

**School of InfoComm Technology**

**Deep Learning Assignment**

Diploma in DS / FI / IT

Year 2 (2022/23), Semester 4

**ASSIGNMENT 1**

(30% of DL Module)

**Submission Deadline:**

**Presentation: 18th Dec 2022 11:59PM**

**Report and Code: 18th Dec 2022 11:59PM**

|  |  |  |
| --- | --- | --- |
| **Tutorial Group** | **:** | **T01 / T02** |
| **Student Name** | **:** | **Chong Xin Le** |
| **Student Number** | **:** | **S10221973D** |

**Penalty for late submission:**

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

**NO** submission will be accepted after 24th Dec 2022, 11:59PM.

**Table of Contents**

[1. Overview 3](#_Toc122375699)

[2. Data Pre-processing and Data Loading 4](#_Toc122375700)

[3. Developing Image Classification Model 5](#_Toc122375701)

[3.1 Model trained from scratch using conv2D & dense layers 5](#_Toc122375702)

[3.2 Model utilizing pre-trained models 12](#_Toc122375703)

[4. Evaluate models using Test images 16](#_Toc122375704)

[4.1 Compare model’s performance 16](#_Toc122375705)

[4.2 Best Model 17](#_Toc122375706)

[5. Use the Best Model to perform classification 18](#_Toc122375707)

[5.1 Explain how to apply the model on real life images 18](#_Toc122375708)

[5.2 Explain and analyze the model prediction 18](#_Toc122375709)

[6. Summary 20](#_Toc122375710)

# Overview

This assignment aims at enhancing the classification and prediction accuracy of food images through Convolutional Neural Network (CNN) and Artificial Neural Network (ANN) algorithm. Multiple models utilizing cov2D, dense layers as well as pre-trained models are developed and compared. Through this, we find the best model to make our predictions on the food images.

First, since we already separated the data into train and validation and kept it into their respective folders. We will use the Image Data Generator available in Keras to read images in batches directly from these folders and optionally perform data augmentation.

We start with our input layers with 3 channels since we are using RGB images. Then, we utilize the Conv2D class to implement a CNN model. We perform convolution and apply a Relu activation function. This is followed by a fully connected layer and finally the SoftMax function to classify the images. Max-pooling has been used to reduce the parameters and improve our overall computation of the network. Accuracy is enhanced by using regularization, dropouts as well as reducing and adjusting epochs, batch, learning rate, and network size.

As stated, pre-trained models have also been utilized in order to achieve better recognition accuracy. This includes deep learning neural networks like ResNet50, Inception-V3, and VGG-16 are also used to figure out which neural network works best on our data. All models were trained with pretrained imageNet weights. In addition, we tested our accuracy on our pre-trained model with and without data augmentation. Data augmentation can improve the performance of our model by augmenting the data we already have. The data augmentation techniques mainly focus around geometric transformations such as flipping, cropping, rotating and zooming images.

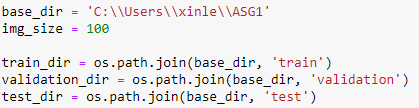
Lastly, we fine-tune our already trained network. This process includes freezing weights of a particular layer, whereby this layer would not be trained.

After every model, the loss and accuracy curve are also plotted. The developed models are also evaluated using test images to obtain an accuracy score. These scores and curves are all compared for us to make changes to better our model. Through this, the best model can be recommended which is used to make our prediction.

 The goal of this project is to experiment with applying CNN techniques to classify our food images accurately. It is important that we know how to handle the overfitting or underfitting problem and apply these techniques to solve this issue. These techniques can include tuning parameters like epochs and learning, using image augmentation and adjusting network size.

# Data Pre-processing and Data Loading

Specify our train, test, and validation image datasets into a directory structure. Img\_size will be 100x100 as our computation power is slower.



ImageDataGenerator is used to load a single dataset in batches. To load our images, the ‘flow\_from \_directory’ function is used while specifying the dataset directory. The function also allows us to configure our target size whereby all our images will be loaded to a specific size, in this case which is (100,100). The batch size is 25, which mean 25 randomly selected image from across the classes in the dataset will be returned in each batch when training. Our classification task via the ‘class\_mode’ argument is a multi-class classification ‘categorical’. The train data augmentation configuration has also been specified.

