

**SA4110: Machine Learning Application Development**

Assignment: Create an Image Classifier

Team 2:

Gao Zhi De

[Htet Aung Ky](https://canvas.nus.edu.sg/courses/39919/users/105016)aw

Lan Qian

Nguyen Tran

Tey Kee Yeow

Xu Zi Yi

Yao Bin

27 May 2023

Table of Contents

[1. Report Overview 4](#_Toc136089170)

[2. Datasets Preparation 5](#_Toc136089171)

[Datasets 5](#_Toc136089172)

[Pre-processing of Data 5](#_Toc136089173)

[Data augmentation 5](#_Toc136089174)

[One-hot Encoding 6](#_Toc136089175)

[Samples of inputs 7](#_Toc136089176)

[3. Model Building 8](#_Toc136089177)

[Model Layers 8](#_Toc136089178)

[Additional Components 9](#_Toc136089179)

[Model Architecture 9](#_Toc136089180)

[Model Structure 1 10](#_Toc136089181)

[Model Structure 2 10](#_Toc136089182)

[4. Model Structure 1 Performance (64 x 64) 11](#_Toc136089183)

[Evaluation Round 1 11](#_Toc136089184)

[Evaluation Round 2 11](#_Toc136089185)

[Evaluation Round 3 12](#_Toc136089186)

[Evaluation Round 4 12](#_Toc136089187)

[Evaluation Round 5 13](#_Toc136089188)

[Evaluation Round 6 14](#_Toc136089189)

[Evaluation Round 7 14](#_Toc136089190)

[Evaluation Round 8 15](#_Toc136089191)

[Evaluation Round 9 16](#_Toc136089192)

[Evaluation Round 10 17](#_Toc136089193)

[Evaluation Round 11 17](#_Toc136089194)

[Evaluation Round 12 18](#_Toc136089195)

[Model Structure 1 Conclusion 19](#_Toc136089196)

[5. Model Structure 2 Performance (256 x 256) 20](#_Toc136089197)

[Observation 20](#_Toc136089198)

[6. Conclusion 22](#_Toc136089199)

[References 23](#_Toc136089200)

# Report Overview

This report presents the outcomes of an image classification task focused on fruits, utilizing a convolutional neural network (CNN) model. The primary objective was to achieve accurate classification of fruit images into four distinct classes: Apple only, Orange only, Banana only, and a mix of the three fruits. The project aimed to overcome challenges such as imbalanced classes and limited labelled data availability, striving for a high level of accuracy.

The report encompasses a general overview of the datasets used, a simple explanation of the model architecture layers, detailed information about the training model built, and evaluations of the model's performance on the test datasets. Additionally, visual aids such as accuracy and loss curves have been included to effectively illustrate the model's learning progress throughout the training phase.

To begin with, the report provides an overview of the datasets employed for training and testing the CNN model. These datasets were meticulously pre-processed, incorporating techniques such as image augmentation, one-hot encoding, and other normalization strategies, in order to ensure a standardized dataset with reduced computational complexity.

The CNN model architecture employed for the image classification task is thoroughly described, highlighting the specific layers, filters, and functions utilized to extract relevant features from the input images. To address the challenge of overfitting, various techniques including the implementation of the Model Checkpoint call-back and early stopping strategies were incorporated to effectively mitigate overfitting concerns.

Furthermore, the report provides a series of evaluations conducted to enhance the model's performance. Several evaluation metrics were employed to assess the model's effectiveness, such as the accuracy, loss and the learning curves. These evaluations and analysis provided valuable insights into the strengths and weaknesses of the CNN model.

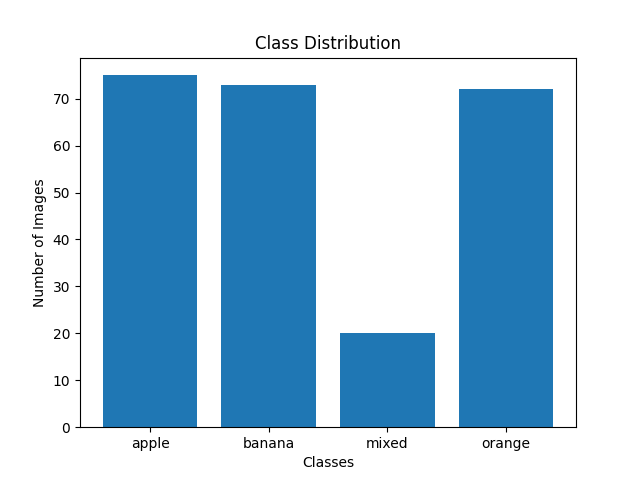
Lastly, report reviewed the entire process and concluded a few points that affect the model’s predicting accuracy.

# Datasets Preparation

## Datasets

Manual check was performed on the dataset to ensure all labelling of the test and train samples are correct to ensure the accuracy of our model.

A class distribution bar chart is plotted to show the classes of the datasets and count the samples in each class for training and testing purpose in the model.

Below figure was generated.

|  |  |
| --- | --- |
| Class | Count |
| Apple | 75 |
| Banana | 73 |
| Orange | 72 |
| Mixed | 20 |

Figure ‑. Class Distribution

The number of samples in ‘Mixed’ Class is significantly less, we will apply data augmentation techniques to improve the sample size and balance the training data.

## Pre-processing of Data

All inputs are given in difference size and different resolution. Thus, we would like all inputs to be sized to the same size for our training purpose.

### Data augmentation

An example of data augmentation before in pre-processing:

train\_datagen = ImageDataGenerator(featurewise\_center=True,

                                   rotation\_range=0.4,

                                   width\_shift\_range=0.3,

                                   horizontal\_flip=True,

                                   preprocessing\_function=preprocess\_input,

                                   zoom\_range=0.4,

                                   shear\_range=0.4)

train\_data = train\_datagen.flow\_from\_directory(directory="./train",target\_size=(256,256),batch\_size=16)

In this example we are using the ImageDataGenerator class from Keras to perform image data augmentation.

### One-hot Encoding

As label (class names) are strings, we adopted the one-hot Encoding method. By using one-hot encoding, our categorical labels are properly represented in the form of binary vectors for machine learning models, enabling them to learn and make predictions accurately.

