# 



**School of InfoComm Technology**



**Deep Learning Assignment**

Diploma in CSF / FI / IT

April 2022 Semester

**ASSIGNMENT 1**

(30% of DL Module)

16th May 2022 – 10th June 2022

**Submission Deadline:**

**Presentation: 10th June 2022 (Sunday), 11:59PM**

**Report and Code: 10th June 2022 (Sunday), 11:59PM**

|  |  |  |
| --- | --- | --- |
| **Tutorial Group** | **:** | **DL T03 April 2022** |
| **Student Name** | **:** | **Seo Shin Youn** |
| **Student Number** | **:** | **S10205100K** |

**Penalty for late submission:**

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

**NO** submission will be accepted after 19th June 2022 (Sunday), 11:59PM

Table of Contents

[1](#_Toc106648777)

[Section 1: Overview - Problem, Objective, and Approach 3](#_Toc106648778)

[Section 2: Data Loading 4](#_Toc106648779)

[Section 3: Data Preprocessing 7](#_Toc106648780)

[Section 4: Image Classification Model 8](#_Toc106648781)

[Model 1A – Model Trained from Scratch without Data Augmentation 8](#_Toc106648782)

[Model 1B – Model Trained from Scratch with Data Augmentation 11](#_Toc106648783)

[Model 2A – Data Augmented VGG19 Model without Fine-Tuning 14](#_Toc106648784)

[Model 2B – Data Augmented VGG19 Model with Fine-Tuning 19](#_Toc106648785)

[Model 3A – Data Augmented InceptionResNetV2 Model without Fine-Tuning 22](#_Toc106648786)

[Model 3B – Data Augmented InceptionResNetV2 Model with Fine-Tuning 26](#_Toc106648787)

[Model 4A – Fine-Tuned Mobile Net Model without Data Augmentation 29](#_Toc106648788)

[Model 4B – Fine-Tuned Mobile Net Model with Data Augmentation 32](#_Toc106648789)

[Section 5: Model Evaluation using Test Images 35](#_Toc106648790)

[Model #1B - Trained from Scratch with Data Augmentation 35](#_Toc106648791)

[Model #2B - Data Augmented VGG19 Model with Fine-Tuning 35](#_Toc106648792)

[Model #3B - Data Augmented InceptionResNetV2 Model with Fine-Tuning 36](#_Toc106648793)

[Model #4B – Data Augmented MobileNet Model with Fine-Tuning 36](#_Toc106648794)

[Ranking from Best to Worst Performing Model 36](#_Toc106648795)

[Section 6: Best model to perform Classification 37](#_Toc106648796)

[Section 7: Summary 39](#_Toc106648797)

[Model Performance Summary 39](#_Toc106648798)

[Suggestions on Further Improvements 40](#_Toc106648799)

# Section 1: Overview - Problem, Objective, and Approach

The purpose of this report seeks to provide readers a comprehensive understanding of how Convolutional Neural Networks could be utilized to perform multi-label image recognition in the context of food items. As a common misconception, unlike a multi-class image Classification which seeks to assign each sample to only one label the problem we are trying to tackle as part of this project involves multi-label classification problem whereby, we would be able to get multiple labels as output,Hence, by splitting the dataset into its training, validation, and testing datasets, the objective of this project would be to classify the following food items including apple\_pie, club\_sandwich, grilled\_cheese\_sandwich, guacamole, pancakes, pizza, prime\_rib, risotto, seaweed\_salad, and spring\_rolls through the utilization and optimization of models built from scratch alongside the usage of pre-trained models.

The approach of the following project would thus firstly encompass data loading whereby we will store our base directory, train directory, validation directory and test directory alongside data preprocessing where we intend to transform the raw images to obtain optimal model performance. Following, to develop the image classification models, we will be building a model trained from scratch, alongside 3 Transfer Learning models encompassing VGG19 – a convolutional neural network that is 19 layers deep, MobileNet – a convolutional neural network designed for mobile and embedded vision applications, and InceptionResNetV2 – a convolutional neural network that builds on the Inception family of architectures but incorporates residual connections

As to optimize the model that has been trained from scratch, some of the techniques that will be utilized as part of this project would include utilizing more Conv2D layers which aids in increasing validation accuracy, alongside adding Batch Normalization layers and Drop Out layers which would assist in reducing overfitting. Additionally, image augmentation techniques would be utilized to increase the number of images observed each epoch which would also increase both accuracy and reduce overfitting.

As to optimize the various pretrained models, we would utilize techniques including freezing the convolutional base, which would prevent their weights from getting updated during training, performing image augmentation which would assist in increasing both accuracy and reduce overfitting, alongside fine-tuning layers that encode more specialized features by unfreezing them instead. Furthermore, besides the various optimization technique previously discussed exclusive to each model, we would also be performing hyperparameter tuning and configuring different batch sizes.

Subsequently, to evaluate which of the following multi-label image classification model we have created would yield the best results, we will be utilizing testing accuracy on our test dataset to determine the best model before using it to make a prediction on unseen images from the internet. Hence, this would allow us to summarize our findings and make recommendations on further improvements to the model that could have potentially been made.

# Section 2: Data Loading

Text, letter

Description automatically generated

**Figure 1**

In the Data Loading stage, our objective would be to extract the 10 food items that we have been assigned with into the train, validation, and test datasets which follows the split ratio we have chosen of 800:100:100 respectively. Hence, with the train, validation, and test datasets defined in our local system, we can obtain the path for each of the dataset by using the OS module as observed from Figure 1. Thus, by using the os.path.join() method which concatenates the respective name of the dataset, it would allow us to obtain the file path of the training directory, validation directory, and test directory.

