

# School of InfoComm Technology

**Deep Learning Assignment**

Diploma in CSF / FI / IT

April 2021 Semester

**ASSIGNMENT 1**

(30% of DL Module)

11st May 2021 – 13rd Jun 2021

**Submission Deadline:**

**Presentation: 13rd Jun 2021 (Sunday), 11:59PM**

**Report: 13rd Jun 2021 (Sunday), 11:59PM**

|  |  |  |
| --- | --- | --- |
| **Tutorial Group** | **:** | **P01** |
| **Student Name** | **:** | Neo Rei Siang Edward |
| **Student Number** | **:** | S10204829 |

**Penalty for late submission:**

10% of the marks will be deducted every calendar day after the deadline.

**NO** submission will be accepted after 20th Jun 2021 (Sunday), 11:59PM.

Table of Contents

[1. Overview 3](#_Toc74517838)

[1.1 Objective of The Assignment 3](#_Toc74517839)

[1.2 Problem of The Assignment 3](#_Toc74517840)

[1.3 The Approach to Complete the Assignment 4](#_Toc74517841)

[2. Data Preprocessing and Data Loading 4](#_Toc74517842)

[3. Developing the Image Classification Models 7](#_Toc74517843)

[3.1 Model Built from Scratch 7](#_Toc74517844)

[3.1.1 Base Model 7](#_Toc74517845)

[3.1.2 Data Augmentation 10](#_Toc74517846)

[3.1.3 Increase Network Size 11](#_Toc74517847)

[3.1.4 Increase Epochs To 50 12](#_Toc74517848)

[3.1.5 Adjusting Learning Rate 13](#_Toc74517849)

[3.1.5.1 Adjusted Learning Rate To 0.001 13](#_Toc74517850)

[3.1.5.2 Adjusted Learning Rate To 0.0005 14](#_Toc74517851)

[3.1.5.3 Adjusted Learning Rate To 0.0001 15](#_Toc74517852)

[3.1.6 Increasing Epochs at Learning Rate 0.0005 16](#_Toc74517853)

[3.1.7 Changing of Network 18](#_Toc74517854)

[3.1.8 Implementing Dropout and Regularizes 19](#_Toc74517855)

[3.2 Pre-Built Model 21](#_Toc74517856)

[3.2.1 Model Built Using VGG16 21](#_Toc74517857)

[3.2.2 Baseline Model 22](#_Toc74517858)

[3.2.3 Data Augmentation Model 24](#_Toc74517859)

[3.2.4 Increase Network size 26](#_Toc74517860)

[3.2.5 Adjusting Learning Rate To 0.00001 27](#_Toc74517861)

[3.2.6 Training VGG16 Model’s Blocks 27](#_Toc74517862)

[3.2.6.1 Fine-Tuning Fifth Block 28](#_Toc74517863)

[3.2.6.2 Fine-Tuning Fourth and Fifth block 29](#_Toc74517864)

[3.2.6.3 Fine-Tuning Third, Fourth, And Fifth Block 30](#_Toc74517865)

[3.2.7 Implementing Regularizes and Dropout 31](#_Toc74517866)

[4. Evaluating Models using Test Images 33](#_Toc74517867)

[4.1 Comparing Both Models Performance 33](#_Toc74517868)

[4.2 Best Model Selection 34](#_Toc74517869)

[5. Performing Classification 34](#_Toc74517870)

[5.1 Performing Classification on Real Life Images 35](#_Toc74517871)

[5.2 Model Classification Results 36](#_Toc74517872)

[6. Summary 38](#_Toc74517873)

# 1. Overview

## 1.1 Objective of The Assignment

For this assignment, the main purpose is to build an image classification model to recognize and classify 10 different classes of foods. A dataset that contains a total of 101 classes of food was provided. For each type of food, the dataset contains 1000 images. Thus, the dataset has a total of 101000 images. Out of the 101 classes of food, only 10 different categories of food were assigned to be used and had to be extracted from the dataset. A total of 10000 images of food will be used and the 10000 images will be split up into three datasets, training, validation, and testing.

The training dataset will have 7500 images, 750 for each food class. The validation dataset will have 2000 images, 200 for each food class. The test dataset will have 500 images, 50 for each food class. Using these three datasets, two models are created, one trained from scratch and the other utilizing a pre-built model. Comparisons between the two models must be made and the better of the two must be used to test to see if the model can accurately classify random images of the food from the internet.

## 1.2 Problem of The Assignment

The problem of the assignment is to correctly determine the class of food present in an image. This is no easy task as there are a considerable number of factors that the model has to look out for to correctly identify the food. The program must train sufficiently to have the ability to differentiate between the features from one food to another. There are 10 different classes of images for the assignment and the model must be able to correctly identify the food class the image belongs to.

## 1.3 The Approach to Complete the Assignment

The approach to complete the assignment consists of a few steps. The first is to load the training, validation, and test data images into Jupyter Notebook and to resize all images to 150 \* 150. The second is to develop the two Image classification models, where one will be built from scratch using conv2D and dense layers, and the other will be built utilizing the pre-trained model VGG16. The universal machine learning workflow will be followed closely to develop and improve the model.

The universal machine learning workflow consists of a few steps. Firstly, the problem needs to be defined and a dataset has to be assembled. In this case, the problem is a multiclass classification problem, and the dataset is the 7500 training images, 2000 validation images, and 500 test images. Secondly, a measure of success has to be chosen. The measure of success used was accuracy since the model is being used to solve a classification problem. Thirdly, an evaluation protocol had to be chosen. The evaluation protocol decides how the evaluation of data will be carried out and for the assignment, maintaining a hold-out validation set will be used. Although the data available is not large, K-fold cross-validation is not used as it requires a large amount of computational power and time. Fourthly, the data has to be prepared. Fifthly, a model that does better than the baseline model has to be developed. Data Augmentation is used to help develop a better model as it increases the diversity of training data available for the model (Kumar, 2019). Sixthly, the model has to scale up by adding more layers, training for more epochs, and adjusting the learning rate. Lastly, modifying the model to regularize it. Modifiers such as adding dropout, changing the number of layers, adding regularization, and adjusting hyperparameters can be done to improve the model.

After being satisfied with the performance of the model, both models will be evaluated using images reserved for testing to compare and see which model performs better. Lastly, using the better-performing model, at least three random food images found online will be used to feed them into the model to see if the model can classify the images correctly.

# 2. Data Preprocessing and Data Loading

For data preprocessing, two packages are required. The first package that needs to be imported is “os”. Importing “os” allows for interaction with the operating system. It is used in data preprocessing to get the current working directory, make directories, joining file paths together, and for getting a list of names in a specified directory (tutorialspoint, n.d.). The second package that needs to be imported is “shutil”. Importing “shutil” allows for high-level operations on files. It is used in data preprocessing to copy items in a file to another file (python, n.d.).



Figure 2.1: Code for Importing Packages

For data Preprocessing, three folders, train, validation, and test were created to store all the images in their corresponding datasets. The names of the food classes assigned are then extracted out into a list called “food\_list”.

