Table of Contents

Overview

Base Model (Non-Pretrained Model)

When beginning the assignment, my problem was to figure out the problem type of this assignment. After finding out that this is a Multiclass-Single Label problem, all the parameters that were affected by the problem type was changed.

The objective of the Base Model was to achieve a testing accuracy of minimum of 0.700 and above, as setting the right values for each hyperparameter is difficult, requiring a lot of time to fine tune each of it. Data augmentation was already added in the base model due to there being not enough images to train, validate and test. After building, training, and testing the models, here are some key data recorded from the model:

Recorded data type	Value	Epoch (Page 22)
Initial training accuracy	0.2136	1st Epoch, Figure 0.1
Initial validation accuracy	0.3130	1st Epoch, Figure 0.1
Final training accuracy	0.8917	150th Epoch, Figure 0.3
Final validation accuracy	0.7415	150th Epoch, Figure 0.3
Initial training loss	2.1371	1st Epoch, Figure 0.1
Initial validation loss	1.9308	1st Epoch, Figure 0.1
Lowest recorded training loss	0.3121	149th Epoch, Figure 0.3
Lowest recorded validation loss	0.8306	67th Epoch, Figure 0.2
Testing Accuracy	**0.714**	**-**

Base Model overfitted on the 25th epoch onwards, showing that it is still unstable and require regularization. However, it has a higher testing accuracy of 71.4%.

Model 1 (Non-Pretrained Model)

As the Base Model accuracy was not high enough, regularization, in the form of a dropout layer was added into Model 1. It was set to a value of 0.5. The other hyperparameters such as the total number of trainable parameters, learning rate and number of epochs all remained the same as this model is to solely test the effects of a dropout layer to the accuracy.

The key data recorded from the model are as follows:

Recorded data type	Value	Epoch (Page 30)
Initial training accuracy	0.2049	1st Epoch, Figure 1.1
Initial validation accuracy	0.3055	1st Epoch, Figure 1.1
Final training accuracy	0.8513	150th Epoch, Figure 1.5
Final validation accuracy	0.8010	150th Epoch, Figure 1.5
Initial training loss	2.1540	1st Epoch, Figure 1.1
Initial validation loss	1.9407	1st Epoch, Figure 1.1
Lowest recorded training loss	0.4387	148th Epoch, Figure 1.5
Lowest recorded validation loss	0.6923	132nd Epoch, Figure 1.4
Testing Accuracy	**0.772**	**-**

Overfitting can be clearly seen from Model 1 as it starts at around the 60th epoch and validation accuracy stayed between 0.7 and 0.8.

Generally, it was a less overfitted model compared to Model 1 while achieving a higher testing accuracy of 77.2%. However, it still needs to be fine-tuned.

Model 2 (Non-Pretrained Model)

Since Model 1 already has a high accuracy, the problem here is how to increase it further, as my objective is to get a higher accuracy after the other for each model. I decided to reduce the total number of parameters for Model 2 as it was a form of regularization too.

The key data recorded from the model are as follows:

Recorded data type	Value	Epoch (Page 37)
Initial training accuracy	0.2051	1st Epoch, Figure 2.1
Initial validation accuracy	0.3115	1st Epoch, Figure 2.1
Final training accuracy	0.8455	150th Epoch, Figure 2.4
Final validation accuracy	0.7085	150th Epoch, Figure 2.4
Initial training loss	2.1638	1st Epoch, Figure 2.1
Initial validation loss	1.9642	1st Epoch, Figure 2.1
Lowest recorded training loss	0.4430	149th Epoch, Figure 2.3
Lowest recorded validation loss	0.6919	146th Epoch, Figure 2.2
Testing Accuracy	**0.692**	**-**

As seen from the table, the final training accuracy and final validation accuracy is lower than Model 2, with the testing accuracy at 69.2% compared to 77.2% of Model 1.

Model 3 (Pretrained Model)

This is the base pretrained model, which is using convolutional neural network as it can predict images with a higher accuracy. The objective of this model is to observe if the pretrained model has a greater accuracy compared to non-pretrained. Approaching this model, VGG16 pretrained model was used as it provides the one of the highest accuracies of all the pretrained models.

For this model, I used data augmentation as well as fine tuning, combined with a sizable trainable parameter and learning rate to try and get the best testing accuracy.

Here are some of the key values recorded from the model:

Recorded data type	Value	Epoch (Page 44)
Initial training accuracy	0.3619	1st Epoch, Figure 3.1
Initial validation accuracy	0.5381	1st Epoch, Figure 3.1
Final training accuracy	0.8629	30th Epoch, Figure 3.3
Final validation accuracy	0.7944	30th Epoch, Figure 3.3
Initial training loss	1.855	1st Epoch, Figure 3.1
Initial validation loss	1.3713	1st Epoch, Figure 3.1
Lowest recorded training loss	0.4109	30th Epoch, Figure 3.3
Lowest recorded validation loss	0.5957	26th Epoch, Figure 3.2
Testing Accuracy	**0.782**	**-**

The testing accuracy of Model 3 is rather low for a pretrained model at only 78.2%.

Model 4 (Pretrained Model)

The problem with Model 3 is that its accuracy is slightly low for a pretrained model. The objective of this model is to increase the accuracy of the model. As a form of regularization, one dense layer was removed, and learning rate was increased by 10 times. This was to attempt to increase the accuracy of the model without increasing the number of epochs.

Here are some of the key values recorded from the model:

Recorded data type	Value	Epoch (Page 52)
Initial training accuracy	0.3867	1st Epoch, Figure 4.1
Initial validation accuracy	0.6315	1st Epoch, Figure 4.1
Final training accuracy	0.9016	30th Epoch, Figure 4.4
Final validation accuracy	0.8175	30th Epoch, Figure 4.4
Initial training loss	1.7488	1st Epoch, Figure 4.1
Initial validation loss	0.6315	1st Epoch, Figure 4.1
Lowest recorded training loss	0.3330	28th Epoch, Figure 4.3
Lowest recorded validation loss	0.6404	7th Epoch, Figure 4.2
Testing Accuracy	**0.800**	**-**

While Model 4 overfitted very quickly, it has an overall higher accuracy compared to Model 3. It achieved a testing accuracy of 80% compared to 78.2% of Model 3.

Data Preprocessing and Data Loading

Image Preprocessing Jupyter File

Before building and training the models, data preprocessing is essential as it prepares data for the model.

1. We first import tensorflow.keras to allow the file to identity we are working with keras. We then set the base directory as the current directory.

```
 #Set the base directory as the current directory
base_dir = os.getcwd()

#Set the base directory as where you save the downloaded food_images
image_dir = 'C:/!Ezra/Ngee Ann Poly/Year 2.1/Assignments/DL/my_food'
```

2. The preprocessor will then create three separate directories called Train, Test and Validation using this set of code. This will serve as the basis of where the images are taken from for the assignment.

```
# Directories for your training, validation and test splits
train_dir = os.path.join(base_dir, 'train')
os.mkdir(train_dir)
validation_dir = os.path.join(base_dir, 'validation')
os.mkdir(validation_dir)
test_dir = os.path.join(base_dir, 'test')
os.mkdir(test_dir)
```

3. The images will then be searched from “food-101” based on the text file that was assigned to each student.

```
# Assign the 10 types of food from your .txt file to a list variable 'food_list'

label_file = os.path.join(base_dir, '32.txt')
# Refer to the report Appendix
# Please enter the name of .txt file which contains a list of food assigned to you
# Make sure you save the .txt file in your base_dir

with open(label_file, 'r') as f:
    x = f.readlines()

food_list =[]
for item in x:
    if item == '\n':
        continue
    else:
        food_list.append(item.strip('\n'))
```

4. The preprocessor will then go into each of the food folders that were assigned to each of the students and separate the first 750 of each food for training. The preprocessor will create a separate folder for the respective foods and paste the images inside the train, validation, and test folder.

```
#copy the first 750 images to train folder
for item in food_list:
    train_food_dir = os.path.join(train_dir, item)
    os.mkdir(train_food_dir)
    img_list = os.listdir(os.path.join(image_dir, item))[:750]
    for fname in img_list:
        src = os.path.join(image_dir, item, fname)
        dst = os.path.join(train_food_dir, fname)
        shutil.copyfile(src, dst)
```

5. The next 200 images of each food are assigned for validation.

```
#copy the following 200 images [750:950] to validation folder
for item in food_list:
    validation_food_dir = os.path.join(validation_dir, item)
    os.mkdir(validation_food_dir)
    img_list = os.listdir(os.path.join(image_dir, item))[750:950]
    for fname in img_list:
        src = os.path.join(image_dir, item, fname)
        dst = os.path.join(validation_food_dir, fname)
        shutil.copyfile(src, dst)
```

6. And the last 50 of each food images are assigned for testing.

```
#copy the remaining 50 images [950:1000] to test folder
for item in food_list:
    test_food_dir = os.path.join(test_dir, item)
    os.mkdir(test_food_dir)
    img_list = os.listdir(os.path.join(image_dir, item))[950:1000]
    for fname in img_list:
        src = os.path.join(image_dir, item, fname)
        dst = os.path.join(test_food_dir, fname)
        shutil.copyfile(src, dst)
```

Assignment 1 Jupyter File Data Loading

Importing Packages

1. The command below will be first executed and imported. This is to tell the Jupyter notebook that we will be utilizing the keras network library.

```
# Import the Required Packages
from tensorflow import keras
```

Data Loading

2. The code represented in such a manner that I can assign my base directory as where the imported images were kept, which was in the train, validation, and test folders. These folders will be assigned and used as the base train, validation, and test directory for the assignment.

```
import os
base_dir = 'C:/!Ezra/Ngee Ann Poly/Year 2.1/Assignments/DL/img'

train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')
```

Develop the Images Classification Models

Base Model (food_model_BASE.h5)

```
# Build the Model
from tensorflow.keras.models import Sequential
from tensorflow.keras import models
from tensorflow.keras import layers

img_size = 150
model = models.Sequential()
model.add(layers.Conv2D(40, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(80, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(160, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(160, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(500, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

model.summary()
```

For my base model, I decided to use MaxPooling2D and Conv2D to augment the capacity of the network, reducing the size of the feature maps so that they are not too large when it reaches the flatten layer, reducing total trainable parameters from 18,000,500 to 3,920,500 in the last dense layer. The role of MaxPooling is to down sample the feature maps, extracting windows from the input feature maps and outputting the max value of each window. A Conv2D layer creates a convolution kernel which helps to produce a tensor of outputs.

An image size of 150 by 150 pixels is used to train and validate the images. A flatten layer is placed to convert the data into a 1-dimension array for inputting it into the next later. The output of the convolutional layer creates a single long feature vector.

In this assignment, since we are using images and is a multiclass, single label problem type. Hence, “softmax” is needed as last layer activation. And since there are 10 food categories, there needs to be 10 output layers to allow the model to accurately categorize the images.

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 148, 148, 40)      1120
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 74, 74, 40)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 72, 72, 80)        28880
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 36, 36, 80)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 34, 34, 160)       115360
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 17, 17, 160)       0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 15, 15, 160)       230560
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 7, 7, 160)         0
_________________________________________________________________
flatten_5 (Flatten)          (None, 7840)              0
_________________________________________________________________
dense_11 (Dense)             (None, 500)               3920500
_________________________________________________________________
dense_12 (Dense)             (None, 10)                5010
=================================================================
Total params: 4,301,430
Trainable params: 4,301,430
Non-trainable params: 0
```

These are the results of the Model 1 parameters summary, with the total parameters coming to just about 4,301,430. This is more than enough and does not need any more methods, like K-fold validation, to try and increase the number of images available.

Training Base Model

```
# Train the Model

from tensorflow.keras import optimizers
from tensorflow.keras.preprocessing.image import ImageDataGenerator

model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(lr = 2e-4),
              metrics=['acc'])