Graphical user interface, text, application

Description automatically generated

**Figure 2**

To then store the various food items from the text file we have been assigned with to a list, we could make use of the os.path.join() method as well to obtain the text file path located in our base directory. Hence, by reading the various food items in the text file, we can then loop through the list of items by subsequently storing them in ‘food\_list’ as observed from Figure 2.

Text

Description automatically generated

**Figure 3**

After which, given that we have previously mentioned that we would like to split the training, validation, and testing dataset according to the ratio of 800:100:100, the code as observed from Figure 3 would do the following by looping through each of the food item for the respective directories and copying 800 of each food item image into the training directory, 100 of each food item image into the validation directory, with 100 of each food item image into the testing directory out of the 1000 food image available for each food item.

# Section 3: Data Preprocessing

In the data preprocessing stage, the objective of the procedure would be to transform the raw images of the various food items to ensure that we are able to obtain optimal classification performance. Without preprocessing the images, it would lead to bad classification performances alongside longer training time as images are not scaled when they are being fed into the network.



Text

Description automatically generated

**Figure 4**

As observed from Figure 4 to perform data preprocessing, we could import the ImageDataGenerator class from the tensorflow.keras.preprocessing.image library which would allow us to transform the images into a NumPy array by decoding the JPEG content to RGB grids of pixels ranging from 0 to 255. Additionally, although we would use different arguments in the ImageDataGenerator class initialized across different models, the models would all have a common argument of rescale = 1./255 which would scale the NumPy array to value ranging from 0 to 1.

For our training and validation dataset, given that convolutional neural networks would require images of the same size as input, we will be using 150x150 as the default resolution for all our images in the training and validation dataset. This ultimately prevents shrinking which may cause deformation of features and patterns inside the image. Additionally, rather than having the entire sample of training and validation data to be propagated through the network at one time, we could utilize batch\_size to control the accuracy of estimate of the error gradient when training neural network. In the above example, we had set the batch\_size to 20 which indicates that 20 samples would be propagated through the network once at a time. Additionally, for our class\_mode, given that we have 10 different food items that are mutually exclusive, the ‘categorical’ class\_mode would be suitable for the categorical cross entropy loss.

# Section 4: Image Classification Model

## Model 1A – Model Trained from Scratch without Data Augmentation

**A picture containing table

Description automatically generated**

**Figure 5**

As observed from Figure 5, we could observe that our convolutional neural network model trained from scratch consist of a stack of alternated Conv2D layers using Rectified Linear Unit activation function, MaxPooling2D layers, alongside Batch Normalization layers. Additionally, as observed from the 2 Dense layers, given that we intend to classify 10 different types of food items, we would utilize 10 units as the end layer with a SoftMax activation function which would assist in assigning decimal probabilities to each class in a multi-class problem. Hence, the reason why we had utilized Batch Normalization layers would be because it would significantly increase the learning rate which further increases the speed at which the network trains. Thus, this would reduce the likelihood of overfitting. Furthermore, by applying layer weight regularization (i.e., kernel\_regularizer = l2(0.00005)) on the second Conv2D layers onwards, this would apply penalties on layer parameters which ultimately reduces chances of overfitting.

A picture containing text

Description automatically generated

**Figure 6**

As illustrated from Figure 6, which shows the compilation of our model, we could observe that in order to optimize the weights, we had utilized an optimizer value of ‘Adam’ – a variation to the classical stochastic gradient descent, and to find the deviation in the learning process, we had utilized a loss function of ‘categorical\_crossentropy’ given that it is suited for multi-class classification task whereby a sample can only belong to one out of the ten possible food items. Additionally, we had utilized a metrics of value ‘accuracy’ which would calculate how often predictions are equivalent to the labels.

Text

Description automatically generated

**Figure 7**

To prepare our food image dataset for training, we could observe Figure 7 which illustrates the data preprocessing procedures taken for our following model. Hence, for this particular model, we had firstly transformed the images into NumPy Array using the ImageDataGenerator class alongside rescaling the array values between 0 to 1 which allows for quicker computation and higher accuracy as part of the training and validation dataset. Additionally, to achieve quicker computation, we had resized the images to 150x150 which prevents issues such as shrinkage which may lead to loss of feature patterns. As for this particular model, we would be utilizing a batch size of 20 alongside a class\_mode of ‘categorical’ given that we have utilized ‘categorical\_crossentropy’ as our loss function.

**Graphical user interface, text, application, chat or text message

Description automatically generated**

**Figure 8**

Hence, to understand how well the following model that we have created generalizes similar data to that on which it was trained, we would use the fit function as observed from Figure 8 which utilizes a steps\_per\_epoch of 400, an iteration of 100 (i.e. epochs = 100), and a validation\_steps of 50. Hence, the steps\_per\_epoch value used could be derived from the following formula of steps per epoch = No. of samples in training set / batch size while the value used for validation steps (i.e., 50) could be derived from the following formula of validation steps = No. of samples in validation set / batch size. Additionally, to prevent the model from going through iterations that are meaningless, we had utilized early stopping which prevents the model from running further epochs once the maximum validation accuracy does not increase any further after 20 epochs or more(i.e., patience = 20).

**Chart

Description automatically generated**

**Graphical user interface, application

Description automatically generated**

**Figure 10**

From the above graphs (Figure 10) which illustrates the training and validation accuracy and loss, we could observe that overfitting has occurred such that while the training accuracy has increased linearly to around 94% accuracy at 85 epochs, the validation accuracy hovers around 55% accuracy. Likewise, by observing the training and validation loss, we could observe that while our validation loss reaches its minimum of around less than 10 epochs, our training loss has decreased consistently even as the rate of change may not be as significant. Additionally, from Figure 10, we could observe that the model has stopped training at 66 epochs although we have set an epoch to 100 as the early stopping callbacks had stopped training given that the maximum validation accuracy had not increase any further for 20 epochs.