Text

Description automatically generated

# Developing Image Classification Model

## Model trained from scratch using conv2D & dense layers

**Baseline Model**

This is my baseline model whereby I will work around this model to improve our accuracy. The first Conv2D layer learns a total of 32 filters. Max pooling is then used to reduce the spatial dimensions of the output volume. We then learn 64 filters on the second and third Conv2D layer. Again, max pooling is used for both layers again. Our kernel size is kept at 3x3 throughout. We also apply a Relu activation function for all Conv2D layers.

A flattened layer and three fully connected layers with 20,20 and 10 nodes respectively are appended to the CNN. Finally, a SoftMax classifier is added to the network.

Text

Description automatically generated

Compiled the model using RMSprop optimizer with a learning rate of 0.0001. Categorical cross entropy function is used to calculate the model’s loss. since the data has more than one label class. Epoch of 30 is used.

Text

Description automatically generated

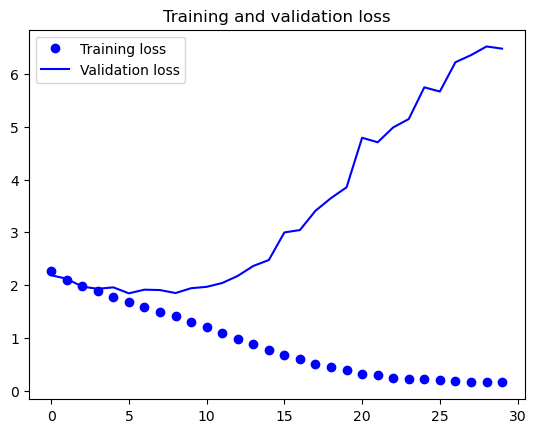
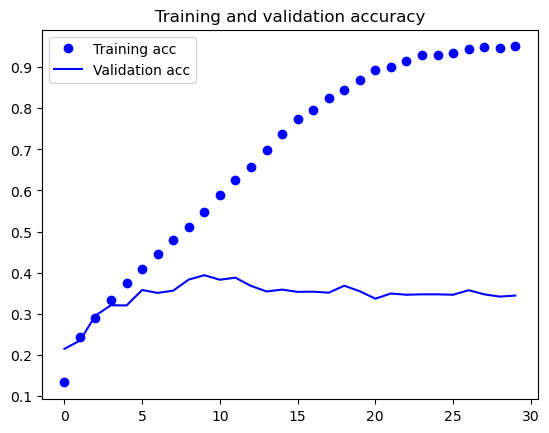
Chart, scatter chart

Description automatically generatedChart, line chart

Description automatically generated

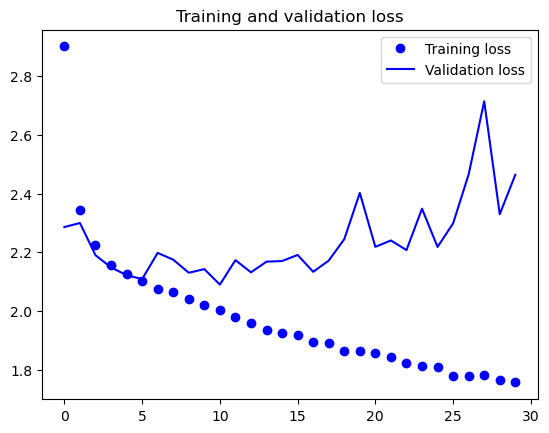
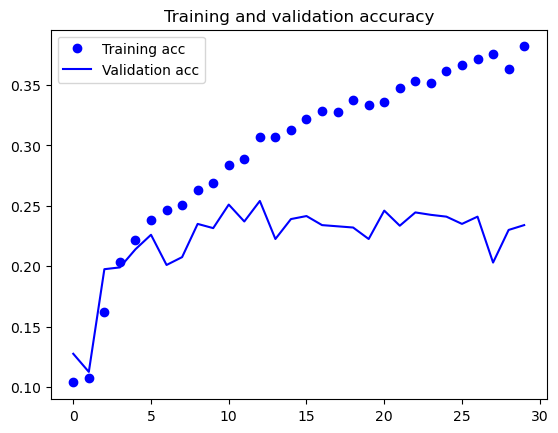
Since we have tried learning rate = 0.0001 for our baseline model, I will try to use different learning rate values to see which one is the best for our model.