An example of one hot encoding:

def encode\_onehot (pos, n\_rows):

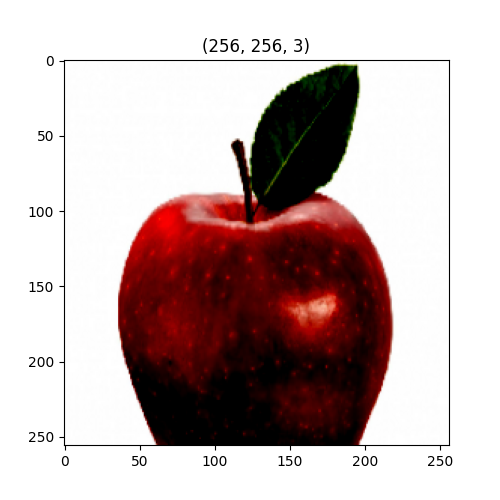
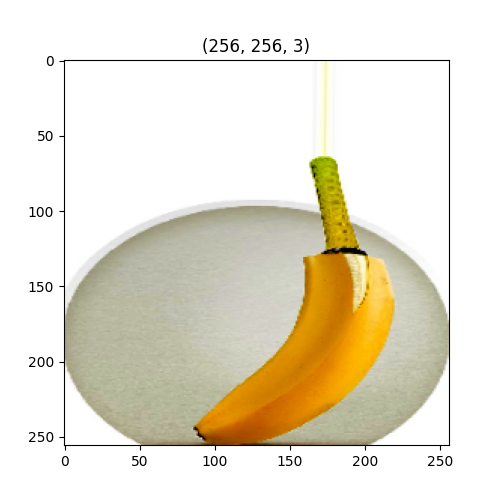
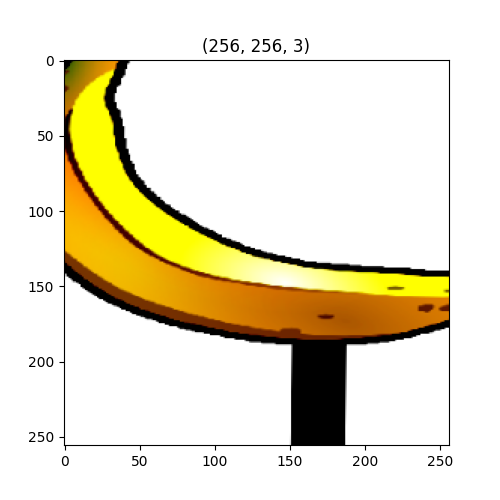
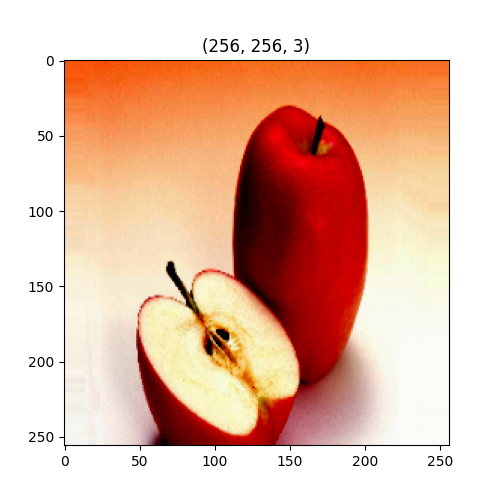
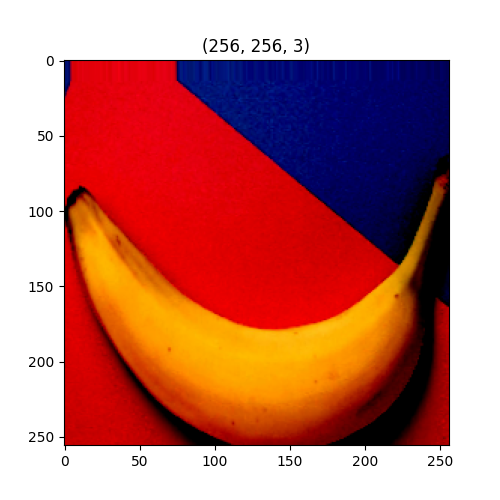
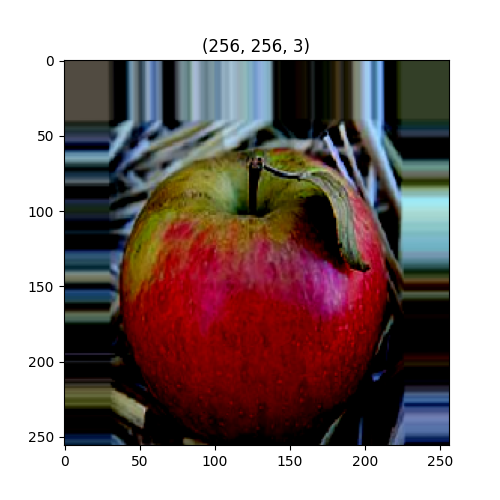
    y\_onehot = [0] \* 4

    y\_onehot[pos] = 1

    y\_onehots = [y\_onehot] \* n\_rows

    return np.array(y\_onehots)

## Samples of inputs



**64 x 64 sample images**

**A picture containing fruit, text, natural foods, diet food

Description automatically generated**

# Model Building

For this CA project, multiple models were built by our teammates to explore the possible structures of an image classifier. In this section, we will discuss some common layers we adopted followed by two sample model codes.

## Model Layers

Here are some of the layering techniques we adopted for this project:

**RandomFlip layer:** Image augmentation layer, introduces random horizontal and vertical flips to the input images during training, creating new variations of the data and improving the model's ability to generalize to different orientations.

**RandomRotation layer:** Image augmentation layer, applies random rotations to the input images within a specified range. It helps to artificially increase the size of a training dataset thus reducing overfitting and improving the generalization capability of the model.

**Conv2D layer:** This layer performs convolutional operations on the input image. This layer extract features from input data, enabling the model to learn spatial patterns and capture local relationships within the data.

**Dropout layer:** Dropout is a regularization technique that randomly sets a fraction of input units to 0 at each update during training. By "dropping out" some units, it reduces overfitting by preventing the model from relying too much on any subset of neurons.

**MaxPool2D layer:** Performs max pooling, which down samples the feature maps by taking the maximum value. Max pooling helps in reducing the spatial dimensions of the feature maps while retaining important features.