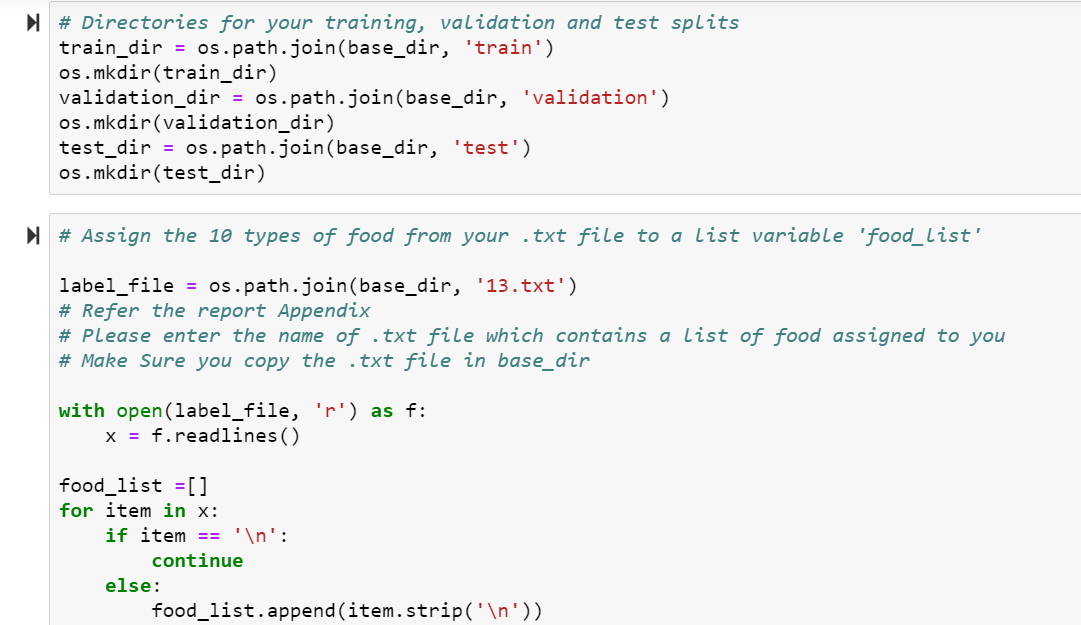


Figure 2.2: Code for Creating Directories and Extracting Food Classes Name

The training dataset was prepared first. For each name in “food\_list”, the program will create a folder with the same name in the training images folder. Since the main dataset that contains 101 classes of food is already separated into different folders by the class names, each name in “food\_list” is used by the program to locate the images in the main dataset. When a name in “food\_list” matches with a name of a folder in the main dataset, the program will take the first 750 image names in that folder and store them in another variable named “img\_list”.

Using the same name of the folder where the images were taken from, the program uses each image name in “img\_list” to locate the image in the main dataset to copy and paste the images into the folder of the training dataset with the same name.

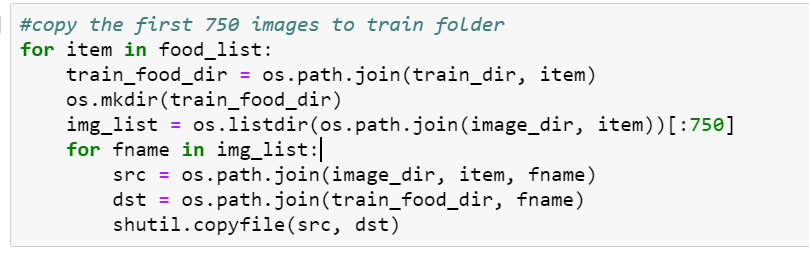
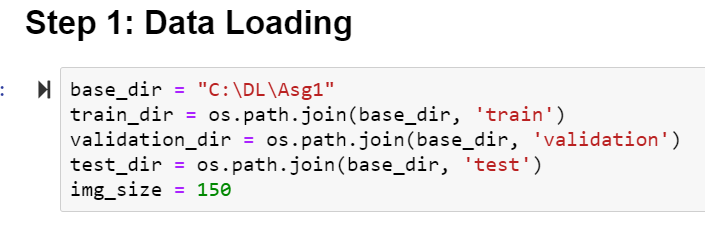


Figure 2.3: Copying Training Images into Training Folders

Likewise, for the validation and test dataset, the process of copying and pasting the images into the datasets are the same. The only difference is that for the validation images dataset, the images being copied will only be 200, and they will be pasted into the same food class name folder but in the validation images folder. Similarly, to the validation images dataset, the test images dataset will only have 50 images copied and they will also be pasted in the same food class name folder but in the test images folder.

Figure 2.4: Code for Extracting Images into Validation And Test Folder

For Data Loading, the package, “os”, was imported and used to specify the training, validation, and test images directory in the program so that the program can locate the training images, validation images, and test images. This will allow the program to use these images at a later point. The image size was also set to have a standard size of 150.

Figure 2.5: Code For Data Loading

# 3. Developing the Image Classification Models

## 3.1 Model Built from Scratch

### 3.1.1 Base Model

For the building of the base model, a total of five conv2D layers, five Maxpooling2D, one flatten layer, and two Dense layers were used. For the conv2D layers, the filter used has a 3x3 shape, and the number of filters used increases from 32 to 512. For the first layer, the shape input is set to 150x150x3. The conv2D layer learns local patterns from the image data using filters and makes predictions based on these patterns. The Maxpooling2D layers have a height and length of two and halves the height and width of the model shape. The main reason for using Maxpooling2D layers is to down-size the size of the model, which lowers the number of parameters that need to be learned, as well as lowers the computational time needed (GeeksforGeeks, 2019). The flatten layer flattens the input data and makes the model have a 1D tensor shape, which is then inputted into the dense layers. One of the dense layers has 256 neurons. The neurons in the dense layer receive input from the previous layer and combine the input at the output layer to form an output shape (Sharma, 2020). The last Dense layer only has 10 nodes as there are 10 classes and the activation was set to ‘softmax’ since the model is being used for a multi-class classification problem. Since there are 10 classes, there are 10 class labels and thus softmax is used (Brownlee, 2020).

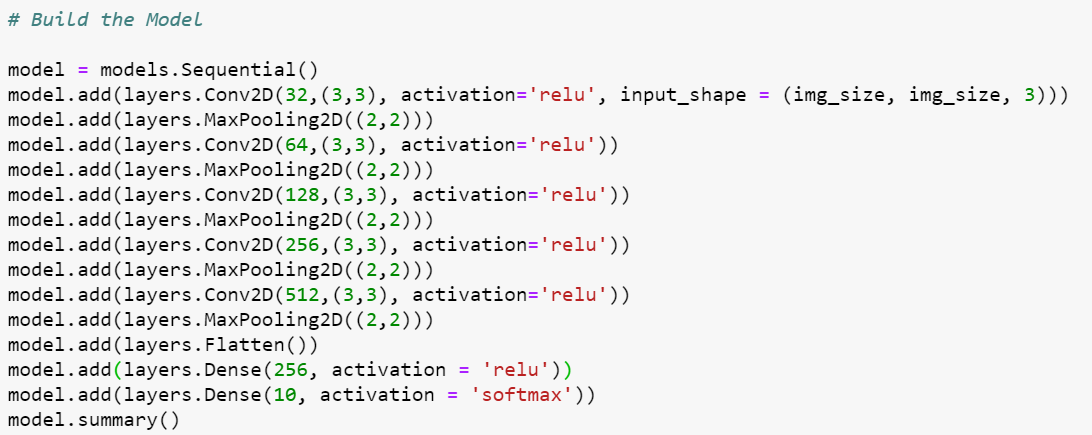


Figure 3.1.1.1: Code for Building Model

For the training of the base model, both the training and validation data sets are rescaled to transform every pixel value from the range of 0 to 255, to 0 to 1. This will make images of different pixel range contribute more evenly to the loss, and also the learning rate will affect the images in a similar way as the pixel range is more limited (Jahn, 2017). The training and validation images are then extracted from the respective directories at 25 images at a time. The class\_mode is set to “categorical” since there are 10 classes.

The base model is then compiled with the optimiser having its default learning rate as “rmsprop”. Loss is set to “categorical\_crossentropy” because there are multiple classes involved. Metrics is set to “acc” as the model is trying to find the accuracy for classification of images.

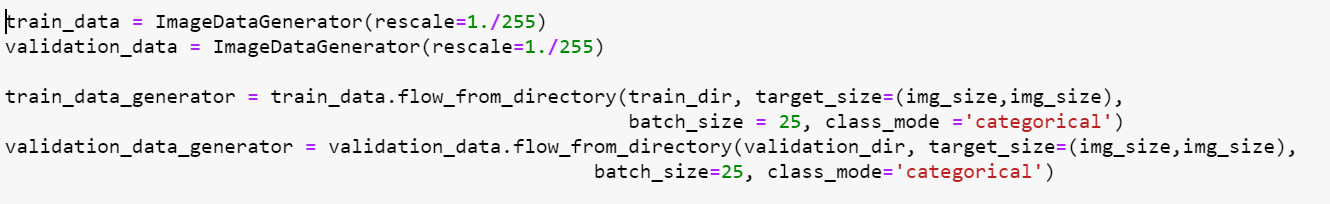


Figure 3.1.2: Code for Training Both Training And Validation Data



Figure 3.1.1.3: Code for Compiling the Model

The base model is then fitted with the training images and compared to the validation images. 300 sets of batches were taken from the training generator per epoch, 80 sets of batches were taken from the validation generator per epoch and the number of epochs running was set to 30.