## Model 1B – Model Trained from Scratch with Data Augmentation

Our main goal for Model 1B would thus be to minimize the overfitting that has occurred in Model 1A while also increasing the validation accuracy of the following mode. To thus reduce overfitting, we could make use of regularization techniques including more dropout layers throughout the sequential model alongside a larger number of batch normalization layers which would assist in reducing overfitting. Likewise, by using more Conv2D layers, this would allow our model to pick up more features which would lead to a higher validation accuracy.

**A picture containing text

Description automatically generated**

**Figure 11**

As observed from Figure 11, our optimized convolutional neural network model trained from scratch consists of a stack of alternated Conv2D layers and Batch Normalization layers twice before MaxPooling2D layers and a Dropout layer of value 0.2 is being utilized. Additionally, as observed from the 3 Dense layers, given that we intend to classify 10 different types of food items, we would utilize 10 units as the end layer with a SoftMax activation function which would assist in assigning decimal probabilities to each class in a multi-class problem. Hence, the reason why we had utilized Batch Normalization layers would be because it would significantly increase the learning rate which further increases the speed at which the network trains. Thus, this would reduce the likelihood of overfitting. Furthermore, by applying a Dropout layer after every MaxPooling2D layer, it would assist in randomly setting 20% of input unit to 0 with a frequency of rate at each step during training time, which prevents overfitting.



**Figure 12**

Additionally, as observed from Figure 12, although most our variables remain constant, the only argument we had changed would be to use a learning rate of 2\*1e-4 which is much faster than the ‘Adam’ optimizer we had previously utilized. Hence, it would allow our model to learn faster which prevents overfitting.

Text, application

Description automatically generated

**Figure 12**

Furthermore, to ensure that overfitting does not occur due to the lack of variety of samples, which renders our model from training unseen data, we could utilize data augmentation which seeks to generate different variations of our food dataset artificially to increase its size after every epoch. Hence, from Figure 12 which illustrates the data augmentation technique we had utilized, we had firstly rescaled the image to a value between 0 and 1 as usual alongside setting a rotation of 40 which randomly rotates the picture by 40 degrees. Furthermore, given that rotated images may lose some of its features, we could replace the missing features using the fill\_mode argument of value ‘nearest’ which copies over the nearest values possible. Additionally, by setting a value of 0.25 to both width and height shift, it would move the object image by 20% horizontally and vertically. Other arguments we had utilized would also contain shear range, we seek to rectify the perception angles, horizontal flip as the name suggest, alongside brightness range which control the bright of the variety of images. As for this particular model, we would be utilizing a batch size of 10 instead of 20 which would lead to higher accuracy.

**Text

Description automatically generated with medium confidence**

**Figure 13**

Hence, to understand how well the following model that we have created generalizes similar data to that on which it was trained, we would use the fit function as observed from Figure 13 which utilizes a steps\_per\_epoch of 800, an iteration of 100 (i.e. epochs = 100), and a validation\_steps of 100. Hence, the steps\_per\_epoch value used could be derived from the following formula of steps per epoch = No. of samples in training set / batch size while the value used for validation steps (i.e., 100) could be derived from the following formula of validation steps = No. of samples in validation set / batch size. Additionally, to prevent the model from going through iterations that are meaningless, we had utilized early stopping which prevents the model from running further epochs once the maximum validation accuracy does not increase any further after 20 epochs or more (i.e., patience = 20).

Line chart

Description automatically generated with medium confidence

Chart

Description automatically generated

**Figure 14**

As a result of applying data augmentation techniques alongside using a larger number of dropout layers and batch normalization layers, it could be observed from Figure 14, which illustrates the training and validation accuracy and loss that the model is no longer overfitting. This is evident from the fact that both training and validation accuracy and loss are rather close to one another which shows that there is not much deviation between the accuracy and loss of training and validation dataset. Furthermore, as a result of using more Conv2D layers alongside a smaller batch size, it could be observed that although our training accuracy has decreased from 94% to approximately 75.35%, we have managed to increase our validation accuracy from 52.40% to over 70% as well. Lastly, from Figure 14, we could observe that although we have set an epoch to 100 the early stopping callbacks had not been triggered.

## Model 2A – Data Augmented VGG19 Model without Fine-Tuning

Of the three pretrained model that will be covered in this report, the first pre-trained convolutional neural network that we would be looking at involves the VGG19 model, where the number 19 stands for the number of layers with trainable weights (i.e., 16 Convolutional layers and 3 Fully Connected layers). Hence, as part of model 2A, we will be performing feature extraction with data augmentation by extending the conv\_base model through adding dense layers and running the whole thing end-to-end on input data. Thus, although the model would be more computationally more expensive to run, it would allow us to leverage data augmentation given that the input images would go through the convolutional base every time it is seen by the VGG19 model.

Text

Description automatically generated

**Figure 15**

As illustrated from Figure 15, to instantiate the VGG19 model, we could firstly import the model through the tensorflow.keras.applications library whereby we can declare the VGG19 class of weights value ‘imagenet’, setting a value of False to input\_top given that we intend to use our own densely-connected classifier of 10 food classes instead of the 1000 classes from ImageNet, as well as configuring the input shape to 150x150 which represents the shape of the image tensors that will be fed into the network.