**Flatten layer:** This layer flattens the 2D feature maps into a 1D vector, which can be fed into a fully connected layer. It reshapes the output of the previous layer into a single long vector, preserving the learned spatial information.

**Batch Normalization layer**: It normalizes the activations of the previous layer. By normalizing the activations, it helps in stabilizing the training process, speeding up convergence, and reducing overfitting.

**Activation layer:** Applies an activation function. Activation layers in CNN introduce non-linearity, enabling the network to learn complex patterns and make nonlinear transformations to the input data, enhancing the model's ability to capture intricate features and improve performance.

**Dense layer:** This layer consists of X neurons. It learns non-linear relationships between the features extracted by the previous layers, enabling more complex modelling.

## Additional Components

In this CA project we also adopted two additional components that can help improve the model accuracy:

**Model Checkpoint:** The Model Checkpoint callback allows us to save the best-performing model during training. By saving the best model, we can avoid overfitting and select the model with the highest accuracy for future use.

**Early Stopping:** The EarlyStopping callback allows you to stop training early if the monitored metric stops improving. Early stopping helps prevent overfitting and saves computational resources by terminating training when the model's performance plateaus.

## Model Architecture

In this section, we present the summary of the model architecture using the **model.summary()** function, which provides detailed information about each layer and the total number of parameters in the model.

By displaying the model summary, we gain insights into the different layers present in the model, such as convolutional layers, flatten layers, and dense layers. This helps the reader understand the basic structure of the model and its purpose in extracting relevant features from the input images and making accurate predictions.

The summary also includes information about the shape of the output at each layer and the total number of trainable parameters in the model, helping the reader understand the structure and design aspects in the model.

## Model Structure 1

Training and test images are split into 4 folders (Folder 1 to Folder 4) separating fruit categories respectively.

def create\_model():

    model = tf.keras.Sequential()

    model.add(tf.keras.layers.RandomFlip(mode = "horizontal\_and\_vertical", seed = 5))

    model.add(tf.keras.layers.RandomRotation(0.2, fill\_mode= "reflect", interpolation="bilinear", seed= 5, fill\_value=0.0))

    model.add(tf.keras.layers.Conv2D(filters =32, kernel\_size=(7,7), padding="same", activation="relu", input\_shape= (64,64,3)))

    model.add(tf.keras.layers.BatchNormalization(axis =1))

    model.add(tf.keras.layers.MaxPooling2D(pool\_size=(2,2)))

    model.add(tf.keras.layers.Dropout(0.8)) #Dropout layers prevent neurons from relying on one input because it might be dropped out at random

    model.add(tf.keras.layers.Conv2D(filters = 32, kernel\_size=(3,3), padding="same", activation = "relu"))

    model.add(tf.keras.layers.MaxPooling2D(pool\_size=(2,2)))

    #model.add(tf.keras.layers.Dropout(0.5)) #Adding a third Dropout Layers dramatically decreases test accuracy

    #model.add(Flatten())

    model.add(tf.keras.layers.Conv2D(filters = 32, kernel\_size = (3,3), padding="same", activation = "relu"))

    model.add(tf.keras.layers.MaxPooling2D(pool\_size=(2,2)))

    model.add(tf.keras.layers.Flatten())

    model.add(tf.keras.layers.Dropout(0.8))

    model.add(tf.keras.layers.Dense(28,activation = "relu"))

    model.add(tf.keras.layers.Dense(4, activation="softmax"))

    model.compile(optimizer = 'adam', loss = 'categorical\_crossentropy', metrics = ['accuracy'])

    #model.summary()

    return model

def train\_model(model, x\_train, y\_train):

    earlyStop = EarlyStopping(monitor = 'loss', patience = 17) #adjust based on numbers of Epoch (lower Epoch requires less patience, and vice versa)

    #batch\_size set to 10 to increase training time leading to better accuracy

    return model.fit(x=x\_train, y=y\_train, epochs = 201, validation\_split = 0.20, batch\_size = 10, callbacks = [earlyStop])

## Model Structure 2

Model 2 was built on top of a InceptionV3 (keras.io, n.d.) Model Architecture.

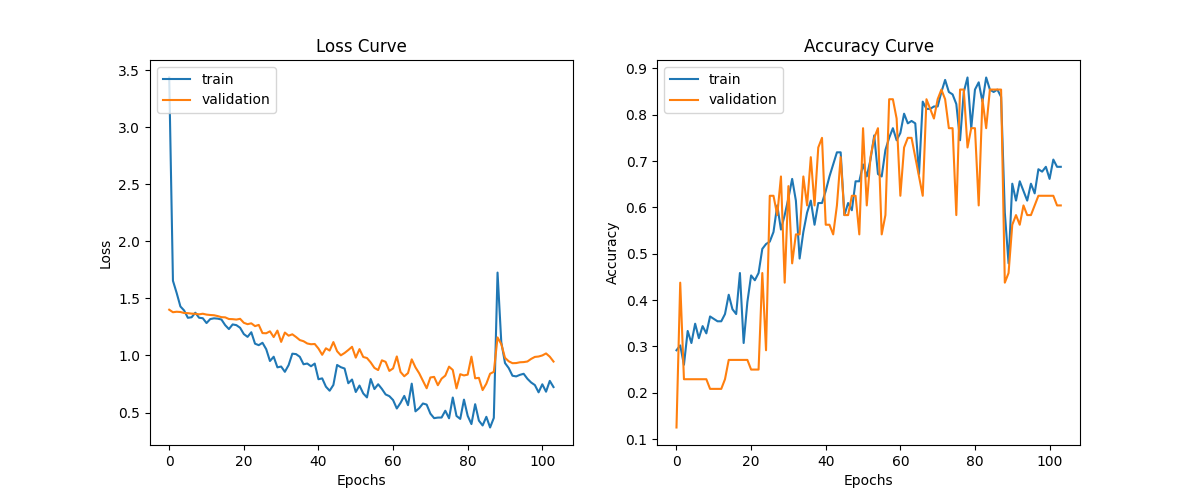
Please refer to the “fruit\_classifier\_Model Structure 2.txt" for the full model structure.

