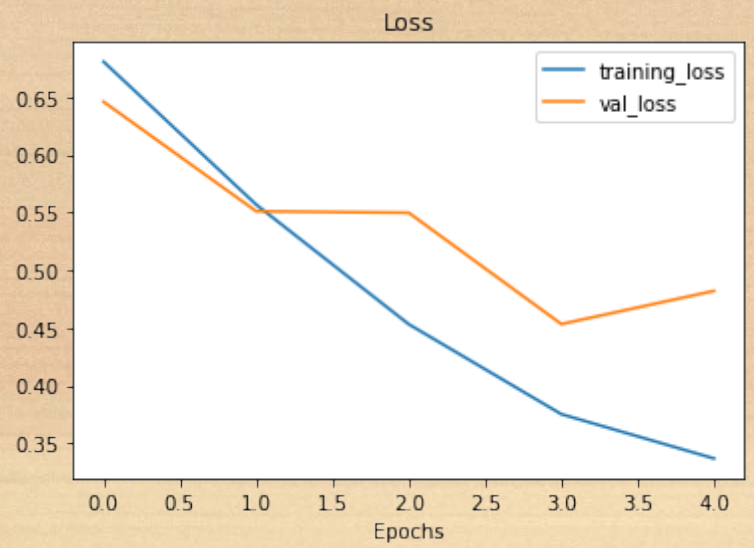
Appendix 1 – Binary Classification Test cases (3/6)

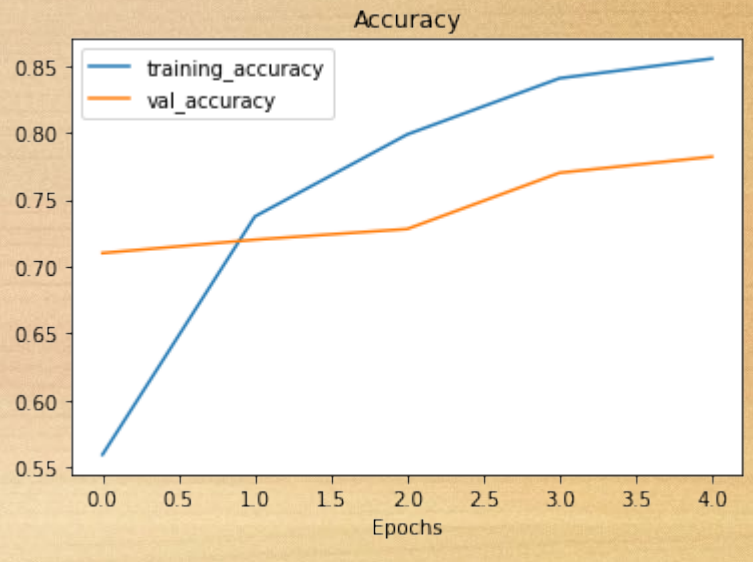
Model 3

```
Epoch 1/5
47/47 [==============================] - 10s 210ms/step - loss: 0.6806 - accuracy: 0.5593 - val_loss: 0.6458 - val_accuracy: 0.7100
Epoch 2/5
47/47 [==============================] - 12s 251ms/step - loss: 0.5567 - accuracy: 0.7373 - val_loss: 0.5509 - val_accuracy: 0.7200
Epoch 3/5
47/47 [==============================] - 6s 131ms/step - loss: 0.4530 - accuracy: 0.7987 - val_loss: 0.5496 - val_accuracy: 0.7280
Epoch 4/5
47/47 [==============================] - 6s 133ms/step - loss: 0.3751 - accuracy: 0.8407 - val_loss: 0.4532 - val_accuracy: 0.7700
Epoch 5/5
47/47 [==============================] - 6s 129ms/step - loss: 0.3367 - accuracy: 0.8553 - val_loss: 0.4819 - val_accuracy: 0.7820
```

```
model_3.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_10 (Conv2D)          (None, 222, 222, 10)      280

 max_pooling2d_2 (MaxPooling  (None, 111, 111, 10)     0
 2D)

 conv2d_11 (Conv2D)          (None, 109, 109, 10)      910

 max_pooling2d_3 (MaxPooling  (None, 54, 54, 10)       0
 2D)

 conv2d_12 (Conv2D)          (None, 52, 52, 10)        910

 max_pooling2d_4 (MaxPooling  (None, 26, 26, 10)       0
 2D)

 flatten_3 (Flatten)         (None, 6760)              0

 dense_3 (Dense)             (None, 1)                 6761

=================================================================
Total params: 8,861
Trainable params: 8,861
Non-trainable params: 0
_________________________________________________________________
```



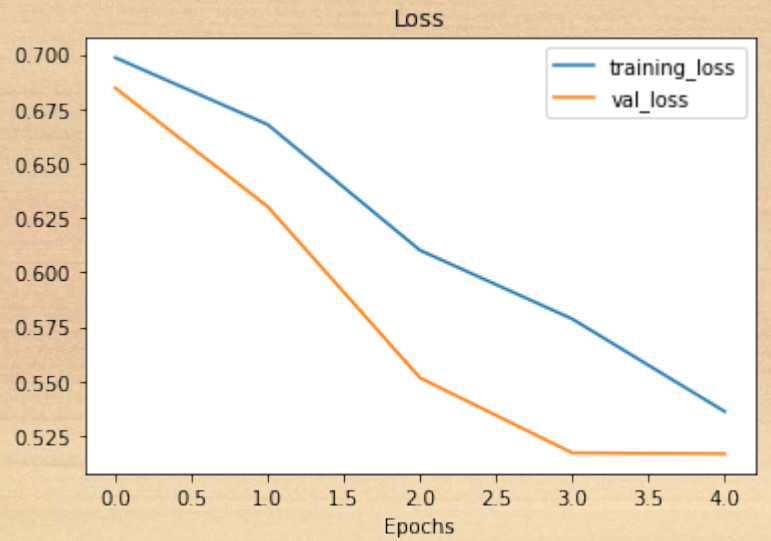


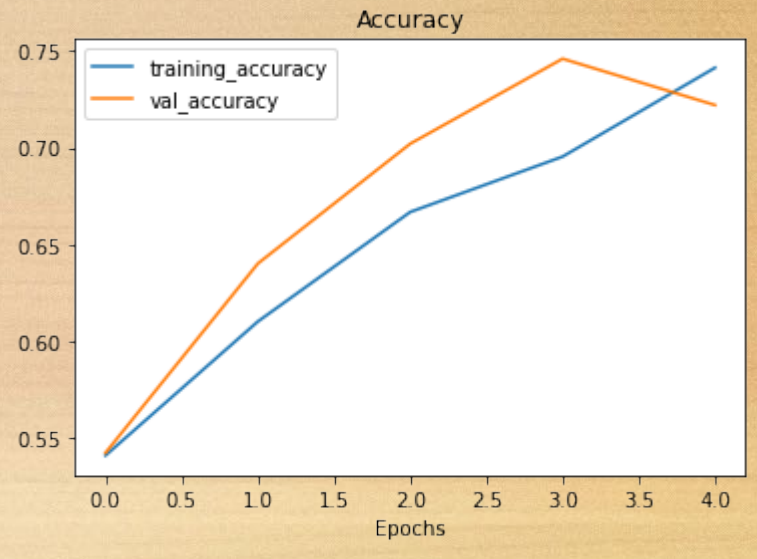
Appendix 1 – Binary Classification Test cases (4/6)

Model 4

```
Epoch 1/5
47/47 [==============================] - 19s 400ms/step - loss: 0.6936 - accuracy: 0.5773 - val_loss: 0.6684 - val_accuracy: 0.5580
Epoch 2/5
47/47 [==============================] - 19s 398ms/step - loss: 0.6559 - accuracy: 0.5967 - val_loss: 0.6071 - val_accuracy: 0.6620
Epoch 3/5
47/47 [==============================] - 19s 401ms/step - loss: 0.5644 - accuracy: 0.7107 - val_loss: 0.5269 - val_accuracy: 0.7220
Epoch 4/5
47/47 [==============================] - 18s 393ms/step - loss: 0.5379 - accuracy: 0.7240 - val_loss: 0.4905 - val_accuracy: 0.7600
Epoch 5/5
47/47 [==============================] - 18s 394ms/step - loss: 0.4828 - accuracy: 0.7727 - val_loss: 0.5246 - val_accuracy: 0.7260
```

```
model_4.summary()

Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 222, 222, 10)      280

 max_pooling2d_3 (MaxPooling  (None, 111, 111, 10)     0
 2D)

 conv2d_4 (Conv2D)           (None, 109, 109, 10)      910

 max_pooling2d_4 (MaxPooling  (None, 54, 54, 10)       0
 2D)

 conv2d_5 (Conv2D)           (None, 52, 52, 10)        910

 max_pooling2d_5 (MaxPooling  (None, 26, 26, 10)       0
 2D)

 flatten_1 (Flatten)         (None, 6760)              0

 dense_1 (Dense)             (None, 1)                 6761

=================================================================
Total params: 8,861
Trainable params: 8,861
Non-trainable params: 0
_________________________________________________________________
```



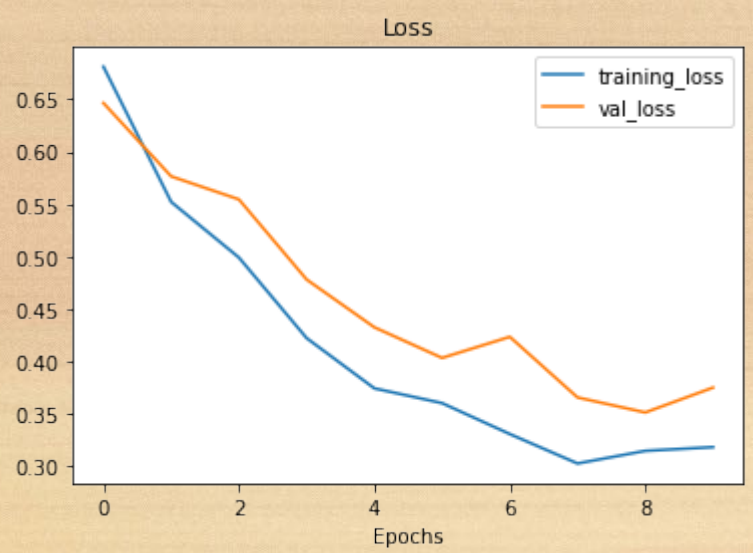


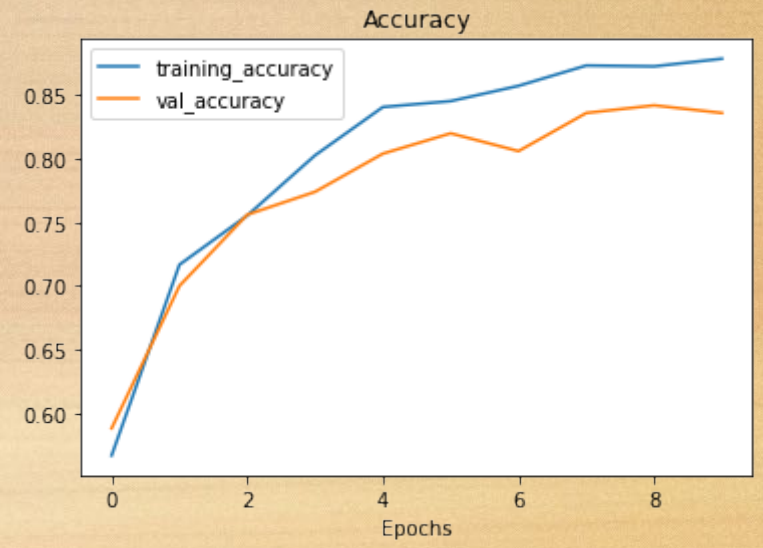
Appendix 1 – Binary Classification Test cases (5/6)

Model 5

```
Epoch 1/10
47/47 [==============================] - 22s 453ms/step - loss: 0.6810 - accuracy: 0.5667 - val_loss: 0.6464 - val_accuracy: 0.5880
Epoch 2/10
47/47 [==============================] - 21s 445ms/step - loss: 0.5522 - accuracy: 0.7167 - val_loss: 0.5765 - val_accuracy: 0.7000
Epoch 3/10
47/47 [==============================] - 21s 446ms/step - loss: 0.4992 - accuracy: 0.7553 - val_loss: 0.5546 - val_accuracy: 0.7560
Epoch 4/10
47/47 [==============================] - 21s 442ms/step - loss: 0.4224 - accuracy: 0.8027 - val_loss: 0.4782 - val_accuracy: 0.7740
Epoch 5/10
47/47 [==============================] - 21s 447ms/step - loss: 0.3745 - accuracy: 0.8407 - val_loss: 0.4327 - val_accuracy: 0.8040
Epoch 6/10
47/47 [==============================] - 21s 446ms/step - loss: 0.3604 - accuracy: 0.8453 - val_loss: 0.4035 - val_accuracy: 0.8200
Epoch 7/10
47/47 [==============================] - 21s 446ms/step - loss: 0.3309 - accuracy: 0.8573 - val_loss: 0.4236 - val_accuracy: 0.8060
Epoch 8/10
47/47 [==============================] - 21s 447ms/step - loss: 0.3029 - accuracy: 0.8733 - val_loss: 0.3658 - val_accuracy: 0.8360
Epoch 9/10
47/47 [==============================] - 21s 449ms/step - loss: 0.3149 - accuracy: 0.8727 - val_loss: 0.3515 - val_accuracy: 0.8420
Epoch 10/10
47/47 [==============================] - 21s 446ms/step - loss: 0.3184 - accuracy: 0.8787 - val_loss: 0.3752 - val_accuracy: 0.8360
```

```
model_5.summary()

Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_16 (Conv2D)          (None, 222, 222, 16)      448

 max_pooling2d_8 (MaxPooling  (None, 111, 111, 16)     0
 2D)

 conv2d_17 (Conv2D)          (None, 109, 109, 32)      4640

 max_pooling2d_9 (MaxPooling  (None, 54, 54, 32)       0
 2D)

 conv2d_18 (Conv2D)          (None, 52, 52, 64)        18496

 max_pooling2d_10 (MaxPoolin  (None, 26, 26, 64)       0
 g2D)

 conv2d_19 (Conv2D)          (None, 24, 24, 128)       73856

 max_pooling2d_11 (MaxPoolin  (None, 12, 12, 128)      0
 g2D)

 flatten_5 (Flatten)         (None, 18432)             0

 dense_5 (Dense)             (None, 1)                 18433

=================================================================
Total params: 115,873
Trainable params: 115,873
Non-trainable params: 0
_________________________________________________________________
```