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(150, 150),
        batch_size=50,
        class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='categorical')

history = model.fit_generator(
      train_generator,
      steps_per_epoch=150,
      epochs=150,
      validation_data=validation_generator,
      validation_steps=100)
```

In this model, categorical crossentropy was used as the loss function and RMS prop as the optimizer with a learning rate of 2e-4.

Data augmentation was also used in this model to further expand the total number of images as the initial number of images in the folders were not enough. This is seen by “train_datagen” variable as it is the code that enables images to be augmented in a way such that it is accepted as new images by the model and valid for training and validation.

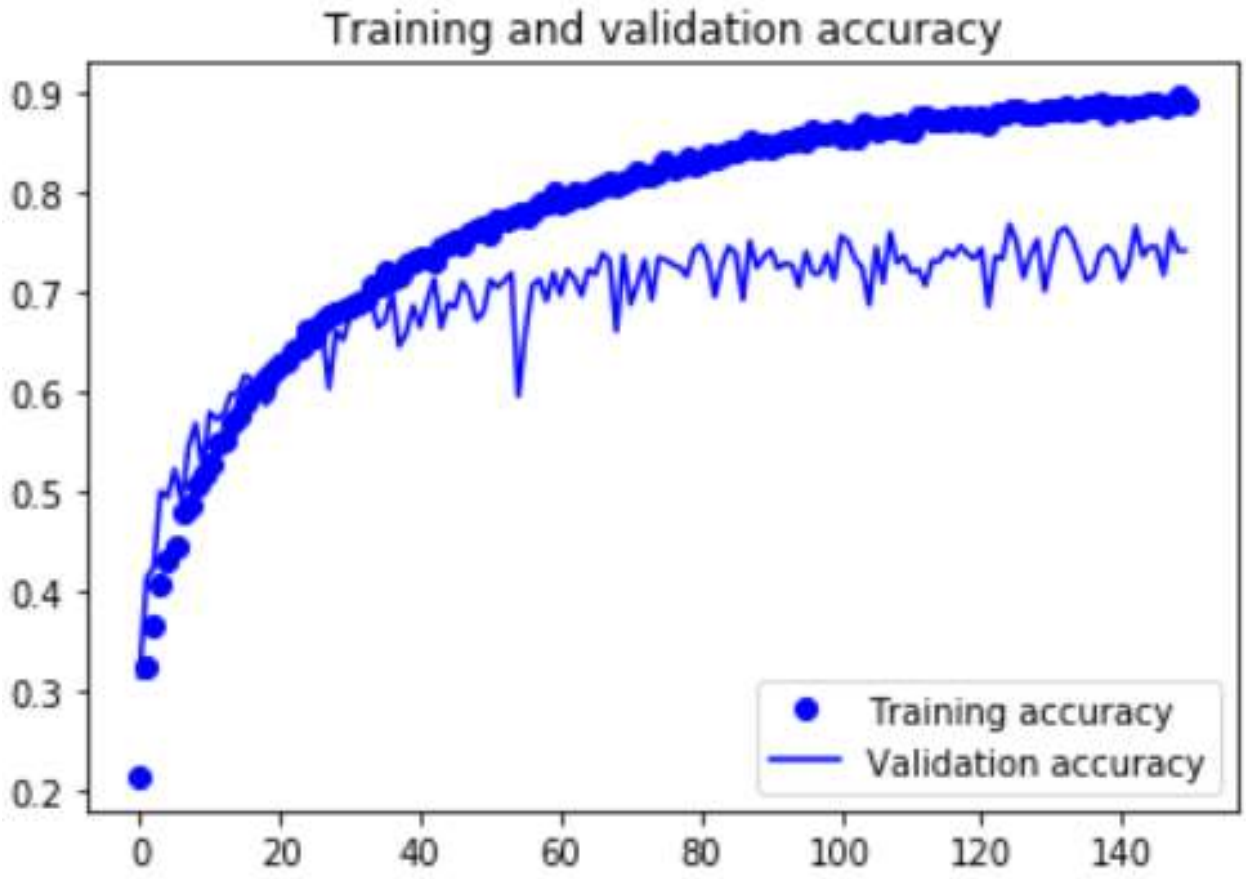
Class mode is set as categorical as it is a multiclass, single label problem type. This is used as the assignment requires us to classify the images and train them and categorize them respective to their own food class, thus categorical class mode was used.

Plotting the graphs of all the models

```
# Plot the Training and Validation Accuracy & Loss Scores

import matplotlib.pyplot as plt
%matplotlib inline

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

This shows the code to plot and show the graph in the Jupyter files. The accuracies and losses for training and validation are captured and plotted on a graph in accordance to the length of epochs determined by us. Labels and the indication points of the graphs can be determined, for example, changing ‘bo’ to ‘b+’ would change a circle point to a crossed point on the graph when it is plotted.

Graph of Training & Validation Accuracy for Base Model



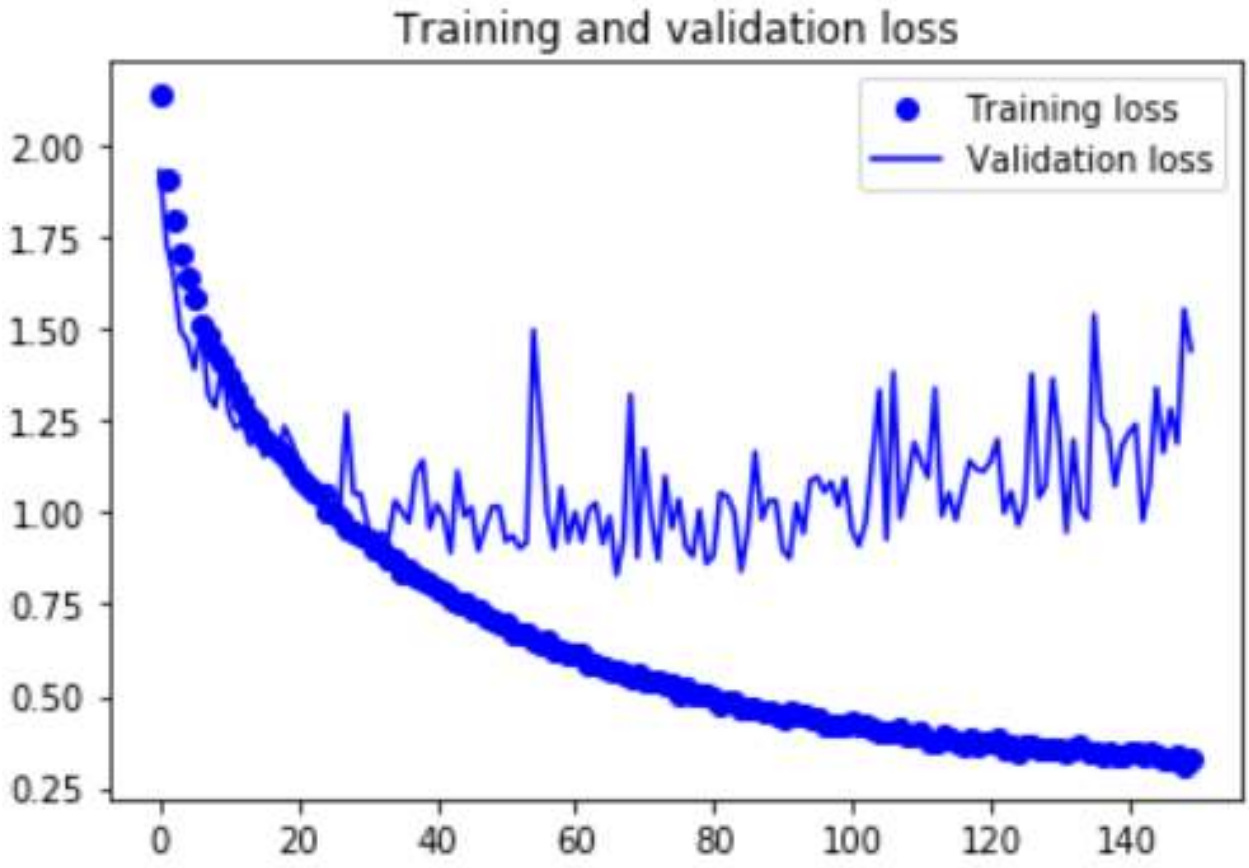
The Base Model shows the training accuracy to be way higher than validation accuracy. This model overfitted on the 25th epoch (Figure 0.2), afterwards, the training accuracy continuously increases stably and consistently, while validation accuracy did not increase much in accuracy. With the initial training and validation accuracy at 0.2136 and 0.3130 respectively (Figure 0.1), this is quite a regular accuracy for a non-pretrained model as the values are slightly lower than a pretrained model. With the final training and validation accuracy hitting 0.8917 and 0.7415 respectively (Figure 0.6), it shows that the model has overfitted by a lot and requires regularization. This can be seen by the validation graph not increasing while the training graph continues to increase.

Base Model also overfitted on the 25th epoch, which is quite early in the training when compared to Model 1 and 2 at 60th and 34th epoch, respectively.

There is a huge dip in validation accuracy in this model, on the 55th epoch (Figure 0.3), which hit an accuracy of 0.5955. This might be caused by images that are not related to the others being placed there, or the images were too hard for the model to learn.

The training and validation accuracies are quite high for a non-pretrained model. In general, the model was rather stable and did not have spikes and dips in accuracy which suggests that this is a stable model that is suitable to be used as the pretrained base model.

Graph of Training & Validation Accuracy for Base Model



This graph shows the training and validation loss of the base model, which has an initial training and validation loss of 0.2138 and 0.3130 (Figure 0.1) and a final loss of 0.3121 on the 149th epoch for training (Figure 0.5) and 0.8306 on the 67th epoch for validation (Figure 0.4). Compared to the pretrained Model 3, this training and validation loss is higher. This is due to the familiarity and the optimization of the pretrained model being better than a non-pretrained model.

Base Model validation accuracy is also increasing slowly after hitting the lowest point for validation loss. This suggests that the model is good at making the decisions as suggested from the training loss graph reaching a very low loss value, but not confident in making decisions, as suggested by the gentle increase in validation graph.

There's a huge increase in accuracy on the 55th epoch (Figure 0.3), which corresponds to the dip in accuracy in the validation graph, as a decrease in accuracy can be seen as a loss in quality of the images, hence the increased loss value.

With the lowest training and validation loss recorded at 0.3121 on the 149th (Figure 0.5) and 0.8306 on the 67th epoch (Figure 0.4), respectively. This model requires high amounts of regularizations to reduce overfitting.

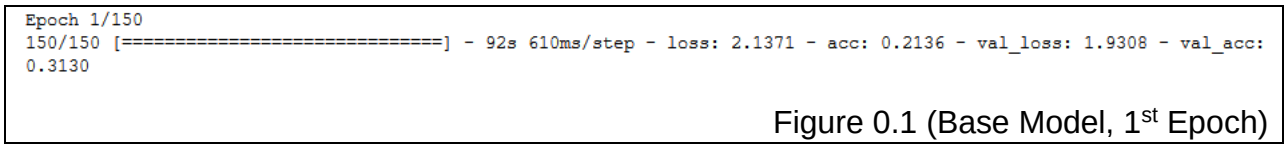


Figure 0.1 (Base Model, 1st Epoch)

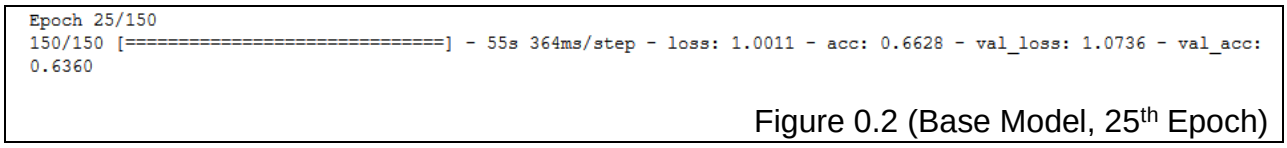


Figure 0.2 (Base Model, 25th Epoch)

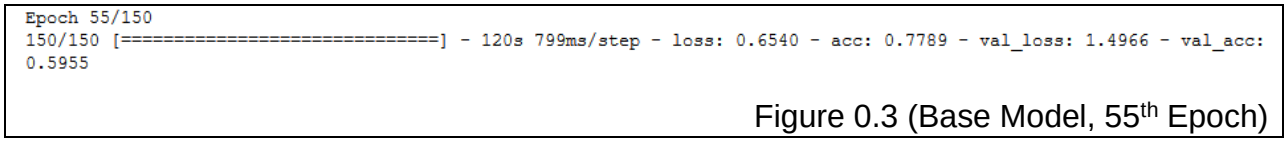


Figure 0.3 (Base Model, 55th Epoch)

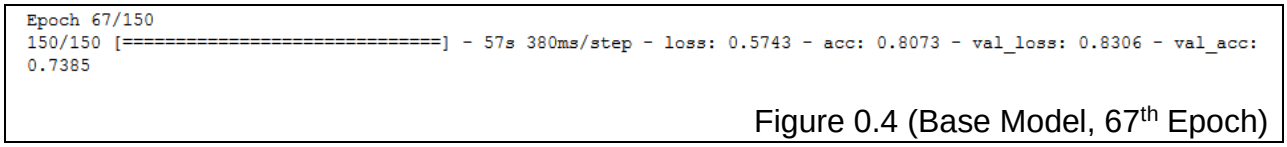


Figure 0.4 (Base Model, 67th Epoch)

Figure 0.5 (Base Model, 149th Epoch)

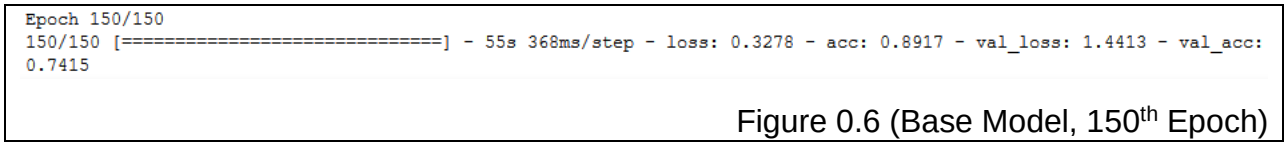


Figure 0.6 (Base Model, 150th Epoch)