# Model Structure 1 Performance (64 x 64)

## Evaluation Round 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Evaluation Round 1 Metrics Summary** | | | | |
| Correct: 39 | Wrong: 21 | Accuracy: 0.64999 | Loss: 0.8712 | Epoch: 201 |

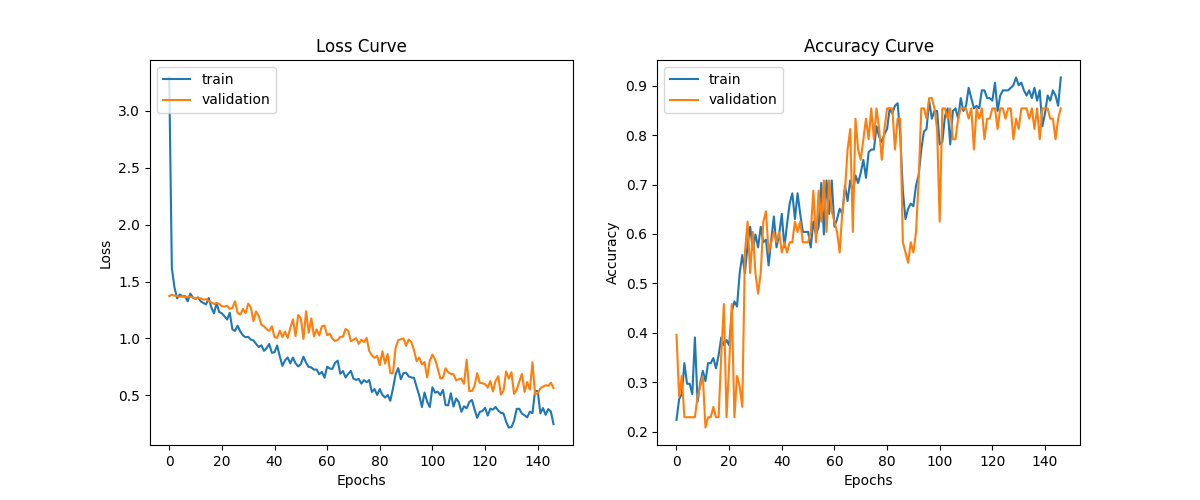
**Observations and Comments**: Accuracy at first attempts sits at 0.65, yet the graph does display both the accuracy and the validation line close together indicating minimal to no overfitting of graphs. Since Epoch starts off at 201, the next attempt will have a reduced Epoch quantity to 150. The goal of the next attempt is to determine the optimum epoch to utilize and obtain the best accuracy.

Evaluation Round 1 Learning Curve

## Evaluation Round 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Evaluation 2 Metrics Summary** | | | | |
| Correct: 50 | Wrong: 10 | Accuracy: 0.83333 | Loss: 0.6082 | Epoch:  150 |

**Observations and Comments**: Just by reducing Epoch to 150, the training model was able to obtain an accuracy of 0.833. This indication shows that the lower number of Epoch could provide a better accuracy. The goal of this training model is to be able to achieve above 90%. Therefore, the next attempt will have a reduced Epoch to 140, this is so that we can find whether the direction in which the model can achieve 90% is close to 150 Epoch.

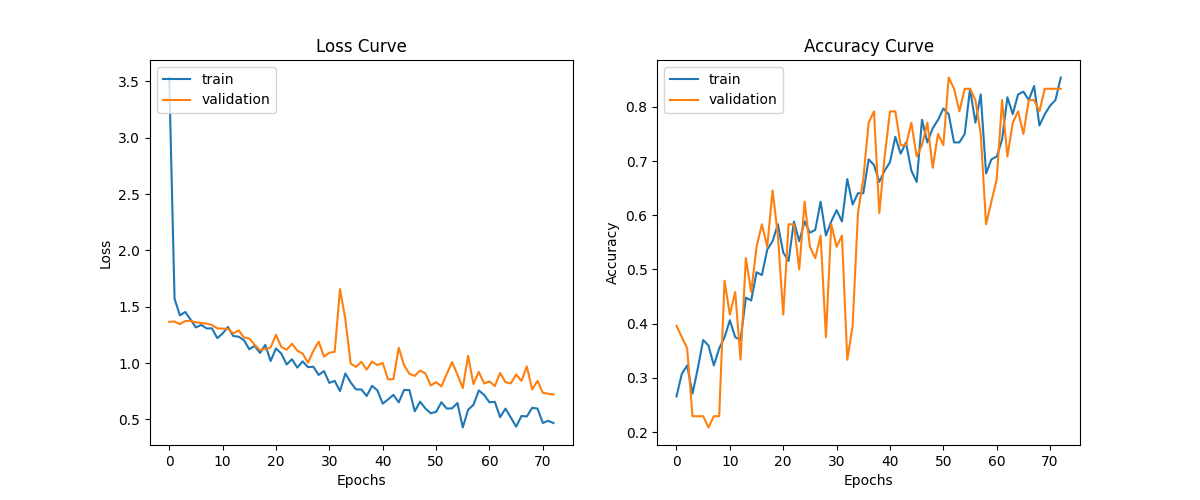
**Adjustment from previous result**: (Epoch: 201 -> 150) Evaluation Round 2 Learning Curve

## Evaluation Round 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Evaluation Round 3 Metrics Summary** | | | | |
| Correct: 48 | Wrong: 12 | Accuracy: 0.8000 | Loss: 0.7507 | Epoch:  140 |

**Observations and Comments**: Reducing Epoch to 140 yielded a lower accuracy than previous result. This indicates that to obtain an accuracy over 90%, having a smaller Epoch than 150 may not be the best direction for this model. Therefore, Epoch will now increase to 160 to see if it yields a better result than Figure\_2.