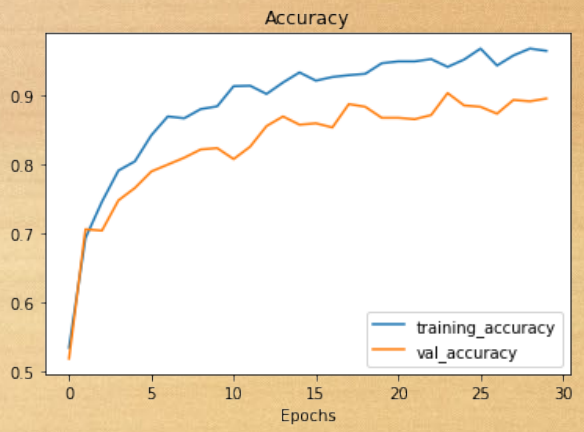


Appendix 1 – Binary Classification Test cases (6/6) Model 6

```
Epoch 1/30
47/47 [==============================] - 22s 451ms/step - loss: 0.6939 - accuracy: 0.5340 - val_loss: 0.6725 - val_accuracy: 0.5180
Epoch 2/30
47/47 [==============================] - 21s 447ms/step - loss: 0.5856 - accuracy: 0.6933 - val_loss: 0.5580 - val_accuracy: 0.7060
Epoch 3/30
47/47 [==============================] - 21s 442ms/step - loss: 0.5063 - accuracy: 0.7460 - val_loss: 0.5355 - val_accuracy: 0.7040
Epoch 4/30
47/47 [==============================] - 21s 446ms/step - loss: 0.4437 - accuracy: 0.7913 - val_loss: 0.4997 - val_accuracy: 0.7480
Epoch 5/30
47/47 [==============================] - 21s 444ms/step - loss: 0.3994 - accuracy: 0.8047 - val_loss: 0.4776 - val_accuracy: 0.7660
Epoch 6/30
47/47 [==============================] - 21s 445ms/step - loss: 0.3503 - accuracy: 0.8427 - val_loss: 0.4082 - val_accuracy: 0.7900
Epoch 7/30
47/47 [==============================] - 21s 443ms/step - loss: 0.3161 - accuracy: 0.8700 - val_loss: 0.4370 - val_accuracy: 0.8000
Epoch 8/30
47/47 [==============================] - 21s 444ms/step - loss: 0.3122 - accuracy: 0.8673 - val_loss: 0.3936 - val_accuracy: 0.8100
Epoch 9/30
47/47 [==============================] - 21s 442ms/step - loss: 0.2779 - accuracy: 0.8807 - val_loss: 0.3812 - val_accuracy: 0.8220
Epoch 10/30
47/47 [==============================] - 21s 445ms/step - loss: 0.2769 - accuracy: 0.8847 - val_loss: 0.3746 - val_accuracy: 0.8240
Epoch 11/30
47/47 [==============================] - 21s 443ms/step - loss: 0.2236 - accuracy: 0.9140 - val_loss: 0.4510 - val_accuracy: 0.8080
Epoch 12/30
47/47 [==============================] - 21s 440ms/step - loss: 0.2067 - accuracy: 0.9147 - val_loss: 0.4248 - val_accuracy: 0.8260
Epoch 13/30
47/47 [==============================] - 21s 439ms/step - loss: 0.2318 - accuracy: 0.9027 - val_loss: 0.3184 - val_accuracy: 0.8560
Epoch 14/30
47/47 [==============================] - 21s 442ms/step - loss: 0.1972 - accuracy: 0.9193 - val_loss: 0.3220 - val_accuracy: 0.8700
Epoch 15/30
47/47 [==============================] - 21s 440ms/step - loss: 0.1689 - accuracy: 0.9340 - val_loss: 0.3386 - val_accuracy: 0.8580
Epoch 16/30
47/47 [==============================] - 21s 439ms/step - loss: 0.2018 - accuracy: 0.9220 - val_loss: 0.3183 - val_accuracy: 0.8600
Epoch 17/30
47/47 [==============================] - 21s 444ms/step - loss: 0.1856 - accuracy: 0.9273 - val_loss: 0.3171 - val_accuracy: 0.8540
Epoch 18/30
47/47 [==============================] - 21s 439ms/step - loss: 0.1713 - accuracy: 0.9300 - val_loss: 0.3061 - val_accuracy: 0.8880
Epoch 19/30
47/47 [==============================] - 21s 439ms/step - loss: 0.1582 - accuracy: 0.9320 - val_loss: 0.2762 - val_accuracy: 0.8840
Epoch 20/30
47/47 [==============================] - 21s 442ms/step - loss: 0.1322 - accuracy: 0.9473 - val_loss: 0.3218 - val_accuracy: 0.8680
Epoch 21/30
47/47 [==============================] - 21s 444ms/step - loss: 0.1375 - accuracy: 0.9500 - val_loss: 0.3758 - val_accuracy: 0.8680
Epoch 22/30
47/47 [==============================] - 21s 443ms/step - loss: 0.1305 - accuracy: 0.9500 - val_loss: 0.2906 - val_accuracy: 0.8660
Epoch 23/30
47/47 [==============================] - 21s 437ms/step - loss: 0.1207 - accuracy: 0.9533 - val_loss: 0.3671 - val_accuracy: 0.8720
Epoch 24/30
47/47 [==============================] - 21s 439ms/step - loss: 0.1464 - accuracy: 0.9420 - val_loss: 0.2594 - val_accuracy: 0.9040
Epoch 25/30
47/47 [==============================] - 21s 443ms/step - loss: 0.1223 - accuracy: 0.9527 - val_loss: 0.3019 - val_accuracy: 0.8860
Epoch 26/30
47/47 [==============================] - 21s 437ms/step - loss: 0.0923 - accuracy: 0.9687 - val_loss: 0.3333 - val_accuracy: 0.8840
Epoch 27/30
47/47 [==============================] - 21s 438ms/step - loss: 0.1375 - accuracy: 0.9440 - val_loss: 0.3652 - val_accuracy: 0.8740
Epoch 28/30
47/47 [==============================] - 21s 441ms/step - loss: 0.1046 - accuracy: 0.9587 - val_loss: 0.3382 - val_accuracy: 0.8940
Epoch 29/30
47/47 [==============================] - 20s 435ms/step - loss: 0.0871 - accuracy: 0.9687 - val_loss: 0.2976 - val_accuracy: 0.8920
Epoch 30/30
47/47 [==============================] - 21s 438ms/step - loss: 0.0885 - accuracy: 0.9653 - val_loss: 0.3559 - val_accuracy: 0.8960
```

```
model_6.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_25 (Conv2D)          (None, 222, 222, 16)      448

 max_pooling2d_17 (MaxPoolin  (None, 111, 111, 16)     0
 g2D)

 conv2d_26 (Conv2D)          (None, 109, 109, 32)      4640

 max_pooling2d_18 (MaxPoolin  (None, 54, 54, 32)       0
 g2D)

 conv2d_27 (Conv2D)          (None, 52, 52, 64)        18496

 max_pooling2d_19 (MaxPoolin  (None, 26, 26, 64)       0
 g2D)

 conv2d_28 (Conv2D)          (None, 24, 24, 128)       73856

 max_pooling2d_20 (MaxPoolin  (None, 12, 12, 128)      0
 g2D)

 conv2d_29 (Conv2D)          (None, 10, 10, 256)       295168

 max_pooling2d_21 (MaxPoolin  (None, 5, 5, 256)        0
 g2D)

 flatten_7 (Flatten)         (None, 6400)              0

 dense_7 (Dense)             (None, 1)                 6401

=================================================================
Total params: 399,009
Trainable params: 399,009
Non-trainable params: 0
_________________________________________________________________
```



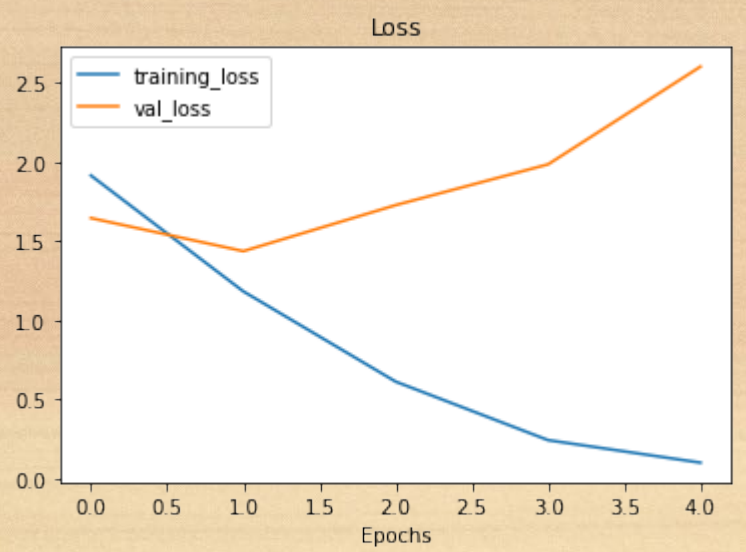
Appendix 2 – Multi-Class Classification Test cases (1/7)

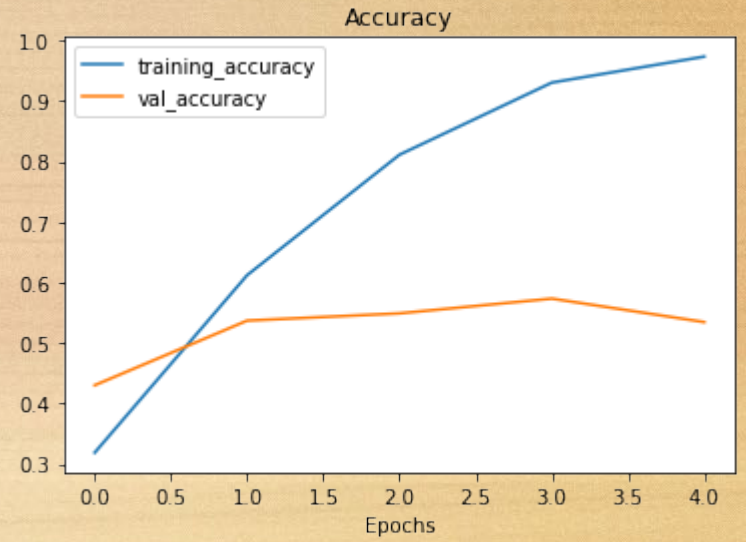
Model 1

```
Epoch 1/5
235/235 [==============================] - 44s 144ms/step - loss: 1.9457 - accuracy: 0.2981 - val_loss: 1.6967 - val_accuracy: 0.4032
Epoch 2/5
235/235 [==============================] - 33s 140ms/step - loss: 1.2313 - accuracy: 0.5928 - val_loss: 1.4616 - val_accuracy: 0.5176
Epoch 3/5
235/235 [==============================] - 33s 139ms/step - loss: 0.6352 - accuracy: 0.8025 - val_loss: 1.4772 - val_accuracy: 0.5728
Epoch 4/5
235/235 [==============================] - 33s 140ms/step - loss: 0.2448 - accuracy: 0.9264 - val_loss: 1.7247 - val_accuracy: 0.5764
Epoch 5/5
235/235 [==============================] - 33s 141ms/step - loss: 0.0867 - accuracy: 0.9793 - val_loss: 2.0259 - val_accuracy: 0.5736
```

```
model_1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 10)      280

 conv2d_1 (Conv2D)           (None, 220, 220, 10)      910

 max_pooling2d (MaxPooling2D  (None, 110, 110, 10)     0
 )

 conv2d_2 (Conv2D)           (None, 108, 108, 10)      910

 conv2d_3 (Conv2D)           (None, 106, 106, 10)      910

 max_pooling2d_1 (MaxPooling  (None, 53, 53, 10)       0
 2D)

 flatten (Flatten)           (None, 28090)             0

 dense (Dense)               (None, 10)                280910

=================================================================
Total params: 283,920
Trainable params: 283,920
Non-trainable params: 0
_________________________________________________________________
```





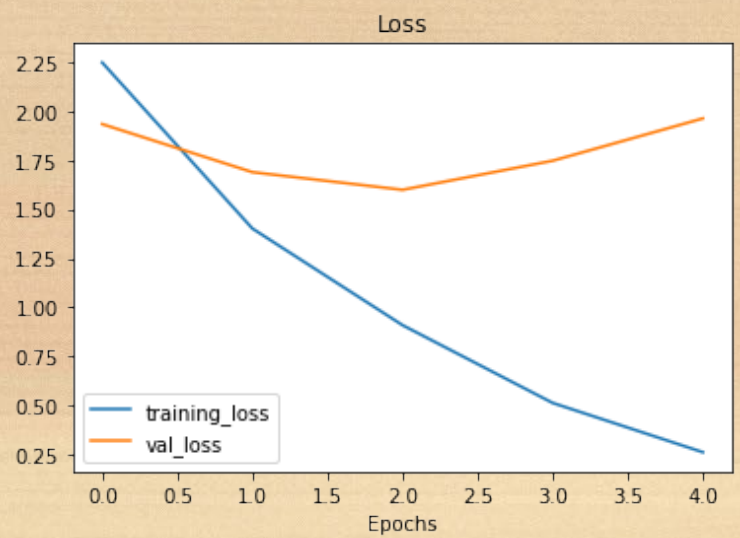
Appendix 2 – Multi-Class Classification Test cases (2/7)

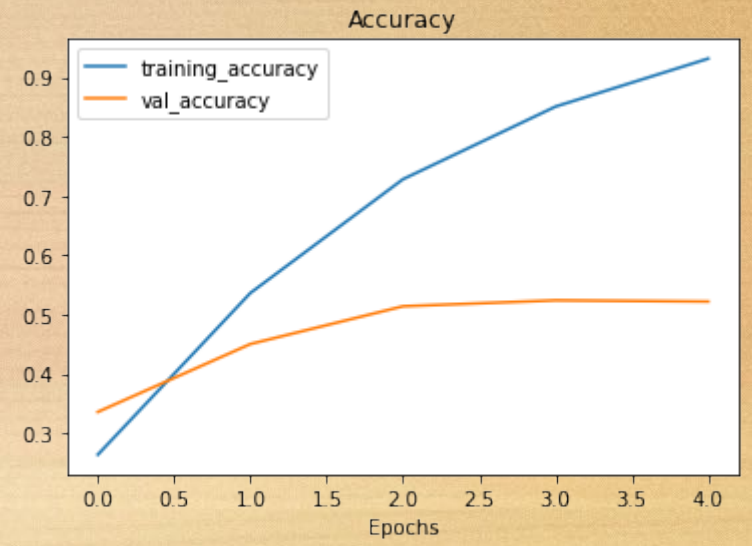
Model 2

```
Epoch 1/5
235/235 [==============================] - 33s 139ms/step - loss: 1.7927 - accuracy: 0.3805 - val_loss: 1.6063 - val_accuracy: 0.4528
Epoch 2/5
235/235 [==============================] - 33s 139ms/step - loss: 1.0828 - accuracy: 0.6505 - val_loss: 1.4303 - val_accuracy: 0.5384
Epoch 3/5
235/235 [==============================] - 32s 136ms/step - loss: 0.6521 - accuracy: 0.8047 - val_loss: 1.3706 - val_accuracy: 0.5648
Epoch 4/5
235/235 [==============================] - 32s 136ms/step - loss: 0.3701 - accuracy: 0.8963 - val_loss: 1.6121 - val_accuracy: 0.5592
Epoch 5/5
235/235 [==============================] - 32s 135ms/step - loss: 0.2174 - accuracy: 0.9425 - val_loss: 1.5935 - val_accuracy: 0.5804
```

```
model_2.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 10)      280

 max_pooling2d (MaxPooling2D  (None, 111, 111, 10)     0
 )

 conv2d_1 (Conv2D)           (None, 109, 109, 10)      910

 max_pooling2d_1 (MaxPooling  (None, 54, 54, 10)       0
 2D)

 flatten (Flatten)           (None, 29160)             0

 dense (Dense)               (None, 10)                291610

=================================================================
Total params: 292,800
Trainable params: 292,800
Non-trainable params: 0
```





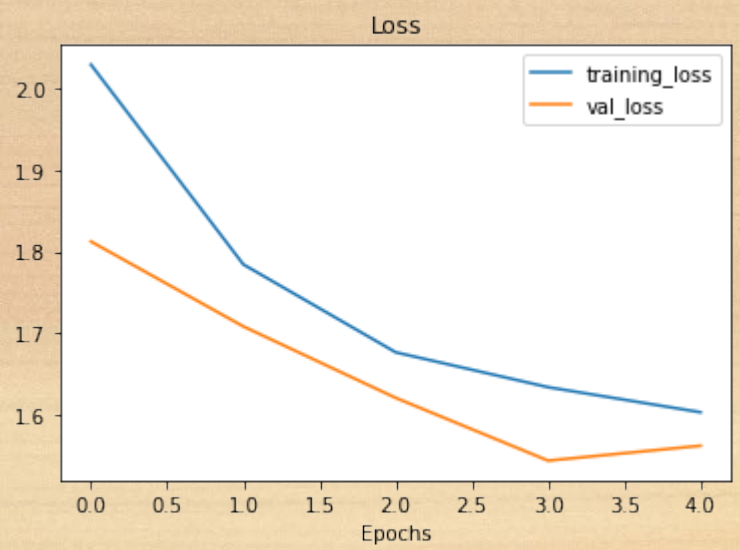
Appendix 2 – Multi-Class Classification Test cases (3/7)

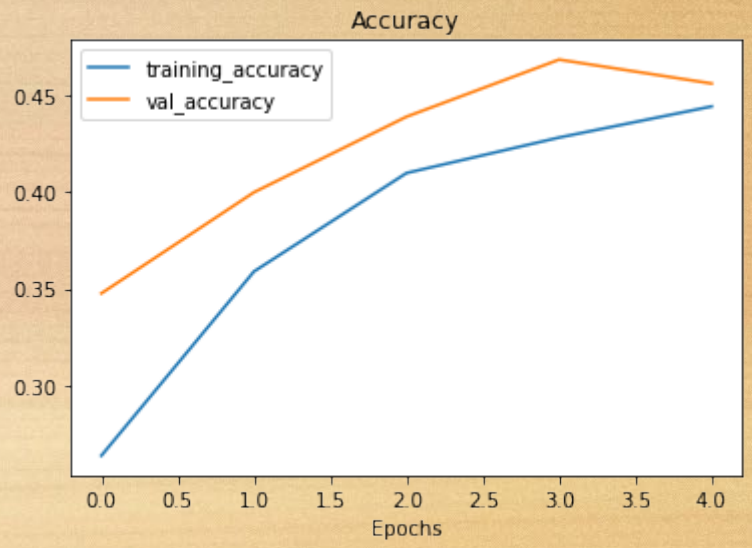
Model 3

```
Epoch 1/5
235/235 [==============================] - 108s 457ms/step - loss: 2.0690 - accuracy: 0.2519 - val_loss: 1.8383 - val_accuracy: 0.3348
Epoch 2/5
235/235 [==============================] - 107s 455ms/step - loss: 1.8106 - accuracy: 0.3532 - val_loss: 1.6659 - val_accuracy: 0.4144
Epoch 3/5
235/235 [==============================] - 105s 447ms/step - loss: 1.7027 - accuracy: 0.3993 - val_loss: 1.5714 - val_accuracy: 0.4412
Epoch 4/5
235/235 [==============================] - 105s 448ms/step - loss: 1.6296 - accuracy: 0.4280 - val_loss: 1.6453 - val_accuracy: 0.4472
Epoch 5/5
235/235 [==============================] - 105s 446ms/step - loss: 1.5927 - accuracy: 0.4419 - val_loss: 1.4698 - val_accuracy: 0.4920
```

```
model_3.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 222, 222, 10)      280

 max_pooling2d_2 (MaxPooling  (None, 111, 111, 10)     0
 2D)

 conv2d_5 (Conv2D)           (None, 109, 109, 10)      910

 max_pooling2d_3 (MaxPooling  (None, 54, 54, 10)       0
 2D)

 flatten_1 (Flatten)         (None, 29160)             0

 dense_1 (Dense)             (None, 10)                291610

=================================================================
Total params: 292,800
Trainable params: 292,800
Non-trainable params: 0
_________________________________________________________________
```