**Model #1 – learning rate: 0.001**



As you can see, our model starts overfitting much earlier compared to when using a learning rate of 0.0001.

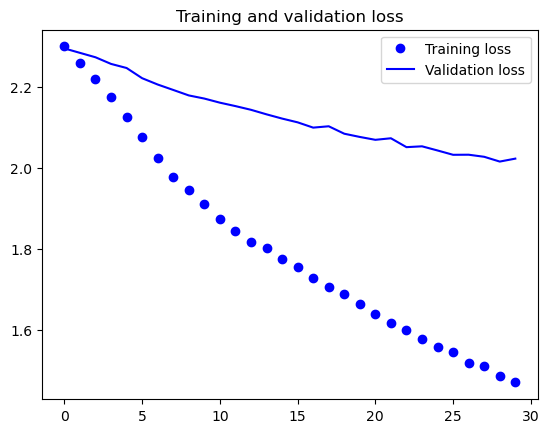
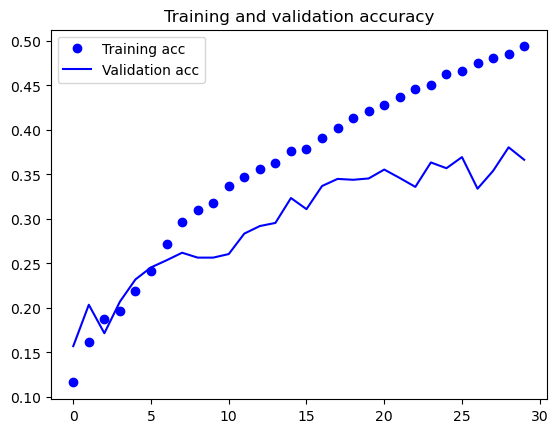
**Model #2 – learning rate: 0.01**



Validation accuracy and loss is fluctuating after increasing our learning rate

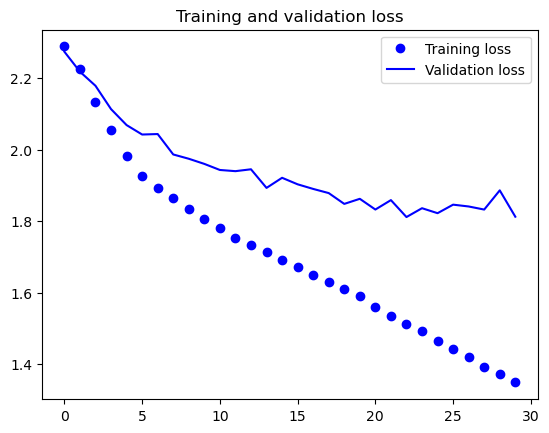
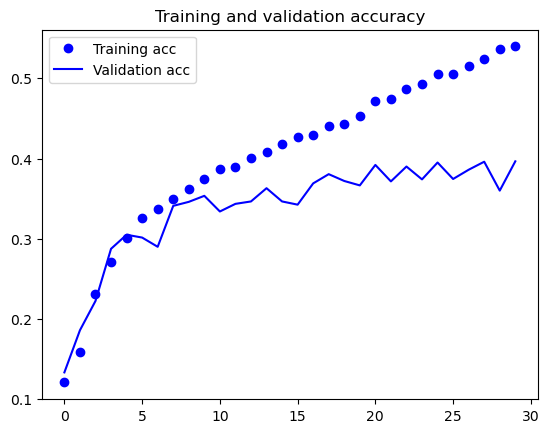
Dropout is a regularization technique that is used to reduce overfitting. I will try to add dropouts to our baseline model to test whether our accuracy improves.

**Model #3 – dropout: 0.5**



Validation loss starts to overfit immediately at epoch 0.