**Adjustment from previous result**: (Epoch: 150 -> 140)



Evaluation Round 3 Learning Curve

## Evaluation Round 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Evaluation Round 4 Metrics Summary** | | | | |
| Correct: 52 | Wrong: 8 | Accuracy: 0.86667 | Loss: 0.79705 | Epoch:  160 |

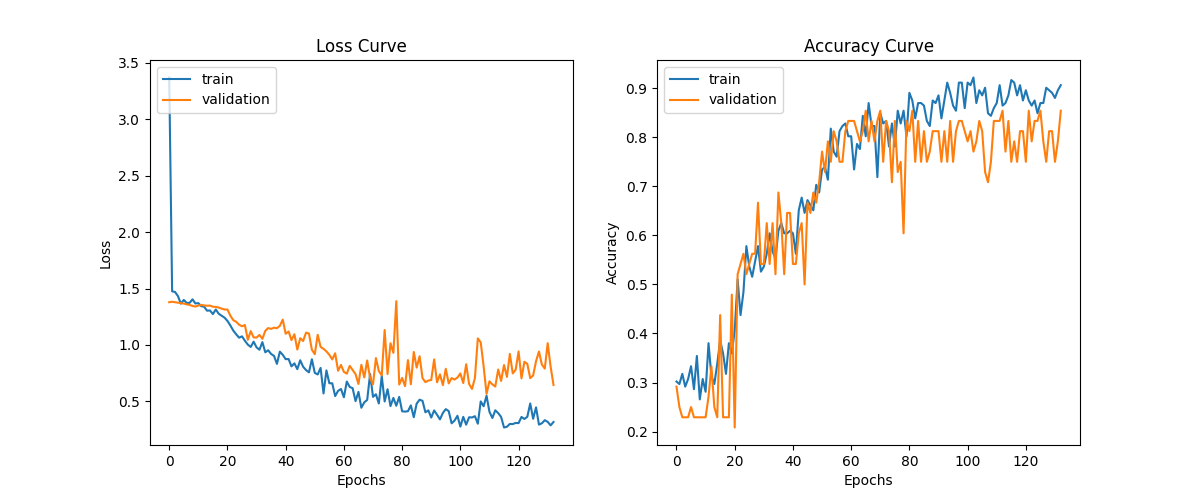
**Observations and Comments**: Epoch 160 yields a better result than Figure\_2 affirms the theory mentioned in the result of Figure\_3. For the next result, Epoch will increase to 170. If the result continues to increase, then the desired epoch is likely between 200 and 170.

Epoch 129/160

2/2 [==============================] - 0s 22ms/step - loss: 0.7970 - accuracy: 0.8667

loss = 0.7970461249351501 accuracy = 0.8666666746139526

**Adjustment from previous result**: (Epoch: 140 -> 160)



Evaluation Round 4 Learning Curve

## Evaluation Round 5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Evaluation Round 5 Metrics Summary** | | | | |
| Correct: 49 | Wrong:  11 | Accuracy: 0.81667 | Loss: 0.7558 | Epoch:  170 |

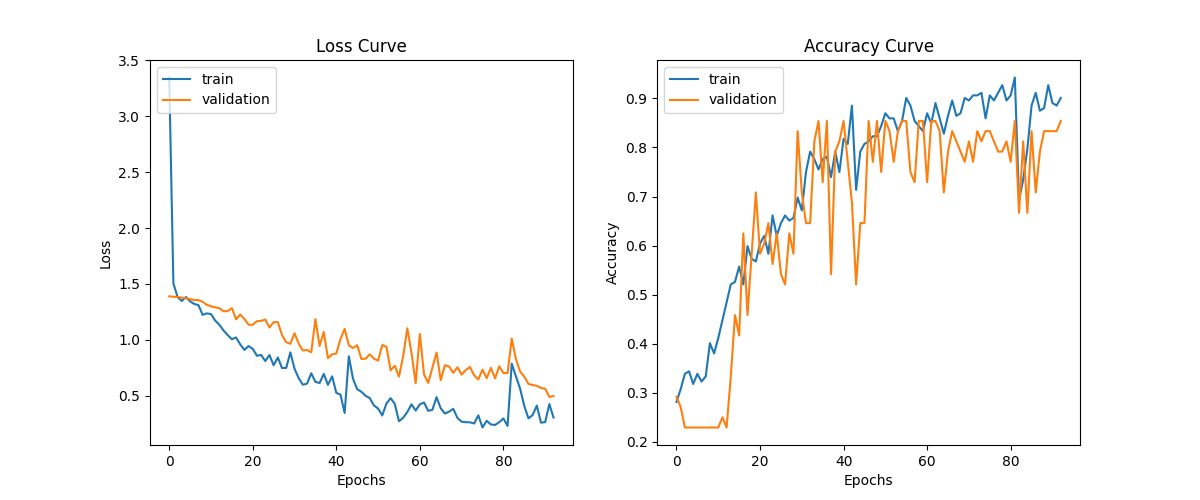
**Observations and Comments**: Epoch 160 yields a better result than Figure\_2 affirms the theory mentioned in the result of Figure\_3. For the next result, Epoch will increase to 170. If the result continues to increase, then the desired epoch is likely between 200 and 170.

Epoch 89/170

2/2 [==============================] - 0s 22ms/step - loss: 0.7558 - accuracy: 0.8167

loss = 0.7557841539382935 accuracy = 0.8166666626930237

**Adjustment from previous result**: (Epoch: 160 -> 170)

Evaluation Round 5 Learning Curve

## Evaluation Round 6

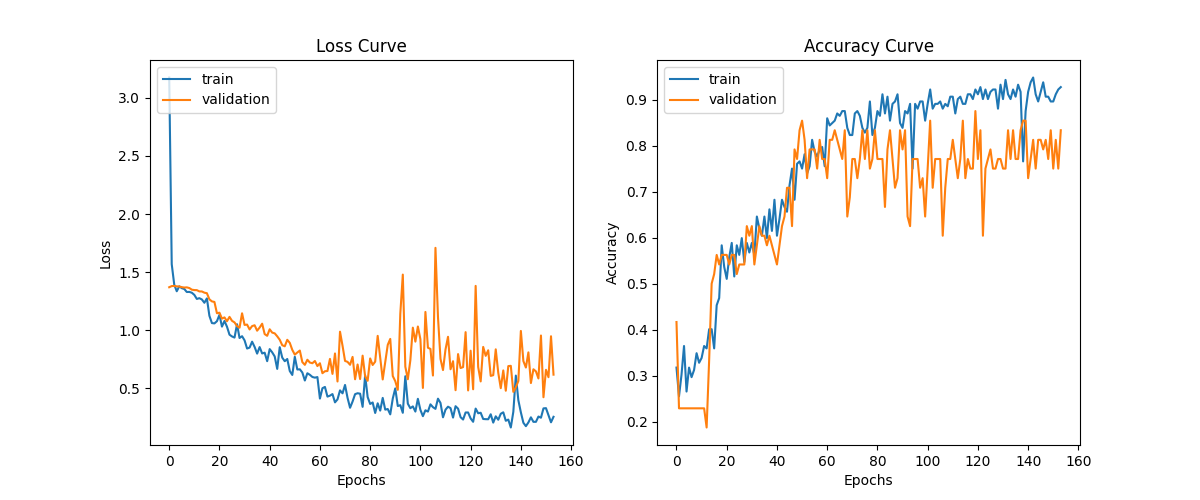
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Evaluation Round 6 Metrics Summary** | | | | |
| Correct: 50 | Wrong: 10 | Accuracy: 0.83333 | Loss: 0.79845 | Epoch:  155 |