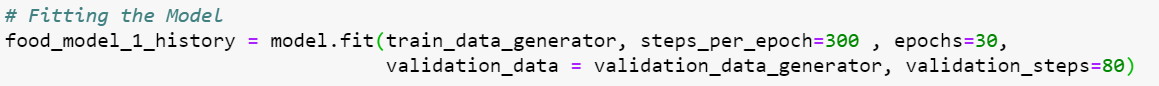


Figure 3.1.1.4: Code for Fitting the Model

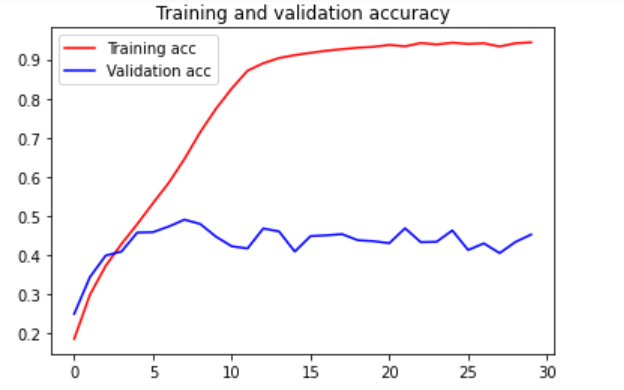


Figure 3.1.1.5: Accuracy Graph of Baseline Model

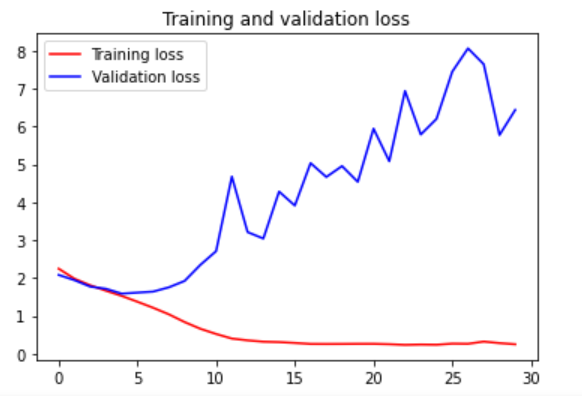


Figure 3.1.1.6: Loss Graph of Baseline Model

### 3.1.2 Data Augmentation

The first change made to improve the model is to augment the training dataset. The images are set to randomly rotate between 0 and 40 degrees. The images are also set to have a width\_shift\_rate of 0.2 and height\_shift\_range of 0.2 which means the images will be shifted by moving all pixels of the image in one direction horizontally and then vertically. Shear\_range was also set to be 0.2, which shifts one part of the image like a parallelogram. Another command used is zoom\_range. It is set at 0.2 and randomly zooms the images in at 80% and zooms out at 120%. Horizontal\_flip is set to true which results in random images being flipped horizontally (Brownlee, How to Configure Image Data Augmentation in Keras, 2019). With the data being augmented, a better model was developed as the overfit decreased and validation accuracy increased.

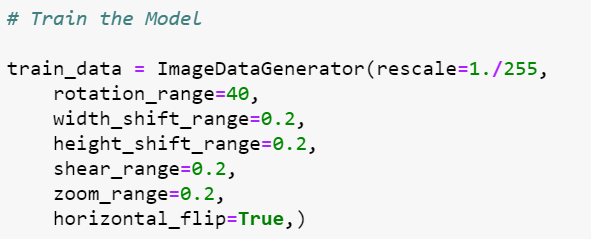


Figure 3.1.2.1: Code for Data Augmentation

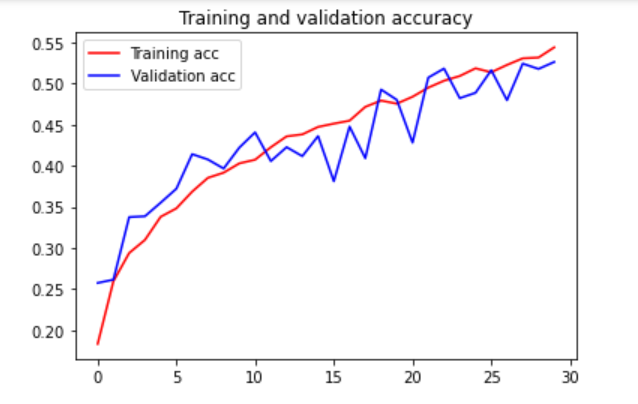


Figure 3.1.2.2: Accuracy Graph of Data Augmented Model

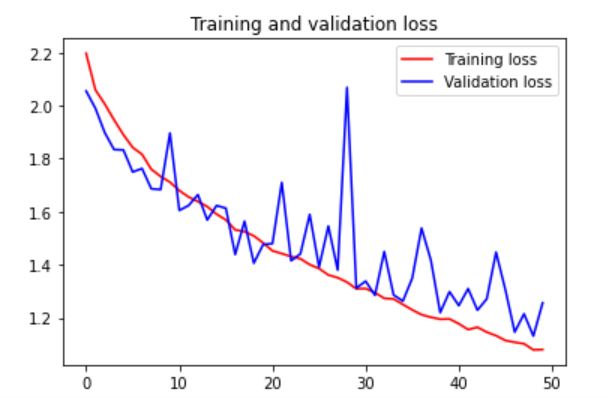


Figure 3.1.2.3: Accuracy Graph of Data Augmented Model

### 3.1.3 Increase Network Size

According to the universal workflow of machine learning, the next step is to scale up the model and train it until it overfits again. Thus, another dense layer with 256 nodes was added to the network. However, the addition of this dense layer was not enough to make the model overfit and only increased the validation accuracy.

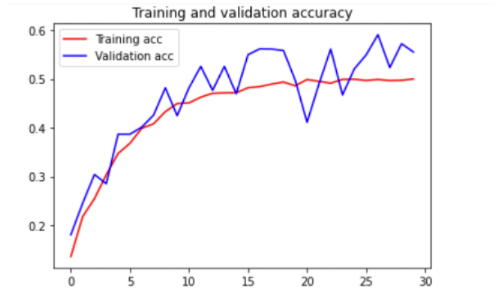


Figure 3.1.3.1: Accuracy Graph of Increased Network Size Model

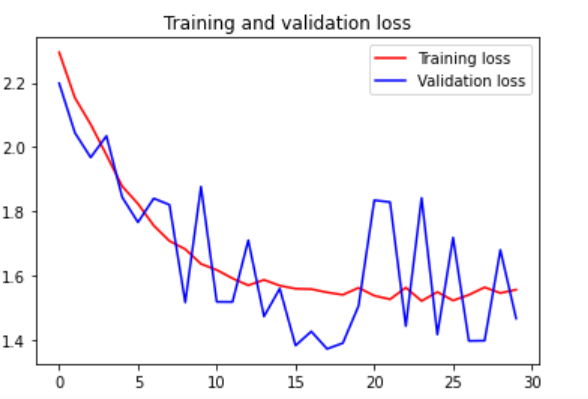


Figure 3.1.3.2: Loss Graph of Increased Network Size Model

### 3.1.4 Increase Epochs To 50

The model was also trained for more epochs to see if the model would overfit if it was trained longer. As shown in the graph below, the model did not overfit.