Text

Description automatically generated with medium confidence

**Figure 16**

To add our own densely connected classifier, we could utilize the VGG19 model the same way a sequential model is being built. Hence, on top of convolutional blocks of the VGG19 model, we will be adding a flatten layer which flattens the multi-dimensional input tensors into a single dimension before feeding it into the dense layers which has a unit of 10 to classify the 10 different food images. Furthermore, as illustrated from the model summary in Figure 16, we could observe that the convolutional base of VGG19 has over 20 million parameters on top of the 4 million parameters of the classifier. Hence, to ensure that the weights of the convolutional base do not get updated, we can perform freezing so that the representations do not get modified during training. Otherwise, given that we have not trained our classifier, the error signal that is propagated through the network would be too large.

Table

Description automatically generated

**Figure 17**

Hence, to freeze the convolutional base of VGG19, we could simply do so by setting its trainable attribute to False as illustrated from Figure 17. Thus, as illustrated from the model summary before and after freezing, we could observe that the number of trainable parameters has decreased from over 24 million parameters to around 4 million trainable parameters.

Text

Description automatically generated

**Figure 18**

Thereafter, to ensure that overfitting does not occur due to the lack of variety of samples, which renders our model from training unseen data, we could utilize data augmentation on only our training data which seeks to generate different variations of our food dataset artificially to increase its size after every epoch. Hence, from Figure 18 which illustrates the data augmentation technique we had utilized, we had firstly rescaled the image to a value between 0 and 1 as usual alongside setting a rotation of 40 which randomly rotates the picture by 40 degrees. Furthermore, given that rotated images may lose some of its features, we could replace the missing features using the fill\_mode argument of value ‘nearest’ which copies over the nearest values possible. Additionally, by setting a value of 0.25 to both width and height shift, it would move the object image by 20% horizontally and vertically. Other arguments we had utilized would also contain shear range, we seek to rectify the perception angles, horizontal flip as the name suggest, alongside brightness range which control the bright of the variety of images.

As for this particular VGG19 model, we would be utilizing a batch size of 20 alongside a class\_mode of ‘categorical’ given that we have utilized ‘categorical\_crossentropy’ as our loss function.

Hence, to understand how well the following model that we have created generalizes similar data to that on which it was trained, we would use the fit function as observed from Figure 18 which utilizes a steps\_per\_epoch of 400, an iteration of 60 (i.e. epochs = 60), and a validation\_steps of 50. Hence, the steps\_per\_epoch value used could be derived from the following formula of steps per epoch = No. of samples in training set / batch size while the value used for validation steps (i.e., 50) could be derived from the following formula of validation steps = No. of samples in validation set / batch size. Additionally, to prevent the model from going through iterations that are meaningless, we had utilized early stopping which prevents the model from running further epochs once the maximum validation accuracy does not increase any further after 10 epochs or more (i.e., patience = 10).

A picture containing shape

Description automatically generated

A picture containing shape

Description automatically generated

**Figure 19**

Thus, by observing the training and validation accuracy and loss as illustrated from Figure 19, we could observe that there are no signs of overfitting given that both training and validation accuracy has similar level of accuracy which may be attributed to the data augmentation we had performed. In fact, from 0 to 50 epochs, we could observe that the model had been underfitting as observed from the higher validation accuracy compared to training accuracy before being resolved afterwards. Hence, as observed at the end of 60 epochs, we managed to receive a training accuracy of 64.33% and a validation accuracy of 64.80%. While on the other hand, by the end of 60 epochs, the model had training loss of 1.03 and a validation loss of 1.03 as well.

## Model 2B – Data Augmented VGG19 Model with Fine-Tuning

Given that there are no issues in Model 2A regarding overfitting or underfitting our objective for model 2B would thus be to mainly increase our overfall training and validation accuracy given that the performance of Model 2A was relatively low (60+% for both training and validation accuracy) compared to the model we had built from scratch in Model 1A and 1B. Hence, given that we have already froze our VGG19 convolutional base and have trained our fully connected classifier as evident Model 2A, the next step would be to unfreeze some layers at the top of the VGG19 model so that we would be able to fine-tune the more specialized features rather than fine-tuning the entire convolutional base which may lead to overfitting and wasting computing resources to train generical features. Thus, this would lead to an overall increase in both training and validation accuracy.

Table

Description automatically generated

**Figure 20**

Looking back at our VGG19 convolutional base, we could observe that the VGG19 base has a total of 0 trainable parameter given that we have previously frozen all of the convolutional layers in Model 2A. Thus, as we would only want to fine-tune layers higher up given that they are able to encode more specialized features, we will set block5\_conv1 layer onwards to be trainable while all previous layers before block5\_conv1 would remain frozen.

Table

Description automatically generated

**Figure 21**

Thus, by setting layers beyond block5\_conv1 to be trainable (unfrozen) while the layers before that remain frozen, we could observe that we have an increase in trainable parameters from over 4 million trainable parameters to over 13 million trainable parameters.

Text

Description automatically generated

**Figure 22**

Hence, to understand how well the following model that we have created generalizes similar data to that on which it was trained, we would use the fit function as observed from Figure 22 which utilizes a steps\_per\_epoch of 400, an iteration of 60 (i.e. epochs = 60), and a validation\_steps of 50. Hence, the steps\_per\_epoch value used could be derived from the following formula of steps per epoch = No. of samples in training set / batch size while the value used for validation steps (i.e., 50) could be derived from the following formula of validation steps = No. of samples in validation set / batch size. Additionally, to prevent the model from going through iterations that are meaningless, we had utilized early stopping which prevents the model from running further epochs once the maximum validation accuracy does not increase any further after 10 epochs or more (i.e., patience = 10).