**Observations and Comments**: Since going above Epoch 160 has yielded a lower result, the result of this training shows that the higher accuracy Epoch range is likely between 155 and 160. Therefore, multiple runs will be made with each one’s Epoch ranging from 156-159 since the result of 155 and 160 is already present.

Epoch 155/155

2/2 [==============================] - 0s 27ms/step - loss: 0.7985 - accuracy: 0.8333loss = 0.7984545826911926 accuracy = 0.8333333134651184

**Adjustment from previous result**: (Epoch: 170 -> 155)



Evaluation Round 6 Learning Curve

## Evaluation Round 7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Evaluation Round 7 Metrics Summary** | | | | |
| Correct: 52 | Wrong: 8 | Accuracy: 0.86667 | Loss: 0.48209 | Epoch: 158 |

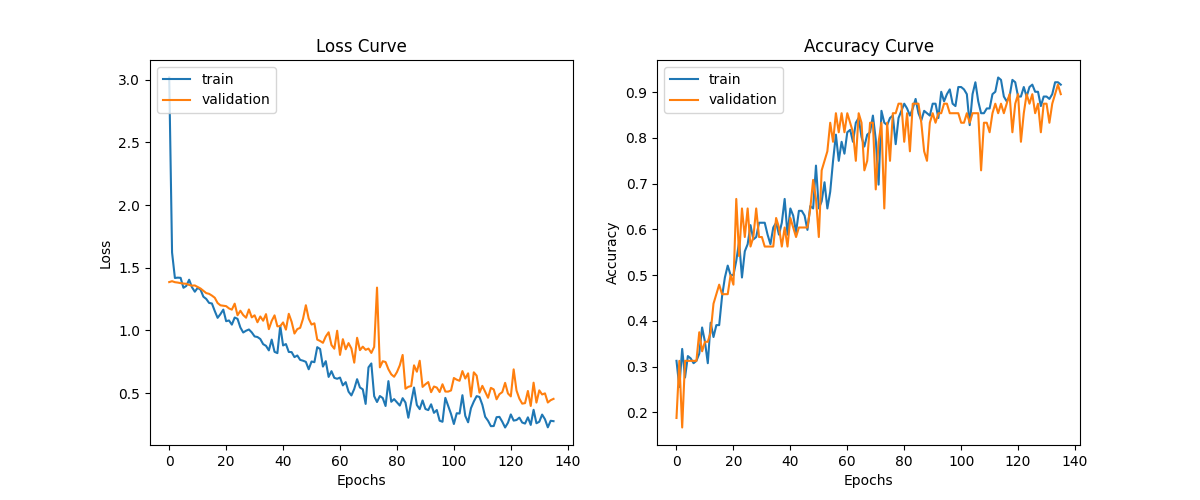
**Observations and Comments**: After testing Epoch range between 156 to 159, the Epoch that yielded the best result is at 158. To push the accuracy further, other values such as Dropout can be adjusted to meet the 90% goal. By default, Dropout value sits at 0.8 for both layers. Therefore, both the Dropout value will reduce to 0.7 to determine the type of changes in the accuracy.

Epoch 139/158

2/2 [==============================] - 0s 28ms/step - loss: 0.4821 - accuracy: 0.8667

loss = 0.48209577798843384 accuracy = 0.8666666746139526

**Adjustment from previous result**: (Epoch: 155 -> 158)

Evaluation Round 7 Learning Curve

## Evaluation Round 8

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Evaluation Round 8 Metrics Summary** | | | | |
| Correct: 53 | Wrong: 7 | Accuracy: 0.88333 | Loss: 0.88886 | Epoch:  158 |

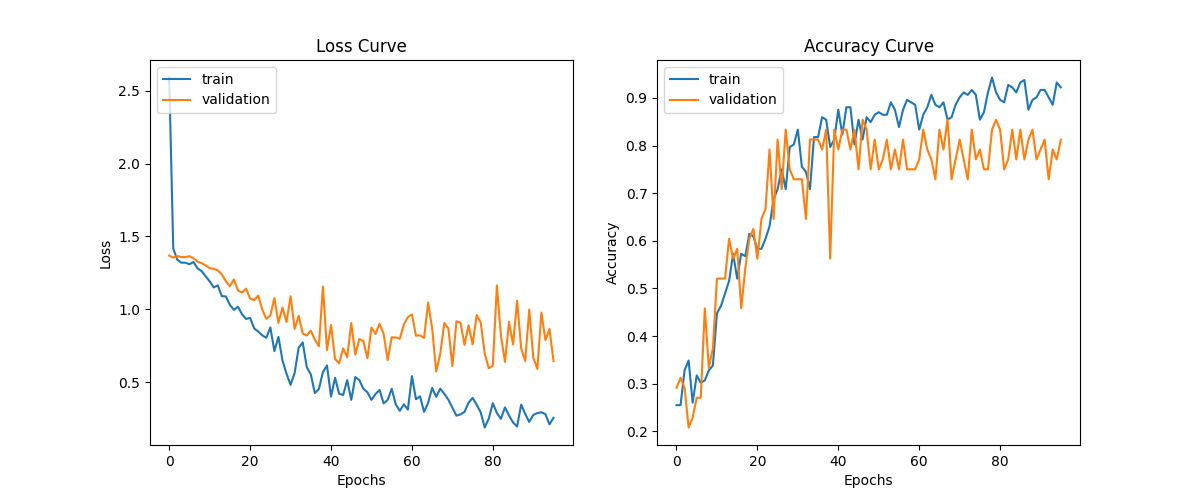
**Observations and Comments**: Reducing the Dropout layers to 0.7 increased the accuracy by roughly 2 percent. However, the graph displays overfitting as the validation and train line deviates from each other. In order to see if the model can achieve accuracy at least 90%, Dropout layers will be reduced to 0.6 despite becoming overfit.