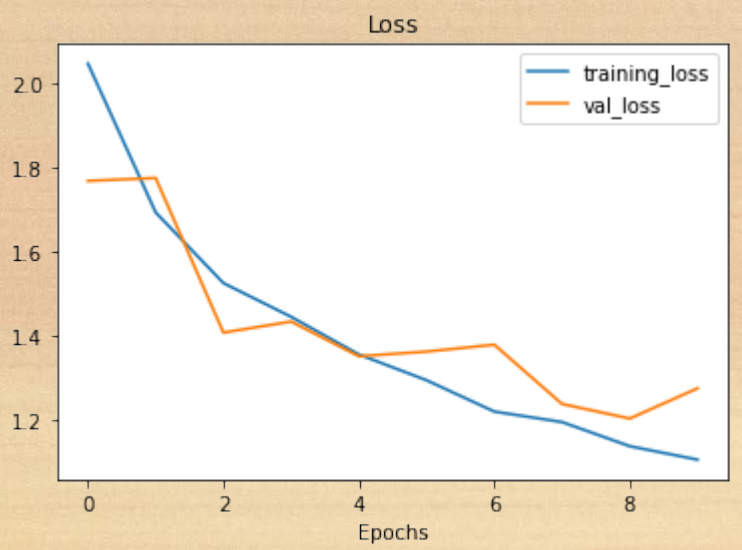


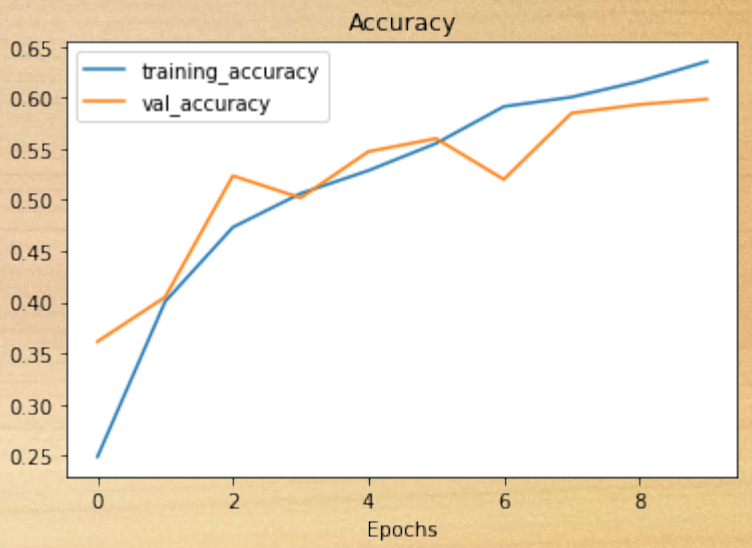
Appendix 2 – Multi-Class Classification Test cases (4/7) Model 4

```
Epoch 1/10
235/235 [==============================] - 109s 461ms/step - loss: 2.0453 - accuracy: 0.2491 - val_loss: 1.7674 - val_accuracy: 0.3616
Epoch 2/10
235/235 [==============================] - 108s 459ms/step - loss: 1.6927 - accuracy: 0.4008 - val_loss: 1.7742 - val_accuracy: 0.4052
Epoch 3/10
235/235 [==============================] - 108s 459ms/step - loss: 1.5253 - accuracy: 0.4733 - val_loss: 1.4079 - val_accuracy: 0.5236
Epoch 4/10
235/235 [==============================] - 108s 459ms/step - loss: 1.4453 - accuracy: 0.5063 - val_loss: 1.4341 - val_accuracy: 0.5020
Epoch 5/10
235/235 [==============================] - 108s 459ms/step - loss: 1.3561 - accuracy: 0.5287 - val_loss: 1.3518 - val_accuracy: 0.5472
Epoch 6/10
235/235 [==============================] - 107s 457ms/step - loss: 1.2950 - accuracy: 0.5552 - val_loss: 1.3630 - val_accuracy: 0.5600
Epoch 7/10
235/235 [==============================] - 106s 452ms/step - loss: 1.2208 - accuracy: 0.5912 - val_loss: 1.3793 - val_accuracy: 0.5200
Epoch 8/10
235/235 [==============================] - 106s 450ms/step - loss: 1.1960 - accuracy: 0.6004 - val_loss: 1.2389 - val_accuracy: 0.5848
Epoch 9/10
235/235 [==============================] - 106s 452ms/step - loss: 1.1387 - accuracy: 0.6157 - val_loss: 1.2043 - val_accuracy: 0.5932
Epoch 10/10
235/235 [==============================] - 107s 453ms/step - loss: 1.1069 - accuracy: 0.6352 - val_loss: 1.2759 - val_accuracy: 0.5984
```

```
model_4.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_10 (Conv2D)          (None, 222, 222, 32)      896

 max_pooling2d_6 (MaxPooling  (None, 111, 111, 32)     0
 2D)

 conv2d_11 (Conv2D)          (None, 109, 109, 32)      9248

 max_pooling2d_7 (MaxPooling  (None, 54, 54, 32)       0
 2D)

 conv2d_12 (Conv2D)          (None, 52, 52, 32)        9248

 max_pooling2d_8 (MaxPooling  (None, 26, 26, 32)       0
 2D)

 conv2d_13 (Conv2D)          (None, 24, 24, 32)        9248

 max_pooling2d_9 (MaxPooling  (None, 12, 12, 32)       0
 2D)

 flatten_3 (Flatten)         (None, 4608)              0

 dense_3 (Dense)             (None, 10)                46090

=================================================================
Total params: 74,730
Trainable params: 74,730
Non-trainable params: 0
_________________________________________________________________
```





Appendix 2 – Multi-Class Classification Test cases (5/7) Model 5

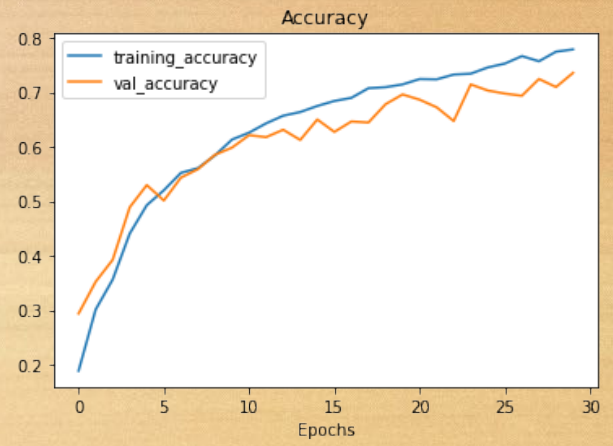
```
Epoch 1/30
235/235 [==============================] - 117s 497ms/step - loss: 2.1587 - accuracy: 0.1885 - val_loss: 1.9572 - val_accuracy: 0.2936
Epoch 2/30
235/235 [==============================] - 117s 496ms/step - loss: 1.9329 - accuracy: 0.3009 - val_loss: 1.8206 - val_accuracy: 0.3520
Epoch 3/30
235/235 [==============================] - 115s 489ms/step - loss: 1.7880 - accuracy: 0.3561 - val_loss: 1.7348 - val_accuracy: 0.3924
Epoch 4/30
235/235 [==============================] - 115s 489ms/step - loss: 1.5789 - accuracy: 0.4405 - val_loss: 1.4770 - val_accuracy: 0.4888
Epoch 5/30
235/235 [==============================] - 115s 488ms/step - loss: 1.4660 - accuracy: 0.4927 - val_loss: 1.3463 - val_accuracy: 0.5296
Epoch 6/30
235/235 [==============================] - 115s 489ms/step - loss: 1.3912 - accuracy: 0.5200 - val_loss: 1.5128 - val_accuracy: 0.5012
Epoch 7/30
235/235 [==============================] - 114s 486ms/step - loss: 1.3172 - accuracy: 0.5520 - val_loss: 1.3530 - val_accuracy: 0.5432
Epoch 8/30
235/235 [==============================] - 114s 486ms/step - loss: 1.2670 - accuracy: 0.5607 - val_loss: 1.2630 - val_accuracy: 0.5588
Epoch 9/30
235/235 [==============================] - 115s 488ms/step - loss: 1.2223 - accuracy: 0.5841 - val_loss: 1.2113 - val_accuracy: 0.5852
Epoch 10/30
235/235 [==============================] - 115s 490ms/step - loss: 1.1469 - accuracy: 0.6132 - val_loss: 1.1673 - val_accuracy: 0.5984
Epoch 11/30
235/235 [==============================] - 115s 490ms/step - loss: 1.1078 - accuracy: 0.6259 - val_loss: 1.1309 - val_accuracy: 0.6212
Epoch 12/30
235/235 [==============================] - 114s 487ms/step - loss: 1.0719 - accuracy: 0.6429 - val_loss: 1.1854 - val_accuracy: 0.6176
Epoch 13/30
235/235 [==============================] - 114s 484ms/step - loss: 1.0192 - accuracy: 0.6569 - val_loss: 1.1184 - val_accuracy: 0.6312
Epoch 14/30
235/235 [==============================] - 113s 480ms/step - loss: 0.9928 - accuracy: 0.6637 - val_loss: 1.1424 - val_accuracy: 0.6124
Epoch 15/30
235/235 [==============================] - 113s 482ms/step - loss: 0.9744 - accuracy: 0.6748 - val_loss: 1.0992 - val_accuracy: 0.6500
Epoch 16/30
235/235 [==============================] - 114s 485ms/step - loss: 0.9348 - accuracy: 0.6840 - val_loss: 1.0836 - val_accuracy: 0.6272
Epoch 17/30
235/235 [==============================] - 114s 486ms/step - loss: 0.9217 - accuracy: 0.6897 - val_loss: 1.0433 - val_accuracy: 0.6464
Epoch 18/30
235/235 [==============================] - 113s 481ms/step - loss: 0.8804 - accuracy: 0.7075 - val_loss: 1.1219 - val_accuracy: 0.6444
Epoch 19/30
235/235 [==============================] - 113s 482ms/step - loss: 0.8742 - accuracy: 0.7093 - val_loss: 0.9540 - val_accuracy: 0.6780
Epoch 20/30
235/235 [==============================] - 113s 483ms/step - loss: 0.8484 - accuracy: 0.7144 - val_loss: 0.9454 - val_accuracy: 0.6960
Epoch 21/30
235/235 [==============================] - 113s 481ms/step - loss: 0.8195 - accuracy: 0.7243 - val_loss: 0.9394 - val_accuracy: 0.6864
Epoch 22/30
235/235 [==============================] - 113s 482ms/step - loss: 0.8161 - accuracy: 0.7237 - val_loss: 1.0360 - val_accuracy: 0.6724
Epoch 23/30
235/235 [==============================] - 115s 487ms/step - loss: 0.8036 - accuracy: 0.7325 - val_loss: 1.1704 - val_accuracy: 0.6472
Epoch 24/30
235/235 [==============================] - 115s 488ms/step - loss: 0.7903 - accuracy: 0.7341 - val_loss: 0.8734 - val_accuracy: 0.7144
Epoch 25/30
235/235 [==============================] - 115s 490ms/step - loss: 0.7543 - accuracy: 0.7456 - val_loss: 0.9199 - val_accuracy: 0.7032
Epoch 26/30
235/235 [==============================] - 115s 488ms/step - loss: 0.7473 - accuracy: 0.7527 - val_loss: 0.9407 - val_accuracy: 0.6976
Epoch 27/30
235/235 [==============================] - 115s 487ms/step - loss: 0.7129 - accuracy: 0.7664 - val_loss: 0.9610 - val_accuracy: 0.6936
Epoch 28/30
235/235 [==============================] - 115s 488ms/step - loss: 0.7370 - accuracy: 0.7569 - val_loss: 0.8789 - val_accuracy: 0.7244
Epoch 29/30
235/235 [==============================] - 115s 489ms/step - loss: 0.6943 - accuracy: 0.7745 - val_loss: 0.9089 - val_accuracy: 0.7096
Epoch 30/30
235/235 [==============================] - 114s 487ms/step - loss: 0.6700 - accuracy: 0.7787 - val_loss: 0.8375 - val_accuracy: 0.7356
```

```
model_5.summary()
```

Model: "sequential_8"

Layer (type)	Output Shape	Param #
conv2d_30 (Conv2D)	(None, 222, 222, 200)	5600
max_pooling2d_26 (MaxPooling2D)	(None, 111, 111, 200)	0
conv2d_31 (Conv2D)	(None, 109, 109, 100)	180100
max_pooling2d_27 (MaxPooling2D)	(None, 54, 54, 100)	0
conv2d_32 (Conv2D)	(None, 52, 52, 100)	90100
max_pooling2d_28 (MaxPooling2D)	(None, 26, 26, 100)	0
conv2d_33 (Conv2D)	(None, 24, 24, 50)	45050
max_pooling2d_29 (MaxPooling2D)	(None, 12, 12, 50)	0
flatten_8 (Flatten)	(None, 7200)	0
dense_8 (Dense)	(None, 10)	72010

Total params: 392,860
Trainable params: 392,860
Non-trainable params: 0



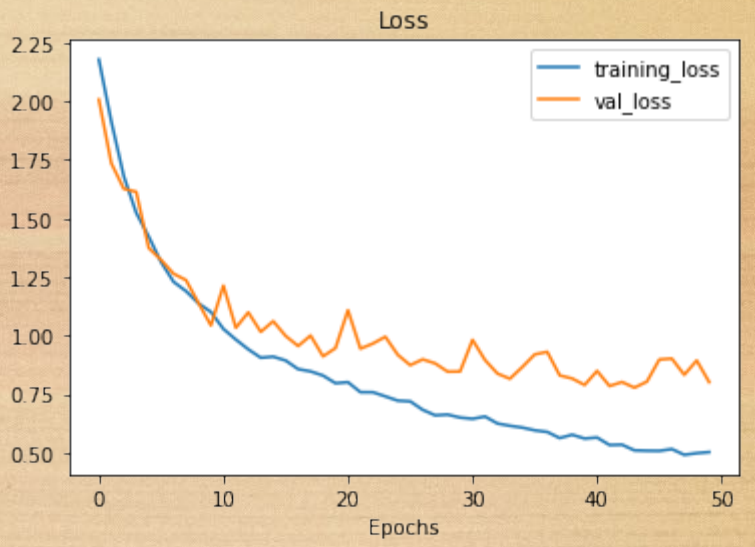
Appendix 2 – Multi-Class Classification Test cases (6/7) Model 6 Part 1