These are some key values to take note of for Base Model:

Recorded data type	Value	Epoch (Page 22)
Initial training accuracy	0.2136	1st Epoch, Figure 0.1
Initial validation accuracy	0.3130	1st Epoch, Figure 0.1
Final training accuracy	0.8917	150th Epoch, Figure 0.6
Final validation accuracy	0.7415	150th Epoch, Figure 0.6
Initial training loss	2.1371	1st Epoch, Figure 0.1
Initial validation loss	1.9308	1st Epoch, Figure 0.1
Lowest recorded training loss	0.3121	149th Epoch, Figure 0.5
Lowest recorded validation loss	0.8306	67th Epoch, Figure 0.4
Testing Accuracy	**0.714**	**-**

First Model (food_model_1.h5)

Building Model 1

```
# Build the Model
from tensorflow.keras.models import Sequential
from tensorflow.keras import models
from tensorflow.keras import layers

img_size = 150
model1 = models.Sequential()
model1.add(layers.Conv2D(40, (3, 3), activation='relu',
                         input_shape=(150, 150, 3)))
model1.add(layers.MaxPooling2D((2, 2)))
model1.add(layers.Conv2D(80, (3, 3), activation='relu'))
model1.add(layers.MaxPooling2D((2, 2)))
model1.add(layers.Conv2D(160, (3, 3), activation='relu'))
model1.add(layers.MaxPooling2D((2, 2)))
model1.add(layers.Conv2D(160, (3, 3), activation='relu'))
model1.add(layers.MaxPooling2D((2, 2)))
model1.add(layers.Flatten())
model1.add(layers.Dropout(0.5))
model1.add(layers.Dense(500, activation='relu'))
model1.add(layers.Dense(10, activation='softmax'))

model1.summary()
```

Model 1 has a similar build to the Base Model, except that it has a dropout layer included. This is to reduce the training capabilities of the model by randomly dropping out certain neurons, including hidden ones. By doing so, the model will be required to rely on itself to make the predictions of the image classification without the help of the neurons. Hence, increasing the overall accuracy of the model, as well as decreasing the overfitting of the model.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 40)      1120
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 40)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 80)        28880
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 80)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 160)       115360
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 160)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 160)       230560
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 160)         0
_________________________________________________________________
flatten (Flatten)            (None, 7840)              0
_________________________________________________________________
dropout (Dropout)            (None, 7840)              0
_________________________________________________________________
dense (Dense)                (None, 500)               3920500
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5010
=================================================================
Total params: 4,301,430
Trainable params: 4,301,430
Non-trainable params: 0
```

As seen in this Model 1 summary, there was no difference between the number of trainable parameters of Base Model and Model 1, since it has the same build, with the only difference being the inclusion of a dropout layer.

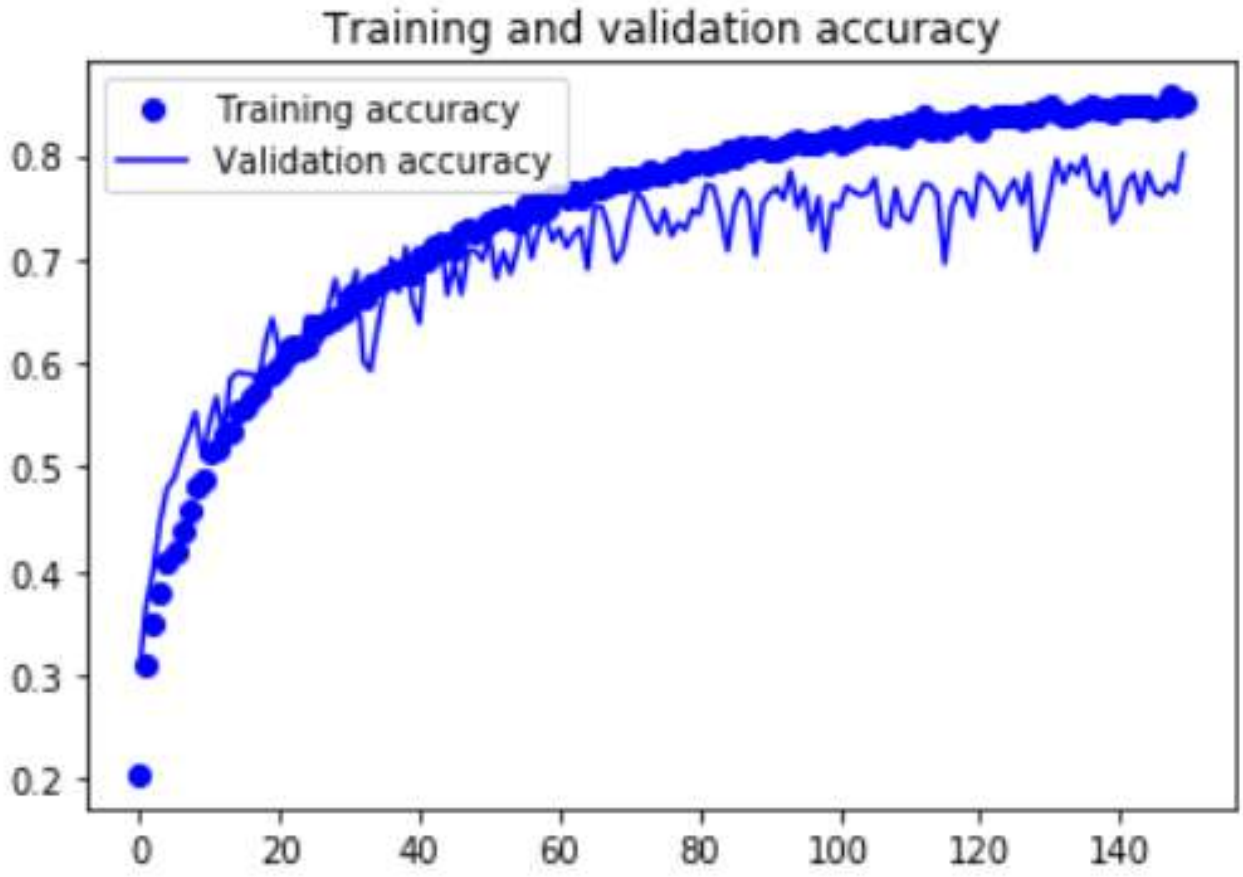
That is the main difference between Base Model and Model 1, as all other hyperparameters such as learning rate, number of epochs and number of total trainable parameters remain the same.

Training Model 1

```
# Train the Model

from tensorflow.keras import optimizers
from tensorflow.keras.preprocessing.image import ImageDataGenerator

model1.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(lr = 2e-4),
              metrics=['acc'])

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(150, 150),
        batch_size=50,
        class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='categorical')

history = model1.fit_generator(
      train_generator,
      steps_per_epoch=150,
      epochs=150,
      validation_data=validation_generator,
      validation_steps=100)
```

Model 1 training code is the same as the one found in Base Model. Hence, this model was made to solely see the effects of a dropout layer on a model accuracy and its overfitting.

Graph of Model 1 (Training and Validation Accuracy)



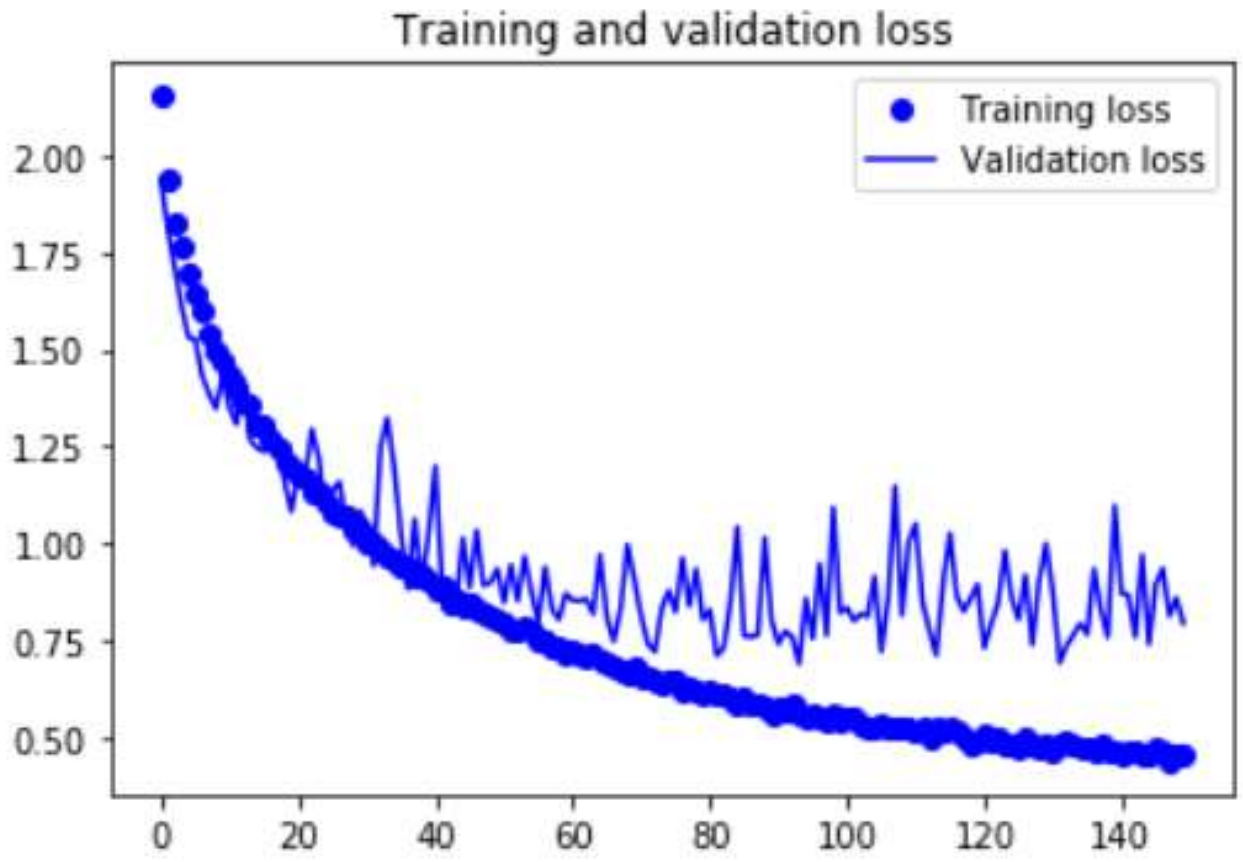
This graph shows the stable increase of accuracy of the model while training. With the initial training accuracy of 0.2049 (Figure 1.1) and final training accuracy of 0.8513 (Figure 1.6). The model does not show drastic changes or show signs of instability during training. This might be because of the low learning rate of the model, hence contributing to the well-shaped graph.

The validation accuracy starts at 0.3055 (Figure 1.1) and has a final accuracy of 0.8010 (Figure 1.6). The validation accuracy was initially higher than the training accuracy but as the model trains further, the training accuracy starts to have a higher accuracy starting from around the 23rd epoch (Figure 1.3).

Model 1 has a higher overall accuracy compared to Base Model. This is likely due to the model using its prediction capabilities and not the memorization capabilities that comes from the training images when dropout was included in the model.

This model show signs of overfitting as it is evident from the 60th epoch onwards. This proves that this model is much more resistant to overfitting compared to the Base Model as Model 1 overfitted much later compared to Base Model, showing the positive effects of a dropout layer for a model.

Graph of Model 1 (Training and Validation Loss)



This graph shows the training and validation loss. As seen, the training validation loss was higher than validation loss initially at 2.1540 and 1.9407 respectively (Figure 1.1). However, training loss is lower than validation loss starting at the 13th epoch, where training loss was 1.3703 and validation loss was 1.4268 (Figure 1.2). Model 1 shows a lowest recordedd loss of 0.4387 on the 148th epoch for training (Figure 1.5) and 0.6923 on the 132rd epoch for validation (Figure 1.4). Although towards the end of the model loss graph seems that the validation loss is higher than training loss, the gradual loss of the training and validation seems rather stable.

The initial point of training and validation loss is slightly higher compared to Base Model, while Base Model had the lowest recorded training loss while Model 1 had the lowest recorded validation loss between the two models.

In general, Model 1 has a good testing accuracy and is more resistent to overfitting than Base Model. Thus, it is a good model to be used as the best model.

Figure 1.1 (Model 1, 1st Epoch)

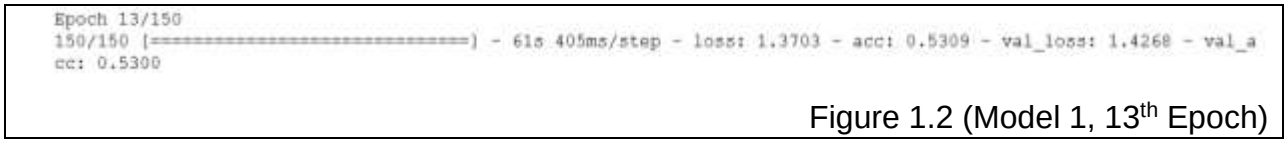


Figure 1.2 (Model 1, 13th Epoch)

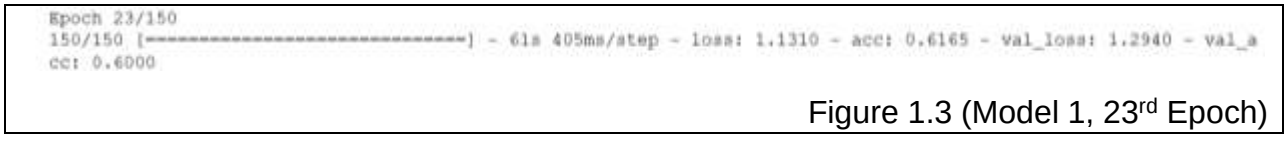


Figure 1.3 (Model 1, 23rd Epoch)

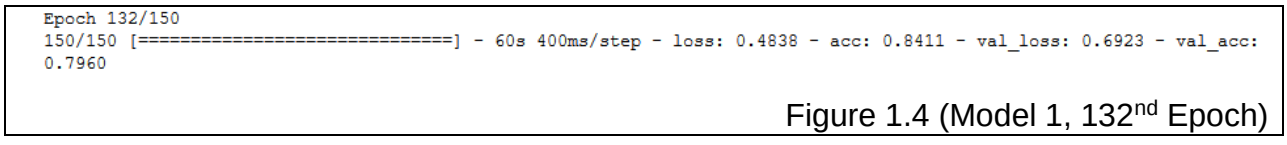


Figure 1.4 (Model 1, 132nd Epoch)

Figure 1.5 (Model 1, 148th Epoch)

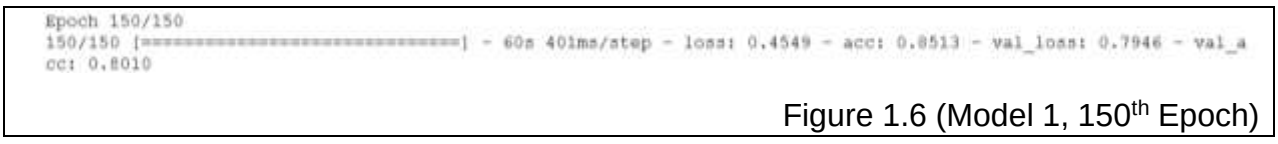


Figure 1.6 (Model 1, 150th Epoch)

These are some key values to take note of for Model 1:

Recorded data type	Value	Epoch (Page 30)
Initial training accuracy	0.2049	1st Epoch, Figure 1.1
Initial validation accuracy	0.3055	1st Epoch, Figure 1.1
Final training accuracy	0.8513	150th Epoch, Figure 1.5
Final validation accuracy	0.8010	150th Epoch, Figure 1.5
Initial training loss	2.1540	1st Epoch, Figure 1.1
Initial validation loss	1.9407	1st Epoch, Figure 1.1
Lowest recorded training loss	0.4387	148th Epoch, Figure 1.5
Lowest recorded validation loss	0.6923	132nd Epoch, Figure 1.4
Testing Accuracy	**0.772**	**-**

Second Model (food_model_2-1.h5)

Building Model 2

