Machine Learning Application Development- Image Classifier Report

By Team 4

Nang Shri Kham Merng
Nguyen Binh Tran
Jiang Bingyu
Yeo Min Xiu Sherlin
Khant Zaw Hein
Wong Chee Kiat
Lu Bo Zhen

Fruit Classification Experiments:

1) **Using (default) provided image set**
2) **Using provided image set + image augmentation**
3) **Using additional training images**
4) **Using additional training images + image augmentation**

1) Experiments with given images

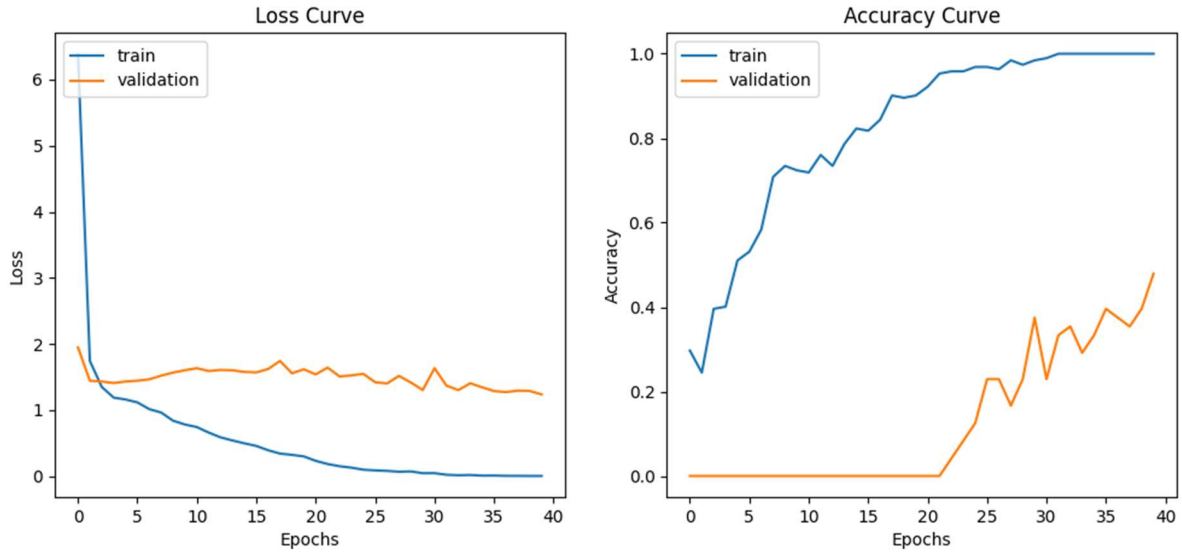
Experiment 1:

Actions: Initial model with 3 Conv2D layer, 2 Dense layers, epoch = 40, image size = 256

Model Sequence:

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=(5,5),activation='relu',
input_shape=(256,256,3)))
model.add(tf.keras.layers.BatchNormalization(axis=1))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2,2)))
model.add(tf.keras.layers.Dropout(0.25))
model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=(3,3), activation =
'relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2,2)))
model.add(tf.keras.layers.Conv2D(filters=32,kernel_size=(3,3),activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2,2)))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dropout(0.25))
model.add(tf.keras.layers.Dense(units=32, activation='relu'))
model.add(tf.keras.layers.Dense(units=4, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam',
```

Results:



Epoch 40/40
3/3 [==============================] - 12s 4s/step - loss: 0.0044 - accuracy: 1.0000 - val_loss: 1.2381 - val_accuracy: 0.4792

Auto-evaluation of model with test data:
loss = 0.6259
accuracy = 0.8000

Manual Evaluation of our model with test data:
correct: 48, wrong: 12
accuracy = 0.8

Interpretation of Results:
Looking at the validation results, this model seemed to have performed extremely poorly, as a validation accuracy of approximately only 48% was achieved. However surprisingly with this initial model, we managed to achieve a testing accuracy of 80%. In our next experiment, we will test the effects of reducing image size.

Experiment 2:
Actions: Image resized to 128.

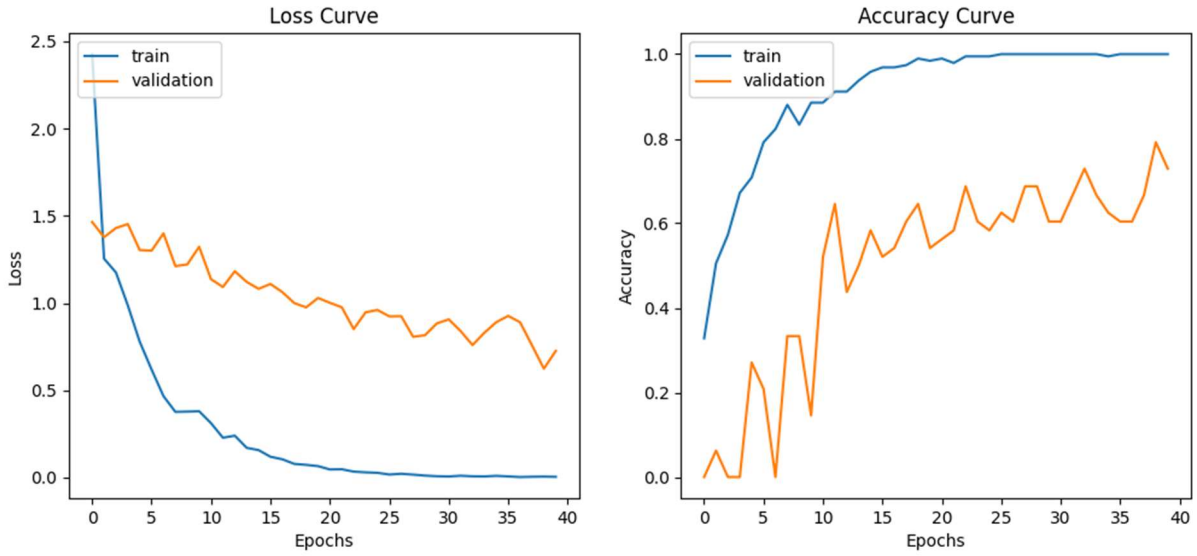
Model Sequence (changes):

```
img_resized = im.resize((128,128))

return np.reshape(x_train, (-1,128,128,3)), np.reshape(y_train, (-1, 4))