Chart

Description automatically generated with medium confidence

Chart

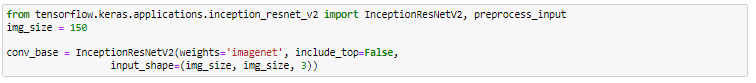
Description automatically generated

**Figure 23**

Thus, by observing the training and validation accuracy and loss as illustrated from Figure 23, we could observe that there are no signs of overfitting or underfitting given that both training and validation accuracy has similar level of accuracy which may be attributed to the fine-tuning that we had performed. In fact, from 0 to 6 epochs, although we could observe that the model had been underfitting as observed from the higher validation accuracy compared to training accuracy, it has been resolved afterwards. Hence, as observed at the end of 17 epochs, we managed to receive a training accuracy of 72.48% and a validation accuracy of 72.00%. While on the other hand, at the end of 17 epochs, the model had training loss of 0.9067 and a validation loss of 1.1785 as well.

## Model 3A – Data Augmented InceptionResNetV2 Model without Fine-Tuning

The next pre-trained convolutional neural network that we will be looking at involves InceptionResNetV2, which is a convolutional neural network that is trained on more than a million images from the ImageNet database. Additionally, the network is 164 layers deep which is far more complex than the VGG19 model previously discussed given that it has a total of over 54 million trainable parameters compared to the VGG19 model which had a total of over 24 million trainable parameters. Hence, as part of model 3A, we will be performing feature extraction with data augmentation by extending the conv\_base model through adding dense layers and running the whole thing end-to-end on input data. Thus, although the model would be more computationally more expensive to run, it would allow us to leverage data augmentation given that the input images would go through the convolutional base every time it is seen by the InceptionResNetV2 model.



**Figure 24**

As illustrated from Figure 24, to instantiate the InceptionResNetV2 model, we could firstly import the model through the tensorflow.keras.applications.inception\_resnet\_v2 library whereby we can declare the InceptionResNetV2 class of weights value ‘imagenet’, setting a value of False to input\_top given that we intend to use our own densely-connected classifier of 10 food classes instead of the 1000 classes from ImageNet, as well as configuring the input shape to 150x150 which represents the shape of the image tensors that will be fed into the network.

Text, email

Description automatically generated

**Figure 25**

To add our own densely connected classifier, we could utilize the InceptionResNetV2 model the same way a sequential model is being built. Hence, on top of convolutional blocks of the InceptionResNetV2 model, we will be adding a flatten layer which flattens the multi-dimensional input tensors into a single dimension before feeding it into the dense layers which has a unit of 10 to classify the 10 different food images. Furthermore, as illustrated from the model summary in Figure 25, we could observe that the convolutional base of InceptionResNetV2 has over 54 million parameters on top of the 1.3 million parameters of the classifier. Hence, to ensure that the weights of the convolutional base do not get updated, we can perform freezing so that the representations do not get modified during training. Otherwise, given that we have not trained our classifier, the error signal that is propagated through the network would be too large.

Text, table

Description automatically generated

**Figure 26**

Hence, to freeze the convolutional base of InceptionResNetV2, we could simply do so by setting its trainable attribute to False as illustrated from Figure 26. Thus, as illustrated from the model summary before and after freezing, we could observe that the number of trainable parameters has decreased from over 55 million parameters to around 1.3 million trainable parameters.

Text

Description automatically generated

**Figure 27**

Thereafter, to ensure that overfitting does not occur due to the lack of variety of samples, which renders our model from training unseen data, we could utilize data augmentation on only our training data which seeks to generate different variations of our food dataset artificially to increase its size after every epoch. Hence, from Figure 27 which illustrates the data augmentation technique we had utilized, we had firstly rescaled the image to a value between 0 and 1 as usual alongside setting a rotation of 40 which randomly rotates the picture by 40 degrees. Furthermore, given that rotated images may lose some of its features, we could replace the missing features using the fill\_mode argument of value ‘nearest’ which copies over the nearest values possible. Additionally, by setting a value of 0.25 to both width and height shift, it would move the object image by 25% horizontally and vertically. Other arguments we had utilized would also contain shear range, we seek to rectify the perception angles, horizontal flip as the name suggest, alongside brightness range which control the bright of the variety of images. Moreover, as for this particular InceptionResNetV2model, we would be utilizing a batch size of 20 alongside a class\_mode of ‘categorical’ given that we have utilized ‘categorical\_crossentropy’ as our loss function.

Hence, to understand how well the following model that we have created generalizes similar data to that on which it was trained, we would use the fit function as observed from Figure 27 which utilizes a steps\_per\_epoch of 400, an iteration of 60 (i.e. epochs = 60), and a validation\_steps of 50. Hence, the steps\_per\_epoch value used could be derived from the following formula of steps per epoch = No. of samples in training set / batch size while the value used for validation steps (i.e., 50) could be derived from the following formula of validation steps = No. of samples in validation set / batch size. Additionally, to prevent the model from going through iterations that are meaningless, we had utilized early stopping which prevents the model from running further epochs once the maximum validation accuracy does not increase any further after 15 epochs or more (i.e., patience = 15).

A picture containing text

Description automatically generated

A picture containing shape

Description automatically generated

**Figure 27**

Thus, by observing the training and validation accuracy and loss as illustrated from Figure 27, we could observe that although there are no signs of overfitting, we could observe that the InceptionResNetV2 model is slightly underfitting as observed from the marginally higher validation accuracy compared to the training accuracy. Hence, at the end of 60 epochs, we could observe that the model is able to produce a training accuracy of 76.35% and a validation accuracy of 79.30%. On the other hand, by the end of 60 epochs, the model had a training loss of 0.6876 and a validation loss of 0.6560 as well.