**Model #4 – dropout: 0.2**



Dropout 0.2 is a better option compared to 0.5, however, it still does overfit earlier compared to our baseline model.

I added weight regularization to every Conv2D layer. L2 and L1 weight regularization of different values are tested.

**Model #5 - L2 regularization: 0.0001**

Chart, line chart, scatter chart

Description automatically generatedChart, line chart

Description automatically generated

**Model #6 - L2 regularization: 0.001**

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated

**Model #7 - L2 regularization: 0.01**

Chart, line chart

Description automatically generatedChart

Description automatically generated

Generally, a lower L2 regularization value causes our model to be less overfitted compared to when using a larger L2 value. However, the accuracy and loss perform better on a higher L2 value.

**Model #8 – L1 regularization: 0.001**

Chart, scatter chart

Description automatically generatedA picture containing chart

Description automatically generated

There is almost no overfitting which is very good, however accuracy score only reaches around 34%.

**Model #9 – L1 regularization: 0.01**

Chart, scatter chart

Description automatically generatedGraphical user interface

Description automatically generated

Likewise, there is almost no overfitting, but model performance is not good.

Our baseline model uses RMSprop optimizer, I will replace this with Adam and SGD to test our model on different optimizers.

**Model #10 – Adam**

Chart, line chart, scatter chart

Description automatically generatedChart, line chart

Description automatically generated

Starts to overfit at around 15 epochs when we look at our training and validation loss. Compared to RMSprop, Adam seems to perform worse since it overfits more severely.

**Model #11 – SGD**

Chart, scatter chart

Description automatically generatedChart

Description automatically generated

Validation accuracy is extremely unstable and noisy, and accuracy reach a low score.

**Model #12 – Increasing network size to (64,64 10)**

A screenshot of a computer

Description automatically generated with medium confidence

Chart, line chart

Description automatically generated Chart, line chart

Description automatically generated

Validation accuracy stops increasing at 5 epochs. Overfitting occurs around the 20th epochs.

**Model #13 – epoch: 60**

**Chart, scatter chart

Description automatically generated** Chart

Description automatically generated

Accuracy stops increasing at around 30 epochs while validation loss stops decreasing at 1.9.

## Model utilizing pre-trained models

**Model #1 – VGG16**

This parameter will be used as a baseline whereby there are two dense layers with 256 units and 10 units respectively. In between, I have also added a dropout layer of 0.5. An Adam optimizer with a learning rate of 0.0001 is used. Data freezing is performed on all models except for fine tuning.

I will use a lower learning rate of 1e-5 for finetuning because I am using a much larger model than in the first round of training. We want to limit the magnitude of the modifications we make to the representations of the layers that we are fine-tuning. Updates that are too large may harm these representations.

Text

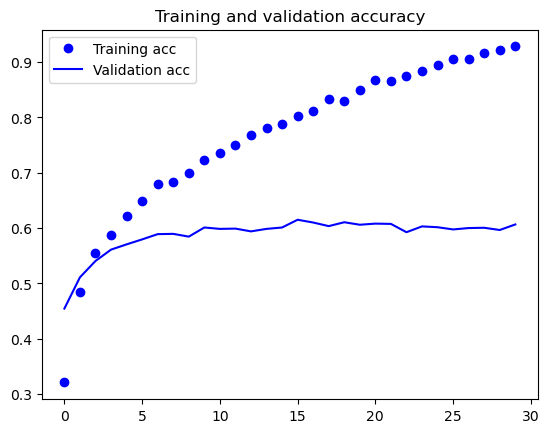
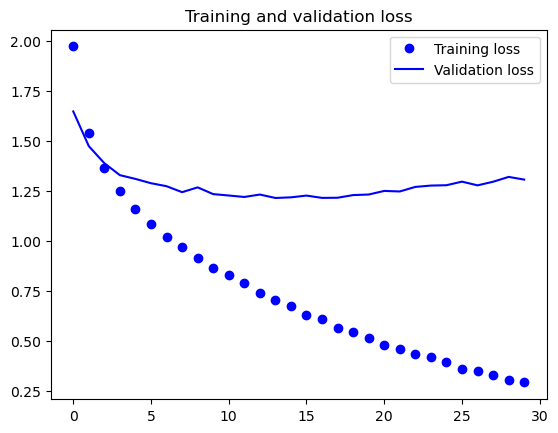
Description automatically generated