```
# Build the Model
from tensorflow.keras.models import Sequential
from tensorflow.keras import models
from tensorflow.keras import layers

img_size = 150
model2 = models.Sequential()
model2.add(layers.Conv2D(40, (3, 3), activation='relu',
                         input_shape=(150, 150, 3)))
model2.add(layers.MaxPooling2D((2, 2)))
model2.add(layers.Conv2D(80, (3, 3), activation='relu'))
model2.add(layers.MaxPooling2D((2, 2)))
model2.add(layers.Conv2D(160, (3, 3), activation='relu'))
model2.add(layers.MaxPooling2D((2, 2)))
model2.add(layers.Conv2D(160, (3, 3), activation='relu'))
model2.add(layers.MaxPooling2D((2, 2)))
model2.add(layers.Flatten())
model2.add(layers.Dropout(0.5))
model2.add(layers.Dense(350, activation='relu'))
model2.add(layers.Dense(10, activation='softmax'))

model2.summary()
```

Model 2 has a similar build to Base Model and Model 1. However, it has a lower overall number of parameters compared to the previous two models. This is because the second last dense layer filter numbers are changed from 500 on Base Model and Model 1 to 350 on Model 2.

This reduction in filters is to test the effects reducing trainable parameters on the model. All other hyperparameters in Model 2 is the same as Model 1, even the dropout layer, as this model is a continuation of Model 1.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 148, 148, 40)      1120
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 74, 74, 40)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 72, 72, 80)        28880
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 36, 36, 80)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 34, 34, 160)       115360
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 17, 17, 160)       0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 15, 15, 160)       230560
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 7, 7, 160)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 7840)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 7840)              0
_________________________________________________________________
dense_2 (Dense)              (None, 350)               2744350
_________________________________________________________________
dense_3 (Dense)              (None, 10)                3510
=================================================================
Total params: 3,123,780
Trainable params: 3,123,780
Non-trainable params: 0
```

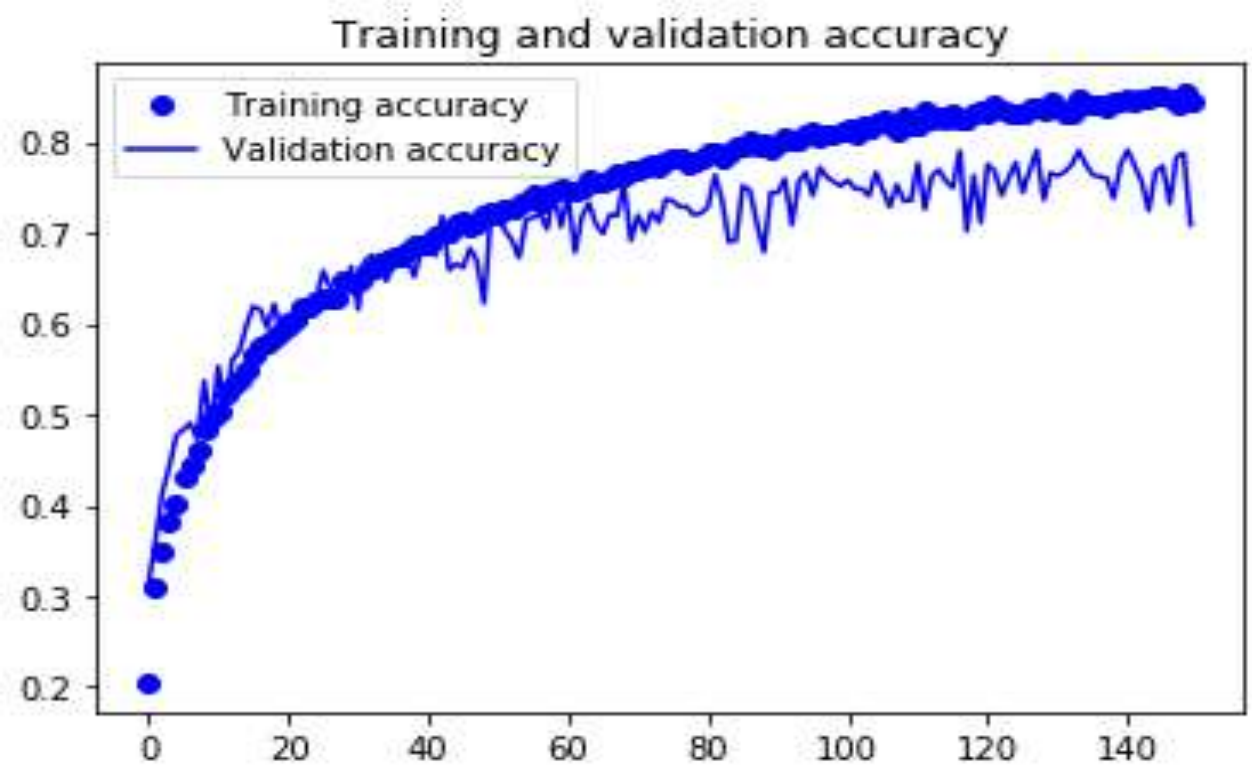
This shows the model summary of Model 2, with the total trainable parameter adding up to 3,123,780 due to 350 filters used on the second last dense layer. This is far lesser compared to the 4,301,430 trainable parameters found in Base Model and Model 1.

Training Model 2

```
# Train the Model

from tensorflow.keras import optimizers
from tensorflow.keras.preprocessing.image import ImageDataGenerator

model2.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(lr = 2e-4),
              metrics=['acc'])

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(150, 150),
        batch_size=50,
        class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='categorical')

history = model2.fit_generator(
      train_generator,
      steps_per_epoch=150,
      epochs=150,
      validation_data=validation_generator,
      validation_steps=100)
```

The learning rate remained at 2e-4, with 150 epochs, which is the same as Base Model and Model 1.

Graph of Model 2 (Training and Validation Accuracy)

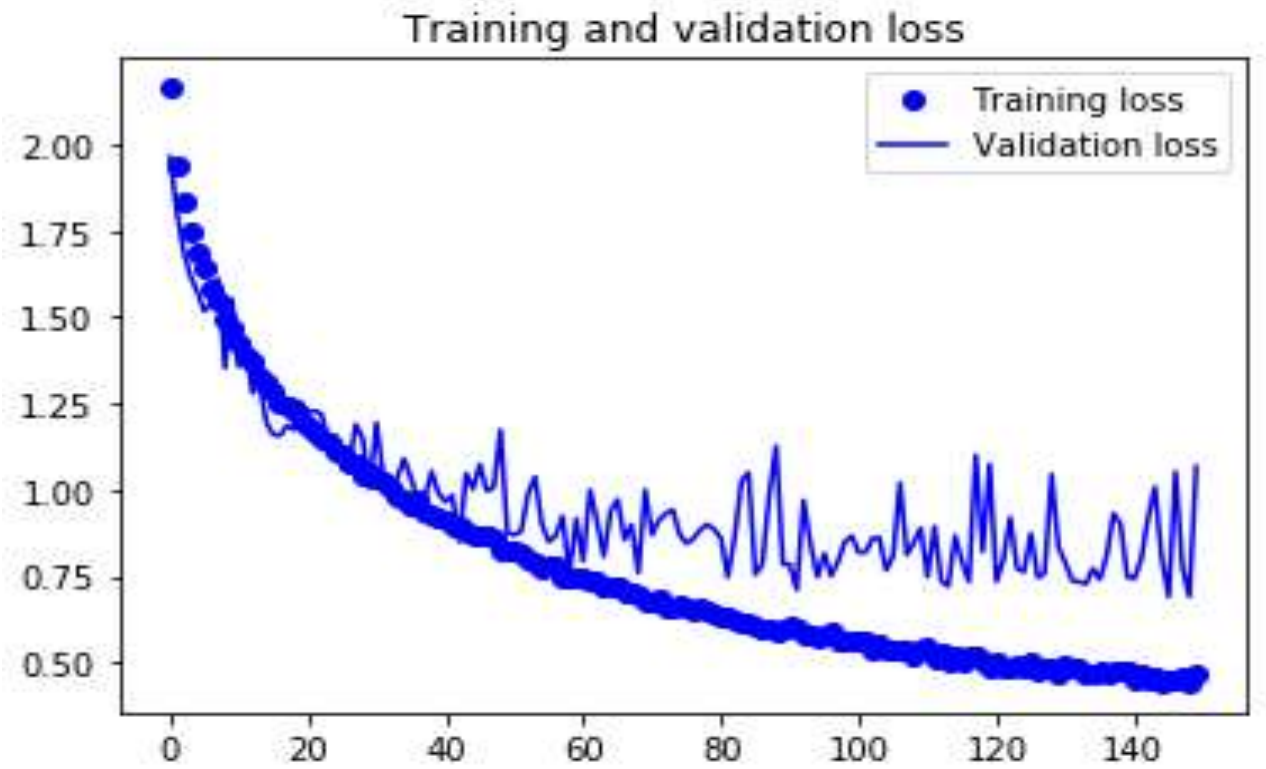


This is the training and accuracy graph of Model 2. Overfitting can be clearly seen, but it is not at much as Base Model. Though Model 2 have a high starting accuracy, at 0.2051 and 0.3115 for training and validation accuracy respectively (Figure 2.1). This did not progress well since the final training and validation accuracy reached only 0.8455 and 0.7085 respectively (Figure 2.4). This is lower in comparsion to Base Model and Model 1 final validation accuracy, at 0.7415 (Figure 0.6) and 0.8010 (Figure 1.5) respectively. The decrease in trainable parameters and the dropout layer might have caused the model to lose too much information, thus was not able to validate the images efficiently.

The final testing accuracy for Model 2 was 69.2%, compared to 77.2% of Model 1 and 71.4% of Base Model.

From the failure of this model, a lesson learnt is that too much regularization can harm the model instead of doing good, by producing models with lower accuracies.