## Model 3B – Data Augmented InceptionResNetV2 Model with Fine-Tuning

Given that the Model 3A is only marginally underfitting, our objective for model 3B would thus be to mainly increase our overall training and validation accuracy given that the performance of Model 3A was relatively average (70+% for both training and validation accuracy) compared to the model we had built from scratch in Model 1A and 1B. Hence, given that we have already frozen our InceptionResNetV2 convolutional base and have trained our fully connected classifier as evident Model 3A, the next step would be to unfreeze some layers at the top of the InceptionResNetV2 model so that we would be able to fine-tune the more specialized features rather than fine-tuning the entire convolutional base which may lead to overfitting and wasting computing resources to train generical features. Thus, this would lead to an overall increase in both training and validation accuracy.

Table

Description automatically generated

**Figure 28**

Looking back at our InceptionResNetV2 convolutional base, we could observe that the InceptionResNetV2 base has a total of 0 trainable parameter given that we have previously frozen all of the convolutional layers in Model 3A. Thus, as we would only want to fine-tune layers higher up given that they are able to encode more specialized features, we will set block8\_9\_conv layer onwards to be trainable while all previous layers before block8\_9\_conv would remain frozen.

Table

Description automatically generated with medium confidence

**Figure 29**

Thus, by setting layers beyond block8\_9\_conv to be trainable (unfrozen) while the layers before that remain frozen, we could observe that we have an increase in trainable parameters from over 3 million trainable parameters to over 7 million trainable parameters in the model.

Text

Description automatically generated

**Figure 30**

Hence, to understand how well the following model that we have created generalizes similar data to that on which it was trained, we would use the fit function as observed from Figure 30 which utilizes a steps\_per\_epoch of 400, an iteration of 60 (i.e. epochs = 60), alongside a validation\_steps of 50. Hence, the steps\_per\_epoch value used could be derived from the following formula of steps per epoch = No. of samples in training set / batch size while the value used for validation steps (i.e., 50) could be derived from the following formula of validation steps = No. of samples in validation set / batch size. Additionally, to prevent the model from going through iterations that are meaningless, we had utilized early stopping which prevents the model from running further epochs once the maximum validation accuracy does not increase any further after 15 epochs or more (i.e., patience = 15).

A picture containing line chart

Description automatically generated

A picture containing graphical user interface

Description automatically generated

**Figure 31**

Thus, by observing the training and validation accuracy and loss as illustrated from Figure 31, we could observe that although there are no signs of overfitting or underfitting in the first 18 epochs, we could observe that the model has started to overfit afterwards as observe from the increasing deviation between the training and validation accuracy. Additionally, by observing the Training and Validation Loss from Figure 31, we could likewise observe that the model if starting to overfit at around 18 epochs as observed from the increasing deviation between the training and validation loss.

Hence, as observed at the end of 46 out of the 60 epochs to be run due to early stoppage, we managed to receive a training accuracy of 86.26% and a validation accuracy of 78.60%. While on the other hand, at the end of 46 out of the 60 epochs, the model had training loss of 0.4075 and a validation loss of 0.7280 as well.

## Model 4A – Fine-Tuned Mobile Net Model without Data Augmentation

The fourth and last convolutional neural network that we will be covering involves Mobile Net, which is a convolutional neural network designed for embedded vision and mobile applications. Hence, its unique characteristic would be that it is a lightweight deep neural network that has relatively few parameters compared to most pre-trained models while still maintaining a high classification accuracy. Unlike how we have previously built our pre-trained models, we will firstly perform fine-tuning before performing data augmentation as our optimization technique.

Graphical user interface, text, application

Description automatically generated Table

Description automatically generated with medium confidence

**Figure 32**

Hence, to instantiate the Mobile Net model, we could make use of the keras.applications.mobilenet.MobileNet() and storing it under the variable mobile. Thus, by observing the model summary, we could observe that the Mobile Net Convolutional base has a total of over 4 million parameters to be trained which is significantly less than what we have observed from the VGG19 and InceptionResNetV2 convolutional base.

Table

Description automatically generated with medium confidence

**Figure 33**

Subsequently, given that we would like to add our own fully connected dense layer in order to classify the 10 different types of food images, we had removed the 6 last layers of the Mobile Net Model which includes the classifier for the 1000 different classes of the ImageNet database alongside adding our own output layer that contains 10 output nodes which corresponds to each of the food item. Thus, by building the model with the Keras API and printing out the model summary, we could observe that we have added in our dense layer of 10 units which represents the size of the output. Additionally, the number of parameters has decreased to 3 million parameters given that we have removed the last 6 layers as previously discussed.

**Graphical user interface, application, Word

Description automatically generated with medium confidence**

**Figure 33**

As we would like to keep what the original Mobile Net model has already learned from the ImageNet database, we can do so by freezing the weights of the majority of layers. However, as we would also like to fine-tune some of the layers from the Mobile Net convolutional base in order to perform transfer learning on our custom food dataset, we can do so by training only the last 23 layers of the convolutional base as observed from Figure 23. Hence, while the last 23 layers are trainable, all layers before the last 23 layers would be untrainable.

Graphical user interface, text, application

Description automatically generated

**Figure 34**

Hence, to understand how well the following model that we have created generalizes similar data to that on which it was trained, we would use the fit function as observed from Figure 34 which utilizes a steps\_per\_epoch of 400, an iteration of 60 (i.e. epochs = 60), and a validation\_steps of 50. Hence, the steps\_per\_epoch value used could be derived from the following formula of steps per epoch = No. of samples in training set / batch size while the value used for validation steps (i.e., 50) could be derived from the following formula of validation steps = No. of samples in validation set / batch size. Additionally, to prevent the model from going through iterations that are meaningless, we had utilized early stopping which prevents the model from running further epochs once the maximum validation accuracy does not increase any further after 10 epochs or more (i.e., patience = 15).