Text

Description automatically generated with low confidence

****

**No data augmentation**

** **

Manage to reach a validation accuracy of about 60%. However, our plots also indicate that we are overfitting slightly despite using dropout with a fairly large rate. This is because this technique does not leverage data augmentation, which is essential to prevent overfitting with small image datasets

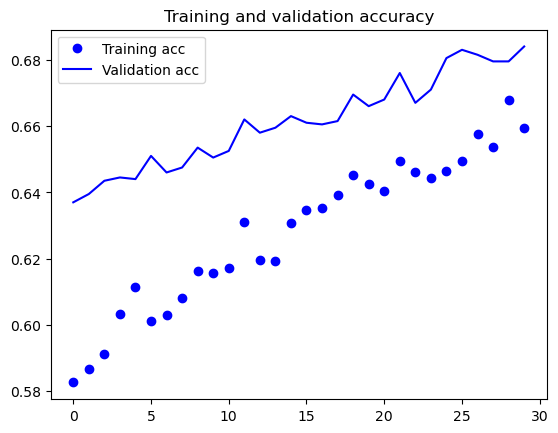
**With data augmentation**

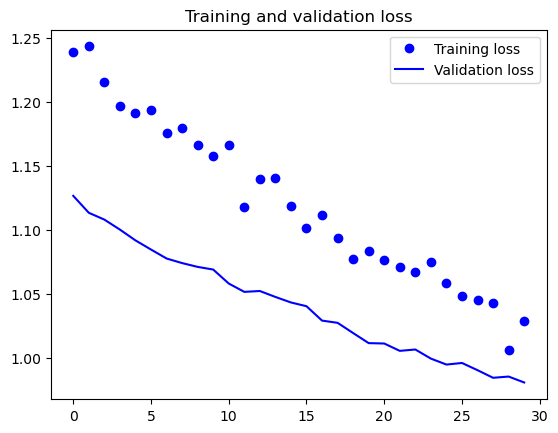
**Chart, scatter chart

Description automatically generated Chart

Description automatically generated**

Hit a slightly higher validation accuracy of around 63%. It is clear that with data augmentation, our model is more resistant to overfitting

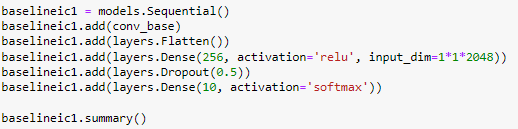
**Fine tuning**

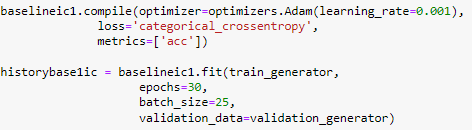
****

Validation accuracy reached 68% which is an improved value after fine tuning. It definitely does seem that validation accuracy and loss can still be improved since it is constantly increasing and decreasing respectively.

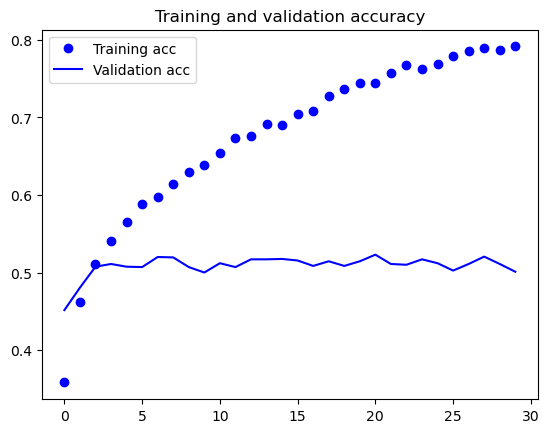
**Model #2 – InceptionV3**

Likewise, baseline parameters are similar except the dimensions are now 1x1x2048 and learning rate is 0.001.