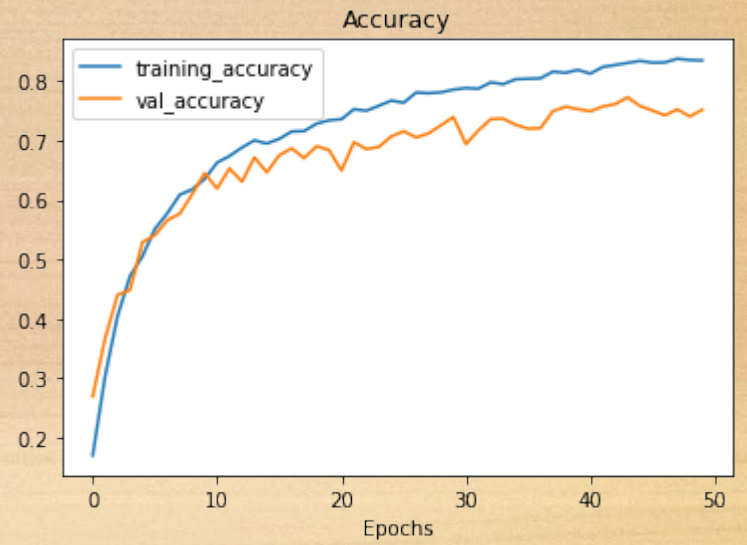
```
Epoch 1/50
235/235 [==============================] - 116s 489ms/step - loss: 2.1784 - accuracy: 0.1699 - val_loss: 2.0075 - val_accuracy: 0.2692
Epoch 2/50
235/235 [==============================] - 115s 488ms/step - loss: 1.9137 - accuracy: 0.3036 - val_loss: 1.7354 - val_accuracy: 0.3672
Epoch 3/50
235/235 [==============================] - 114s 487ms/step - loss: 1.6806 - accuracy: 0.4044 - val_loss: 1.6264 - val_accuracy: 0.4400
Epoch 4/50
235/235 [==============================] - 115s 489ms/step - loss: 1.5235 - accuracy: 0.4716 - val_loss: 1.6135 - val_accuracy: 0.4480
Epoch 5/50
235/235 [==============================] - 115s 489ms/step - loss: 1.4227 - accuracy: 0.5049 - val_loss: 1.3738 - val_accuracy: 0.5288
Epoch 6/50
235/235 [==============================] - 115s 489ms/step - loss: 1.3128 - accuracy: 0.5509 - val_loss: 1.3224 - val_accuracy: 0.5404
Epoch 7/50
235/235 [==============================] - 115s 490ms/step - loss: 1.2291 - accuracy: 0.5772 - val_loss: 1.2636 - val_accuracy: 0.5656
Epoch 8/50
235/235 [==============================] - 115s 490ms/step - loss: 1.1892 - accuracy: 0.6084 - val_loss: 1.2363 - val_accuracy: 0.5764
Epoch 9/50
235/235 [==============================] - 115s 488ms/step - loss: 1.1371 - accuracy: 0.6173 - val_loss: 1.1385 - val_accuracy: 0.6096
Epoch 10/50
235/235 [==============================] - 115s 488ms/step - loss: 1.1004 - accuracy: 0.6349 - val_loss: 1.0428 - val_accuracy: 0.6448
Epoch 11/50
235/235 [==============================] - 115s 487ms/step - loss: 1.0278 - accuracy: 0.6624 - val_loss: 1.2128 - val_accuracy: 0.6188
Epoch 12/50
235/235 [==============================] - 114s 487ms/step - loss: 0.9826 - accuracy: 0.6735 - val_loss: 1.0329 - val_accuracy: 0.6528
Epoch 13/50
235/235 [==============================] - 115s 488ms/step - loss: 0.9409 - accuracy: 0.6879 - val_loss: 1.0987 - val_accuracy: 0.6304
Epoch 14/50
235/235 [==============================] - 114s 486ms/step - loss: 0.9050 - accuracy: 0.6999 - val_loss: 1.0156 - val_accuracy: 0.6712
Epoch 15/50
235/235 [==============================] - 113s 479ms/step - loss: 0.9095 - accuracy: 0.6945 - val_loss: 1.0609 - val_accuracy: 0.6460
Epoch 16/50
235/235 [==============================] - 114s 483ms/step - loss: 0.8930 - accuracy: 0.7024 - val_loss: 0.9980 - val_accuracy: 0.6748
Epoch 17/50
235/235 [==============================] - 113s 480ms/step - loss: 0.8564 - accuracy: 0.7148 - val_loss: 0.9550 - val_accuracy: 0.6864
Epoch 18/50
235/235 [==============================] - 113s 481ms/step - loss: 0.8467 - accuracy: 0.7155 - val_loss: 0.9998 - val_accuracy: 0.6700
Epoch 19/50
235/235 [==============================] - 111s 474ms/step - loss: 0.8289 - accuracy: 0.7280 - val_loss: 0.9109 - val_accuracy: 0.6900
Epoch 20/50
235/235 [==============================] - 110s 469ms/step - loss: 0.7962 - accuracy: 0.7336 - val_loss: 0.9463 - val_accuracy: 0.6836
Epoch 21/50
235/235 [==============================] - 111s 472ms/step - loss: 0.8006 - accuracy: 0.7355 - val_loss: 1.1081 - val_accuracy: 0.6492
Epoch 22/50
235/235 [==============================] - 112s 475ms/step - loss: 0.7579 - accuracy: 0.7523 - val_loss: 0.9436 - val_accuracy: 0.6968
Epoch 23/50
235/235 [==============================] - 111s 473ms/step - loss: 0.7577 - accuracy: 0.7497 - val_loss: 0.9655 - val_accuracy: 0.6852
Epoch 24/50
235/235 [==============================] - 111s 473ms/step - loss: 0.7401 - accuracy: 0.7580 - val_loss: 0.9946 - val_accuracy: 0.6888
Epoch 25/50
235/235 [==============================] - 111s 474ms/step - loss: 0.7217 - accuracy: 0.7668 - val_loss: 0.9174 - val_accuracy: 0.7064
Epoch 26/50
235/235 [==============================] - 109s 464ms/step - loss: 0.7188 - accuracy: 0.7632 - val_loss: 0.8733 - val_accuracy: 0.7152
Epoch 27/50
235/235 [==============================] - 109s 463ms/step - loss: 0.6826 - accuracy: 0.7803 - val_loss: 0.8981 - val_accuracy: 0.7048
Epoch 28/50
235/235 [==============================] - 109s 464ms/step - loss: 0.6592 - accuracy: 0.7792 - val_loss: 0.8808 - val_accuracy: 0.7116
Epoch 29/50
235/235 [==============================] - 109s 464ms/step - loss: 0.6622 - accuracy: 0.7807 - val_loss: 0.8456 - val_accuracy: 0.7252
Epoch 30/50
235/235 [==============================] - 109s 462ms/step - loss: 0.6495 - accuracy: 0.7852 - val_loss: 0.8462 - val_accuracy: 0.7392
Epoch 31/50
235/235 [==============================] - 109s 464ms/step - loss: 0.6442 - accuracy: 0.7879 - val_loss: 0.9813 - val_accuracy: 0.6936
Epoch 32/50
235/235 [==============================] - 109s 463ms/step - loss: 0.6540 - accuracy: 0.7867 - val_loss: 0.8954 - val_accuracy: 0.7164
Epoch 33/50
235/235 [==============================] - 109s 462ms/step - loss: 0.6243 - accuracy: 0.7975 - val_loss: 0.8374 - val_accuracy: 0.7356
Epoch 34/50
235/235 [==============================] - 109s 463ms/step - loss: 0.6145 - accuracy: 0.7944 - val_loss: 0.8152 - val_accuracy: 0.7364
Epoch 35/50
235/235 [==============================] - 109s 462ms/step - loss: 0.6067 - accuracy: 0.8027 - val_loss: 0.8655 - val_accuracy: 0.7260
Epoch 36/50
235/235 [==============================] - 109s 465ms/step - loss: 0.5950 - accuracy: 0.8035 - val_loss: 0.9194 - val_accuracy: 0.7196
Epoch 37/50
235/235 [==============================] - 109s 463ms/step - loss: 0.5881 - accuracy: 0.8043 - val_loss: 0.9300 - val_accuracy: 0.7204
Epoch 38/50
235/235 [==============================] - 109s 463ms/step - loss: 0.5621 - accuracy: 0.8156 - val_loss: 0.8295 - val_accuracy: 0.7492
Epoch 39/50
235/235 [==============================] - 109s 464ms/step - loss: 0.5762 - accuracy: 0.8135 - val_loss: 0.8163 - val_accuracy: 0.7564
Epoch 40/50
235/235 [==============================] - 109s 464ms/step - loss: 0.5599 - accuracy: 0.8184 - val_loss: 0.7887 - val_accuracy: 0.7524
Epoch 41/50
235/235 [==============================] - 109s 461ms/step - loss: 0.5643 - accuracy: 0.8121 - val_loss: 0.8487 - val_accuracy: 0.7488
Epoch 42/50
235/235 [==============================] - 109s 463ms/step - loss: 0.5323 - accuracy: 0.8233 - val_loss: 0.7841 - val_accuracy: 0.7568
Epoch 43/50
235/235 [==============================] - 109s 464ms/step - loss: 0.5337 - accuracy: 0.8267 - val_loss: 0.8012 - val_accuracy: 0.7608
Epoch 44/50
235/235 [==============================] - 109s 463ms/step - loss: 0.5095 - accuracy: 0.8301 - val_loss: 0.7772 - val_accuracy: 0.7724
Epoch 45/50
235/235 [==============================] - 109s 462ms/step - loss: 0.5079 - accuracy: 0.8336 - val_loss: 0.8029 - val_accuracy: 0.7576
Epoch 46/50
235/235 [==============================] - 109s 463ms/step - loss: 0.5073 - accuracy: 0.8305 - val_loss: 0.8975 - val_accuracy: 0.7500
Epoch 47/50
235/235 [==============================] - 109s 462ms/step - loss: 0.5151 - accuracy: 0.8311 - val_loss: 0.9013 - val_accuracy: 0.7420
Epoch 48/50
235/235 [==============================] - 109s 464ms/step - loss: 0.4905 - accuracy: 0.8372 - val_loss: 0.8321 - val_accuracy: 0.7520
Epoch 49/50
235/235 [==============================] - 109s 465ms/step - loss: 0.4975 - accuracy: 0.8349 - val_loss: 0.8933 - val_accuracy: 0.7400
Epoch 50/50
235/235 [==============================] - 109s 466ms/step - loss: 0.5021 - accuracy: 0.8344 - val_loss: 0.8008 - val_accuracy: 0.7512
```

Appendix 2 – Multi-Class Classification Test cases (6/7) Model 6 Part 2

```
model_6.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_34 (Conv2D)          (None, 222, 222, 200)     5600

 max_pooling2d_30 (MaxPoolin  (None, 111, 111, 200)    0
 g2D)

 conv2d_35 (Conv2D)          (None, 109, 109, 100)     180100

 max_pooling2d_31 (MaxPoolin  (None, 54, 54, 100)      0
 g2D)

 conv2d_36 (Conv2D)          (None, 52, 52, 100)       90100

 max_pooling2d_32 (MaxPoolin  (None, 26, 26, 100)      0
 g2D)

 conv2d_37 (Conv2D)          (None, 24, 24, 50)        45050

 max_pooling2d_33 (MaxPoolin  (None, 12, 12, 50)       0
 g2D)

 flatten_9 (Flatten)         (None, 7200)              0

 dense_9 (Dense)             (None, 10)                72010

=================================================================
Total params: 392,860
Trainable params: 392,860
Non-trainable params: 0
_________________________________________________________________
```





Appendix 2 – Multi-Class Classification Test cases (7/7) Model 7 Part 1

```
Epoch 1/100
235/235 [==============================] - ETA: 0s - loss: 2.3961 - accuracy: 0.1025
Epoch 00001: val_loss improved from inf to 2.33791, saving model to Multi_classification.hdf5
235/235 [==============================] - 100s 390ms/step - loss: 2.3961 - accuracy: 0.1025 - val_loss: 2.3379 - val_accuracy: 0.1096 - lr: 0.0010
Epoch 2/100
235/235 [==============================] - ETA: 0s - loss: 2.3097 - accuracy: 0.1253
Epoch 00002: val_loss improved from 2.33791 to 2.29667, saving model to Multi_classification.hdf5
235/235 [==============================] - 90s 383ms/step - loss: 2.3097 - accuracy: 0.1253 - val_loss: 2.2967 - val_accuracy: 0.1184 - lr: 0.0010
Epoch 3/100
235/235 [==============================] - ETA: 0s - loss: 2.1824 - accuracy: 0.1773
Epoch 00003: val_loss improved from 2.29667 to 2.13017, saving model to Multi_classification.hdf5
235/235 [==============================] - 90s 383ms/step - loss: 2.1824 - accuracy: 0.1773 - val_loss: 2.1302 - val_accuracy: 0.2168 - lr: 0.0010
Epoch 4/100
235/235 [==============================] - ETA: 0s - loss: 2.1090 - accuracy: 0.2103
Epoch 00004: val_loss improved from 2.13017 to 2.09215, saving model to Multi_classification.hdf5
235/235 [==============================] - 90s 385ms/step - loss: 2.1090 - accuracy: 0.2103 - val_loss: 2.0921 - val_accuracy: 0.2144 - lr: 0.0010
Epoch 5/100
235/235 [==============================] - ETA: 0s - loss: 2.0656 - accuracy: 0.2193
Epoch 00005: val_loss improved from 2.09215 to 2.05835, saving model to Multi_classification.hdf5
235/235 [==============================] - 90s 384ms/step - loss: 2.0656 - accuracy: 0.2193 - val_loss: 2.0583 - val_accuracy: 0.2248 - lr: 0.0010
Epoch 6/100
235/235 [==============================] - ETA: 0s - loss: 2.0322 - accuracy: 0.2381
Epoch 00006: val_loss did not improve from 2.05835
235/235 [==============================] - 90s 383ms/step - loss: 2.0322 - accuracy: 0.2381 - val_loss: 2.0680 - val_accuracy: 0.2404 - lr: 0.0010
Epoch 7/100
235/235 [==============================] - ETA: 0s - loss: 1.9650 - accuracy: 0.2767
Epoch 00007: val_loss improved from 2.05835 to 1.91100, saving model to Multi_classification.hdf5
235/235 [==============================] - 90s 383ms/step - loss: 1.9650 - accuracy: 0.2767 - val_loss: 1.9110 - val_accuracy: 0.3128 - lr: 0.0010
Epoch 8/100
235/235 [==============================] - ETA: 0s - loss: 1.9194 - accuracy: 0.2916
Epoch 00008: val_loss did not improve from 1.91100
235/235 [==============================] - 90s 382ms/step - loss: 1.9194 - accuracy: 0.2916 - val_loss: 1.9271 - val_accuracy: 0.2908 - lr: 0.0010
Epoch 9/100
235/235 [==============================] - ETA: 0s - loss: 1.8315 - accuracy: 0.3160
Epoch 00009: val_loss improved from 1.91100 to 1.84493, saving model to Multi_classification.hdf5
235/235 [==============================] - 90s 382ms/step - loss: 1.8315 - accuracy: 0.3160 - val_loss: 1.8449 - val_accuracy: 0.3348 - lr: 0.0010
Epoch 10/100
235/235 [==============================] - ETA: 0s - loss: 1.7741 - accuracy: 0.3451
Epoch 00010: val_loss improved from 1.84493 to 1.73410, saving model to Multi_classification.hdf5
Epoch 11/100
235/235 [==============================] - ETA: 0s - loss: 1.6888 - accuracy: 0.3884
Epoch 00011: val_loss improved from 1.73410 to 1.66738, saving model to Multi_classification.hdf5
235/235 [==============================] - 89s 380ms/step - loss: 1.6888 - accuracy: 0.3884 - val_loss: 1.6674 - val_accuracy: 0.3852 - lr: 0.0010
Epoch 12/100
235/235 [==============================] - ETA: 0s - loss: 1.6310 - accuracy: 0.4024
Epoch 00012: val_loss improved from 1.66738 to 1.58931, saving model to Multi_classification.hdf5
235/235 [==============================] - 90s 382ms/step - loss: 1.6310 - accuracy: 0.4024 - val_loss: 1.5893 - val_accuracy: 0.4228 - lr: 0.0010
Epoch 13/100
235/235 [==============================] - ETA: 0s - loss: 1.5601 - accuracy: 0.4360
Epoch 00013: val_loss improved from 1.58931 to 1.58062, saving model to Multi_classification.hdf5
235/235 [==============================] - 89s 381ms/step - loss: 1.5601 - accuracy: 0.4360 - val_loss: 1.5806 - val_accuracy: 0.4380 - lr: 0.0010
Epoch 14/100
235/235 [==============================] - ETA: 0s - loss: 1.4676 - accuracy: 0.4795
Epoch 00014: val_loss improved from 1.58062 to 1.46581, saving model to Multi_classification.hdf5
235/235 [==============================] - 90s 382ms/step - loss: 1.4676 - accuracy: 0.4795 - val_loss: 1.4658 - val_accuracy: 0.4888 - lr: 0.0010
Epoch 15/100
235/235 [==============================] - ETA: 0s - loss: 1.3661 - accuracy: 0.5209
Epoch 00015: val_loss improved from 1.46581 to 1.39756, saving model to Multi_classification.hdf5
235/235 [==============================] - 90s 382ms/step - loss: 1.3661 - accuracy: 0.5209 - val_loss: 1.3976 - val_accuracy: 0.5112 - lr: 0.0010
Epoch 16/100
235/235 [==============================] - ETA: 0s - loss: 1.2970 - accuracy: 0.5521
Epoch 00016: val_loss did not improve from 1.39756
235/235 [==============================] - 90s 384ms/step - loss: 1.2970 - accuracy: 0.5521 - val_loss: 1.4633 - val_accuracy: 0.4884 - lr: 0.0010
Epoch 17/100
235/235 [==============================] - ETA: 0s - loss: 1.1820 - accuracy: 0.5983
Epoch 00017: val_loss improved from 1.39756 to 1.20500, saving model to Multi_classification.hdf5
235/235 [==============================] - 90s 383ms/step - loss: 1.1820 - accuracy: 0.5983 - val_loss: 1.2050 - val_accuracy: 0.5956 - lr: 0.0010
Epoch 18/100
235/235 [==============================] - ETA: 0s - loss: 1.1471 - accuracy: 0.6175
Epoch 00018: val_loss did not improve from 1.20500
235/235 [==============================] - 90s 382ms/step - loss: 1.1471 - accuracy: 0.6175 - val_loss: 1.2980 - val_accuracy: 0.5548 - lr: 0.0010
Epoch 19/100
235/235 [==============================] - ETA: 0s - loss: 1.0673 - accuracy: 0.6443
Epoch 00019: val_loss did not improve from 1.20500
235/235 [==============================] - 90s 383ms/step - loss: 1.0673 - accuracy: 0.6443 - val_loss: 1.2310 - val_accuracy: 0.5892 - lr: 0.0010
Epoch 20/100
235/235 [==============================] - ETA: 0s - loss: 1.0212 - accuracy: 0.6605
Epoch 00020: val_loss improved from 1.20500 to 1.15781, saving model to Multi_classification.hdf5
```

```
Epoch 21/100
235/235 [==============================] - ETA: 0s - loss: 0.9599 - accuracy: 0.6825
Epoch 00021: val_loss did not improve from 1.15781
235/235 [==============================] - 90s 383ms/step - loss: 0.9599 - accuracy: 0.6825 - val_loss: 1.1883 - val_accuracy: 0.6044 - lr: 0.0010
Epoch 22/100
235/235 [==============================] - ETA: 0s - loss: 0.9329 - accuracy: 0.6860
Epoch 00022: val_loss did not improve from 1.15781
235/235 [==============================] - 90s 382ms/step - loss: 0.9329 - accuracy: 0.6860 - val_loss: 1.1581 - val_accuracy: 0.6336 - lr: 0.0010
Epoch 23/100
235/235 [==============================] - ETA: 0s - loss: 0.8884 - accuracy: 0.7012
Epoch 00023: val_loss improved from 1.15781 to 1.08805, saving model to Multi_classification.hdf5
235/235 [==============================] - 90s 382ms/step - loss: 0.8884 - accuracy: 0.7012 - val_loss: 1.0881 - val_accuracy: 0.6484 - lr: 0.0010
Epoch 24/100
235/235 [==============================] - ETA: 0s - loss: 0.8470 - accuracy: 0.7220
Epoch 00024: val_loss improved from 1.08805 to 1.00324, saving model to Multi_classification.hdf5
235/235 [==============================] - 90s 382ms/step - loss: 0.8470 - accuracy: 0.7220 - val_loss: 1.0032 - val_accuracy: 0.6692 - lr: 0.0010
Epoch 25/100
235/235 [==============================] - ETA: 0s - loss: 0.8443 - accuracy: 0.7200
Epoch 00025: val_loss did not improve from 1.00324
235/235 [==============================] - 90s 382ms/step - loss: 0.8443 - accuracy: 0.7200 - val_loss: 1.0066 - val_accuracy: 0.6852 - lr: 0.0010
Epoch 26/100
235/235 [==============================] - ETA: 0s - loss: 0.7878 - accuracy: 0.7437
Epoch 00026: val_loss improved from 1.00324 to 1.00253, saving model to Multi_classification.hdf5
235/235 [==============================] - 90s 382ms/step - loss: 0.7878 - accuracy: 0.7437 - val_loss: 1.0025 - val_accuracy: 0.6788 - lr: 0.0010
Epoch 27/100
235/235 [==============================] - ETA: 0s - loss: 0.7598 - accuracy: 0.7492
Epoch 00027: val_loss improved from 1.00253 to 0.93924, saving model to Multi_classification.hdf5
235/235 [==============================] - 90s 381ms/step - loss: 0.7598 - accuracy: 0.7492 - val_loss: 0.9392 - val_accuracy: 0.6884 - lr: 0.0010
Epoch 28/100
235/235 [==============================] - ETA: 0s - loss: 0.7363 - accuracy: 0.7569
Epoch 00028: val_loss did not improve from 0.93924
235/235 [==============================] - 89s 379ms/step - loss: 0.7363 - accuracy: 0.7569 - val_loss: 0.9567 - val_accuracy: 0.7028 - lr: 0.0010
Epoch 29/100
235/235 [==============================] - ETA: 0s - loss: 0.7181 - accuracy: 0.7653
Epoch 00029: val_loss did not improve from 0.93924
235/235 [==============================] - 90s 381ms/step - loss: 0.7181 - accuracy: 0.7653 - val_loss: 1.0198 - val_accuracy: 0.6908 - lr: 0.0010
Epoch 30/100
235/235 [==============================] - ETA: 0s - loss: 0.7030 - accuracy: 0.7661
Epoch 31/100
235/235 [==============================] - ETA: 0s - loss: 0.6571 - accuracy: 0.7867
Epoch 00031: val_loss did not improve from 0.90281
235/235 [==============================] - 90s 381ms/step - loss: 0.6571 - accuracy: 0.7867 - val_loss: 0.9721 - val_accuracy: 0.7080 - lr: 0.0010
Epoch 32/100
235/235 [==============================] - ETA: 0s - loss: 0.6359 - accuracy: 0.7877
Epoch 00032: val_loss improved from 0.90281 to 0.87984, saving model to Multi_classification.hdf5
235/235 [==============================] - 89s 381ms/step - loss: 0.6359 - accuracy: 0.7877 - val_loss: 0.8798 - val_accuracy: 0.7324 - lr: 0.0010
Epoch 33/100
235/235 [==============================] - ETA: 0s - loss: 0.6297 - accuracy: 0.7985
Epoch 00033: val_loss improved from 0.87984 to 0.84967, saving model to Multi_classification.hdf5
235/235 [==============================] - 90s 382ms/step - loss: 0.6297 - accuracy: 0.7985 - val_loss: 0.8497 - val_accuracy: 0.7448 - lr: 0.0010
Epoch 34/100
235/235 [==============================] - ETA: 0s - loss: 0.6129 - accuracy: 0.7987
Epoch 00034: val_loss did not improve from 0.84967
235/235 [==============================] - 90s 381ms/step - loss: 0.6129 - accuracy: 0.7987 - val_loss: 0.9134 - val_accuracy: 0.7240 - lr: 0.0010
Epoch 35/100
235/235 [==============================] - ETA: 0s - loss: 0.5858 - accuracy: 0.8091
Epoch 00035: val_loss did not improve from 0.84967
235/235 [==============================] - 90s 382ms/step - loss: 0.5858 - accuracy: 0.8091 - val_loss: 0.9003 - val_accuracy: 0.7300 - lr: 0.0010
Epoch 36/100
235/235 [==============================] - ETA: 0s - loss: 0.5820 - accuracy: 0.8096
Epoch 00036: val_loss improved from 0.84967 to 0.84280, saving model to Multi_classification.hdf5
235/235 [==============================] - 90s 385ms/step - loss: 0.5820 - accuracy: 0.8096 - val_loss: 0.8428 - val_accuracy: 0.7416 - lr: 0.0010
Epoch 37/100
235/235 [==============================] - ETA: 0s - loss: 0.5742 - accuracy: 0.8103
Epoch 00037: val_loss did not improve from 0.84280
235/235 [==============================] - 89s 381ms/step - loss: 0.5742 - accuracy: 0.8103 - val_loss: 0.9611 - val_accuracy: 0.7200 - lr: 0.0010
Epoch 38/100
235/235 [==============================] - ETA: 0s - loss: 0.5462 - accuracy: 0.8185
Epoch 00038: val_loss did not improve from 0.84280
235/235 [==============================] - 89s 379ms/step - loss: 0.5462 - accuracy: 0.8185 - val_loss: 0.8707 - val_accuracy: 0.7372 - lr: 0.0010
Epoch 39/100
235/235 [==============================] - ETA: 0s - loss: 0.5243 - accuracy: 0.8297
Epoch 00039: val_loss improved from 0.84280 to 0.81465, saving model to Multi_classification.hdf5
235/235 [==============================] - 89s 381ms/step - loss: 0.5243 - accuracy: 0.8297 - val_loss: 0.8146 - val_accuracy: 0.7636 - lr: 0.0010
Epoch 40/100
235/235 [==============================] - ETA: 0s - loss: 0.5178 - accuracy: 0.8303
Epoch 00040: val_loss did not improve from 0.81465
235/235 [==============================] - 90s 383ms/step - loss: 0.5178 - accuracy: 0.8303 - val_loss: 0.9321 - val_accuracy: 0.7324 - lr: 0.0010
```