Chart

Description automatically generated

Chart, line chart

Description automatically generated

**Figure 35**

Thus, as observed from Figure 35 which shows that the model has stopped running further after 22 epochs due to the callback argument we had utilized, we could observe that the Mobile Net model has reached a training and validation accuracy of 99.89% and 79.80% respectively. Additionally, by observing the training and validation loss plot, we could observe that the model has a training loss close to 0 with an increasing validation loss of 0.8. Hence, as illustrated from Figure 35, we could observe that the model is significantly overfitted as evident of the large and increasing deviation between the training and validation accuracy. Thus, to solve this issue we could try to perform data augmentation in Model 4b so as to increase the variation of food images observed by the model each epoch which would lead to an increase in validation accuracy and reduce overfitting.

## Model 4B – Fine-Tuned Mobile Net Model with Data Augmentation

Hence, given that we have previously observed from Model 4A that the ImageNet model is overfitting and has a low validation accuracy, this can be resolved by leveraging data augmentation which would increase the variation of food images observed by the model after each iteration. Additionally, we would also be performing hyperparameter tuning that may reduce the likelihood of overfitting.

Graphical user interface, text, application

Description automatically generated

**Figure 36**

Similar to how we have built the ImageNet model from Model 4A, the only modification we would make would be to introduce l2 regularization technique on the fully connected Dense layer that we will be adding onto the convolutional base. Hence, by using a value of l2(0.001) on the Dense layer argument of activity\_regularizer (i.e., regularization function applied to the output layer) and kernel\_regularizer (i.e., regularization function applied to the output layer), this would reduce the likelihood of overfitting the model.



**Figure 37**

Likewise, as observed from Model 4A, as we would like to keep what the original Mobile Net model has already learned from the ImageNet database, we can do so by freezing the weights of the majority of layers. However, as we would also like to fine-tune some of the layers from the Mobile Net convolutional base in order to perform transfer learning on our custom food dataset, we can do so by training only the last 23 layers of the convolutional base as observed from Figure 23. Hence, while the last 23 layers are trainable, all layers before the last 23 layers would be untrainable.

Text

Description automatically generated

**Figure 37**

Thereafter, to ensure that overfitting does not occur due to the lack of variety of samples, which renders our model from training unseen data, we could utilize data augmentation on only our training data which seeks to generate different variations of our food dataset artificially to increase its size after every epoch. Hence, from Figure 37 which illustrates the data augmentation technique we had utilized, we had firstly rescaled the image to a value between 0 and 1 as usual alongside setting a rotation of 40 which randomly rotates the picture by 40 degrees. Furthermore, given that rotated images may lose some of its features, we could replace the missing features using the fill\_mode argument of value ‘nearest’ which copies over the nearest values possible. Additionally, by setting a value of 0.25 to both width and height shift, it would move the object image by 25% horizontally and vertically. Other arguments we had utilized would also contain shear range, we seek to rectify the perception angles, horizontal flip as the name suggest, alongside brightness range which control the bright of the variety of images. Moreover, as for this particular MobileNet model, we would be utilizing a batch size of 10 alongside a class\_mode of ‘categorical’ given that we have utilized ‘categorical\_crossentropy’ as our loss function.

Hence, to understand how well the following model that we have created generalizes similar data to that on which it was trained, we would use the fit function as observed from Figure 27 which utilizes a steps\_per\_epoch of 800, an iteration of 60 (i.e. epochs = 60), and a validation\_steps of 100. Hence, the steps\_per\_epoch value used could be derived from the following formula of steps per epoch = No. of samples in training set / batch size while the value used for validation steps (i.e., 100) could be derived from the following formula of validation steps = No. of samples in validation set / batch size. Additionally, to prevent the model from going through iterations that are meaningless, we had utilized early stopping which prevents the model from running further epochs once the maximum validation accuracy does not increase any further after 20 epochs or more (i.e., patience = 20).

Chart

Description automatically generated

A picture containing graphical user interface

Description automatically generated

**Figure 38**

Thus, as observed from Figure 38 which shows that the model has stopped running further after 44 epochs due to the callback argument we had utilized, we could observe that the Mobile Net model has reached a training and validation accuracy of 92.63% and 84.00% respectively. Additionally, by observing the training and validation loss plot, we could observe that the model has a training loss of 0.2451 with a constant validation loss of 0.6334. Hence, as illustrated from Figure 35, although we could observe that the model is model is still overfitting, the deviation of overfitting has decreased significantly between the training and testing accuracy. Thus, it could be observed that Data Augmentation and hyperparameter tuning plays an effective role in minimizing overfitting.

# Section 5: Model Evaluation using Test Images

With the various convolutional neural network models that we have built and optimized, this section of the report seeks to illustrate how the various models trained from scratch and pre-trained models of VGG19, IncepionResNetV2, and MobileNet would perform on our food image test dataset. Hence, the metrics that we will primarily be focusing on involves testing accuracy with our secondary metrics being the testing loss of each model.

**Graphical user interface, text, application

Description automatically generated**

**Figure 39**

Hence, to pass the test images into our environment, we could make use of the flow\_from\_directory() method which takes a path of the test images directory and generates batches of augmented data of batch size 20. Additionally, by utilizing a class\_mode of value ‘categorical’, it would allow us to generate a 2D one-hot encoded labels.

## Model #1B - Trained from Scratch with Data Augmentation

**Text

Description automatically generated**

By loading the data augmented model that has been **trained from scratch** alongside utilizing the evaluate() method, we could observe that the convolutional neural network has a test accuracy of 72.900% (3 d.p.) with a test loss score of 0.953(3 d.p.).