model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=(5,5),
activation='relu', input_shape=(128,128,3)))
```

Results:



Epoch 40/40
3/3 [==============================] - 3s 953ms/step - loss: 0.0046 - accuracy: 1.0000 - val_loss: 0.7259 - val_accuracy: 0.7292

Auto-evaluation of model with test data:
loss = 0.4840
accuracy = 0.8500

Manual Evaluation of our model with test data:
correct: 51, wrong: 9
accuracy = 0.85

Interpretation of Results:
With image size decreased to 64, it is observed that the accuracy has increased by roughly 5%. Theoretically, a smaller image size would likely increase the accuracy further.

Experiment 3:
Actions: Image resized to 64.

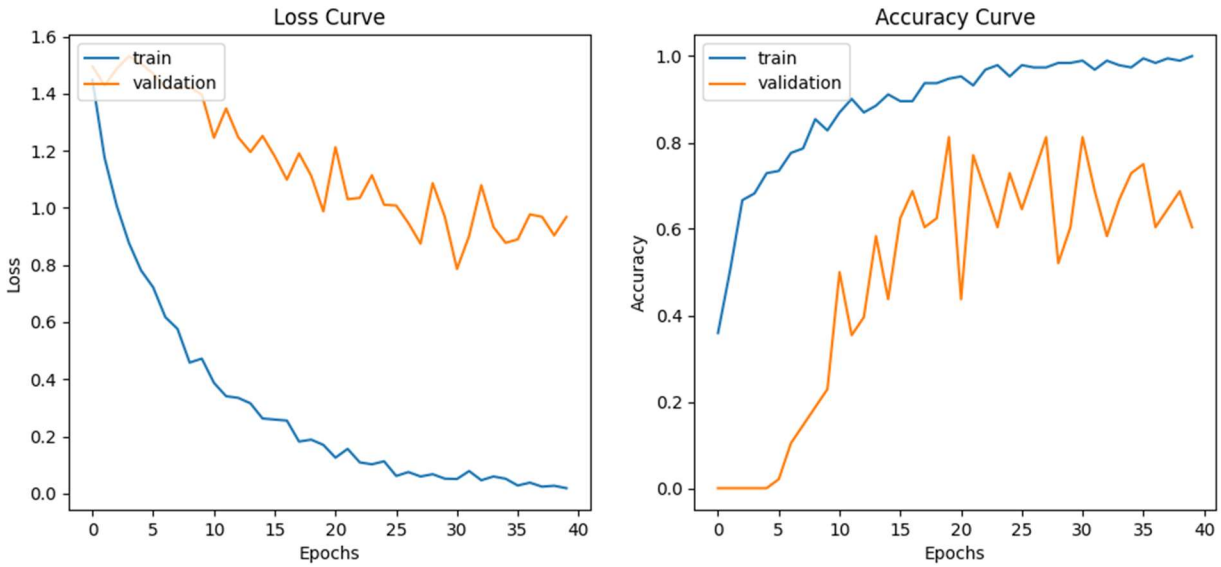
Model Sequence (changes):

```
img_resized = im.resize((64,64))

return np.reshape(x_train, (-1,64,64,3)), np.reshape(y_train, (-1, 4))