Appendix 2 – Multi-Class Classification Test cases (7/7) Model 7 Part 2

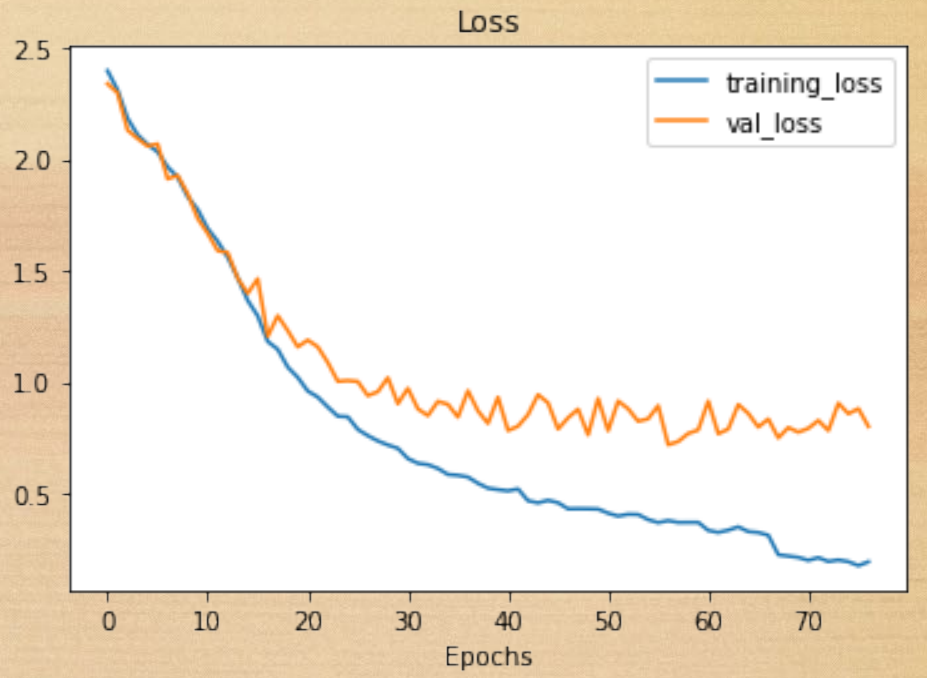
```
Epoch 41/100
235/235 [==============================] - ETA: 0s - loss: 0.5124 - accuracy: 0.8364
Epoch 00041: val_loss improved from 0.81465 to 0.78264, saving model to Multi_classification.hdf5
235/235 [==============================] - 90s 382ms/step - loss: 0.5124 - accuracy: 0.8364 - val_loss: 0.7826 - val_accuracy: 0.7568 - lr: 0.0010
Epoch 42/100
235/235 [==============================] - ETA: 0s - loss: 0.5210 - accuracy: 0.8313
Epoch 00042: val_loss did not improve from 0.78264
235/235 [==============================] - 90s 381ms/step - loss: 0.5210 - accuracy: 0.8313 - val_loss: 0.8021 - val_accuracy: 0.7668 - lr: 0.0010
Epoch 43/100
235/235 [==============================] - ETA: 0s - loss: 0.4681 - accuracy: 0.8448
Epoch 00043: val_loss did not improve from 0.78264
235/235 [==============================] - 90s 383ms/step - loss: 0.4681 - accuracy: 0.8448 - val_loss: 0.8542 - val_accuracy: 0.7628 - lr: 0.0010
Epoch 44/100
235/235 [==============================] - ETA: 0s - loss: 0.4586 - accuracy: 0.8515
Epoch 00044: val_loss did not improve from 0.78264
235/235 [==============================] - 90s 381ms/step - loss: 0.4586 - accuracy: 0.8515 - val_loss: 0.9436 - val_accuracy: 0.7344 - lr: 0.0010
Epoch 45/100
235/235 [==============================] - ETA: 0s - loss: 0.4697 - accuracy: 0.8492
Epoch 00045: val_loss did not improve from 0.78264
235/235 [==============================] - 90s 381ms/step - loss: 0.4697 - accuracy: 0.8492 - val_loss: 0.9069 - val_accuracy: 0.7548 - lr: 0.0010
Epoch 46/100
235/235 [==============================] - ETA: 0s - loss: 0.4592 - accuracy: 0.8532
Epoch 00046: val_loss did not improve from 0.78264
235/235 [==============================] - 90s 381ms/step - loss: 0.4592 - accuracy: 0.8532 - val_loss: 0.7894 - val_accuracy: 0.7720 - lr: 0.0010
Epoch 47/100
235/235 [==============================] - ETA: 0s - loss: 0.4316 - accuracy: 0.8591
Epoch 00047: val_loss did not improve from 0.78264
235/235 [==============================] - 91s 387ms/step - loss: 0.4316 - accuracy: 0.8591 - val_loss: 0.8398 - val_accuracy: 0.7620 - lr: 0.0010
Epoch 48/100
235/235 [==============================] - ETA: 0s - loss: 0.4322 - accuracy: 0.8565
Epoch 00048: val_loss did not improve from 0.78264
235/235 [==============================] - 90s 382ms/step - loss: 0.4322 - accuracy: 0.8565 - val_loss: 0.8786 - val_accuracy: 0.7656 - lr: 0.0010
Epoch 49/100
235/235 [==============================] - ETA: 0s - loss: 0.4317 - accuracy: 0.8595
Epoch 00049: val_loss improved from 0.78264 to 0.76335, saving model to Multi_classification.hdf5
235/235 [==============================] - 89s 381ms/step - loss: 0.4317 - accuracy: 0.8595 - val_loss: 0.7633 - val_accuracy: 0.7704 - lr: 0.0010
Epoch 50/100
235/235 [==============================] - ETA: 0s - loss: 0.4307 - accuracy: 0.8639
Epoch 00050: val_loss did not improve from 0.76335
Epoch 51/100
235/235 [==============================] - ETA: 0s - loss: 0.4123 - accuracy: 0.8656
Epoch 00051: val_loss did not improve from 0.76335
235/235 [==============================] - 89s 381ms/step - loss: 0.4123 - accuracy: 0.8656 - val_loss: 0.7821 - val_accuracy: 0.7736 - lr: 0.0010
Epoch 52/100
235/235 [==============================] - ETA: 0s - loss: 0.3992 - accuracy: 0.8683
Epoch 00052: val_loss did not improve from 0.76335
235/235 [==============================] - 89s 381ms/step - loss: 0.3992 - accuracy: 0.8683 - val_loss: 0.9153 - val_accuracy: 0.7616 - lr: 0.0010
Epoch 53/100
235/235 [==============================] - ETA: 0s - loss: 0.4074 - accuracy: 0.8693
Epoch 00053: val_loss did not improve from 0.76335
235/235 [==============================] - 89s 381ms/step - loss: 0.4074 - accuracy: 0.8693 - val_loss: 0.8802 - val_accuracy: 0.7612 - lr: 0.0010
Epoch 54/100
235/235 [==============================] - ETA: 0s - loss: 0.4062 - accuracy: 0.8687
Epoch 00054: val_loss did not improve from 0.76335
235/235 [==============================] - 90s 381ms/step - loss: 0.4062 - accuracy: 0.8687 - val_loss: 0.8244 - val_accuracy: 0.7664 - lr: 0.0010
Epoch 55/100
235/235 [==============================] - ETA: 0s - loss: 0.3834 - accuracy: 0.8743
Epoch 00055: val_loss did not improve from 0.76335
235/235 [==============================] - 90s 383ms/step - loss: 0.3834 - accuracy: 0.8743 - val_loss: 0.8354 - val_accuracy: 0.7680 - lr: 0.0010
Epoch 56/100
235/235 [==============================] - ETA: 0s - loss: 0.3700 - accuracy: 0.8819
Epoch 00056: val_loss did not improve from 0.76335
235/235 [==============================] - 90s 382ms/step - loss: 0.3700 - accuracy: 0.8819 - val_loss: 0.8943 - val_accuracy: 0.7820 - lr: 0.0010
Epoch 57/100
235/235 [==============================] - ETA: 0s - loss: 0.3788 - accuracy: 0.8753
Epoch 00057: val_loss improved from 0.76335 to 0.71891, saving model to Multi_classification.hdf5
235/235 [==============================] - 90s 381ms/step - loss: 0.3788 - accuracy: 0.8753 - val_loss: 0.7189 - val_accuracy: 0.7900 - lr: 0.0010
Epoch 58/100
235/235 [==============================] - ETA: 0s - loss: 0.3708 - accuracy: 0.8821
Epoch 00058: val_loss did not improve from 0.71891
235/235 [==============================] - 90s 382ms/step - loss: 0.3708 - accuracy: 0.8821 - val_loss: 0.7335 - val_accuracy: 0.7856 - lr: 0.0010
Epoch 59/100
235/235 [==============================] - ETA: 0s - loss: 0.3711 - accuracy: 0.8827
Epoch 00059: val_loss did not improve from 0.71891
235/235 [==============================] - 90s 381ms/step - loss: 0.3711 - accuracy: 0.8827 - val_loss: 0.7695 - val_accuracy: 0.7984 - lr: 0.0010
Epoch 60/100
235/235 [==============================] - ETA: 0s - loss: 0.3711 - accuracy: 0.8808
Epoch 00060: val_loss did not improve from 0.71891
```