Epoch 96/158

2/2 [==============================] - 0s 25ms/step - loss: 0.8889 - accuracy: 0.8833

loss = 0.8888686895370483 accuracy = 0.8833333253860474

**Adjustment from previous result**: (Dropout 1st: 0.8 -> 0.7), (Dropout 2nd: 0.8 -> 0.7)

Evaluation Round 8 Learning Curve

## Evaluation Round 9

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Evaluation Round 9 Metrics Summary** | | | | |
| Correct: 55 | Wrong: 5 | Accuracy: 0.91667 | Loss: 1.0054 | Epoch:  158 |

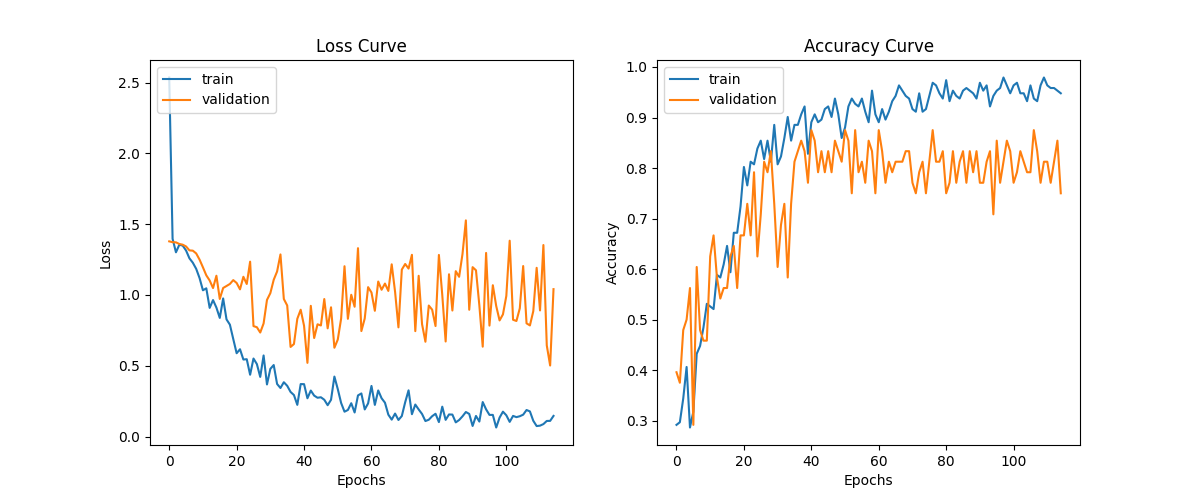
**Observations and Comments**: Having achieved above 91.6% comes with a drawback of overfitting the graph. The challenge now is to maintain accuracy above 90% while reducing overfitting on the graph. In this case, batch\_size will be increased since it is one of the ways to reduce overfitting other than changing the Dropout layer again. Early Stopping patience will be set lower to avoid over fitting the data when accuracy runs into diminishing return.

Epoch 115/158

2/2 [==============================] - 0s 25ms/step - loss: 1.0054 - accuracy: 0.9167

loss = 1.0053658485412598 accuracy = 0.9166666865348816

**Adjustment from previous result**: (Dropout 2nd: 0.7 -> 0.6)



Evaluation Round 9 Learning Curve

## Evaluation Round 10

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Evaluation Round 10 Metrics Summary** | | | | |
| Correct: 54 | Wrong: 6 | Accuracy: 0.89999 | Loss: 0.5254 | Epoch:  158 |

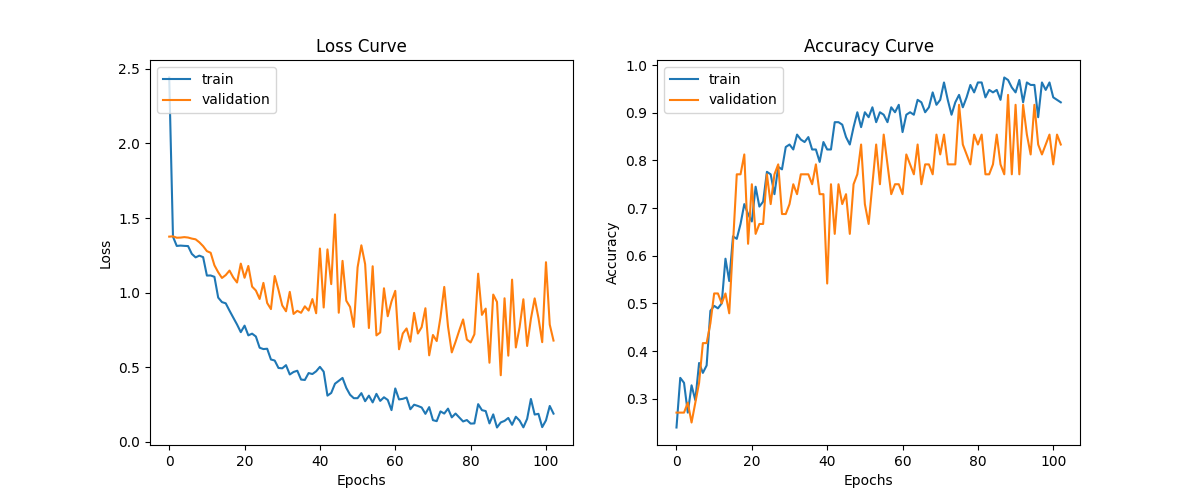
**Observations and Comments**: The increase in batch\_size as well as the change in Early Stopping have reduced the accuracy to exactly 90%. However, the issues of overfitting have become slightly less as a result.