Graph of Model 2 (Training and Validation Loss)



The graph shows a stable decrease of training loss while validation loss still sees some spikes towards the second half of the epochs. This can also be explained by the loss of information of the images due to dropout and less trainable parameters. It started with a training and validation loss of 2.1638 and 1.9642 (Figure 2.1), which is only slightly higher than both Base Model and Model 1. The lowest recorded training and validation loss is 0.4430 on the 149th epoch (Figure 2.3) and 0.6919 on the 146th epoch (Figure 2.2). This is slightly lower than both training and accuracy for Model 1, but only lower than the validation of Base Model. Base Model still has lower training loss as it does not have a dropout layer.

In general, this model was negatively regularized in a sense, that the accuracy turned for the worse due to over-regularization.

Figure 2.1 (Model 2, 1st Epoch)

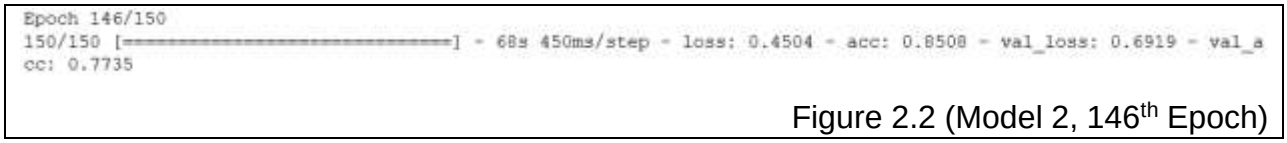


Figure 2.2 (Model 2, 146th Epoch)

Figure 2.3 (Model 2, 149th Epoch)

Figure 2.4 (Model 2, 150th Epoch)

These are some key values to take note of for Model 2:

Recorded data type	Value	Epoch (Page 37)
Initial training accuracy	0.2051	1st Epoch, Figure 2.1
Initial validation accuracy	0.3115	1st Epoch, Figure 2.1
Final training accuracy	0.8455	150th Epoch, Figure 2.4
Final validation accuracy	0.7085	150th Epoch, Figure 2.4
Initial training loss	2.1638	1st Epoch, Figure 2.1
Initial validation loss	1.9642	1st Epoch, Figure 2.1
Lowest recorded training loss	0.4430	149th Epoch, Figure 2.3
Lowest recorded validation loss	0.6919	146th Epoch, Figure 2.2
Testing Accuracy	**0.692**	**-**

Third Model (food_model_3.h5)

Building Model 3

```
# Build the Model
import os
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras.applications import VGG16
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import regularizers
from tensorflow.keras import optimizers

img_size = 150

vgg16base1 = VGG16(weights='imagenet',
                   include_top=False,
                   input_shape=(150, 150, 3))

model3 = models.Sequential()
model3.add(vgg16base2)
model3.add(layers.Flatten())
model3.add(layers.Dense(512, activation='relu'))
model3.add(layers.Dense(256, activation='relu'))
model3.add(layers.Dense(10, activation='softmax'))

vgg16base1.trainable = True

set_trainable = False
for layer in vgg16base1.layers:
    if layer.name == 'block5_conv1':
        set_trainable = True # after black5_conv1, set_trainable becomes True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False

model3.compile(loss='categorical_crossentropy',
        optimizer=optimizers.RMSprop(lr=2e-5),
        metrics=['acc'])

model3.summary()
```

Model 3 is also known as my pretrained base model, which is using the VGG16 pretrained model. The image size used is 150 by 150 pixels. Data augmentation was used as the number of images in the folder are not enough. The learning rate was set to 2e-5 since I ran it running using a learning rate of 2e-4 before and it overfitted very quickly.

Training Model 3

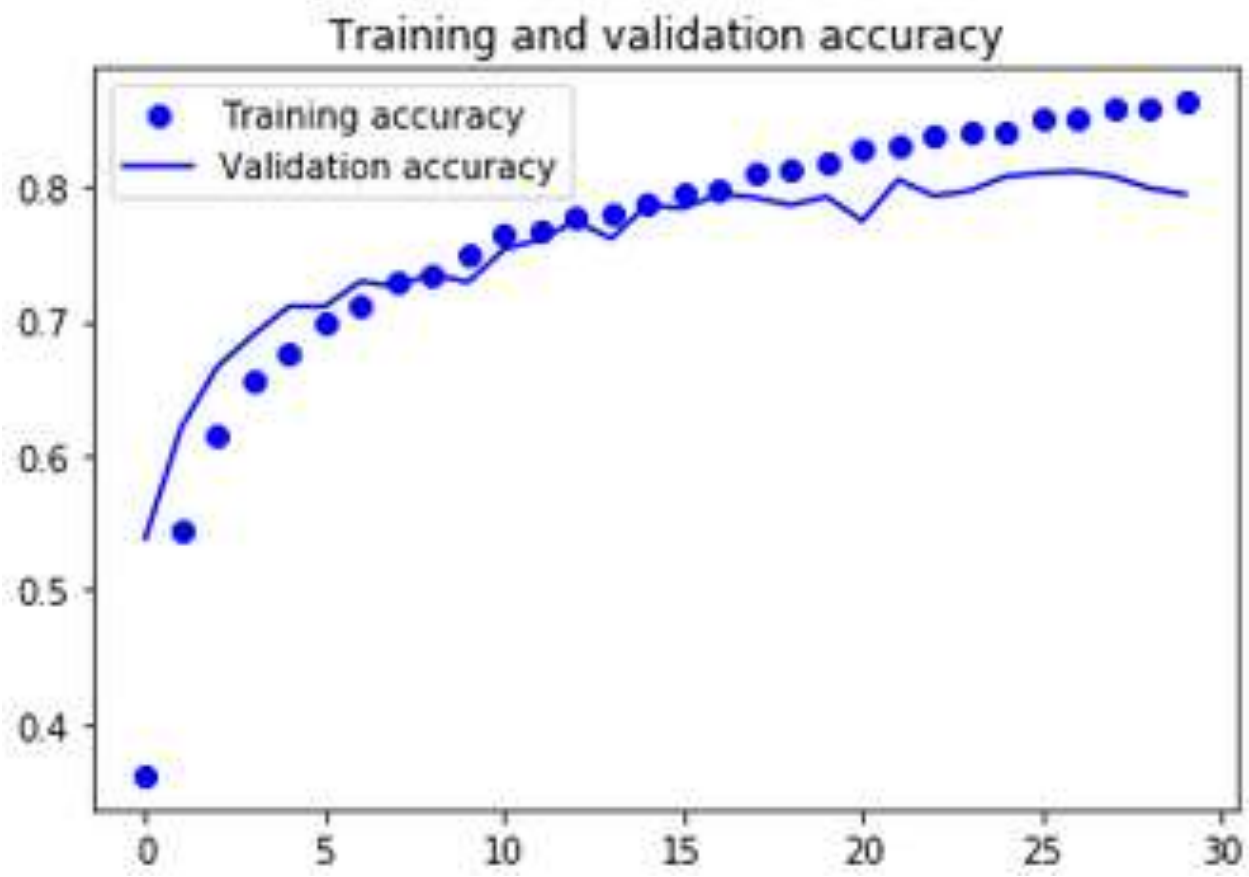
```
# Train the Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import optimizers

train_datagen = ImageDataGenerator(
      rescale=1./255,
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(150, 150),
        batch_size=50,
        class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=50,
        class_mode='categorical')

model3.compile(loss='categorical_crossentropy',
        optimizer=optimizers.RMSprop(lr=2e-5),
        metrics=['acc'])

history = model3.fit_generator(
      train_generator,
      steps_per_epoch=150,
      epochs=30,
      validation_data=validation_generator,
      validation_steps=40,
      verbose=1)
```

This code is almost like the one found in Base Model, except for the changes in the learning rate from 2e-4 to 2e-5, and the number of epochs has been changed from 150 to 30 as a form of regularization called Early Stopping.

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_2 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_4 (Dense)              (None, 512)               4194816
_________________________________________________________________
dense_5 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_6 (Dense)              (None, 10)                2570
=================================================================
Total params: 19,043,402
Trainable params: 11,408,138
Non-trainable params: 7,635,264
```

This shows the model summary of Model 3. And there is a huge increase of parameters in the pretrained models compared to non-pretrained. While some of the parameters remain untrainable, 11,408,138 parameters have been set to trainable just by unfreezing the last three layers of the model. This has allowed for the model to be built for a multiclass, single label problem by changing the final activation function to “softmax”.

Graph of Model 3 (Training and Validation Accuracy)

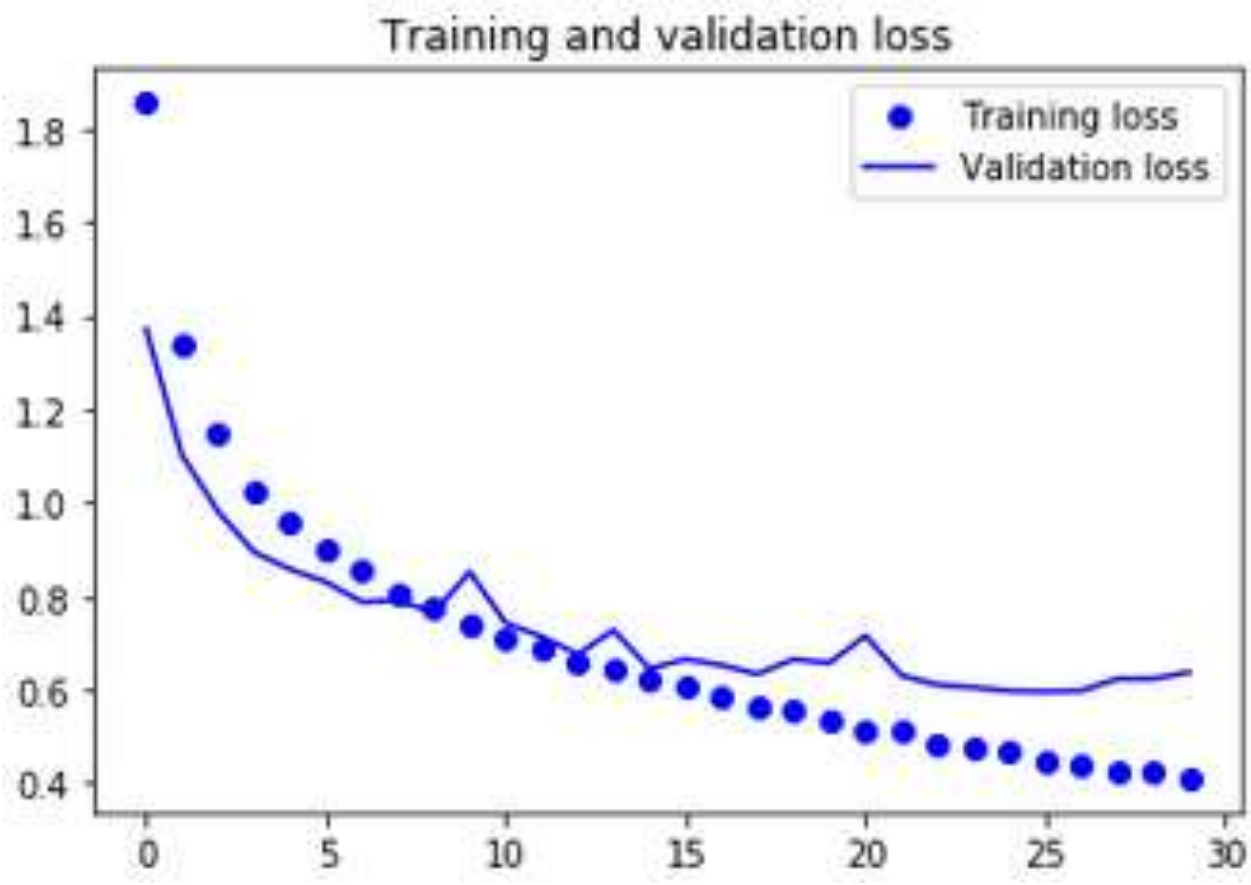


The graph shows a stable increase in training accuracy, which almost flattens nearing the end of the model, showing the stability. The validation accuracy had some minor spikes and dips in accuracy which are negligible. The starting accuracies are 0.3619 for training and 0.5381 for validation (Figure 3.1). This is far higher compared to non-pretrained Base Model, which started at 0.2136 and 0.3130 for training and validation (Figure 2.1), respectively. This tells me that a pretrained model is already more well optimized even before having any form of regularization added to it. Without any regularization, it overfitted on the 8th epoch. This means that the model can be far more optimized to achieve greater accuracy.

The final training and validation accuracy are 0.8629 and 0.7944. For Model 3, the final training accuracy is lower while final validation accuracy is higher than Base Model. The lower final training accuracy may mean that Model 3 is more resistant to overfitting without regularization compared to Base Model.

The testing accuracy of Model 3 is 78.2%, which is higher compared to 71.4% of Base Model. This shows that the training accuracy of models matters, but not as much as validation accuracy, since it is a way the models use to “test” its accuracy in preparation for the testing images.

Graph of Model 3 (Training and Validation Accuracy)



This training loss graph is rather stable compared to the validation loss graph, which has three spikes of accuracy. The initial values of training and validation graphs are 0.8629 and 0.7944 respectively (Figure 3.1). This is rather low compared to Base Model at 2.1371 and 1.9308, respectively (Figure 0.1). This is because the pretrained model has a better optimization. The validation accuracy continues to decrease throughout the epochs and only slightly increased in loss towards the end of the epochs. The lowest recorded training and validation loss are 0.4109 and 0.5957, respectively. The lowest training loss value is not lower than Base Model, most likely due to Base Model not having any optimization and is able to memorize most of the details of the training images well enough that the training accuracy is lower than Model 3. However, Model 3 has the lower lowest validation

loss between both Base Model and Model 3 due to the model accurately predicting the images after training, finding patterns and details in the images. Instead of Base Model, which trains by memorizing the features of the images, and then performs not as well on new images as they might not have the exact same features.

Figure 3.1 (Model 3, 1st Epoch)

Figure 3.2 (Model 3, 26th Epoch)

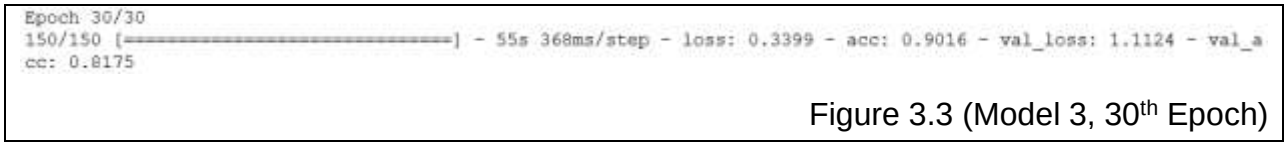


Figure 3.3 (Model 3, 30th Epoch)

These are some key values to take note of for Model 3:

Recorded data type	Value	Epoch (Page 44)
Initial training accuracy	0.3619	1st Epoch, Figure 3.1
Initial validation accuracy	0.5381	1st Epoch, Figure 3.1
Final training accuracy	0.8629	30th Epoch, Figure 3.3
Final validation accuracy	0.7944	30th Epoch, Figure 3.3
Initial training loss	1.855	1st Epoch, Figure 3.1
Initial validation loss	1.3713	1st Epoch, Figure 3.1
Lowest recorded training loss	0.4109	30th Epoch, Figure 3.3
Lowest recorded validation loss	0.5957	26th Epoch, Figure 3.2
Testing Accuracy	**0.782**	**-**

Fourth Model (food_model_4.h5)

Building Model 4