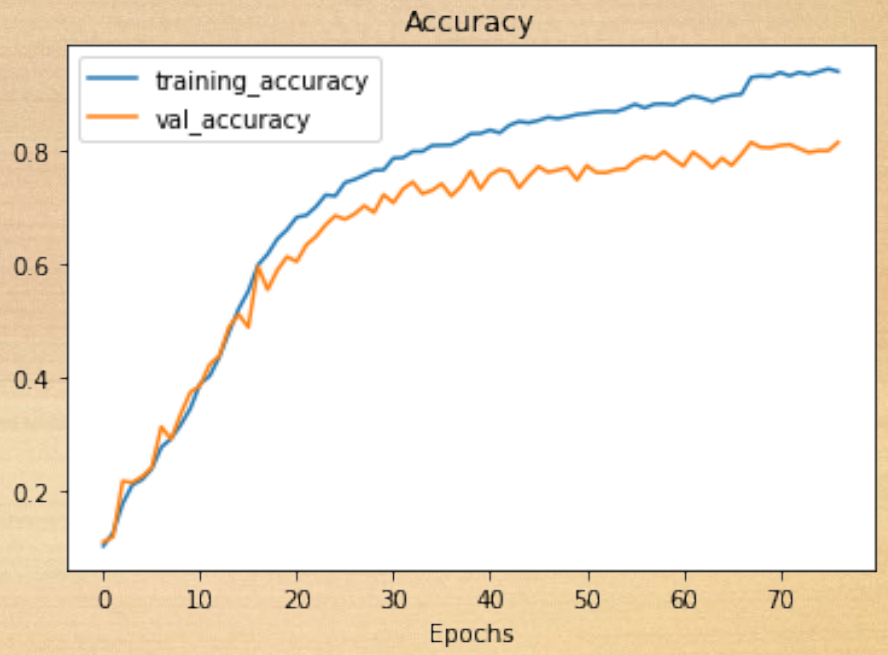
```
Epoch 61/100
235/235 [==============================] - ETA: 0s - loss: 0.3351 - accuracy: 0.8904
Epoch 00061: val_loss did not improve from 0.71891
235/235 [==============================] - 89s 380ms/step - loss: 0.3351 - accuracy: 0.8904 - val_loss: 0.9148 - val_accuracy: 0.7728 - lr: 0.0010
Epoch 62/100
235/235 [==============================] - ETA: 0s - loss: 0.3260 - accuracy: 0.8965
Epoch 00062: val_loss did not improve from 0.71891
235/235 [==============================] - 90s 381ms/step - loss: 0.3260 - accuracy: 0.8965 - val_loss: 0.7684 - val_accuracy: 0.7972 - lr: 0.0010
Epoch 63/100
235/235 [==============================] - ETA: 0s - loss: 0.3361 - accuracy: 0.8925
Epoch 00063: val_loss did not improve from 0.71891
235/235 [==============================] - 89s 381ms/step - loss: 0.3361 - accuracy: 0.8925 - val_loss: 0.7894 - val_accuracy: 0.7852 - lr: 0.0010
Epoch 64/100
235/235 [==============================] - ETA: 0s - loss: 0.3511 - accuracy: 0.8868
Epoch 00064: val_loss did not improve from 0.71891
235/235 [==============================] - 90s 381ms/step - loss: 0.3511 - accuracy: 0.8868 - val_loss: 0.8993 - val_accuracy: 0.7696 - lr: 0.0010
Epoch 65/100
235/235 [==============================] - ETA: 0s - loss: 0.3297 - accuracy: 0.8939
Epoch 00065: val_loss did not improve from 0.71891
235/235 [==============================] - 90s 381ms/step - loss: 0.3297 - accuracy: 0.8939 - val_loss: 0.8594 - val_accuracy: 0.7868 - lr: 0.0010
Epoch 66/100
235/235 [==============================] - ETA: 0s - loss: 0.3257 - accuracy: 0.8975
Epoch 00066: val_loss did not improve from 0.71891
235/235 [==============================] - 89s 381ms/step - loss: 0.3257 - accuracy: 0.8975 - val_loss: 0.7984 - val_accuracy: 0.7740 - lr: 0.0010
Epoch 67/100
235/235 [==============================] - ETA: 0s - loss: 0.3134 - accuracy: 0.8996
Epoch 00067: val_loss did not improve from 0.71891
235/235 [==============================] - 90s 382ms/step - loss: 0.3134 - accuracy: 0.8996 - val_loss: 0.8344 - val_accuracy: 0.7932 - lr: 0.0010
Epoch 68/100
235/235 [==============================] - ETA: 0s - loss: 0.2256 - accuracy: 0.9293
Epoch 00068: val_loss did not improve from 0.71891
235/235 [==============================] - 90s 383ms/step - loss: 0.2256 - accuracy: 0.9293 - val_loss: 0.7496 - val_accuracy: 0.8148 - lr: 5.0000e-04
Epoch 69/100
235/235 [==============================] - ETA: 0s - loss: 0.2198 - accuracy: 0.9316
Epoch 00069: val_loss did not improve from 0.71891
235/235 [==============================] - 90s 381ms/step - loss: 0.2198 - accuracy: 0.9316 - val_loss: 0.7963 - val_accuracy: 0.8060 - lr: 5.0000e-04
Epoch 70/100
235/235 [==============================] - ETA: 0s - loss: 0.2140 - accuracy: 0.9305
Epoch 00070: val_loss did not improve from 0.71891
235/235 [==============================] - 90s 383ms/step - loss: 0.2140 - accuracy: 0.9305 - val_loss: 0.7754 - val_accuracy: 0.8052 - lr: 5.0000e-04
Epoch 71/100
235/235 [==============================] - ETA: 0s - loss: 0.2008 - accuracy: 0.9380
Epoch 00071: val_loss did not improve from 0.71891
235/235 [==============================] - 90s 383ms/step - loss: 0.2008 - accuracy: 0.9380 - val_loss: 0.7926 - val_accuracy: 0.8092 - lr: 5.0000e-04
Epoch 72/100
235/235 [==============================] - ETA: 0s - loss: 0.2129 - accuracy: 0.9321
Epoch 00072: val_loss did not improve from 0.71891
235/235 [==============================] - 90s 383ms/step - loss: 0.2129 - accuracy: 0.9321 - val_loss: 0.8280 - val_accuracy: 0.8108 - lr: 5.0000e-04
Epoch 73/100
235/235 [==============================] - ETA: 0s - loss: 0.1957 - accuracy: 0.9383
Epoch 00073: val_loss did not improve from 0.71891
235/235 [==============================] - 90s 383ms/step - loss: 0.1957 - accuracy: 0.9383 - val_loss: 0.7826 - val_accuracy: 0.8036 - lr: 5.0000e-04
Epoch 74/100
235/235 [==============================] - ETA: 0s - loss: 0.2024 - accuracy: 0.9341
Epoch 00074: val_loss did not improve from 0.71891
235/235 [==============================] - 90s 383ms/step - loss: 0.2024 - accuracy: 0.9341 - val_loss: 0.9056 - val_accuracy: 0.7968 - lr: 5.0000e-04
Epoch 75/100
235/235 [==============================] - ETA: 0s - loss: 0.1948 - accuracy: 0.9393
Epoch 00075: val_loss did not improve from 0.71891
235/235 [==============================] - 90s 383ms/step - loss: 0.1948 - accuracy: 0.9393 - val_loss: 0.8581 - val_accuracy: 0.8000 - lr: 5.0000e-04
Epoch 76/100
235/235 [==============================] - ETA: 0s - loss: 0.1771 - accuracy: 0.9444
Epoch 00076: val_loss did not improve from 0.71891
235/235 [==============================] - 90s 384ms/step - loss: 0.1771 - accuracy: 0.9444 - val_loss: 0.8808 - val_accuracy: 0.8000 - lr: 5.0000e-04
Epoch 77/100
235/235 [==============================] - ETA: 0s - loss: 0.1943 - accuracy: 0.9397
Epoch 00077: val_loss did not improve from 0.71891
235/235 [==============================] - 90s 384ms/step - loss: 0.1943 - accuracy: 0.9397 - val_loss: 0.8009 - val_accuracy: 0.8148 - lr: 5.0000e-04
```

Appendix 2 – Multi-Class Classification Test cases (7/7) Model 7 Part 3

```
model_7.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 112, 112, 32)      2432

 conv2d_1 (Conv2D)           (None, 56, 56, 32)        25632

 max_pooling2d (MaxPooling2D  (None, 28, 28, 32)       0
 )

 dropout (Dropout)           (None, 28, 28, 32)        0

 conv2d_2 (Conv2D)           (None, 28, 28, 64)        18496

 conv2d_3 (Conv2D)           (None, 28, 28, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 14, 14, 64)       0
 2D)

 dropout_1 (Dropout)         (None, 14, 14, 64)        0

 conv2d_4 (Conv2D)           (None, 14, 14, 128)       32896

 conv2d_5 (Conv2D)           (None, 14, 14, 128)       65664

 max_pooling2d_2 (MaxPooling  (None, 7, 7, 128)        0
 2D)

 dropout_2 (Dropout)         (None, 7, 7, 128)         0

 conv2d_6 (Conv2D)           (None, 7, 7, 256)         131328

 conv2d_7 (Conv2D)           (None, 7, 7, 256)         262400

 global_average_pooling2d (G  (None, 256)              0
 lobalAveragePooling2D)

 dense (Dense)               (None, 512)               131584

 dropout_3 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 10)                5130

=================================================================
Total params: 712,490
Trainable params: 712,490
Non-trainable params: 0
_________________________________________________________________
```





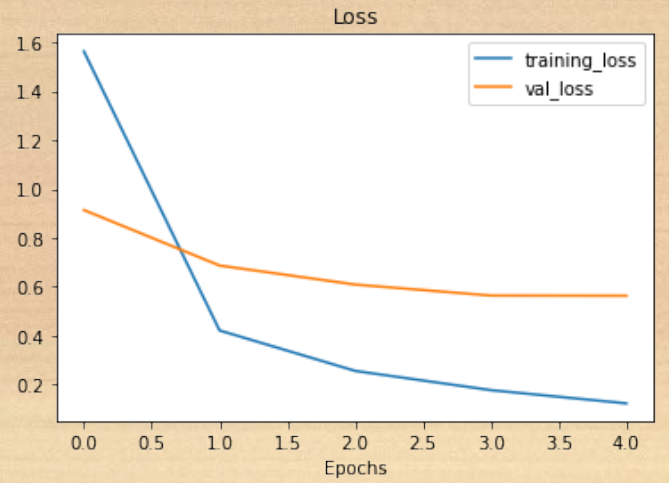
Appendix 3 – Transfer Learning_v1(20 food classes)cases (1/2)

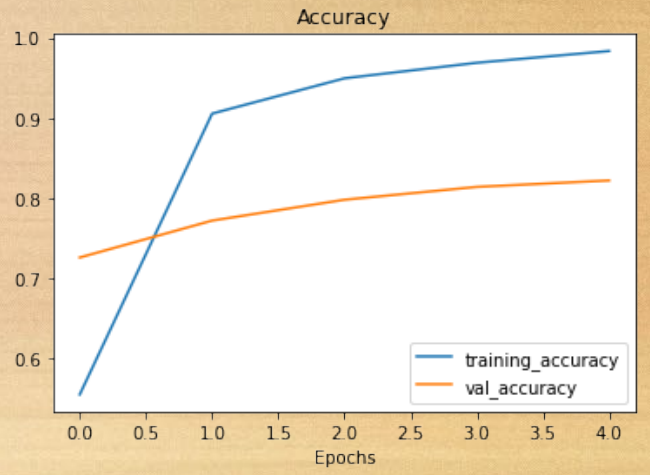
ResNetV1 Model

```
Saving TensorBoard log files to: tensorflow_hub/resnet50V2/20220214-123252
Epoch 1/5
 3/47 [>.............................] - ETA: 40s - loss: 3.8061 - accuracy: 0.0625/usr/local/lib/python3.7/dist-packages/PIL/Image.py:960: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA images
  "Palette images with Transparency expressed in bytes should be "
25/47 [==============>...............] - ETA: 23s - loss: 2.2514 - accuracy: 0.3455/usr/local/lib/python3.7/dist-packages/PIL/TiffImagePlugin.py:788: UserWarning: Corrupt EXIF data.  Expecting to read 4 bytes but only got 0.
  warnings.warn(str(msg))
47/47 [==============================] - 58s 1s/step - loss: 1.6210 - accuracy: 0.5360 - val_loss: 0.9370 - val_accuracy: 0.7180
Epoch 2/5
47/47 [==============================] - 53s 1s/step - loss: 0.4292 - accuracy: 0.8993 - val_loss: 0.6827 - val_accuracy: 0.7900
Epoch 3/5
47/47 [==============================] - 53s 1s/step - loss: 0.2562 - accuracy: 0.9500 - val_loss: 0.6179 - val_accuracy: 0.7940
Epoch 4/5
47/47 [==============================] - 53s 1s/step - loss: 0.1756 - accuracy: 0.9740 - val_loss: 0.5908 - val_accuracy: 0.8160
Epoch 5/5
47/47 [==============================] - 53s 1s/step - loss: 0.1253 - accuracy: 0.9887 - val_loss: 0.5683 - val_accuracy: 0.8200
```

```
resnet_model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 feature_extraction_layer (K  (None, 2048)             23564800
 erasLayer)

 output_layer (Dense)        (None, 20)                40980

=================================================================
Total params: 23,605,780
Trainable params: 40,980
Non-trainable params: 23,564,800
_________________________________________________________________
```



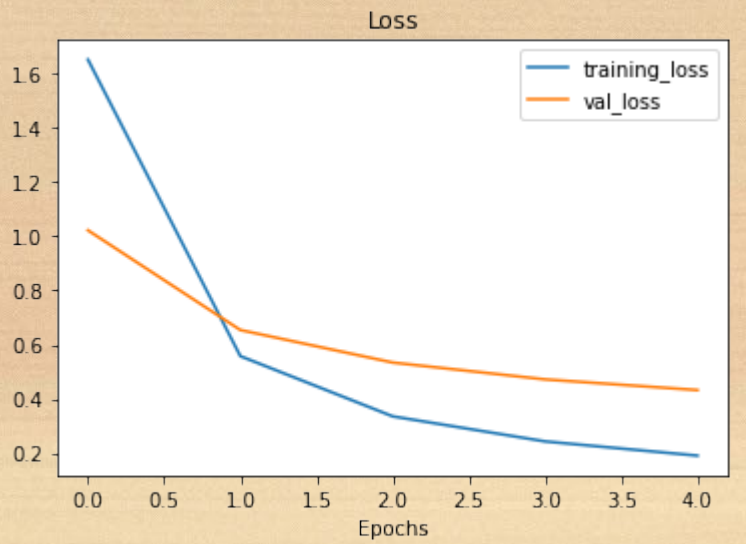


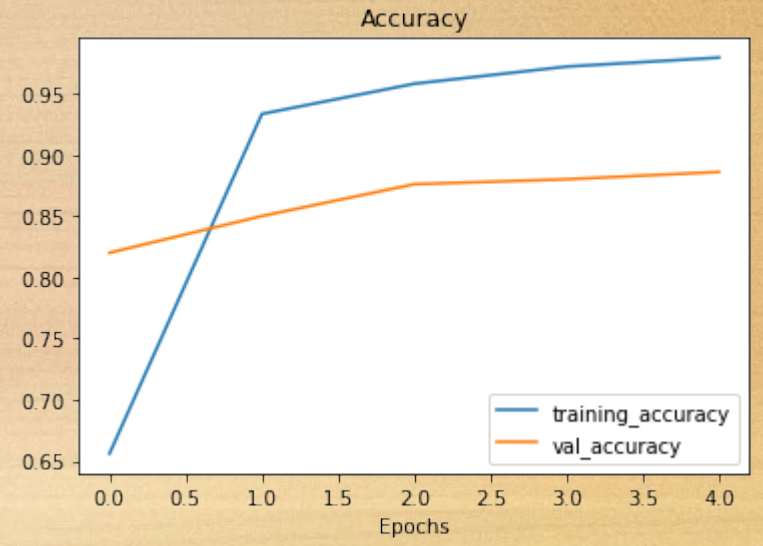
Appendix 3 – Transfer Learning_v1(20 food classes)cases (2/2)

EfficientNetB0

```
Saving TensorBoard log files to: tensorflow_hub/efficientnetB0/20220215-102534
Epoch 1/5
 4/47 [=>............................] - ETA: 37s - loss: 2.9133 - accuracy: 0.0859/usr/local/lib/python3.7/dist-packages/PIL/Image.py:960: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA images
  "Palette images with Transparency expressed in bytes should be "
17/47 [=========>....................] - ETA: 36s - loss: 2.4407 - accuracy: 0.3272/usr/local/lib/python3.7/dist-packages/PIL/TiffImagePlugin.py:788: UserWarning: Corrupt EXIF data.  Expecting to read 4 bytes but only got 0.
  warnings.warn(str(msg))
47/47 [==============================] - 81s 1s/step - loss: 1.6872 - accuracy: 0.6253 - val_loss: 1.0072 - val_accuracy: 0.8040
Epoch 2/5
47/47 [==============================] - 61s 1s/step - loss: 0.5717 - accuracy: 0.9307 - val_loss: 0.6577 - val_accuracy: 0.8520
Epoch 3/5
47/47 [==============================] - 61s 1s/step - loss: 0.3436 - accuracy: 0.9553 - val_loss: 0.5227 - val_accuracy: 0.8780
Epoch 4/5
47/47 [==============================] - 61s 1s/step - loss: 0.2481 - accuracy: 0.9687 - val_loss: 0.4618 - val_accuracy: 0.8840
Epoch 5/5
47/47 [==============================] - 62s 1s/step - loss: 0.1953 - accuracy: 0.9807 - val_loss: 0.4187 - val_accuracy: 0.8940
```

```
efficientnet_model_1.summary()

Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 feature_extraction_layer (K  (None, 1280)             4049564
 erasLayer)

 output_layer (Dense)        (None, 20)                25620

=================================================================
Total params: 4,075,184
Trainable params: 25,620
Non-trainable params: 4,049,564
_________________________________________________________________
```