Epoch 103/158

2/2 [==============================] - 0s 30ms/step - loss: 0.5254 - accuracy: 0.9000

loss = 0.5253992676734924 accuracy = 0.8999999761581421

**Adjustment from previous result**: (Early Stopping patience: 17 -> 15), (batch\_size: 10 -> 16)



Evaluation Round 10 Learning Curve

## Model Structure 1 Conclusion

Throughout these sets of evaluation, it has been understood that there are many main factors in which the accuracy rating depends on. The following in which includes the number of training Epoch being used, the patience setting utilized by Early Stopping, the batch size in which controls the number of samples processed before model is updated, and the Dropout layer which nullifies the contribution of some neurons in the network before it goes towards the next layer. At the beginning, this structure starts out with an Epoch of 201 which immediately proven that it is too far from the optimum training quantity. This was also the reason that it constitutes to a low accuracy score since the program has become too good at predicting its training data that it becomes inaccurate when faced with new outside data. This was later remedied when the optimal Epoch was found through a series of trial and error. The graph was also kept from overfitting too much thanks to the utilization of 2 layers of Dropout in which initially both has an input value of 0.8. However, because of the Dropout layer, the accuracy could not achieve higher than 88.33% and are both decrease to 0.7 respectively to see if it was able to achieve over 90%. It is not until the second Dropout layer dropping to 0.6 percent that the accuracy has increased to 91.6%. This comes with a drawback because of the lowering the Dropout value, the graph becomes increasingly overfitted and would require the changing of other values to put the validation and the accuracy/loss closer together. The changing of the batch size from 10 to 16 and putting Early Stopping patience’s value to from 17 to 15 help with this effect even if minorly. Overall, it is a matter of balancing the values and parameters as well as some altering/reorganizing of the train/test data set that contributes to the highest and most optimal results.

# Model Structure 2 Performance (256 x 256)

**Model 2**

The pre-processing of this model structure 2 is different compare to model one, as the minimal input size for the inception model is (75 x 75).

mc = ModelCheckpoint(filepath="./best\_model.h5", monitor="accuracy", verbose=1, save\_best\_only=True)

es = EarlyStopping(monitor="accuracy", min\_delta=0.01, patience=7, verbose=1)

callbacks = [mc, es]

n the second model, we utilized ModelCheckpoint to save the best-performing model, ensuring its preservation for further analysis and utilization.

## Observation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Evaluation Round 12 Metrics Summary** | | | | |
| Correct: 58 | Wrong 6 | Accuracy: 0.91667 | Loss: 1.0991 | Epoch:  50 |

During the training process, the model's accuracy steadily improved over the epochs. The highest achieved accuracy was 96.87% during epoch 6. This demonstrates that the model learned to recognize fruit images effectively.

10/10 [==============================] - 11s 1s/step - loss: 0.4710 - accuracy: 0.9438 - val\_loss: 1.0991 - val\_accuracy: 0.9167

Epoch 12: early stopping

The training was stopped early at epoch 11 as the model's accuracy did not improve further. This shows that the model has reached its optimal performance and further training would not gain significant improvements.

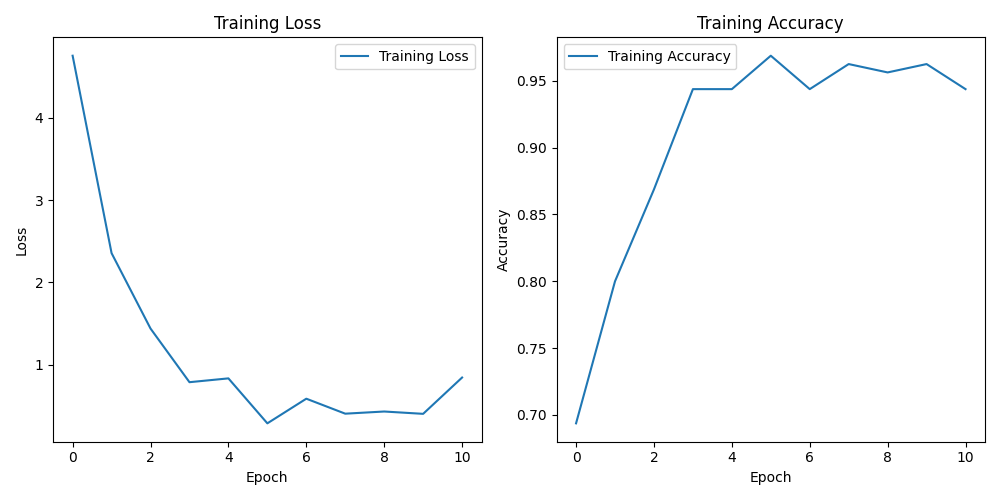


Fig 2. Training Loss VS. Training Accuracy

Base on the graph above we can significantly identified the trend of the loss and accuracy were dramatically improving over each epoch and it stays there steady after the 5th round, reaches the highest at 6th epoch. On the other hand, the lose graph was improved vastly at the 2nd epoch.

A graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of

Description automatically generated with low confidenceModel 2 Learning Curve

# Conclusion

Multiple variables can exert an influence on the outcome of our test, encompassing a range of contributing factors:

**Quality and quantity of datasets**

For this assignment, data augmentation techniques was employed to address class imbalances and augment the volume of data without altering the original dataset. By leveraging these techniques, we aim to ensure a balanced representation of each data class and enrich the dataset, thereby providing the model with a better learning experience.

**Complexity of model**

A model that is excessively simplistic may encounter difficulties in capturing intricate patterns within the data, resulting in underfitting. Conversely, an excessively complex model has the potential to overfit the training data, impeding its ability to generalize effectively to new, unseen data.

In this assignment, we constructed a second model featuring a highly intricate structure. However, despite its complexity, this model did not yield a substantial improvement in accuracy.

**Hyperparameters**

The performance of the model can be greatly influenced by key hyperparameters such as the learning rate, batch size, number of layers, and number of neurons in each layer. In this assignment, an exploration of these hyperparameters was conducted through manual testing, allowing us to select the optimal values that yield a more accurate and less prone to overfitting model.

**Regularization**

Dropout regularization and early stopping techniques were implemented to mitigate the issue of overfitting and enhance the generalization capabilities of the model. These methods effectively regulate the model's capacity to capture noise within the training data, thereby diminishing the risk of overfitting. Through systematic experimentation in this assignment, various dropout rates and early stopping strategies were tested, allowing for the identification and selection of the optimal parameters that best suited our specific model architecture and dataset characteristics.

# References

*keras.io*. (n.d.). Retrieved from InceptionV3: https://keras.io/api/applications/inceptionv3/