model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=(5,5),
activation='relu', input_shape=(64,64,3)))
```

Results:



Epoch 40/40
3/3 [==============================] - 0s 157ms/step - loss: 0.0185 - accuracy: 1.0000 - val_loss: 0.9684 - val_accuracy: 0.6042

Auto-evaluation of model with test data:
loss = 0.6009
accuracy = 0.8666

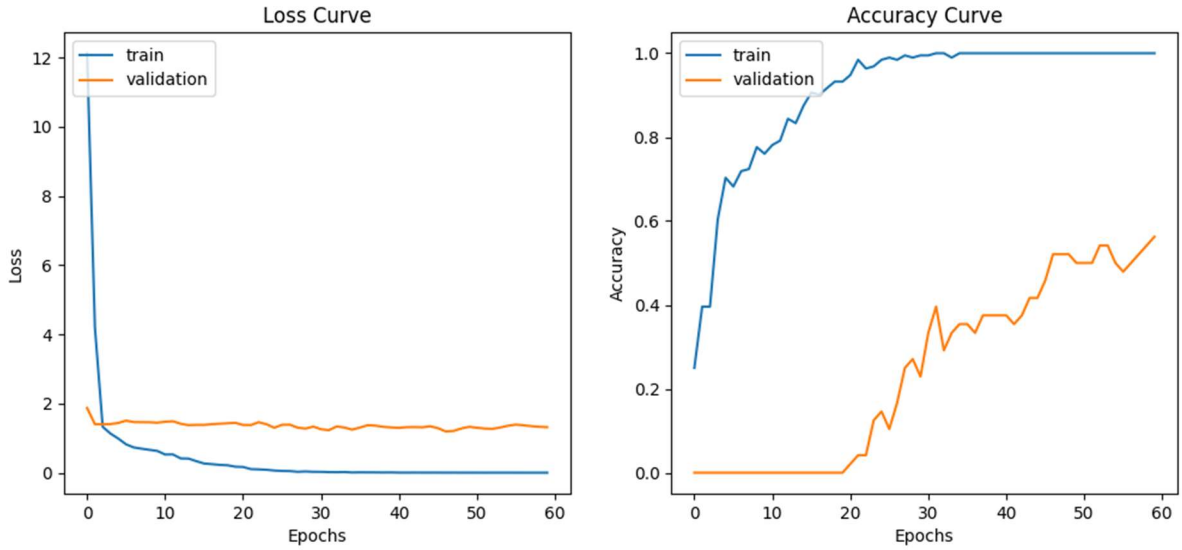
Manual Evaluation of our model with test data:
correct: 52, wrong: 8
accuracy = 0.87

Interpretation of Results:
With the smallest image size of 64, it is observed that the accuracy only increased by roughly 1.6%.

Experiment 4:
Actions:

a. increased epoch to 60
b. image size changed back to 256

Results:



Epoch 60/60
3/3 [==============================] - 11s 4s/step - loss: 0.0013 - accuracy: 1.0000 - val_loss: 1.3164 - val_accuracy: 0.5625

Auto-evaluation of model with test data:
loss = 0.46201959252357483
accuracy = 0.7833333611488342

Manual Evaluation of our model with test data:
correct: 47, wrong: 13
accuracy = 0.7833333333333333

Interpretation of Results:
We also tested to see if model accuracy can also be improved by properties apart from image size. In this experiment, we tested the effects of an increased number of epochs. As accuracy during training stayed at 1.0000 for many epochs towards the end, the decrease in accuracy indicates possible overfitting.

Experiment 5:
Actions:

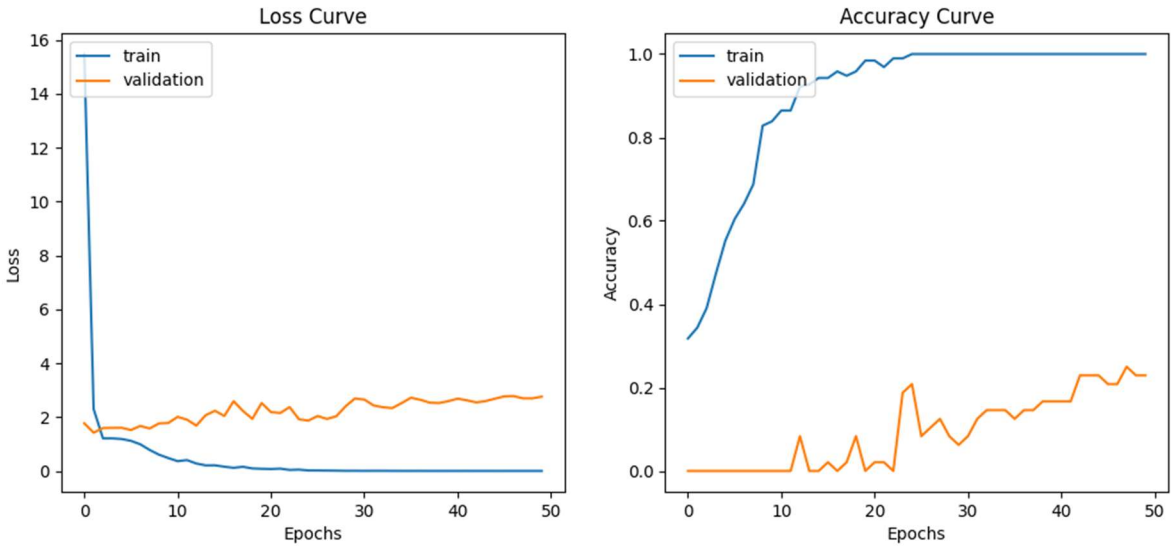
a. filter size changed to 64
b. changed first dense layer units to 64
c. epoch reduced to 50

Model Sequence (changes):