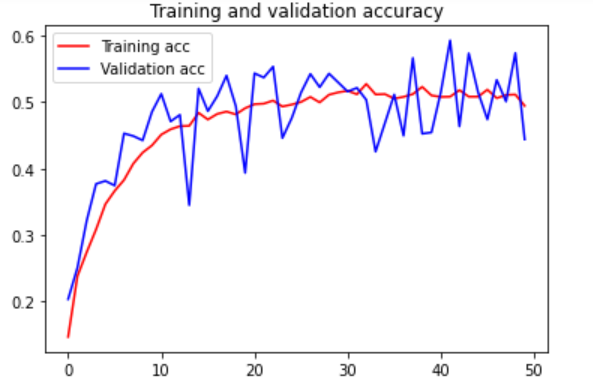


Figure 3.1.4.1: Accuracy Graph of Increased Network Size Model At 50 Epochs

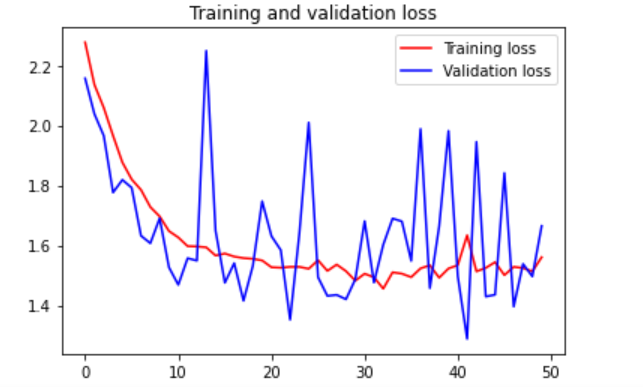


Figure 3.1.4.2: Loss Graph of Increased Network Size Model At 50 Epochs

### 3.1.5 Adjusting Learning Rate

### 3.1.5.1 Adjusted Learning Rate To 0.001

Thus, the learning rate was modified to see if it can help scale up the model instead. The first learning rate used was 0.001, which did not have favourable outcomes as the model did not change much.

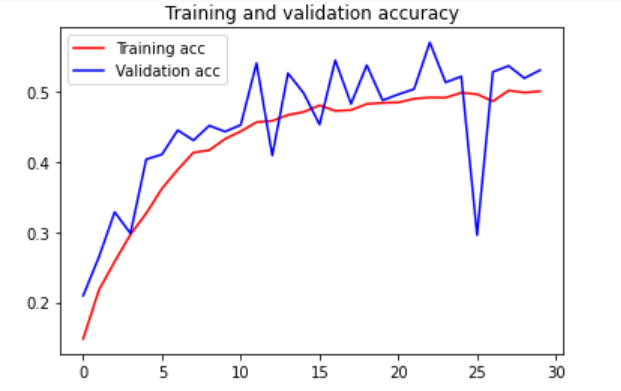


Figure 3.1.5.1.1: Accuracy Graph at Learning Rate 0.001

Figure 3.1.5.1.2: Loss Graph at Learning Rate 0.001

### 3.1.5.2 Adjusted Learning Rate To 0.0005

The second learning rate used decreased to 0.0005 to see if a lower learning rate will improve the model’s performance. Although the model has not yet started overfitting, the training accuracy and validation accuracy increased. Since the lowered learning rate made the model perform better, lower learning rates were tested.

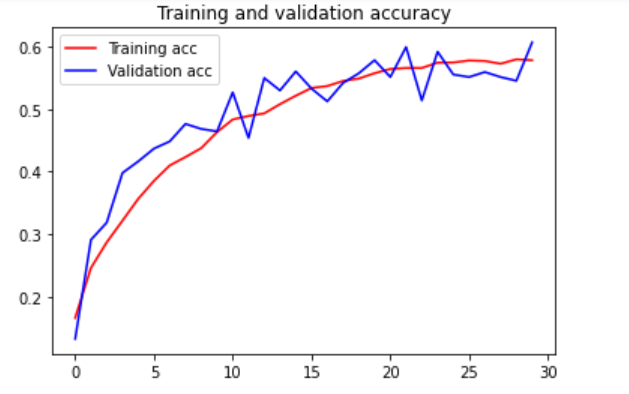


Figure 3.1.5.2.1: Accuracy Graph at Learning Rate 0.0005

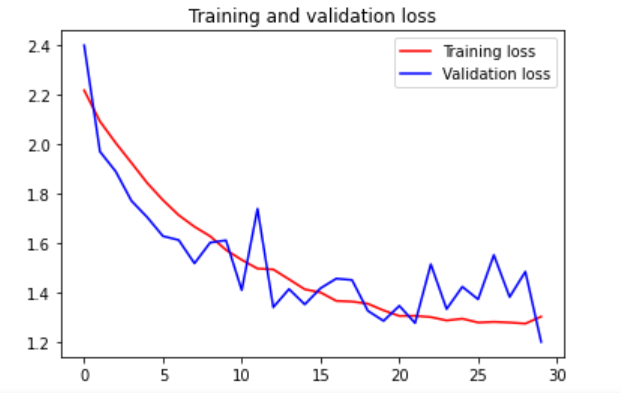


Figure 3.1.5.2.2: Loss Graph at Learning Rate 0.0005

### 3.1.5.3 Adjusted Learning Rate To 0.0001

The learning rate was lowered again to see if the model would perform better at a lower learning rate, but the validation accuracy produced decreased, which does not fulfill the aim of increasing the training accuracy so that the model overfits. This means that the model performed best at 0.0005.

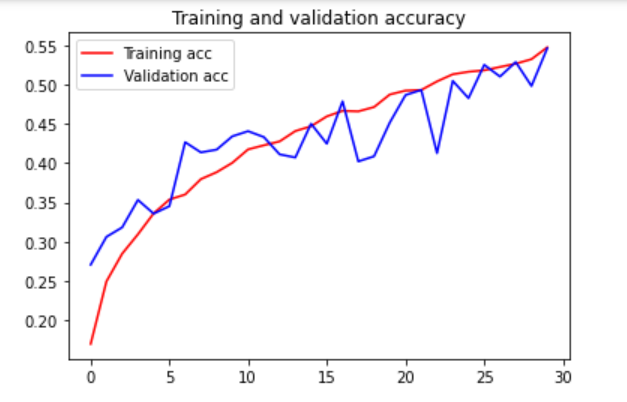


Figure 3.1.5.3.1: Accuracy Graph at Learning Rate 0.0001

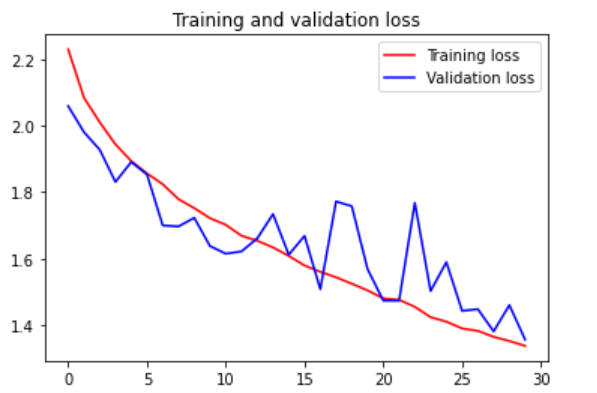


Figure 3.1.5.3.2: Loss Graph at Learning Rate 0.0001

### 3.1.6 Increasing Epochs at Learning Rate 0.0005

Therefore, using the learning rate of 0.0005, the epochs once were increased to check if the model would overfit. However, the training accuracy was limited to 0.6 and could no longer increase. This shows that the model used was too simple, and a more complex model is needed.