```
# Build the Model
import os
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras.applications import VGG16
from tensorflow.keras.preprocessing.image import ImageDataGenerator

img_size = 150

vgg16base1 = VGG16(weights='imagenet',
                   include_top=False,
                   input_shape=(150, 150, 3))

model4 = models.Sequential()
model4.add(vgg16base1)
model4.add(layers.Flatten())
model4.add(layers.Dense(256, activation='relu'))
model4.add(layers.Dense(10, activation='softmax'))

vgg16base1.trainable = True

set_trainable = False
for layer in vgg16base1.layers:
    if layer.name == 'block5_conv1':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False

model4.summary()
```

Model 4 has a similar build to Model 3, with the primary difference being one removed dense layer and reduced learning rate. The other hypermeters used are the same as Model 3.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               2097408
_________________________________________________________________
dense_3 (Dense)              (None, 10)                2570
=================================================================
Total params: 16,814,666
Trainable params: 9,179,402
Non-trainable params: 7,635,264
```

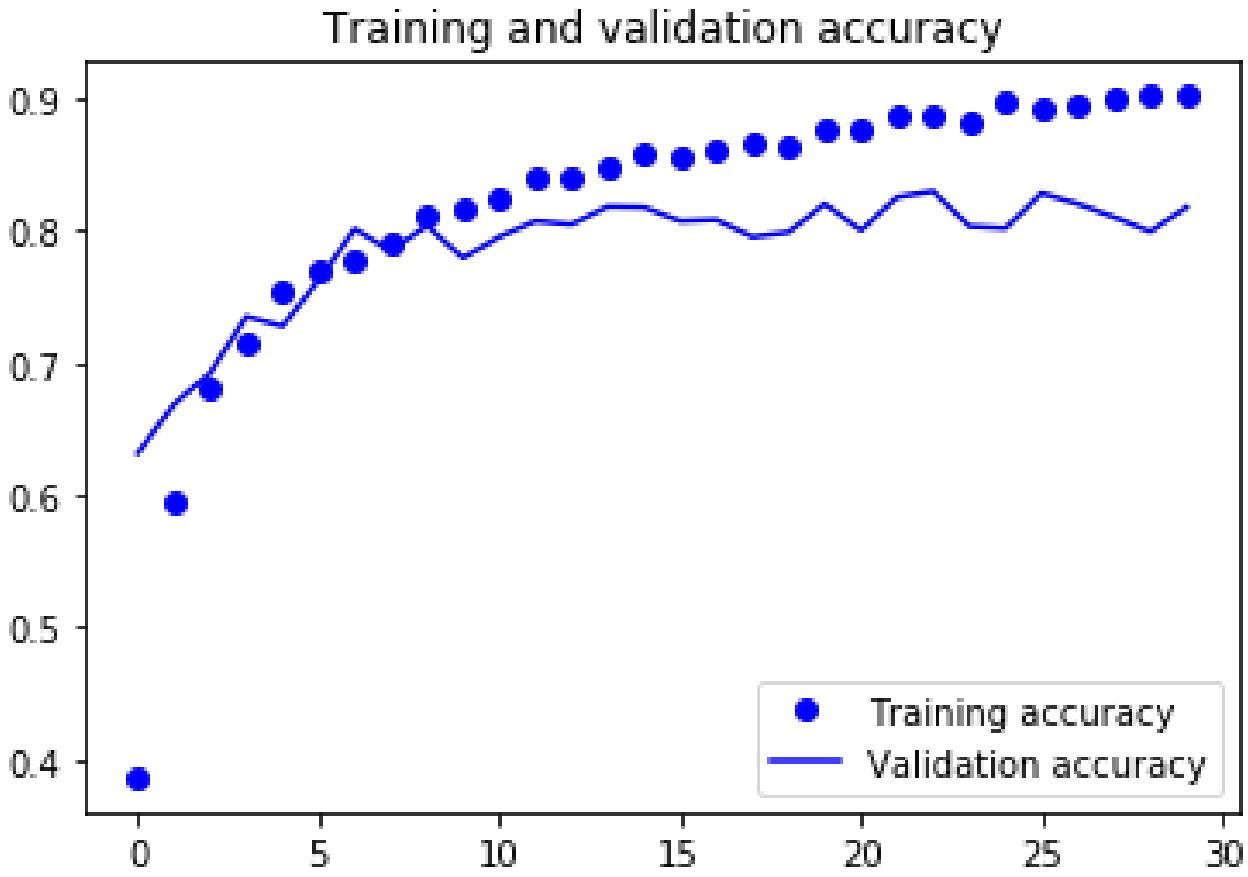
As seen here, the total parameters have been reduced from 19,043,402 on Model 3 to 16,814,666 on Model 4 due to the removal of a dense layer. The number of trainable parameters has also changed from 11,408,138 to 9,179,402. This is a regularization method which reduces the number of trainable parameters of a model, which might reduce overfitting and increase the accuracy of the model. This is tried on Model 2, which failed, which is why I tried again on a pretrained model to see if it will produce a different outcome.

Training Model 4

```
# Train the Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import optimizers

train_datagen = ImageDataGenerator(
      rescale=1./255,
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(150, 150),
        batch_size=50,
        class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='categorical')

model4.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(lr=2e-4),
              metrics=['acc'])

history = model4.fit_generator(
      train_generator,
      steps_per_epoch=150,
      epochs=30,
      validation_data=validation_generator,
      validation_steps=100,
      verbose=1)
```

The learning rate here has been changed from 2e-4 to 2e-5. This is because the model overfitted severely when I first ran it with 2e-4 as the learning rate. After trying a few other learning rates, I found that 2e-5 was the right value to use.

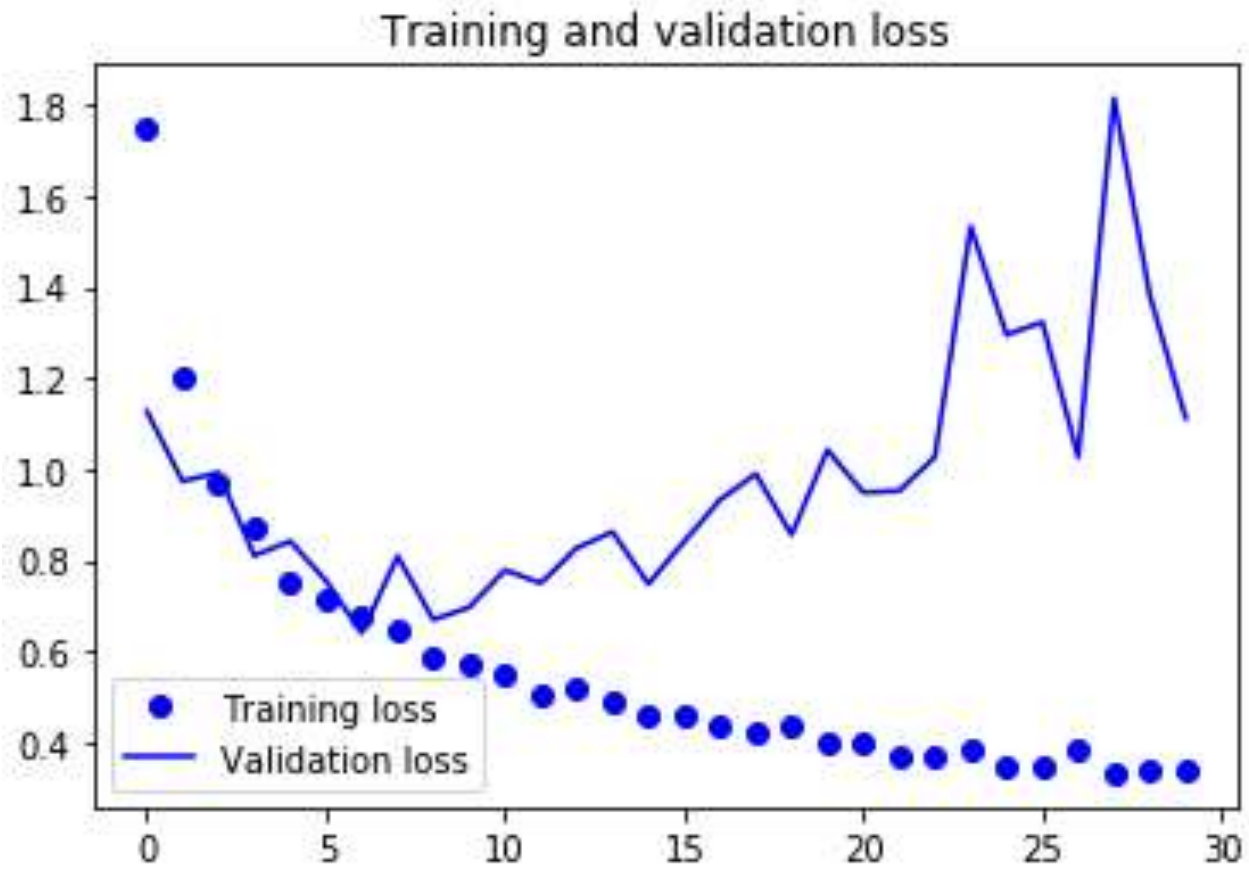
Graph of Model 4 (Training and Validation Accuracy)



The increase of training accuracy of Model 4 is rather stable, while validation accuracy was floating around with an average value of 0.8 after the 10th epoch. The initial training and validation accuracy of Model 4 was 0.3867 and 0.6315, respectively (Figure 4.1). This is higher than the initial values of Model 3. The lower number of trainable parameters might have contributed to the overall increased accuracy of the model. The final training and validation accuracy were 0.9016 and 0.8175, respectively (Figure 4.4). This is higher than Model 3 in both training and validation accuracy. This proves that the reduction of parameters and the decrease of learning rate while retaining all the same hyperparameters increases the accuracy of a pretrained model.

This model overfitted on the 5th epoch, compared to the 8th epoch of Model 3. This shows that the model is not the most regularized yet as the model still overfitted too early. But even as it overfitted early, it still has a testing accuracy of 80%, which is greater than the 78.2% obtained on Model 3.

Graph of Model 4 (Training and Validation Loss)



This training loss is decreasing steadily, while the validation loss decreased to the lowest point and increased drastically. The decreasing training loss and increasing validation loss shows that the model can make decisions, but it is not confident while making the decisions. This means that it will have a high accuracy of the image but may not be able to predict it accurately. The initial training and validation loss are 1.7488 and 0.6315 respectively (Figure 4.1), which is far lower than Model 3. The lowest recorded training and validation loss values are 0.3330 (Figure 4.3) and 0.6404 (Figure 4.2). The lowest training loss value is lower than Model 3, but Model 3 lowest validation loss is 0.5957 (Figure 3.2), which is lower than that of Model 4.