```
model.add(tf.keras.layers.Conv2D(filters=64, kernel_size=(5,5),
activation='relu', input_shape=(256,256,3)))

model.add(tf.keras.layers.Dense(units=64, activation='relu'))
```

Results:



Epoch 50/50
3/3 [==============================] - 23s 8s/step - loss: 0.0014 - accuracy: 1.0000 - val_loss: 2.7580 - val_accuracy: 0.2292

Auto-evaluation of model with test data:
loss = 1.26399827003479
accuracy = 0.6166666746139526

Manual Evaluation of our model with test data:
correct: 47, wrong: 13
accuracy = 0.6166666746139526

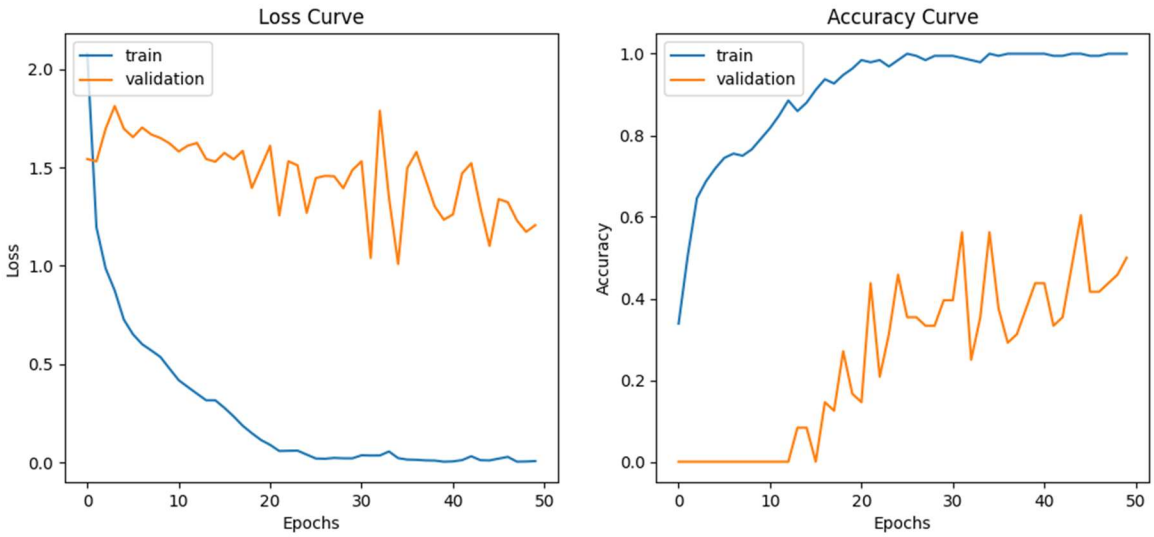
Interpretation of Results:

Based on our previous experiment, we decreased the number of epochs in order to reduce the chances of overfitting. We also tested the possible effects of changing filter size and the dense layer. However, increasing the filter and density to 64 reduced the accuracy down to 61.6%. Theoretically, decreasing the filter and density back down to 32 and setting the epoch to 50 should improve the result. Changing the image size to 128 or 64 should theoretically improve the accuracy which was indicated by previous experiments.

Experiment 6:
Actions:

a. filter size changed back to 32
b. changed first dense layer units back to 32
c. image size reduced to 128
c. switched from MaxPooling to AveragePooling

Results:



Epoch 50/50
3/3 [==============================] - 2s 860ms/step - loss: 0.0064 - accuracy: 1.0000 - val_loss: 1.2067 - val_accuracy: 0.5000

Auto-evaluation of model with test data:
loss = 0.6482096314430237
accuracy = 0.800000011920929

Manual Evaluation of our model with test data:
correct: 48, wrong: 12
accuracy = 0.8

Interpretation of Results:
Returning the filter as well as density back to 32, then switching maxpooling2d for averagepooling2d improves the result significantly when compared to the previous experiment.

Experiment 7:
Actions:

a. Image size reduced to 64
b. Kernel size increased to 7

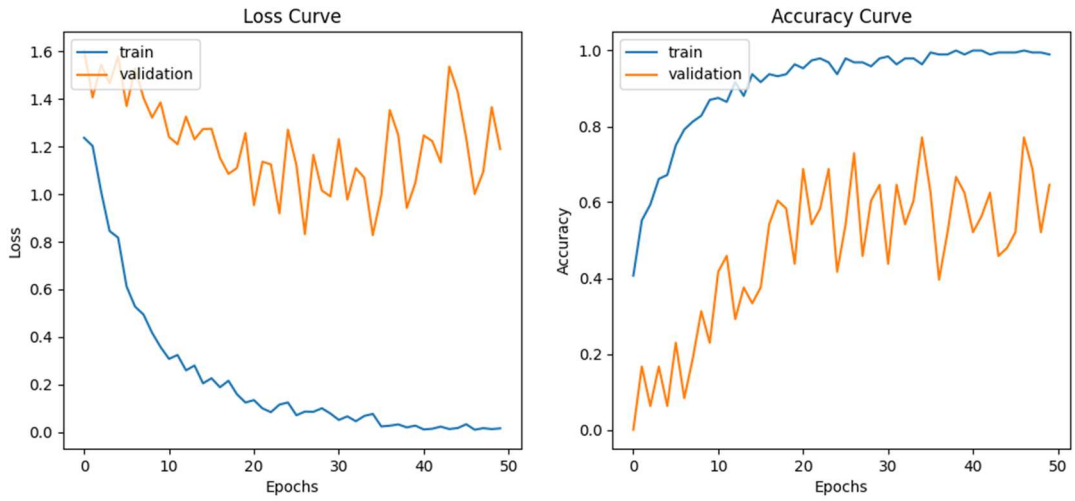
Model Sequence (changes):