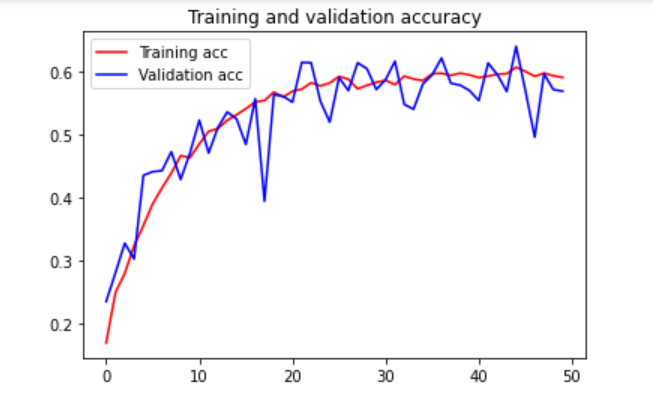


Figure 3.1.6.1: Accuracy Graph of Learning Rate 0.0005 At 50 Epochs

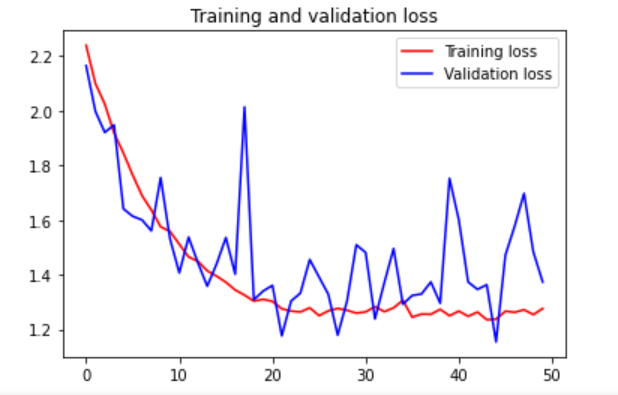


Figure 3.1.6.2: Loss Graph of Learning Rate 0.0005 At 50 Epochs

### 3.1.7 Changing of Network

The network of the model was adjusted and the number of filters in each Conv2D layer went from 32,64,128,256,512 to 64,128,256,512,512. With a new network, the steps above were repeated and the model was finally found to overfit when the learning rate used was 0.0001 and the epochs set to 50.

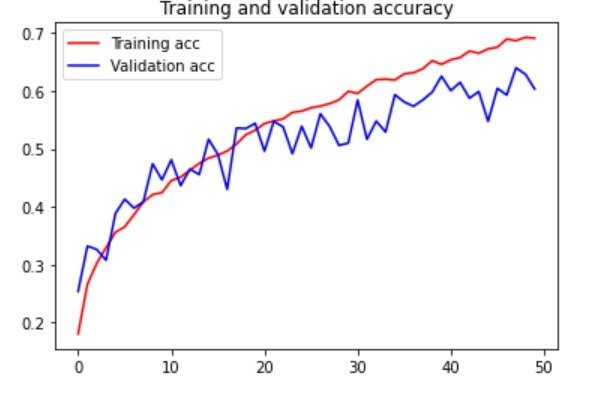


Figure 3.1.7.1: Accuracy Graph of New Model

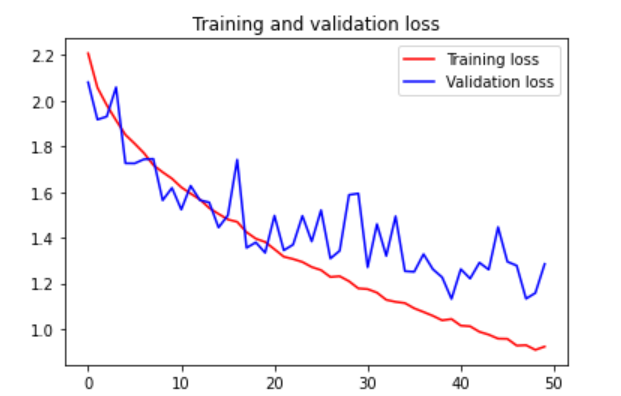


Figure 3.1.7.2: Loss Graph of New Model

### 3.1.8 Implementing Dropout and Regularizes

The last step of the universal workflow of machine learning is to regularize the model by tuning the hyperparameters. Dropout and L2 regularization were added to the model. After a series of testing, the best combination of Dropout and L2 regularization used was two dropouts with values of 0.3 and three L2 regularizations with values of 0.0001.

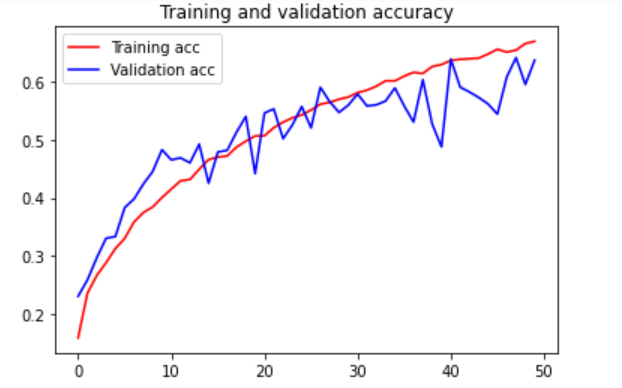


Figure 3.1.8.1: Accuracy Graph After Adding Dropout and Regularizations

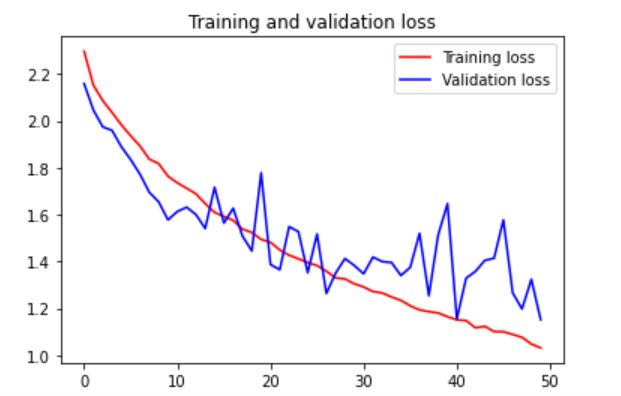


Figure 3.1.8.2: Loss Graph After Adding Dropout and Regularizations

As the Loss of the graph is not yet flattened, it showed that there is room for improvement with the model. Thus, the model has trained again with 70 epochs to maximize the performance of the model.

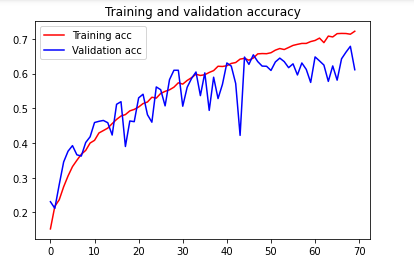


Figure 3.1.8.3: Accuracy Graph of Final Model

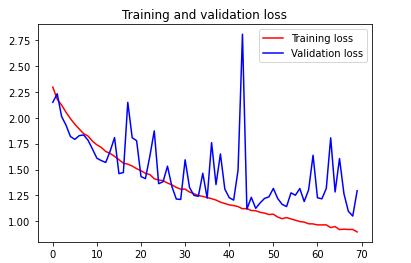
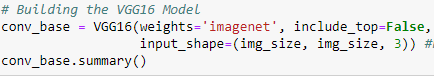


Figure 3.1.8.4: Loss Graph of Final Model

## 3.2 Pre-Built Model

### 3.2.1 Model Built Using VGG16

For the pre-trained model, the VGG16 model was selected for use. The reasons for using VGG16 include it being widely popular and it being easy to implement and use. (Hassan, 2018). For the building of the VGG16 Model, the weights were set to ‘imagnet’. Include\_top was also set to False since the dataset used only consists of 10 classes. Input\_shape was also set to have a shape of 150x150x3. The VGG16 model consists of blocks, with each block having two to three stacks of the same Conv2D layer and one Maxpooling2D layer. The final shape of the VGG16 model is 4x4x512.

  
Figure 3.2.1.1: Code for Building the VGG16 Model

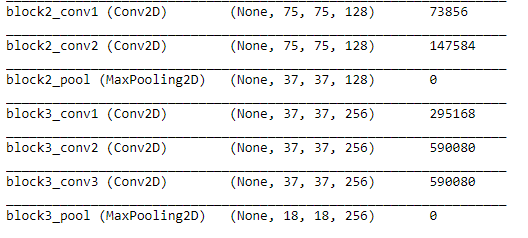


Figure 3.2.1.2: Vgg16 Model’s Summary for Block 2 And Block 3

### 3.2.2 Baseline Model

For the building of the model, two Dense layers were used. One has 512 neurons with an activation function set as “relu”, and the other has 10 neurons with an activation function set as “softmax”.