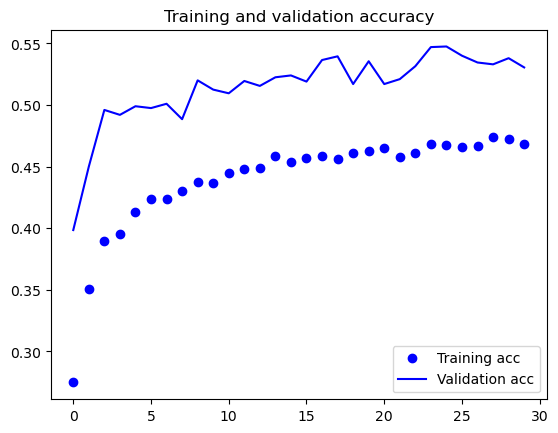
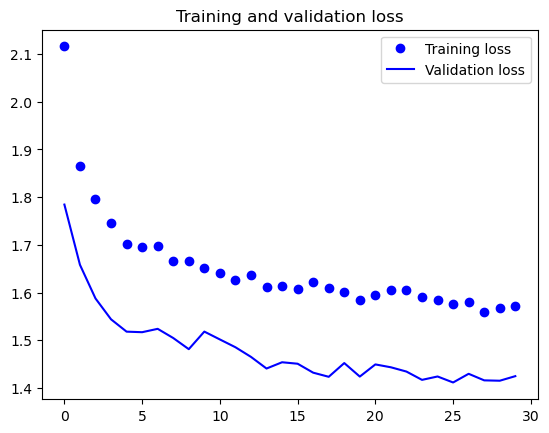
**No data augmentation**

** Chart, line chart

Description automatically generated**

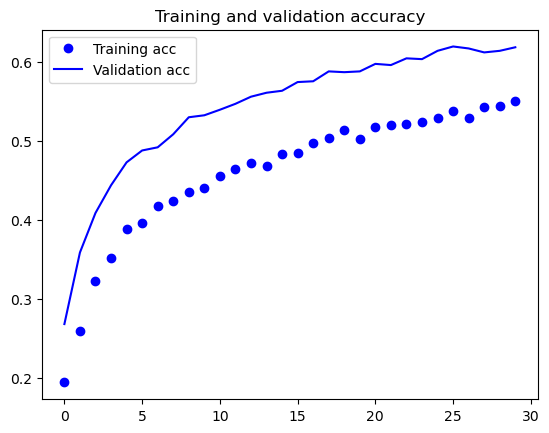
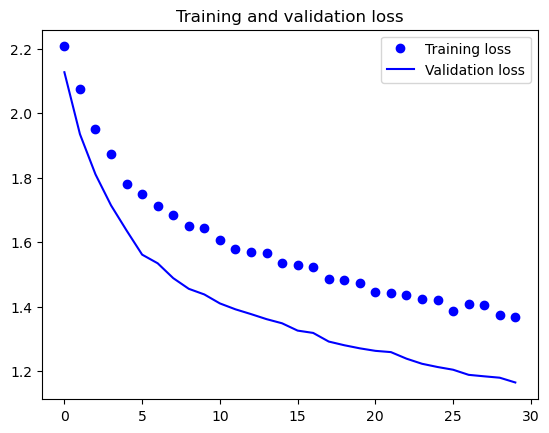
Our plots indicate that we are almost overfitting from the start despite using dropout with a fairly large rate. Validation accuracy is around 52%.

**With data augmentation**

** **

As you can see, we reach a validation accuracy of about 55%. This is much better compared to doing it without data augmentation which caused our model to overfit severely.

**Fine tuning**

** **

Validation accuracy increased to 60%. Validation accuracy and loss is also less noisy. Fine tuning has helped to improve model’s performance

# Evaluate models using Test images

## 4.1 Compare model’s performance

**Baseline Model (Conv2D)**



**Model #1 – learning rate: 0.001**



**Model #2 – learning rate: 0.01**



Learning rate of 0.0001 was used for our baseline model and it performed the best hence I proceeded to use 0.0001 as our learning rate.

**Model #3 – dropout: 0.5**



**Model #4 – dropout: 0.2**



The use of dropout did not seem to improve our model and will not be used.