```
img_resized = im.resize((64,64))

return np.reshape(x_train, (-1,64,64,3)), np.reshape(y_train, (-1, 4))

model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=(7,7),
activation='relu', input_shape=(64,64,3)))
```

Results:



Epoch 50/50
3/3 [==============================] - 1s 233ms/step - loss: 0.0164 - accuracy: 0.9896 - val_loss: 1.1895 - val_accuracy: 0.6458

Auto-evaluation of model with test data:
loss = 0.5427579879760742
accuracy = 0.8166666626930237

Manual Evaluation of our model with test data:
correct: 49, wrong: 11
accuracy = 0.8166666666666667

Interpretation of Results:
Lowering the image size and increasing the kernel size to (7,7) helps improve the accuracy by roughly 1.6%. In conclusion, the best results tend to appear more as the image size is smaller. Other factors which can also contribute would be a bigger kernel size as well as keeping the epoch value lower.

2) Experiments with given images + image augmentation

Experiment 1:
Actions:

a. **Accounted for mislabeled data:**

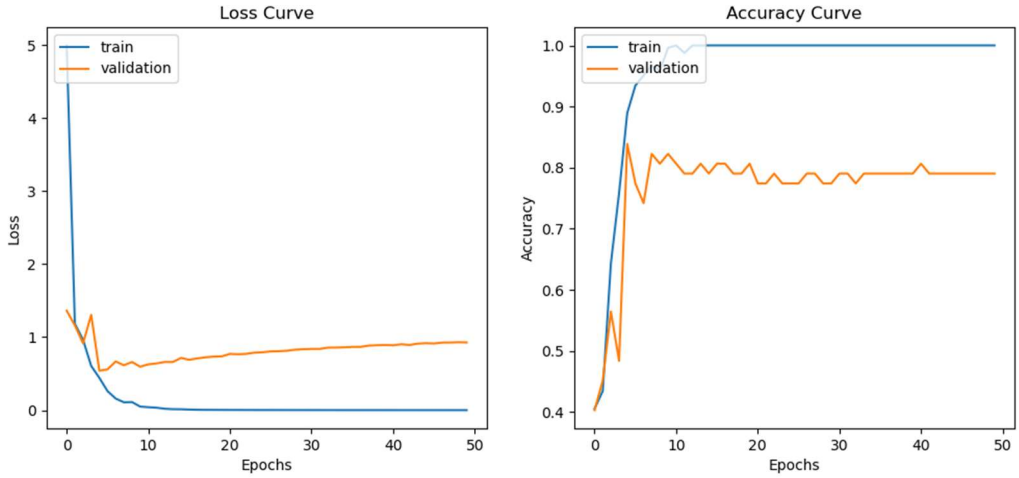
For training sample:
Banana_35 (misclassified: should be mixed fruits)
Banana_61 (misclassified: should be mixed fruits)

b. **Reclassified these two images to mixed fruits**
c. **Given that Apple, Banana and Orange class have about 70+ images each and Mixed Fruit class has only 20+ images, data augmentation is done for Mixed Fruit class.**

Method is written such that for each Mixed Fruit image, image rotation of 90 degrees and image rotation of 180 degrees and image flipping of top to bottom are done to create 3 more copies of each image. Hence, after data augmentation, mixed fruit class has about 80+ images, close to that of Apple, Banana and Orange. This resolves the imbalance in the sample data for each class.