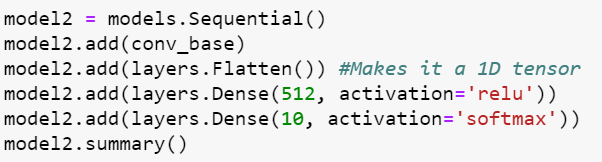


Figure 3.2.2.1: Code for Building Baseline Model

For the training of the model, it is very similar to the model built from scratch, where both the training and validation datasets are rescaled to transform every pixel value from the range of 0 to 255 to 0 to 1. The training and validation images are then extracted from the respective directories at 25 images at a time. The images extracted are set to have a size of 150 x 150. The class\_mode is set to “categorical” as well since there are 10 classes. The VGG16 model had its layers frozen to prevent the weights from getting updated during training. The dense layers are randomly initialized, which can result in very large weight updates propagating throughout the network. Thus, if the layers are not frozen, the weight updates will destroy the representations previously learned.

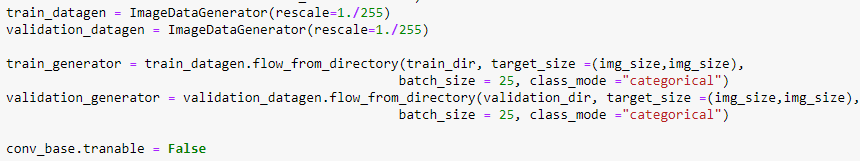


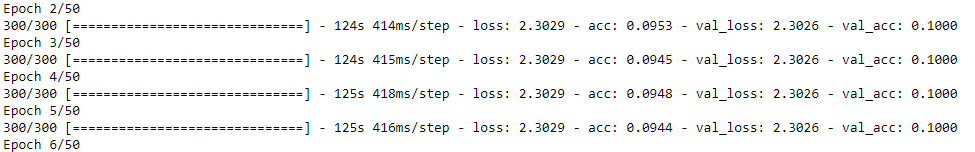
Figure 3.2.2.2: Code for Training Baseline Model

The model is then compiled with the optimizer having its default learning rate as “rmsprop”. Loss is set to “categorical\_crossentropy” because there are multiple classes involved. Metrics is set to “acc” as the model is trying to find the accuracy for the classification of images.

The model is then fitted with the training images and compared to the validation images. 300 sets of image batches were taken from the training generator per epoch, 80 sets of image batches were taken from the validation generator per epoch, and the number of epochs running was set to 30.



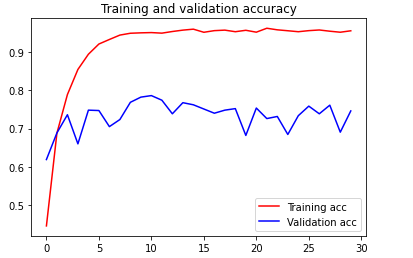
Figure 3.2.2.3: Code for Fitting Baseline Model

  
Figure 3.2.2.4: Output from Running Baseline Model

The results produced from using the default learning rate gave a consistent validation accuracy of 0.1000, which means the model is either learning too slowly or quickly. Thus, the learning rate was adjusted and many values such as 0.01 and 0.0005 were tried but it still showed the same result. The first learning rate that produced a suitable graph was 0.00005.



Figure 3.2.2.4: Code for Compiling Baseline Model With New Learning Rate

  
Figure 3.2.2.5: Accuracy Graph of Baseline Model

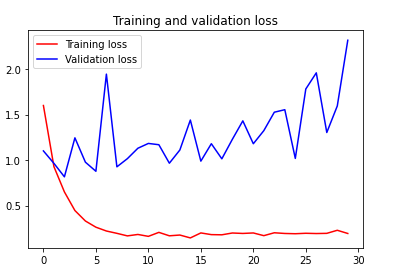


Figure 3.2.2.6: Loss Graph of Baseline Model

### 3.2.3 Data Augmentation Model

After creating a baseline model, another model that performs better than the baseline model must be developed. Thus, data augmentation was added to the training generator. The data augmentation added was similar to the one added for the model built from scratch, except for the addition of one command, which is fill\_mode = “nearest”. This command fills the points outside the boundaries of the input image with the nearest pixel at the boundary (Keras, n.d.). The model after the training data has been augmented is better as the validation accuracy is higher and the overfitting is not as much.

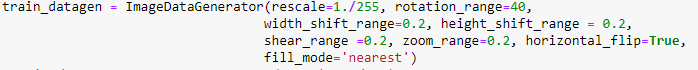


Figure 3.2.3.1: Code for Data Augmentation

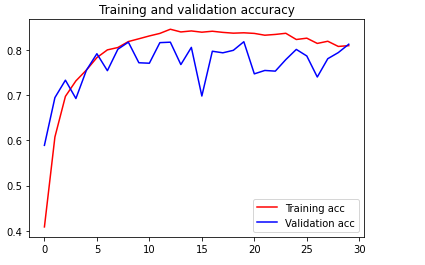


Figure 3.2.3.2: Accuracy Graph of Data Augmented Mode

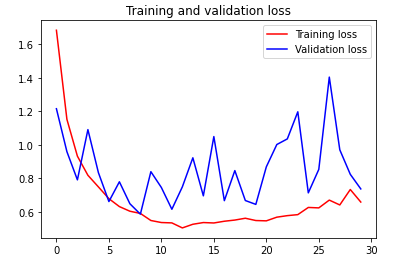


Figure 3.2.3.3: Loss Graph of Data Augmented Mode

### 3.2.4 Increase Network size

In order to scale up the network, the first change which was made is to increase the network size. One Dense Layer with 256 nodes was added to the network. The addition of the network did help increase the training accuracy and the graph started overfitting more. Since the accuracy was limited at 0.8, the number of epochs was not increased. Next, the learning rate of the model was adjusted to see if the results can improve.

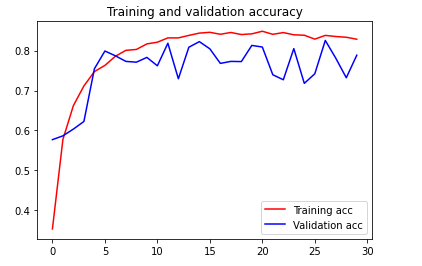


Figure 3.2.4.1: Accuracy Graph of Increased Network Size

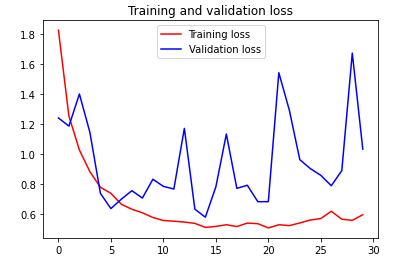


Figure 3.2.4.2: Loss Graph of Increased Network Size

### 3.2.5 Adjusting Learning Rate To 0.00001

After the learning rate was decreased to 0.00001, both the training and validation accuracy increased. The validation accuracy increased slightly and was more consistent as well. Since the training accuracy is quite high, there is not much point in decreasing or increasing the learning rate further, and thus, this learning rate was used for the model.

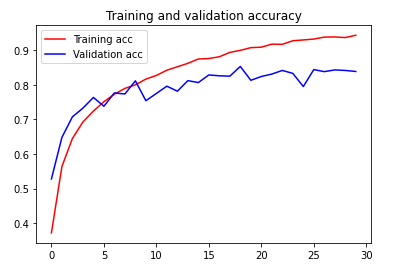


Figure 3.2.5.1: Accuracy Graph of Learning Rate 0.00001

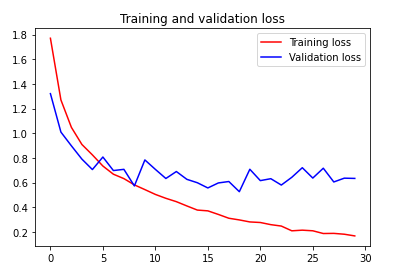


Figure 3.2.5.2: Loss Graph of Learning Rate 0.00001

### 3.2.6 Training VGG16 Model’s Blocks

After identifying the learning rate to use for the model, some of the layers in the VGG16 model are set to unfreeze and set to be trainable to allow for fine-tuning. The layers are fine-tuned starting from the end because the end layers encode more specialised features. Therefore, it is more useful to fine-tune these specialised features to improve the accuracy of the model.