In general, this model has a high accuracy but with a cost of being unable to make decisions confidently. It can be a good thing as it is able to predict the images with high accuracy, but the prediction rate will be low due to low confidence.

```
Epoch 1/30
150/150 [==============================] - 58s 390ms/step - loss: 1.7488 - acc: 0.3867 - val_loss: 1.1274 - val_a
cc: 0.6315
```

Figure 4.1

```
Epoch 7/30
150/150 [==============================] - 55s 364ms/step - loss: 0.6791 - acc: 0.7775 - val_loss: 0.6404 - val_a
cc: 0.8010
```

Figure 4.2

```
Epoch 28/30
150/150 [==============================] - 55s 369ms/step - loss: 0.3330 - acc: 0.9001 - val_loss: 1.8149 - val_a
cc: 0.8100
```

Figure 4.3

```
Epoch 30/30
150/150 [==============================] - 55s 368ms/step - loss: 0.3399 - acc: 0.9016 - val_loss: 1.1124 - val_a
cc: 0.8175
```

Figure 4.4

Recorded data type	Value	Epoch (Page 52)
Initial training accuracy	0.3867	1st Epoch, Figure 4.1
Initial validation accuracy	0.6315	1st Epoch, Figure 4.1
Final training accuracy	0.9016	30th Epoch, Figure 4.4
Final validation accuracy	0.8175	30th Epoch, Figure 4.4
Initial training loss	1.7488	1st Epoch, Figure 4.1
Initial validation loss	0.6315	1st Epoch, Figure 4.1
Lowest recorded training loss	0.3330	28th Epoch, Figure 4.3
Lowest recorded validation loss	0.6404	7th Epoch, Figure 4.2
Testing Accuracy	**0.800**	**-**

Evaluate models using Test Images

Evaluating Base Model (Non-Pretrained)

```
# Model BASE
from tensorflow import keras
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import optimizers

base_dir = 'C:/!Ezra/Ngee Ann Poly/Year 2.1/Assignments/DL/img'
test_dir = os.path.join(base_dir, 'test')

model.load_weights('food_model_BASE.h5')

model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(lr = 2e-4),
              metrics=['acc'])

test_datagen = ImageDataGenerator(rescale=1./255)

test_generator = test_datagen.flow_from_directory(
        test_dir,
        target_size=(150, 150),
        batch_size=50,
        class_mode='categorical')

test_loss, test_acc = model.evaluate_generator(test_generator, steps = 10)
print('test acc:', test_acc)

# Base Model has a final accuracy of 71.4%
```

```
Found 500 images belonging to 10 classes.
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
test acc: 0.714
```

Base Model achieved a testing accuracy of 71.4%. This is a value that is slightly lower compared to the validation accuracy of the model. However, for a non-pretrained model without any regularization, it has a rather high accuracy.

Evaluating Model 1 (Non-Pretrained)

```
# Model 1
from tensorflow import keras
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import optimizers

base_dir = 'C:/!Ezra/Ngee Ann Poly/Year 2.1/Assignments/DL/img'
test_dir = os.path.join(base_dir, 'test')

model1.load_weights('food_model_1.h5')

model1.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(lr = 2e-4),
              metrics=['acc'])

test_datagen = ImageDataGenerator(rescale=1./255)

test_generator = test_datagen.flow_from_directory(
        test_dir,
        target_size=(150, 150),
        batch_size=50,
        class_mode='categorical')

test_loss, test_acc = model1.evaluate_generator(test_generator, steps = 10)
print('test acc:', test_acc)

# Model 1 has a final accuracy of 77.2%
```

```
Found 500 images belonging to 10 classes.
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
test acc: 0.772
```

The accuracy of Model 1 is rather nice start since it obtained a testing accuracy of 77.2%. This is a 5.8% increase in testing accuracy compared to Base Model. This shows that dropout can reduce the overfitting of the model and increase the accuracy of the model as it continues training. The learning rate is set the same as Base Model at 2e-4.

Evaluating Model 2 (Non-Pretrained)

```
# Model 2
from tensorflow import keras
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import optimizers

base_dir = 'C:/!Ezra/Ngee Ann Poly/Year 2.1/Assignments/DL/img'
test_dir = os.path.join(base_dir, 'test')

model2.load_weights('food_model_2-1.h5')

model2.compile(loss='categorical_crossentropy',
               optimizer=optimizers.RMSprop(lr = 2e-4),
               metrics=['acc'])

test_datagen = ImageDataGenerator(rescale=1./255)

test_generator = test_datagen.flow_from_directory(
        test_dir,
        target_size=(150, 150),
        batch_size=50,
        class_mode='categorical')

test_loss, test_acc = model2.evaluate_generator(test_generator, steps = 10)
print('test acc:', test_acc)
```

```
Found 500 images belonging to 10 classes.
WARNING:tensorflow:From <ipython-input-3-ad4fdea88bf7>:24: Model.evaluate_generatc
training) is deprecated and will be removed in a future version.
Instructions for updating:
Please use Model.evaluate, which supports generators.
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
test acc: 0.692
```

The accuracy of Model 2 is only 69.2%, which is an 8.0% decrease from Model 2 and a 2.2% decrease from Base Model. This can be due to it having too much regularization such as a 0.5 dropout value added and reduction of trainable parameters from 4,430,780 to 3,123,130, which can do more harm to the model than good. The learning rate is set at 2e-4, like Base Model. Model 2 is not a suitable model for the best model due to the low testing accuracy.

Evaluating Model 3 (Pretrained)

```
# Model 3
from tensorflow import keras
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import optimizers

base_dir = 'C:/!Ezra/Ngee Ann Poly/Year 2.1/Assignments/DL/img'
test_dir = os.path.join(base_dir, 'test')

model3.load_weights('food_model_3.h5')

model3.compile(loss='categorical_crossentropy',
               optimizer=optimizers.RMSprop(lr = 2e-5),
               metrics=['acc'])

test_datagen = ImageDataGenerator(rescale=1./255)

test_generator = test_datagen.flow_from_directory(
        test_dir,
        target_size=(150, 150),
        batch_size=50,
        class_mode='categorical')

test_loss, test_acc = model3.evaluate_generator(test_generator, steps = 10)
print('test acc:', test_acc)

# Model 3 has a final accuracy of 78.2%
```

```
Found 500 images belonging to 10 classes.
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
test acc: 0.782
```

The testing accuracy of Model 3 is quite high, hitting 78.2%. This is to be expected this result is produced by a pretrained model. When comparing Model 3 and Base Model testing accuracy, Model 3 emerged with a higher testing accuracy compared to Base Model. One difference needed to be taken note is that both are using different learning rates, which is respective to their models. Base Model uses a learning rate of 2e-4 while Model 3 uses 2e-5.

Even though Model 3 overfitted earlier than Base Model, it still performed better in terms of testing accuracy, with 78.2% for Model 3 and 71.4% for Base Model.

Evaluating Model 4 (Pretrained)

```
# Model 4
from tensorflow import keras
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import optimizers

base_dir = 'C:/!Ezra/Ngee Ann Poly/Year 2.1/Assignments/DL/img'
test_dir = os.path.join(base_dir, 'test')

model4.load_weights('food_model_4.h5')

model4.compile(loss='categorical_crossentropy',
               optimizer=optimizers.RMSprop(lr = 2e-5),
               metrics=['acc'])

test_datagen = ImageDataGenerator(rescale=1./255)

test_generator = test_datagen.flow_from_directory(
        test_dir,
        target_size=(150, 150),
        batch_size=50,
        class_mode='categorical')

test_loss, test_acc = model4.evaluate_generator(test_generator, steps = 10)
print('test acc:', test_acc)

# Model 4 has a final accuracy of 80.0%
```

```
Found 500 images belonging to 10 classes.
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
test acc: 0.8
```

The testing accuracy of Model 4 of 80.0% is higher than a Model 3. This is also the first model to hit a testing accuracy of 80%. This is after fine tuning the model for quite a while, changing the model learning rate and removing a dense layer from the model.

Comparing this model testing accuracy with Model 1 and Model 2, Model 4 has a higher testing accuracy than both, obtaining 80%, while the other two have 71.4% and 77.2% respectively. The accuracy of Model 4 is better because like Model 3, the weights have trained and set such that when used to classify images, the weights it is using is already the most optimized and efficient compared to non-pretrained Model 1 and 2.

Use the Best Model to perform classification

Model 4 is used as the best model to perform classification. This was because it has the highest validation and testing accuracy of all the models, obtaining 81.75% for validation and 80% for testing accuracy. Even though Model 4 has increasing validation loss after hitting the lowest point, which indicates lesser confidence in making decisions of the image, the higher accuracy will enable the model to choose the right image, just that it has a lower prediction rate when compared to other models. There was an error with my testing model during the period of my video presentation recording. Thus, I have re-tested different images on Model 4, and most of the images came out with a prediction rate of 1.0. Although most images show a prediction rate of 1.0, there are prediction values of the other images, just that they are so small that it is negligible. An example of such would be the Guacamole and Hamburger. This negligible prediction values might have come from the uncertainty of the model due to the increasing validation loss.

Here are the prediction rates for all the images used for image classification:

Image Number - Food Type	**Prediction**
Image 01 - Caprese Salad	1.0
Image 02 - Cheesecake	0.925761
Image 03 - Deviled Eggs	1.0
Image 04 - Escargots	1.0
Image 05 - Fish & Chips	0.848613
Image 06 - Guacamole	1.0
Image 07 - Hamburger	0.99932
Image 08 - Pancakes	0.001912
Image 09 - Red Velvet Cake	1.0
Image 10 - Spaghetti Carbonara	1.0

Surprisingly, the classification for Pancakes was incorrect, and the model predicted it as Caprese Salad.

While the 60% if the images are predicted with perfect accuracy, 3 of the images were predicted with high, but not perfect accuracy. These images are Cheesecake, Fish & Chips and Hamburger.

There were some food class that had a particularly good prediction that all the other prediction ratings were 0.0, while some other food classes had an extremely high rating that the rating of other foods was so small it was negligible. Some examples of the former are Caprese Salad, Escargots and Spaghetti Carbonara. While the latter consists of images of Deviled Eggs, Guacamole, Red Velvet Cake.

I think that the Model was able to easily predict certain images such as Caprese Salad and Spaghetti Carbonara is because it has such defining features of the dish. For example, Caprese Salad almost always has tomato, leaves and mozzarella plated nicely, which the model has learned and trained over time. This allows it to define it and predict it with high accuracy, which is also contributed by the fact that there are no other images in the group that has such similar defining features.

I believe the model confuses some of the images with high prediction rates such as Cheesecake and Fish & Chips as they both have a yellowish hue which is a feature. But the model was still able to predict it correctly.

There is only an image that the model made an inaccurate prediction, and it is Pancake. I think that the model has gotten some of the features of the image wrongly, perhaps the toppings look like a Caprese Salad topping in some of the images. This can result in the model learning and eventually predicting the image wrongly.

Overall, I believe that the images were predicted and classified mostly correctly except for a few minor losses in details that may affect the accuracy and prediction of the images.