## Model #2B - Data Augmented VGG19 Model with Fine-Tuning

**Graphical user interface, text, application, email

Description automatically generated**

By loading the data augmented **VGG19 model** that has been fine-tuned alongside utilizing the evaluate() method, we could observe that the pre-trained convolutional neural network has a test accuracy of 72.000% (3 d.p.) with a test loss score of 1.072(3 d.p.).

## Model #3B - Data Augmented InceptionResNetV2 Model with Fine-Tuning

**Graphical user interface, text, application

Description automatically generated**

By loading the data augmented **InceptionResNetV2** model that has been fine-tuned alongside utilizing the evaluate() method, we could observe that the pre-trained convolutional neural network has a test accuracy of 80.000% (3 d.p.) with a test loss score of 0.655(3 d.p.).

## Model #4B – Data Augmented MobileNet Model with Fine-Tuning

**Graphical user interface, text, application

Description automatically generated**

By loading the data augmented **MobileNet** model that has been fine-tuned alongside utilizing the evaluate() method, we could observe that the pre-trained convolutional neural network has a test accuracy of 85.700% (3 d.p.) with a test loss score of 0.515(3 d.p.).

## Ranking from Best to Worst Performing Model

Thus, by observing the various models and their respective test accuracy and test loss, we could observe that the MobileNet model would be our recommended model given that it has the highest test accuracy of 85.70% alongside the lowest test loss score of 0.52. Hence, the following model would be able to produce the highest accuracy on unseen images while also having the lowest frequency of misclassification.

The second-best model that we have built and optimized involves the InceptionV2ResNet model which had a second-highest test accuracy of 80% and a second lowest test loss score of 0.66.

The third-best performing model that we have built and optimized involves the model we had trained from scratch which had a third-highest test accuracy of 72.90% alongside the second highest test loss score of 0.95

On the other hand, the worst performing model can be attributed to the VGG19 model which had the lowest test accuracy of 72.00% alongside the highest loss score of 1.07.

# Section 6: Best model to perform Classification

**Graphical user interface, text

Description automatically generated**

**Figure 40**

Hence, to apply the Data Augmented MobileNet model which has been Fine-Tuned on real life images from the internet, we could observe Figure 40 which illustrates the various sequences of process. Hence, from the first line of code, given that we have previously saved the model under ‘food\_model\_best.h5’, we can load the model back by utilizing the load\_model() method. To then review the food items that we have been assigned with, we could make use of the readline() method which would read the text data from ’60.txt’, in which we will append the items into a list named, food\_list. Subsequently, we will creating two functions whereby image\_process() would load and resize the input image to 224x224 given that our MobileNet model had also been trained using the input image size of 224x224 while the prediction() method would assist in displaying each food item and their respective prediction probability.

**Graphical user interface, text

Description automatically generated**

**Figure 41**

To then make a prediction with a food image (i.e., Spring Rolls) we had downloaded from the internet, we could make use of the following lines of command as observed from Figure 41 which would display the label of the image that the MobileNet model has predicted alongside the respective probabilities of each label regarding the image chosen. Thus, from Figure 41, with regards to the spring roll image we have chosen, we could observe that the MobileNet model has been able to correctly classify the food image to its label with a 96.85% accuracy, while the rest of the probabilities is split among other various food items.

# Section 7: Summary

## Model Performance Summary

|  |  |  |
| --- | --- | --- |
| **CNN Model** | **Test Accuracy** | **Test Loss** |
| Model 1A – Model Trained from Scratch | NA | NA |
| Model 1B – Model Trained from Scratch with Data Augmentation | 72.90% | 0.95 |
| Model 2A – VGG19 Model with Data Augmentation | NA | NA |
| Model 2B – Fine-Tuned VGG19 Model with Data Augmentation | 72.00% | 1.07 |
| Model 3A – InceptionResNetV2 Model with Data Augmentation | NA | NA |
| Model 3B – Fine-Tuned InceptionResNetV2 Model with Data Augmentation | 80.00% | 0.65 |
| Model 4A - Fine-Tuned Mobile Net Model | NA | NA |
| Model 4B – Fine-Tuned Mobile Net model with Data Augmentation | 85.7% | 0.52 |
| Best Model (Model 4B) on internet images of ‘Spring Rolls’ | 96.85% | NA |

Note – Model with ‘A’ as its identification represent the unoptimized model. Thus, we will not be analyzing them in terms of their testing accuracy and loss as represented by ‘NA’.

## Suggestions on Further Improvements

One of the improvements that could be made would be to test on larger pre-trained models. Given that we have already experimented with InceptionResNetV2 which already has over 50 million trainable parameters, we could explore other pre-trained models in the interest of time such as NasNet-A and AmoebaNet-A which are models with larger parameters are able to produce higher level of accuracy as observed from their performance on the ImageNet database.

Additionally, besides the data augmentation technique we have performed including rotating the image, shifting the image vertically and horizontally, and controlling the brightness of the image from ImageDataGenerator, we could make use of MixUp to augment our images in order to further reduce overfitting and increasing overall accuracy. Hence, how this process works would be that two images from the training dataset would be placed on top of one another whereby their transparency could be adjusted to form new images. Hence, this would allow us to improve the generalization of the neural network architecture that we have created.

The last suggestion on further improvements that could be taken would be to instead test on individual types of images on top of tracking the overall accuracy of the model on all types of food. Hence, this would allow us to determine which types of food the model is able to accurately classify while also being able to determine which types of food the model is mostly misclassifying. Thus, this would allow us to make an informed decision on what hyperparameter to tune in order to better classify a certain type of food image rather than performing hyperparameter tuning using trial and error.

**THE END**