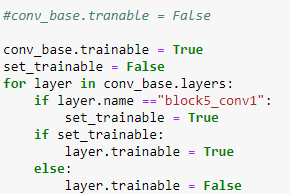


Figure 3.2.6a: Code Used for Training VGG16 Model’s Block

### 3.2.6.1 Fine-Tuning Fifth Block

Firstly, the fifth block of the VGG16 model was set to be trainable. The graph after training the fifth block in the VGG16 model shows that the accuracy, in general, went down, but the validation accuracy is more consistent.

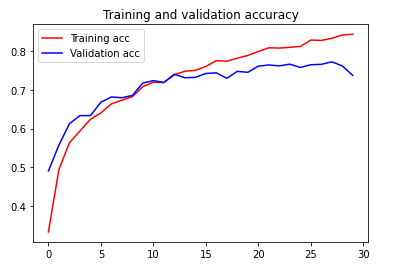


Figure 3.2.6.1.1: Accuracy Graph After Fine-Tuned Fifth Block

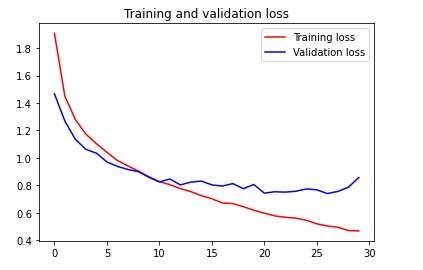


Figure 3.2.6.1.2: Loss Graph After Fine-Tuned Fifth Block

### 3.2.6.2 Fine-Tuning Fourth and Fifth block

The fourth block was also fine-tuned together with the fifth block. This time, both the training accuracy and validation accuracy did improve, but an overfit started to occur from the fifth epoch.

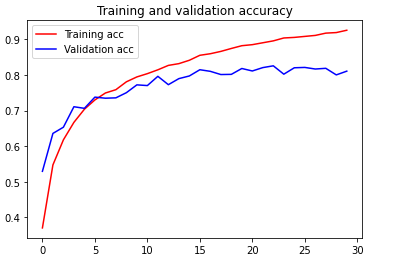


Figure 3.2.6.2.1: Accuracy Graph After Fine-Tuned Fourth and Fifth Block

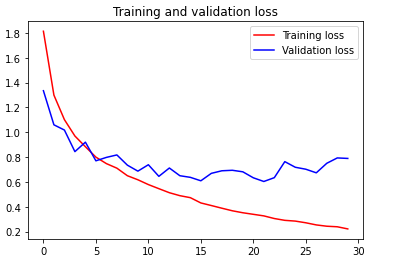


Figure 3.2.6.2.2: Loss Graph After Fine-Tuned Fourth and Fifth Block

### 3.2.6.3 Fine-Tuning Third, Fourth, And Fifth Block

Fine-tuning of the third, fourth and fifth blocks were also tested. There is not much difference in the graphs, but on a closer look, the validation accuracy was more consistent in the 0.80 range after the third, fourth and fifth blocks were fine-tuned. Thus, this model performed better than the model with fine-tuned fourth and fifth blocks. Since the only blocks left are one and two, the model should not be fine-tuned further because blocks one and two are considered earlier layers and encode more generic, reusable features. Thus, fine-tuning earlier layers will result in fast-decreasing returns which should be avoided.

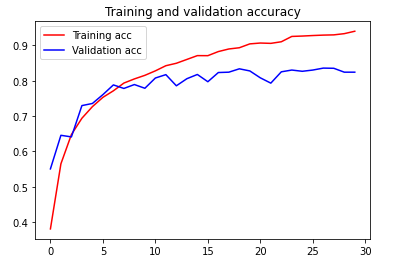


Figure 3.2.6.3.1: Accuracy Graph After Fine-Tuned Third, Fourth, And Fifth Block

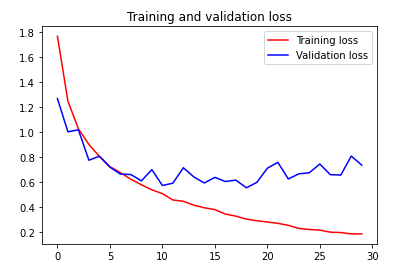


Figure 3.2.6.3.2: Loss Graph After Fine-Tuned Third, Fourth, And Fifth Block

### 3.2.7 Implementing Regularizes and Dropout

After a series amount of testing, it was shown that using three L2 weight Regularizes with values of 0.01, accompanied by two dropout layers of 0.2 and one dropout layer of 0.1 was the best graph the model could produce. Any more increase in dropout and regularization will reduce the performance of the model.

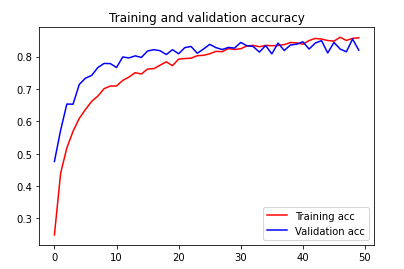


Figure 3.2.7.1: Accuracy Graph After Addition of Regularizes and Dropout

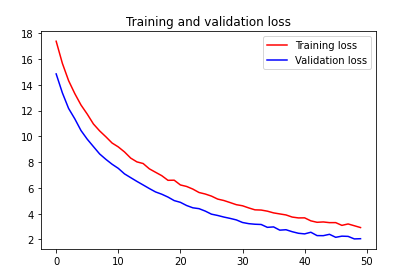


Figure 3.2.7.2: Loss Graph After Addition of Regularizes and Dropout

Since the loss was not flat and was still decreasing, it showed that the model can still perform better. Thus, the model was extended to 70 epochs and the loss became flatter towards the end. Therefore, the final graph is as shown below.

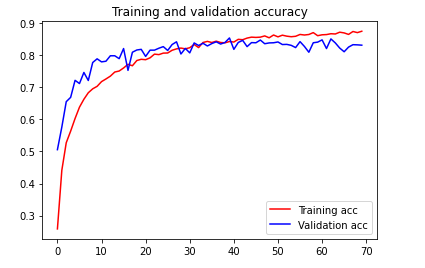


Figure 3.2.7.3: Accuracy Graph of Final Model

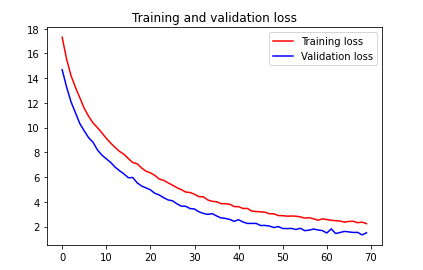


Figure 3.2.7.4: Loss Graph of Final Model

# 4. Evaluating Models using Test Images

In order to evaluate both models using test images, the first action required is to save and load the model weights. For both models, similar to the validation images data set, the test images data set are rescaled to transform every pixel value from the range of 0 to 255 to 0 to 1. The test images are then extracted from their respective directory at 25 images at a time. The images extracted are set to have a size of 150 by 150. The class\_mode is set to “categorical” as well since there are 10 classes. Both models are then evaluated against the test images. A total of 20 sets of batches were taken from the test image generator for evaluation.

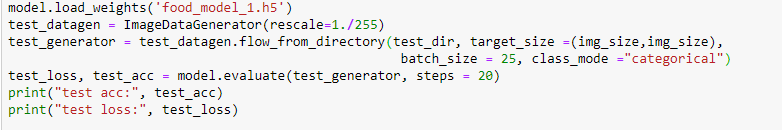


Figure 4a: Code for Test Data Training and Extraction

## 4.1 Comparing Both Models Performance

For the scratch model, the model only had a test accuracy of 0.576.

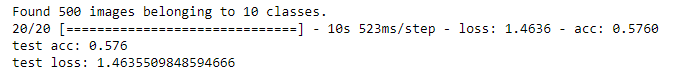


Figure 4.1.1: Test Result for Scratch Model

However, the prebuilt model had a test accuracy of 0.806.

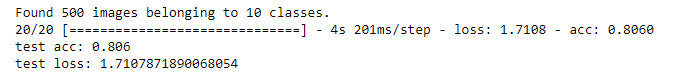


Figure 4.1.2: Test Result for Pre-Built Model

## 4.2 Best Model Selection

Since both models used the same data set, the prebuilt model was the better of the two as it has a higher test accuracy and was thus able to identify the images better. Other than the prebuilt’s model network, the fine-tuning of the prebuilt’s model layers, the learning rate of the model, and a slight difference of commands used in data augmentation, both models have similar hyperparameters.