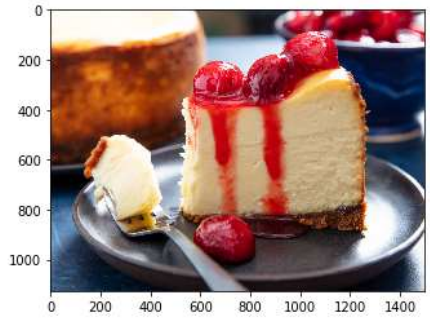
The following images are the screenshots of the testing accuracy of each of the images downloaded from the internet to be used for classification. A summary of the prediction rates can be found on page 59.

```
# Make prediction for the image you downloaded from internet
import matplotlib.pyplot as plt
img = 'C:/!Ezra/Ngee Ann Poly/Year 2.1/Assignments/DL/my_food/test_caprese_salad.jpeg'
plt.imshow(plt.imread(img))
plt.show()

img_array = image_process(img)
prob_df, result = prediction(model4, img_array, food_list)
print('The prediction is: ', result, '\n\n', prob_df)
```


```
The prediction is:  caprese_salad

    caprese_salad  cheesecake  deviled_eggs  escargots  fish_and_chips  \
0             1.0         0.0           0.0        0.0             0.0

   guacamole  hamburger  pancakes  red_velvet_cake  spaghetti_carbonara
0        0.0        0.0       0.0              0.0                  0.0
```

Image 1

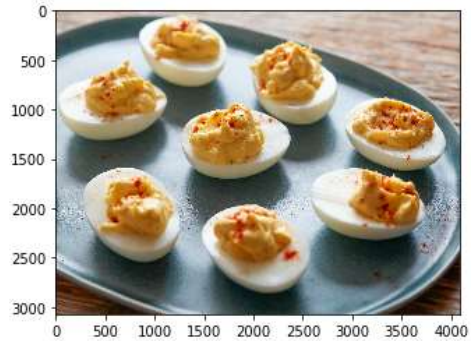
```
# Make prediction for the image you downloaded from internet
import matplotlib.pyplot as plt
img = 'C:/!Ezra/Ngee Ann Poly/Year 2.1/Assignments/DL/my_food/test_cheesecake.jpeg'
plt.imshow(plt.imread(img))
plt.show()

img_array = image_process(img)
prob_df, result = prediction(model4, img_array, food_list)
print('The prediction is: ', result, '\n\n', prob_df)
```




```
The prediction is:  cheesecake

    caprese_salad  cheesecake  deviled_eggs  escargots  fish_and_chips  \
0        0.001204    0.925761      0.000171   0.000947        0.000114

   guacamole  hamburger  pancakes  red_velvet_cake  spaghetti_carbonara
0   0.000228   0.001207  0.015146         0.055214             0.000008
```

Image 2

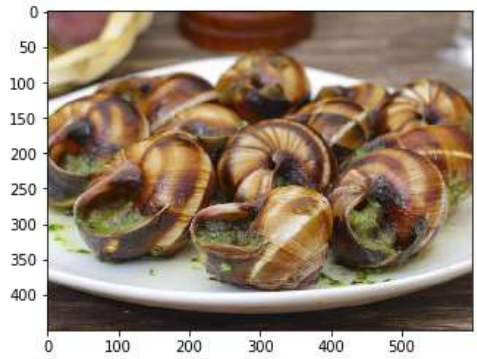
```
# Make prediction for the image you downloaded from internet
import matplotlib.pyplot as plt
img = 'C:/!Ezra/Ngee Ann Poly/Year 2.1/Assignments/DL/my_food/test_deviled_eggs.jpeg'
plt.imshow(plt.imread(img))
plt.show()

img_array = image_process(img)
prob_df, result = prediction(model4, img_array, food_list)
print('The prediction is: ', result, '\n\n', prob_df)
```




```
The prediction is:  deviled_eggs

    caprese_salad    cheesecake  deviled_eggs     escargots  fish_and_chips  \
0   1.224002e-30  1.376676e-30           1.0  2.254277e-28    1.470924e-38

   guacamole     hamburger      pancakes  red_velvet_cake  spaghetti_carbonara
0        0.0  4.963390e-30  5.301885e-26              0.0         6.825224e-35
```

Image 3

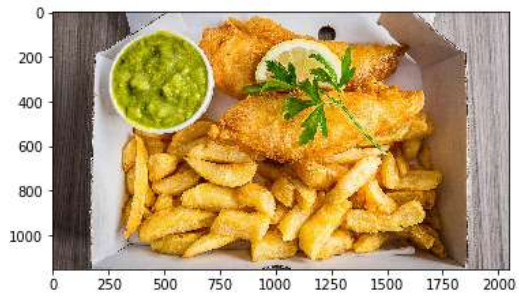
```
# Make prediction for the image you downloaded from internet
import matplotlib.pyplot as plt
img = 'C:/!Ezra/Ngee Ann Poly/Year 2.1/Assignments/DL/my_food/test_escargots.jpeg'
plt.imshow(plt.imread(img))
plt.show()

img_array = image_process(img)
prob_df, result = prediction(model4, img_array, food_list)
print('The prediction is: ', result, '\n\n', prob_df)
```




```
The prediction is:  escargots

    caprese_salad  cheesecake  deviled_eggs  escargots  fish_and_chips  \
0             0.0         0.0           0.0        1.0             0.0

   guacamole  hamburger  pancakes  red_velvet_cake  spaghetti_carbonara
0        0.0        0.0       0.0              0.0                  0.0
```

Image 4

```
# Make prediction for the image you downloaded from internet
import matplotlib.pyplot as plt
img = 'C:/!Ezra/Ngee Ann Poly/Year 2.1/Assignments/DL/my_food/test_fish_and_chips.jpeg'
plt.imshow(plt.imread(img))
plt.show()

img_array = image_process(img)
prob_df, result = prediction(model4, img_array, food_list)
print('The prediction is: ', result, '\n\n', prob_df)
```




```
The prediction is:  fish_and_chips

    caprese_salad  cheesecake  deviled_eggs  escargots  fish_and_chips  \
0        0.003329    0.000022      0.002487    0.00003        0.848613

   guacamole  hamburger  pancakes  red_velvet_cake  spaghetti_carbonara
0   0.116236   0.027464  0.001729     2.638721e-08             0.000091
```

Image 5

```
# Make prediction for the image you downloaded from internet
import matplotlib.pyplot as plt
img = 'C:/!Ezra/Ngee Ann Poly/Year 2.1/Assignments/DL/my_food/test_guacamole.jpeg'
plt.imshow(plt.imread(img))
plt.show()

img_array = image_process(img)
prob_df, result = prediction(model4, img_array, food_list)
print('The prediction is: ', result, '\n\n', prob_df)
```


```
The prediction is:  guacamole

    caprese_salad    cheesecake  deviled_eggs     escargots  fish_and_chips  \
0   1.893673e-25  3.746905e-23  2.025479e-24  1.642480e-17    1.959193e-20

   guacamole     hamburger      pancakes  red_velvet_cake  spaghetti_carbonara
0        1.0  1.540082e-21  1.592753e-19     1.061495e-21         1.935617e-17
```

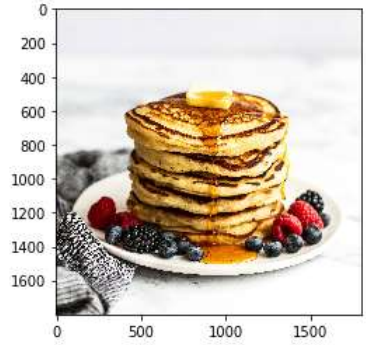
Image 6

```
# Make prediction for the image you downloaded from internet
import matplotlib.pyplot as plt
img = 'C:/!Ezra/Ngee Ann Poly/Year 2.1/Assignments/DL/my_food/test_hamburger.jpeg'
plt.imshow(plt.imread(img))
plt.show()

img_array = image_process(img)
prob_df, result = prediction(model4, img_array, food_list)
print('The prediction is: ', result, '\n\n', prob_df)
```


```
The prediction is:  hamburger

    caprese_salad    cheesecake  deviled_eggs     escargots  fish_and_chips  \
0         0.00068  3.934725e-08  1.885576e-09  9.213124e-08    3.602037e-09

      guacamole  hamburger      pancakes  red_velvet_cake  spaghetti_carbonara
0  5.464986e-10    0.99932  4.185424e-07     4.625310e-09         3.836060e-13
```

Image 7

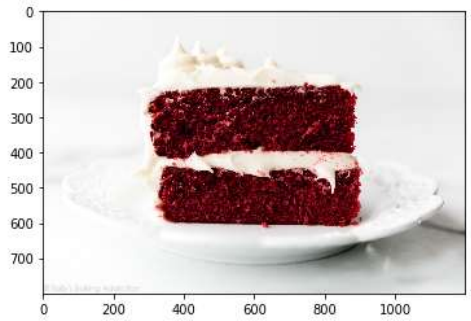
```
# Make prediction for the image you downloaded from internet
import matplotlib.pyplot as plt
img = 'C:/!Ezra/Ngee Ann Poly/Year 2.1/Assignments/DL/my_food/test_pancakes.jpeg'
plt.imshow(plt.imread(img))
plt.show()

img_array = image_process(img)
prob_df, result = prediction(model4, img_array, food_list)
print('The prediction is: ', result, '\n\n', prob_df)
```




```
The prediction is:  caprese_salad

    caprese_salad  cheesecake  deviled_eggs     escargots  fish_and_chips  \
0        0.989764    0.008166  1.058492e-09  7.886342e-07    1.015903e-12

       guacamole     hamburger  pancakes  red_velvet_cake  spaghetti_carbonara
0  3.753645e-13  9.557429e-09  0.001912         0.000158         1.549137e-14
```

Image 8

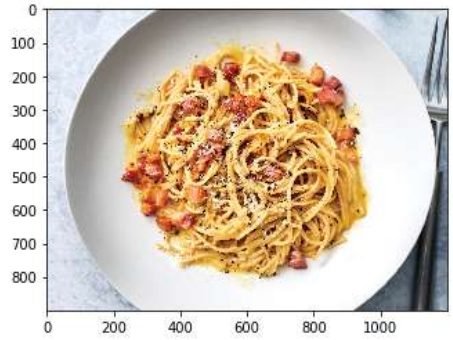
```
# Make prediction for the image you downloaded from internet
import matplotlib.pyplot as plt
img = 'C:/!Ezra/Ngee Ann Poly/Year 2.1/Assignments/DL/my_food/test_red_velvet_cake.jpeg'
plt.imshow(plt.imread(img))
plt.show()

img_array = image_process(img)
prob_df, result = prediction(model4, img_array, food_list)
print('The prediction is: ', result, '\n\n', prob_df)
```




```
The prediction is:  red_velvet_cake

    caprese_salad    cheesecake  deviled_eggs     escargots  fish_and_chips  \
0   4.588769e-25  1.696016e-16  7.769689e-34  8.401281e-29    2.583306e-38

       guacamole     hamburger      pancakes  red_velvet_cake  \
0  3.639191e-34  1.386503e-25  9.039545e-26              1.0

   spaghetti_carbonara
0         1.521508e-35
```

Image 9

```
# Make prediction for the image you downloaded from internet
import matplotlib.pyplot as plt
img = 'C:/!Ezra/Ngee Ann Poly/Year 2.1/Assignments/DL/my_food/test_spaghetti_carbonara.jpeg'
plt.imshow(plt.imread(img))
plt.show()

img_array = image_process(img)
prob_df, result = prediction(model4, img_array, food_list)
print('The prediction is: ', result, '\n\n', prob_df)
```




```
The prediction is:  spaghetti_carbonara

    caprese_salad  cheesecake  deviled_eggs  escargots  fish_and_chips  \
0             0.0         0.0           0.0        0.0             0.0

   guacamole  hamburger  pancakes  red_velvet_cake  spaghetti_carbonara
0        0.0        0.0       0.0              0.0                  1.0
```

Image 10

Summary

In summary, the performance of Model 4 was better than I expected. This was because it managed to classify 9 out of 10 images that it was given. This shows that the model is already good now but can be built better and trained more efficiently such that next time, all the images placed into it can be classified with perfect accuracy. This can be done by using a pretrained model that provides a higher accuracy without fine-tuning. This will give users an idea of high accuracy the model can be in the first place. However, it must be fine-tuned to obtain even higher accuracies, with regularization added to the model such as L2 regularization, dropout layer and early stopping can increase the validation and testing accuracy of Model 4.

One possible build is to include a dropout layer, number of epochs but retain the learning rate of Model 4. The trainable parameters do not need to be changed since it is already reduced from Model 3.

This is shown in the following images in the next two pages.

```
# Build the Model
import os
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras.applications import VGG16
from tensorflow.keras.preprocessing.image import ImageDataGenerator

img_size = 150

vgg16base1 = VGG16(weights='imagenet',
                   include_top=False,
                   input_shape=(150, 150, 3))

model4 = models.Sequential()
model4.add(vgg16base1)
model4.add(layers.Flatten())
model2.add(layers.Dropout(0.4))
model4.add(layers.Dense(256, activation='relu'))
model4.add(layers.Dense(10, activation='softmax'))

vgg16base1.trainable = True

set_trainable = False
for layer in vgg16base1.layers:
    if layer.name == 'block5_conv1':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False

model4.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_3 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_7 (Dense)              (None, 256)               2097408
_________________________________________________________________
dense_8 (Dense)              (None, 10)                2570
=================================================================
Total params: 16,814,666
Trainable params: 9,179,402
Non-trainable params: 7,635,264
```

This shows what would be the edited model that may produce a higher accuracy model that is a continuation of Model 4. A dropout layer is added to the model, with a value of 0.4.

```
# Train the Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import optimizers

train_datagen = ImageDataGenerator(
      rescale=1./255,
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(150, 150),
        batch_size=50,
        class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='categorical')

model4.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(lr=2e-4),
              metrics=['acc'])

history = model4.fit_generator(
      train_generator,
      steps_per_epoch=150,
      epochs=50,
      validation_data=validation_generator,
      validation_steps=100,
      verbose=1)
```

This new model also has a maintained learning rate of 2e-4 while the number of epochs was increased to 50 compared to 30 of Model 4. This is to accommodate the increasing model accuracy at a slower pace due to the addition of a dropout layer. This will see the model overfitting later than Model 4. The number of epochs will not remain at 50 as the model might overfit slightly earlier than expected e.g. at the 40th epoch. The number of epochs will then be changed to 45 as a form of regularization called Early Stopping.

This shows that although Model 4 is a well-trained model, there are still more improvements to be made to ensure a 100% prediction and full confidence in the prediction and classification of images.