**Model #5 - L2 regularization: 0.0001**



**Model #6 - L2 regularization: 0.001**



**Model #7 - L2 regularization: 0.01**



**Model #8 – L1 regularization: 0.001**



**Model #9 – L1 regularization: 0.01**



We know that L1 regularization tends to underfit when used, hence I will stick to using L2 as our weight regularization. As for its value, since 0.0001 obtained the highest accuracy amongst the others, 0.0001 is the best choice.

**Model #10 – Adam**



**Model #11 – SGD**



Amongst the different optimizers tried, RMSprop which is our baseline model performs the best. Adam obtains a close test accuracy to RMSprop, however, it still does overfit more severely compared to RMSprop. Thus, RMSprop will be used as our optimizer.

**Model #12 – Network size (64,64,10)**



Test accuracy did not improve after increasing network size. Hence, the network size would be kept as 20,20,10 following our baseline model.

**Model #13 – Epochs: 60**



Increasing the number of epochs did not improve model performance. This could be because

**Model #1 VGG**



**Model #1 INCEPTION**



Using INCEPTIONV3 seems to produce a slightly lower test accuracy compared to when using VGG16. This could mean that VGG16 just works better on the dataset we have.

## 4.2 Best Model

In general, utilizing pre-train models works much better on our data and produces a higher test accuracy compared to when building a mode from scratch. Pre-train also saves a lot of time compared to spending a serious amount of time training our model from scratch. Not only that, but our data set might also not be large enough for our model to generalize well enough. ImageNet has 1000 classes so pre-trained models have been trained to work on many different things resulting in better accuracy as well.

Since we have decided to use pre-train models, the better model is utilizing VGG16 since it performs better compared to when using other pre-train models.

# Use the Best Model to perform classification

## Explain how to apply the model on real life images

Nowadays, where online shopping is extremely popular, image recognition has also grown in eCommerce. Image recognition is used to recognize places, objects, logos and more using pixels and pattern analysis in images. This is whereby our model can come in handy to consumers looking for a particular food item they wish to purchase but are lazy to type it out, or simply do not know what the item is called. This model will automatically recognize the image uploaded by the consumer and gather the required information they need. Such models can be used in things like Panda mart or Grab mart where the platforms offer groceries to customers which can be delivered to their convenience.

Furthermore, these can also be implemented in food delivery companies like Deliveroo. Consumers can upload images of food they are interested in, while our model identify this food images, Deliveroo can recommend the restaurants which serve this particular type of food.

## 5.2 Explain and analyze the model prediction

A picture containing text, food

Description automatically generatedA picture containing text, pizza, dish, food

Description automatically generated

A picture containing text, food

Description automatically generated

A plate of food

Description automatically generated with medium confidenceA picture containing text, food, dish

Description automatically generated

A picture containing text, food

Description automatically generatedA picture containing text, food, dish

Description automatically generated

From our predictions, we can see that all food images were predicted accurately and its probability for each is above 0.9. This means that the model we have used is at the very least capable of depicting our food images correctly.

# Summary

There is always room for improvement of the training datasets used for our model. Generally, increasing the size of a training data set improves accuracy, but that is not always feasible for real-time datasets. A different approach is to artificially expand the dataset which we have done through applying a bunch of transformations to the original training data such as mirroring, rotating, zooming, and cropping.

Surely, the network design can be improved as well. We only used a few convolution layers and/or a few fully connected layers. This could be too shallow to learn to differentiate the different classes. Hence, adding more convolution layers may help the network identify meaningful patterns.

In conclusion, we focused on tuning our model to find the best model to accurately predict our food images correctly. Our findings from the model build from scratch using convolutional and dense layers has shown us things like RMSprop being a better optimizer for our data, a lower L2 learning rate works well and dropout does not improve our model performance as well as many more.

While building a model from scratch may seem like a good idea since you get a deeper understanding of the algorithm, pre-train models have proved to be far more efficient. This is because pre-train models reduce the cost and effort required for deep learning because not much time is spent to gather and clean the data. Not to mention, the infrastructure and knowledge needed to train the models correctly makes pre-train models stand out from building our model from scratch. We have also shown that VGG16 seemed to work much better compared to Inception when it came to this particular dataset.