The difference of one command in data augmentation should not have caused the huge difference in results and since the learning rate is based on the model network, this shows that the prebuilt model’s network is better than the scratch model’s network. This suggests that the network of the models mainly contributed to the large difference in results.

Due to the difference in the network, the prebuilt model is able to learn better than the scratch model. Being able to learn better, the model was able to train better and thus was able to produce better validation accuracy and test accuracy. Furthermore, the layers in the prebuilt model were fine-tuned, which means that the layers underwent further training, making the model more accurate in its identification of images.\

# 5. Performing Classification

## 5.1 Performing Classification on Real Life Images

The best model’s weights were saved and loaded. The first action that needs to be done to enable the model to perform the classification of images is to extract the list of food names into a list. This is done by taking the text file provided, which contains the name of the 10 food items, and extracting the names from the list. The names must also be sorted accordingly to alphabetical order. Next, two functions were created which will be used.

The first function is called, “image\_process”, which loads the image downloaded from the internet to a PIL format and sets the inputted image to have an image size of 150 x 150 (TensorFlow, 2021). The function also converts the PIL image into a 3D Numpy array (TensorFlow, 2021). The 3D Numpy array is rescaled to transform every pixel value from the range of 0 to 255 to 0 to 1.

The second function is called, “prediction”. Since the activation function of the model is softmax, the model will calculate the probability of the inputted image belonging to a certain class for every class available (Versloot, 2020). A pandas dataframe with 10 columns, each representing one of the food items names, is used to store the respective probabilities of each class in the dataframe. By using maximum argument, the index of the highest probability will be identified for each image inputted (Versloot, 2020). This index will then be used to extract the food item class name from the list of food names created at the start, and the name extracted represents what the model has classified the inputted image as.

The last step is to specify the directory of the images downloaded from the internet and to use the two functions to let the model classify the images.

## 5.2 Model Classification Results

After inputting three images, belonging to the food item classes,” gyoza”, ”omelette”, and “beef tartare”. The model was able to classify the images into their correct classes.

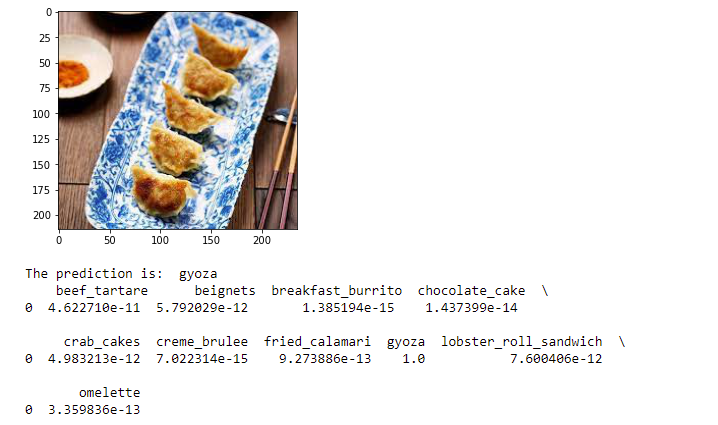


Figure 5.1.1: Gyoza Prediction (Chen, 2021)



Figure 5.1.2: Beef Tartare Prediction (Nick, 2019)

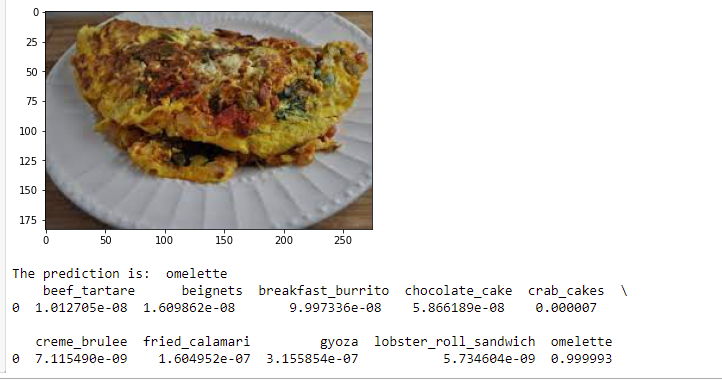


Figure 5.1.3: Omellete Prediction (Food And Life Path)

Due to its high-test accuracy, it shows that the model was able to learn how to differentiate the images from one another, as well as was able to learn the patterns correctly to identify the images. The prediction values shown are also a strong indication that the model learned well. For almost all the three images, the probability prediction for each image food class name was very close to one, which suggests that the model was confident and could accurately identify the images corresponding classes.

# 6. Summary

Overall, the pre-built model performed quite well as it has a well-tested accuracy of 0.806. It was also able to classify the three images downloaded from the internet, which means that it can differentiate the images. However, the scratch model did disappoint as it only achieved a tested accuracy of 0.576. A few improvements could have been made. For instance, a different type of optimizer such as “adam” or “SG” could have been tested out to see if the results would have been improved. For the scratch model, perhaps, more variations of the network could have been tested out to see which would produce better results. For the prebuilt model, comparisons between different types of pre-built models could have been made as well to see which model would have produced the best results. Additionally, more features could have been researched on and used. Some examples include testing out average pooling instead of Max Pooling and testing out bias\_regularizers and activity\_regularizers to see if the model would have performed better.

7. References

Brownlee, J. (2019, April 12). *How to Configure Image Data Augementation in Keras*. Retrieved from machinelearningmastery: https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/

Brownlee, J. (2020, October 19). *Softmax Activation Function with Python*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/softmax-activation-function-with-python/

Chen, N. (2021, May 4). Gyoza(Japanese potstickers) (video).

Food And Life Path. (n.d.). Easy Omellete: How To Make An Omellete With Ham and Cheese.

GeeksforGeeks. (2019, August 26). *CNN | Introduction to Pooling Layer*. Retrieved from GFG: https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/

Hassan, M. u. (2018, November 20). *VGG16 - Convolutional Network For Classification and Detection*. Retrieved from neurohive : https://neurohive.io/en/popular-networks/vgg16/

Jahn, A. (2017, February 16). *Keras Image Preprocessing: Scaling Image Pixels For Training*. Retrieved from Linkedin: https://www.linkedin.com/pulse/keras-image-preprocessing-scaling-pixels-training-adwin-jahn/

Keras. (n.d.). *Image Data Preprocessing*. Retrieved from keras.io: https://keras.io/api/preprocessing/image/

Kumar, S. (2019, July 21). *Data Augmentation Increases Accuracy of your model — But how ?* Retrieved from medium: https://medium.com/secure-and-private-ai-writing-challenge/data-augmentation-increases-accuracy-of-your-model-but-how-aa1913468722

Nick. (2019, January 29). How to Make Perfect Steak Tartare at Home.

python. (n.d.). *shutil - High-level file operations*. Retrieved from docs.python: https://docs.python.org/3/library/shutil.html

Sharma, P. (2020, October 20). *Keras Dense Layer Explained for Beginners*. Retrieved from machinelearningknowledge: https://machinelearningknowledge.ai/keras-dense-layer-explained-for-beginners/

TensorFlow . (2021, May 14). *tf.keras.preprocessing.image.load\_img*. Retrieved from tensorflow.org: https://www.tensorflow.org/api\_docs/python/tf/keras/preprocessing/image/load\_img

TensorFlow. (2021, May 14). *tf.keras.preprocessing.img\_to\_array*. Retrieved from tensorflow: https://www.tensorflow.org/api\_docs/python/tf/keras/preprocessing/image/img\_to\_array

tutorialspoint. (n.d.). *Python os.lsitdir() Method*. Retrieved from tutorialspoint: https://www.tutorialspoint.com/python/os\_listdir.htm

Versloot, C. (2020, February 21). *How to Predict new samples with your TensorFlow / Keras model*. Retrieved from machinecurve: https://www.machinecurve.com/index.php/2020/02/21/how-to-predict-new-samples-with-your-keras-model/