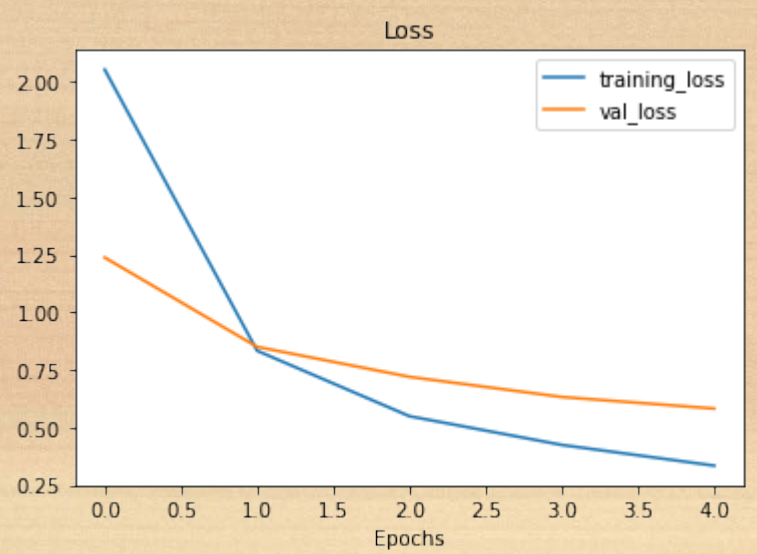


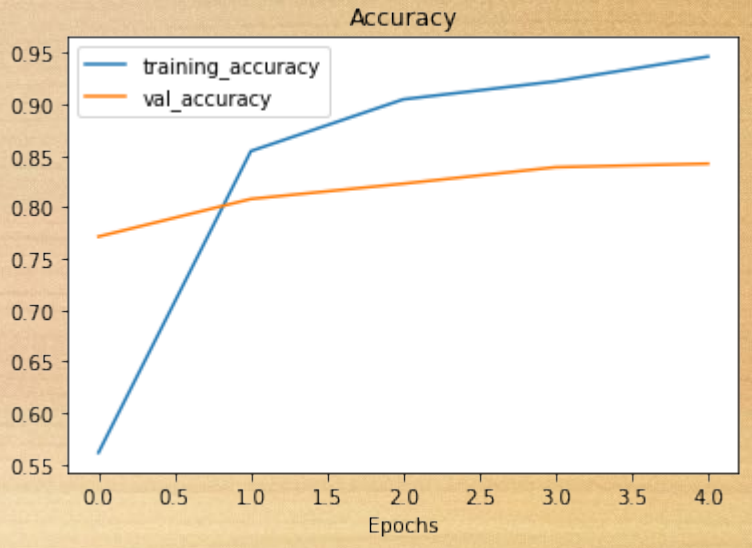
Appendix 4 – Transfer Learning_v2(35 food classes)cases (1/3)

EifficentNetB0 Model 1

```
Saving TensorBoard log files to: tensorflow_hub/efficientnetB0/20220222-042400
Epoch 1/5
18/83 [=====>.....................] - ETA: 54s - loss: 3.1994 - accuracy: 0.1736/usr/local/lib/python3.7/dist-packages/PIL/Image.py:960: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA images
  "Palette images with Transparency expressed in bytes should be "
36/83 [===========>..................] - ETA: 41s - loss: 2.7267 - accuracy: 0.3524/usr/local/lib/python3.7/dist-packages/PIL/TiffImagePlugin.py:788: UserWarning: Corrupt EXIF data.  Expecting to read 4 bytes but only got 0.
  warnings.warn(str(msg))
83/83 [==============================] - 100s 990ms/step - loss: 2.0523 - accuracy: 0.5615 - val_loss: 1.2367 - val_accuracy: 0.7714
Epoch 2/5
83/83 [==============================] - 80s 972ms/step - loss: 0.8322 - accuracy: 0.8545 - val_loss: 0.8491 - val_accuracy: 0.8080
Epoch 3/5
83/83 [==============================] - 78s 947ms/step - loss: 0.5480 - accuracy: 0.9048 - val_loss: 0.7188 - val_accuracy: 0.8229
Epoch 4/5
83/83 [==============================] - 79s 950ms/step - loss: 0.4233 - accuracy: 0.9223 - val_loss: 0.6314 - val_accuracy: 0.8389
Epoch 5/5
83/83 [==============================] - 78s 942ms/step - loss: 0.3334 - accuracy: 0.9463 - val_loss: 0.5811 - val_accuracy: 0.8423
```

```
efficientnet_model_1.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 feature_extraction_layer (K  (None, 1280)             4049564
 erasLayer)

 output_layer (Dense)        (None, 35)                44835

=================================================================
Total params: 4,094,399
Trainable params: 44,835
Non-trainable params: 4,049,564
```



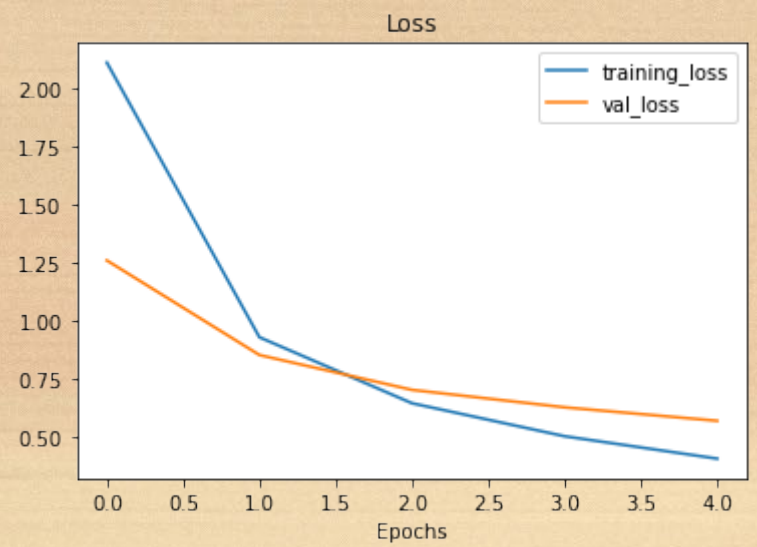


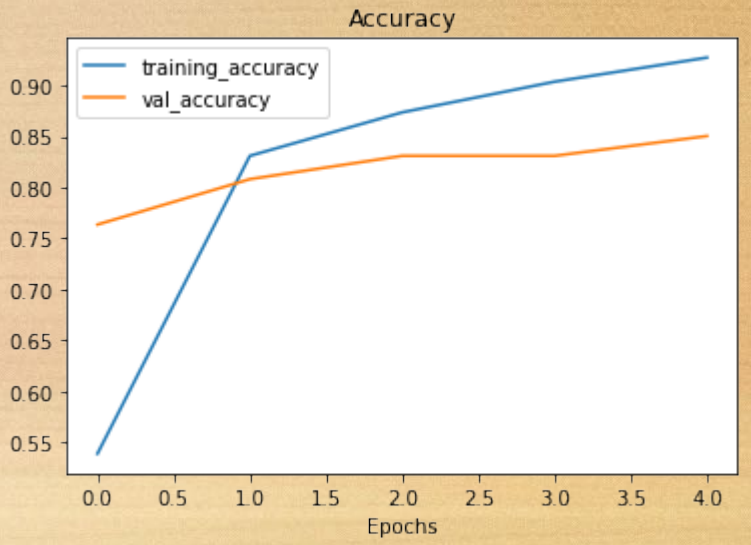
Appendix 4 – Transfer Learning_v2(35 food classes)cases (2/3)

EifficentNetB0 Model 2

```
Saving TensorBoard log files to: tensorflow_hub/efficientnetB0/20220222-052410
Epoch 1/5
40/83 [=============>.............] - ETA: 47s - loss: 2.7246 - accuracy: 0.3483/usr/local/lib/python3.7/dist-packages/PIL/Image.py:960: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA images
  "Palette images with Transparency expressed in bytes should be "
55/83 [==================>...........] - ETA: 30s - loss: 2.4599 - accuracy: 0.4332/usr/local/lib/python3.7/dist-packages/PIL/TiffImagePlugin.py:788: UserWarning: Corrupt EXIF data.  Expecting to read 4 bytes but only got 0.
  warnings.warn(str(msg))
83/83 [==============================] - 110s 1s/step - loss: 2.1141 - accuracy: 0.5387 - val_loss: 1.2613 - val_accuracy: 0.7634
Epoch 2/5
83/83 [==============================] - 101s 1s/step - loss: 0.9301 - accuracy: 0.8309 - val_loss: 0.8536 - val_accuracy: 0.8080
Epoch 3/5
83/83 [==============================] - 101s 1s/step - loss: 0.6456 - accuracy: 0.8735 - val_loss: 0.7032 - val_accuracy: 0.8309
Epoch 4/5
83/83 [==============================] - 101s 1s/step - loss: 0.5034 - accuracy: 0.9036 - val_loss: 0.6280 - val_accuracy: 0.8309
Epoch 5/5
83/83 [==============================] - 100s 1s/step - loss: 0.4068 - accuracy: 0.9272 - val_loss: 0.5699 - val_accuracy: 0.8503
```

```
efficientnet_model_2.summary()

Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 feature_extraction_layer (K  (None, 1280)             4049564
 erasLayer)

 output_layer (Dense)        (None, 35)                44835

=================================================================
Total params: 4,094,399
Trainable params: 44,835
Non-trainable params: 4,049,564
_________________________________________________________________
```





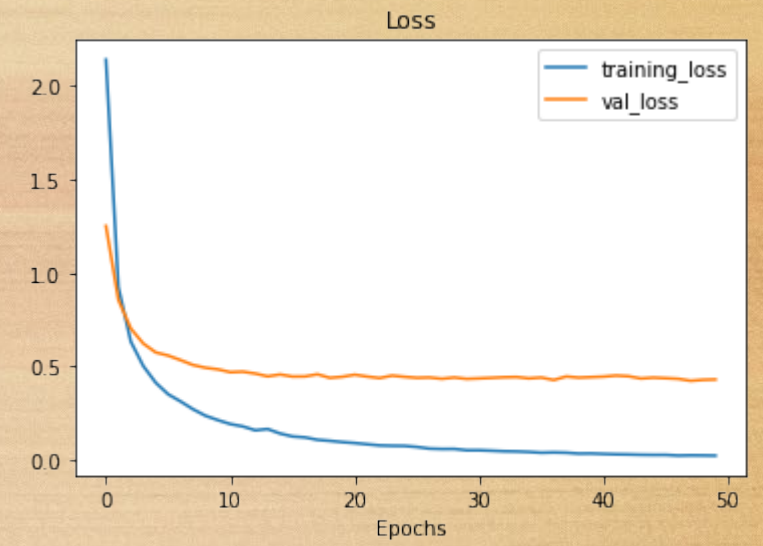
Appendix 4 – Transfer Learning_v2(35 food classes)cases (3/3) EifficentNetB0 Model 3 Part 1

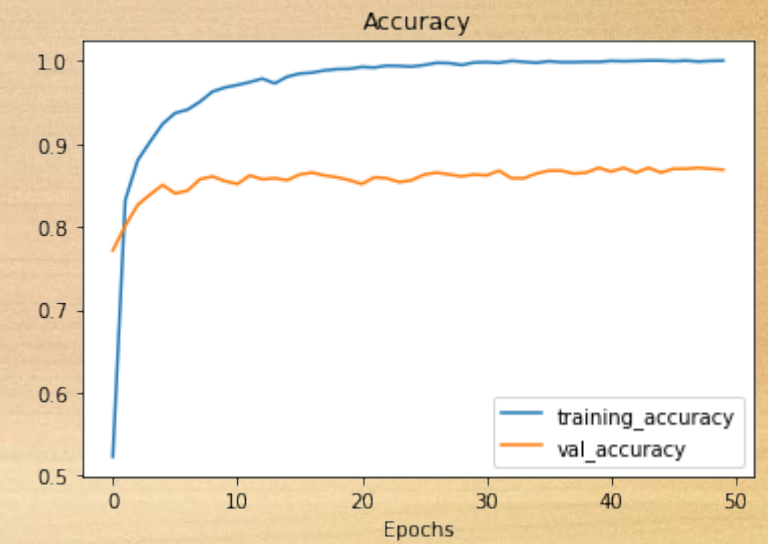
```
Saving TensorBoard log files to: tensorflow_hub/efficientnetB0/20220222-055227
Epoch 1/50
 7/83 [=>............................] - ETA: 1:36 - loss: 3.5449 - accuracy: 0.0848/usr/local/lib/python3.7/dist-packages/PIL/Image.py:960: UserW
  "Palette images with Transparency expressed in bytes should be "
82/83 [============================>.] - ETA: 1s - loss: 2.1548 - accuracy: 0.5187/usr/local/lib/python3.7/dist-packages/PIL/TiffImagePlugin.py:78
  warnings.warn(str(msg))
83/83 [==============================] - ETA: 0s - loss: 2.1415 - accuracy: 0.5230
Epoch 1: val_loss improved from inf to 1.25132, saving model to Transfer_Learning_classification.hdf5
83/83 [==============================] - 125s 1s/step - loss: 2.1415 - accuracy: 0.5230 - val_loss: 1.2513 - val_accuracy: 0.7714 - lr: 0.0010
Epoch 2/50
83/83 [==============================] - ETA: 0s - loss: 0.9241 - accuracy: 0.8320
Epoch 2: val_loss improved from 1.25132 to 0.85615, saving model to Transfer_Learning_classification.hdf5
83/83 [==============================] - 103s 1s/step - loss: 0.9241 - accuracy: 0.8320 - val_loss: 0.8562 - val_accuracy: 0.8011 - lr: 0.0010
Epoch 3/50
83/83 [==============================] - ETA: 0s - loss: 0.6322 - accuracy: 0.8800
Epoch 3: val_loss improved from 0.85615 to 0.70246, saving model to Transfer_Learning_classification.hdf5
83/83 [==============================] - 102s 1s/step - loss: 0.6322 - accuracy: 0.8800 - val_loss: 0.7025 - val_accuracy: 0.8263 - lr: 0.0010
Epoch 4/50
83/83 [==============================] - ETA: 0s - loss: 0.5020 - accuracy: 0.9025
Epoch 4: val_loss improved from 0.70246 to 0.62235, saving model to Transfer_Learning_classification.hdf5
83/83 [==============================] - 113s 1s/step - loss: 0.5020 - accuracy: 0.9025 - val_loss: 0.6223 - val_accuracy: 0.8389 - lr: 0.0010
Epoch 5/50
83/83 [==============================] - ETA: 0s - loss: 0.4129 - accuracy: 0.9238
Epoch 5: val_loss improved from 0.62235 to 0.57476, saving model to Transfer_Learning_classification.hdf5
83/83 [==============================] - 114s 1s/step - loss: 0.4129 - accuracy: 0.9238 - val_loss: 0.5748 - val_accuracy: 0.8503 - lr: 0.0010
Epoch 6/50
83/83 [==============================] - ETA: 0s - loss: 0.3508 - accuracy: 0.9368
Epoch 6: val_loss improved from 0.57476 to 0.55791, saving model to Transfer_Learning_classification.hdf5
83/83 [==============================] - 114s 1s/step - loss: 0.3508 - accuracy: 0.9368 - val_loss: 0.5579 - val_accuracy: 0.8400 - lr: 0.0010
Epoch 7/50
83/83 [==============================] - ETA: 0s - loss: 0.3124 - accuracy: 0.9410
Epoch 7: val_loss improved from 0.55791 to 0.53350, saving model to Transfer_Learning_classification.hdf5
83/83 [==============================] - 115s 1s/step - loss: 0.3124 - accuracy: 0.9410 - val_loss: 0.5335 - val_accuracy: 0.8434 - lr: 0.0010
Epoch 8/50
83/83 [==============================] - ETA: 0s - loss: 0.2707 - accuracy: 0.9505
Epoch 8: val_loss improved from 0.53350 to 0.50740, saving model to Transfer_Learning_classification.hdf5
83/83 [==============================] - 115s 1s/step - loss: 0.2707 - accuracy: 0.9505 - val_loss: 0.5074 - val_accuracy: 0.8571 - lr: 0.0010
Epoch 9/50
83/83 [==============================] - ETA: 0s - loss: 0.2360 - accuracy: 0.9627
Epoch 9: val_loss improved from 0.50740 to 0.49261, saving model to Transfer_Learning_classification.hdf5
83/83 [==============================] - 114s 1s/step - loss: 0.2360 - accuracy: 0.9627 - val_loss: 0.4926 - val_accuracy: 0.8606 - lr: 0.0010
Epoch 10/50
83/83 [==============================] - ETA: 0s - loss: 0.2124 - accuracy: 0.9676
Epoch 10: val_loss improved from 0.49261 to 0.48413, saving model to Transfer_Learning_classification.hdf5
[==============================] - 114s 1s/step - loss: 0.2124 - accuracy: 0.9676 - val_loss: 0.4841 - val_accuracy: 0.8629 - lr: 0.0010
Epoch 11/50
83/83 [==============================] - ETA: 0s - loss: 0.1914 - accuracy: 0.9707
Epoch 11: val_loss improved from 0.48413 to 0.46889, saving model to Transfer_Learning_classification.hdf5
83/83 [==============================] - 114s 1s/step - loss: 0.1914 - accuracy: 0.9707 - val_loss: 0.4689 - val_accuracy: 0.8514 - lr: 0.0010
Epoch 12/50
83/83 [==============================] - ETA: 0s - loss: 0.1788 - accuracy: 0.9741
Epoch 12: val_loss did not improve from 0.46889
83/83 [==============================] - 114s 1s/step - loss: 0.1788 - accuracy: 0.9741 - val_loss: 0.4716 - val_accuracy: 0.8617 - lr: 0.0010
Epoch 13/50
83/83 [==============================] - ETA: 0s - loss: 0.1581 - accuracy: 0.9783
Epoch 13: val_loss improved from 0.46889 to 0.46162, saving model to Transfer_Learning_classification.hdf5
83/83 [==============================] - 115s 1s/step - loss: 0.1581 - accuracy: 0.9783 - val_loss: 0.4616 - val_accuracy: 0.8571 - lr: 0.0010
Epoch 14/50
83/83 [==============================] - ETA: 0s - loss: 0.1648 - accuracy: 0.9726
Epoch 14: val_loss improved from 0.46162 to 0.44668, saving model to Transfer_Learning_classification.hdf5
83/83 [==============================] - 114s 1s/step - loss: 0.1648 - accuracy: 0.9726 - val_loss: 0.4467 - val_accuracy: 0.8583 - lr: 0.0010
Epoch 15/50
83/83 [==============================] - ETA: 0s - loss: 0.1407 - accuracy: 0.9806
Epoch 15: val_loss did not improve from 0.44668
83/83 [==============================] - 109s 1s/step - loss: 0.1407 - accuracy: 0.9806 - val_loss: 0.4567 - val_accuracy: 0.8560 - lr: 0.0010
Epoch 16/50
83/83 [==============================] - ETA: 0s - loss: 0.1250 - accuracy: 0.9844
Epoch 16: val_loss improved from 0.44668 to 0.44555, saving model to Transfer_Learning_classification.hdf5
83/83 [==============================] - 103s 1s/step - loss: 0.1250 - accuracy: 0.9844 - val_loss: 0.4455 - val_accuracy: 0.8629 - lr: 0.0010
Epoch 17/50
83/83 [==============================] - ETA: 0s - loss: 0.1200 - accuracy: 0.9855
Epoch 17: val_loss did not improve from 0.44555
83/83 [==============================] - 102s 1s/step - loss: 0.1200 - accuracy: 0.9855 - val_loss: 0.4462 - val_accuracy: 0.8651 - lr: 0.0010
Epoch 18/50
83/83 [==============================] - ETA: 0s - loss: 0.1072 - accuracy: 0.9882
Epoch 18: val_loss did not improve from 0.44555
83/83 [==============================] - 102s 1s/step - loss: 0.1072 - accuracy: 0.9882 - val_loss: 0.4576 - val_accuracy: 0.8617 - lr: 0.0010
Epoch 19/50
83/83 [==============================] - ETA: 0s - loss: 0.1015 - accuracy: 0.9897
Epoch 19: val_loss improved from 0.44555 to 0.43864, saving model to Transfer_Learning_classification.hdf5
83/83 [==============================] - 102s 1s/step - loss: 0.1015 - accuracy: 0.9897 - val_loss: 0.4386 - val_accuracy: 0.8594 - lr: 0.0010
Epoch 20/50
83/83 [==============================] - ETA: 0s - loss: 0.0949 - accuracy: 0.9901
Epoch 20: val_loss did not improve from 0.43864
83/83 [==============================] - 102s 1s/step - loss: 0.0949 - accuracy: 0.9901 - val_loss: 0.4442 - val_accuracy: 0.8560 - lr: 0.0010
```

```
Epoch 21: val_loss did not improve from 0.43864
83/83 [==============================] - 101s 1s/step - loss: 0.0898 - accuracy: 0.9924 - val_loss: 0.4554 - val_accuracy: 0.8514 - lr: 0.0010
Epoch 22/50
83/83 [==============================] - ETA: 0s - loss: 0.0833 - accuracy: 0.9916
Epoch 22: val_loss did not improve from 0.43864
83/83 [==============================] - 101s 1s/step - loss: 0.0833 - accuracy: 0.9916 - val_loss: 0.4454 - val_accuracy: 0.8594 - lr: 0.0010
Epoch 23/50
83/83 [==============================] - ETA: 0s - loss: 0.0771 - accuracy: 0.9939
Epoch 23: val_loss improved from 0.43864 to 0.43750, saving model to Transfer_Learning_classification.hdf5
83/83 [==============================] - 102s 1s/step - loss: 0.0771 - accuracy: 0.9939 - val_loss: 0.4375 - val_accuracy: 0.8583 - lr: 0.0010
Epoch 24/50
82/83 [============================>.] - ETA: 1s - loss: 0.0750 - accuracy: 0.9935
Epoch 24: val_loss did not improve from 0.43750
83/83 [==============================] - 101s 1s/step - loss: 0.0752 - accuracy: 0.9935 - val_loss: 0.4506 - val_accuracy: 0.8537 - lr: 0.0010
Epoch 25/50
83/83 [==============================] - ETA: 0s - loss: 0.0744 - accuracy: 0.9928
Epoch 25: val_loss did not improve from 0.43750
83/83 [==============================] - 102s 1s/step - loss: 0.0744 - accuracy: 0.9928 - val_loss: 0.4440 - val_accuracy: 0.8560 - lr: 0.0010
Epoch 26/50
83/83 [==============================] - ETA: 0s - loss: 0.0688 - accuracy: 0.9947
Epoch 26: val_loss did not improve from 0.43750
83/83 [==============================] - 101s 1s/step - loss: 0.0688 - accuracy: 0.9947 - val_loss: 0.4388 - val_accuracy: 0.8629 - lr: 0.0010
Epoch 27/50
83/83 [==============================] - ETA: 0s - loss: 0.0600 - accuracy: 0.9973
Epoch 27: val_loss did not improve from 0.43750
83/83 [==============================] - 101s 1s/step - loss: 0.0600 - accuracy: 0.9973 - val_loss: 0.4405 - val_accuracy: 0.8651 - lr: 0.0010
Epoch 28/50
83/83 [==============================] - ETA: 0s - loss: 0.0580 - accuracy: 0.9970
Epoch 28: val_loss improved from 0.43750 to 0.43370, saving model to Transfer_Learning_classification.hdf5
83/83 [==============================] - 102s 1s/step - loss: 0.0580 - accuracy: 0.9970 - val_loss: 0.4337 - val_accuracy: 0.8629 - lr: 0.0010
Epoch 29/50
83/83 [==============================] - ETA: 0s - loss: 0.0583 - accuracy: 0.9947
Epoch 29: val_loss did not improve from 0.43370
83/83 [==============================] - 102s 1s/step - loss: 0.0583 - accuracy: 0.9947 - val_loss: 0.4405 - val_accuracy: 0.8606 - lr: 0.0010
Epoch 30/50
83/83 [==============================] - ETA: 0s - loss: 0.0513 - accuracy: 0.9977
Epoch 30: val_loss improved from 0.43370 to 0.43325, saving model to Transfer_Learning_classification.hdf5
83/83 [==============================] - 102s 1s/step - loss: 0.0513 - accuracy: 0.9977 - val_loss: 0.4333 - val_accuracy: 0.8629 - lr: 0.0010
Epoch 31/50
83/83 [==============================] - ETA: 0s - loss: 0.0513 - accuracy: 0.9981
Epoch 31: val_loss did not improve from 0.43325
83/83 [==============================] - 102s 1s/step - loss: 0.0513 - accuracy: 0.9981 - val_loss: 0.4357 - val_accuracy: 0.8617 - lr: 0.0010
Epoch 32/50
83/83 [==============================] - ETA: 0s - loss: 0.0488 - accuracy: 0.9973
Epoch 32: val_loss did not improve from 0.43325
83/83 [==============================] - 102s 1s/step - loss: 0.0488 - accuracy: 0.9973 - val_loss: 0.4386 - val_accuracy: 0.8674 - lr: 0.0010
Epoch 33/50
83/83 [==============================] - ETA: 0s - loss: 0.0458 - accuracy: 0.9996
Epoch 33: val_loss did not improve from 0.43325
83/83 [==============================] - 101s 1s/step - loss: 0.0458 - accuracy: 0.9996 - val_loss: 0.4418 - val_accuracy: 0.8583 - lr: 0.0010
Epoch 34/50
83/83 [==============================] - ETA: 0s - loss: 0.0446 - accuracy: 0.9985
Epoch 34: val_loss did not improve from 0.43325
83/83 [==============================] - 103s 1s/step - loss: 0.0446 - accuracy: 0.9985 - val_loss: 0.4430 - val_accuracy: 0.8583 - lr: 0.0010
Epoch 35/50
83/83 [==============================] - ETA: 0s - loss: 0.0422 - accuracy: 0.9973
Epoch 35: val_loss did not improve from 0.43325
83/83 [==============================] - 103s 1s/step - loss: 0.0422 - accuracy: 0.9973 - val_loss: 0.4358 - val_accuracy: 0.8640 - lr: 0.0010
Epoch 36/50
83/83 [==============================] - ETA: 0s - loss: 0.0382 - accuracy: 0.9992
Epoch 36: val_loss did not improve from 0.43325
83/83 [==============================] - 102s 1s/step - loss: 0.0382 - accuracy: 0.9992 - val_loss: 0.4402 - val_accuracy: 0.8674 - lr: 0.0010
Epoch 37/50
83/83 [==============================] - ETA: 0s - loss: 0.0397 - accuracy: 0.9981
Epoch 37: val_loss improved from 0.43325 to 0.42710, saving model to Transfer_Learning_classification.hdf5
83/83 [==============================] - 113s 1s/step - loss: 0.0397 - accuracy: 0.9981 - val_loss: 0.4271 - val_accuracy: 0.8674 - lr: 0.0010
Epoch 38/50
83/83 [==============================] - ETA: 0s - loss: 0.0381 - accuracy: 0.9981
Epoch 38: val_loss did not improve from 0.42710
83/83 [==============================] - 114s 1s/step - loss: 0.0381 - accuracy: 0.9981 - val_loss: 0.4459 - val_accuracy: 0.8640 - lr: 0.0010
Epoch 39/50
83/83 [==============================] - ETA: 0s - loss: 0.0334 - accuracy: 0.9985
Epoch 39: val_loss did not improve from 0.42710
83/83 [==============================] - 114s 1s/step - loss: 0.0334 - accuracy: 0.9985 - val_loss: 0.4398 - val_accuracy: 0.8651 - lr: 0.0010
Epoch 40/50
83/83 [==============================] - ETA: 0s - loss: 0.0339 - accuracy: 0.9985
Epoch 40: val_loss did not improve from 0.42710
83/83 [==============================] - 114s 1s/step - loss: 0.0339 - accuracy: 0.9985 - val_loss: 0.4425 - val_accuracy: 0.8709 - lr: 0.0010
```

Appendix 4 – Transfer Learning_v2(35 food classes)cases (3/3) EifficentNetB0 Model 3 Part 2

```
Epoch 41: val_loss did not improve from 0.42710
83/83 [==============================] - 114s 1s/step - loss: 0.0315 - accuracy: 0.9996 - val_loss: 0.4452 - val_accuracy: 0.8663 - lr: 0.0010
Epoch 42/50
83/83 [==============================] - ETA: 0s - loss: 0.0296 - accuracy: 0.9992
Epoch 42: val_loss did not improve from 0.42710
83/83 [==============================] - 114s 1s/step - loss: 0.0296 - accuracy: 0.9992 - val_loss: 0.4508 - val_accuracy: 0.8709 - lr: 0.0010
Epoch 43/50
83/83 [==============================] - ETA: 0s - loss: 0.0287 - accuracy: 0.9996
Epoch 43: val_loss did not improve from 0.42710
83/83 [==============================] - 114s 1s/step - loss: 0.0287 - accuracy: 0.9996 - val_loss: 0.4474 - val_accuracy: 0.8651 - lr: 0.0010
Epoch 44/50
83/83 [==============================] - ETA: 0s - loss: 0.0271 - accuracy: 1.0000
Epoch 44: val_loss did not improve from 0.42710
83/83 [==============================] - 114s 1s/step - loss: 0.0271 - accuracy: 1.0000 - val_loss: 0.4352 - val_accuracy: 0.8709 - lr: 0.0010
Epoch 45/50
83/83 [==============================] - ETA: 0s - loss: 0.0263 - accuracy: 1.0000
Epoch 45: val_loss did not improve from 0.42710
83/83 [==============================] - 114s 1s/step - loss: 0.0263 - accuracy: 1.0000 - val_loss: 0.4396 - val_accuracy: 0.8651 - lr: 0.0010
Epoch 46/50
83/83 [==============================] - ETA: 0s - loss: 0.0264 - accuracy: 0.9992
Epoch 46: val_loss did not improve from 0.42710
83/83 [==============================] - 110s 1s/step - loss: 0.0264 - accuracy: 0.9992 - val_loss: 0.4367 - val_accuracy: 0.8697 - lr: 0.0010
Epoch 47/50
83/83 [==============================] - ETA: 0s - loss: 0.0228 - accuracy: 1.0000
Epoch 47: val_loss did not improve from 0.42710
83/83 [==============================] - 102s 1s/step - loss: 0.0228 - accuracy: 1.0000 - val_loss: 0.4335 - val_accuracy: 0.8697 - lr: 0.0010
Epoch 48/50
83/83 [==============================] - ETA: 0s - loss: 0.0240 - accuracy: 0.9989
Epoch 48: val_loss improved from 0.42710 to 0.42294, saving model to Transfer_Learning_classification.hdf5
83/83 [==============================] - 105s 1s/step - loss: 0.0240 - accuracy: 0.9989 - val_loss: 0.4229 - val_accuracy: 0.8709 - lr: 5.0000e-04
Epoch 49/50
83/83 [==============================] - ETA: 0s - loss: 0.0235 - accuracy: 0.9996
Epoch 49: val_loss did not improve from 0.42294
83/83 [==============================] - 114s 1s/step - loss: 0.0235 - accuracy: 0.9996 - val_loss: 0.4279 - val_accuracy: 0.8697 - lr: 5.0000e-04
Epoch 50/50
83/83 [==============================] - ETA: 0s - loss: 0.0224 - accuracy: 1.0000
Epoch 50: val_loss did not improve from 0.42294
83/83 [==============================] - 114s 1s/step - loss: 0.0224 - accuracy: 1.0000 - val_loss: 0.4299 - val_accuracy: 0.8686 - lr: 5.0000e-04
```






```
efficientnet_model_3.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 feature_extraction_layer (K  (None, 1280)             4049564
 erasLayer)

 output_layer (Dense)        (None, 35)                44835

=================================================================
Total params: 4,094,399
Trainable params: 44,835
Non-trainable params: 4,049,564
_________________________________________________________________
```

Credits

Image Scraper

https://github.com/ohyicong/Google-Image-Scraper - Github source for Google Image Scraper

https://gist.github.com/stephenhouser/c5e2b921c3770ed47eb3b75efbc94799 - Github source for Bing Image Scraper

Streamlit

https://www.youtube.com/watch?v=cF6rMXZcuCs – Youtube channel for streamlit fruit classification

https://github.com/Spidy20/Fruit_Vegetable_Recognition – Github source for streamlit fruit classification

Streamlit documentation – Documentation on how to use streamlit

Learning CNN

CNN Explainer (poloclub.github.io) – A good starting website for beginners to visualize the structure of CNN model

The CNN model contains 1 input layer 1 convolutional layer 1 max pooling layer 1 | Course Hero – Course Notes on CNN in details

(PDF) Food classification from images using convolutional neural networks (researchgate.net) – Research paper specifically on Food Classification

MIT 6.S191 (2020): Convolutional Neural Networks – YouTube – Youtube video from MIT lecturer on CNN

Applied Deep Learning - Part 4: Convolutional Neural Networks | by Arden Dertat | Towards Data Science

Convolutional Neural Networks — Image Classification w. Keras – LearnDataSci

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 - Architecture of Classification Model

Credits

Tensorflow

Food Classification Using Transfer Learning And TensorFlow (analyticsvidhya.com) – related to transfer learning

https://stackoverflow.com/ - Great place to find common solutions encountered during programming

https://tfhub.dev- Repository of trained machine learning models ready for fine-tuning and deployable

https://paperswithcode.com/task/image-classification - Find the rating of current state of the art pre-trained models

https://academy.zerotomastery.io/courses/ - online courses on deep learning

https://github.com/mrdbourke/tensorflow-deep-learning - course notes that teaches in-depth courses about CNN

https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html - Detailed explanation on the structure of EfficientNet and usage

https://drive.google.com/file/d/1UzOC_2dRrRJPN-hE0hiXTpQR0VWJXDMw/view?usp=sharing - Machine Learning with Sickit Learn and TensorFlow

Related Topics

https://www.nutritionix.com/database - Nutrition Info API which provides paid platform as a service for nutritional information for local foods

https://www.moh.gov.sg/resources-statistics/reports/national-survey-2019-20 - National Population Health Survey 2020 for diet related diseases