d. As the image sizes of each image in the training dataset is different, resizing of the image is required to standardize the images for input into the machine for training and learning. A preliminary width and height of 64 by 64 (small size) is chosen for this experiment, as image size 128 by 128 takes too long for each epoch and for the system to train the model and run.
e. A preliminary choice of Epoch 50 is chosen with default batch size of 32.
f. Preliminary CNN model with one convoluted layer (filters=32, kernel size: 7,7; activation: relu), one flatten layer, one dense/output layer (with softmax activation in view of categorical output. Large kernel of 7,7 to extract a bigger feature map, and filters- 32 kernels to run over the input pixels. Activation relu: to convert any negative integers to 0 since all pixels should be 0-255 for images.
g. In model fitting, set the validation split to 20% to set aside 20% of training data for validation so that at each epoch, we can assess the accuracy of our neural network and determine the optimum where there is optimal fitting (no under or overfitting of model to training data)

Results:



Epoch 50/50
8/8 [==============================] - 0s 36ms/step
- loss: 7.8107e-04 - accuracy: 1.0000
- validation loss: 0.9276 – validation accuracy: 0.7903

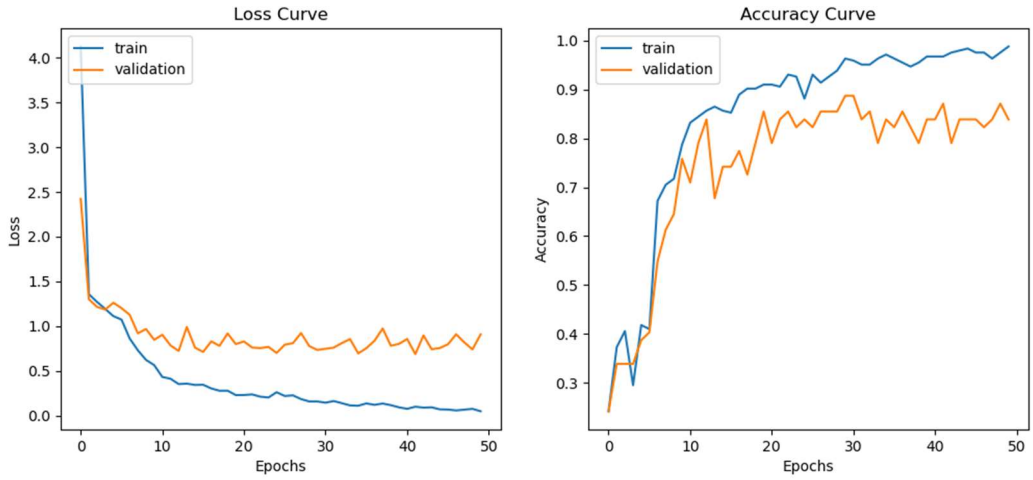
Auto-evaluation of model with test data:
loss = 1.1832400560379028
accuracy = 0.8166666626930237

Manual Evaluation of our model with test data:
correct: 49, wrong: 11
accuracy = 0.8166666666666667

Interpretation of Results:
The model seems to fit well on the training data such that the accuracy score is close to 1.0 and minimum loss close to 0.0 from 20 epoch onwards. However, it does not seem to predict as well on validation and test data with accuracy of 79% to 81%, hence this symbolizes overfitting. And to resolve this, we need more or better training data to help the model to generalize images such that it can predict validation and test data more accurately.

Experiment 2:
Actions:

a. **Added preprocessing layers (RandomFlip and RandomRotation) to the CNN model to do image augmentation to help the model see a greater diversity of images and learn better from them**



Epoch 50/50
8/8 [==============================] - 0s 41ms/step
- loss: 0.0497 - accuracy: 0.9877
- validation loss: 0.9085 – validation accuracy: 0.8387

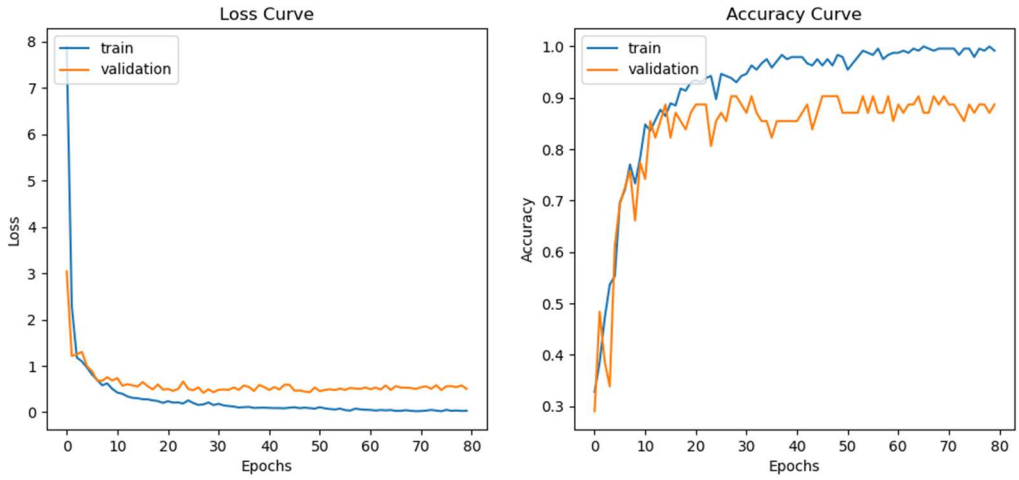
Auto-evaluation of model with test data:
loss = 0.995516836643219
accuracy = 0.800000011920929

Manual Evaluation of our model with test data:
correct: 48, wrong: 12
accuracy = 0.8

Interpretation of Results:
With the preprocessing layers added, it seems that the model is still learning even at epoch 50, with losses continually decreasing. This is understandable since more diverse data are provided to the machine for learning. It is good to observe that the validation loss/accuracy curve is closer to the training loss/accuracy curve than experiment 1. This indicates that the model is fitting generally on the training data and able to predict test data with similar accuracy of 80%. In this case, it seem that the model still need more time for training on the training dataset to further optimize the accuracy.

Experiment 3:
Actions:

a. **Increased the Epoch to 80 and increase batch size to 64 to speed up machine learning**



Epoch 80/80
4/4 [==========] - 0s 80ms/step
- loss: 0.0332 - accuracy: 0.9918 –
-validation loss: 0.5046 – validation accuracy: 0.8871

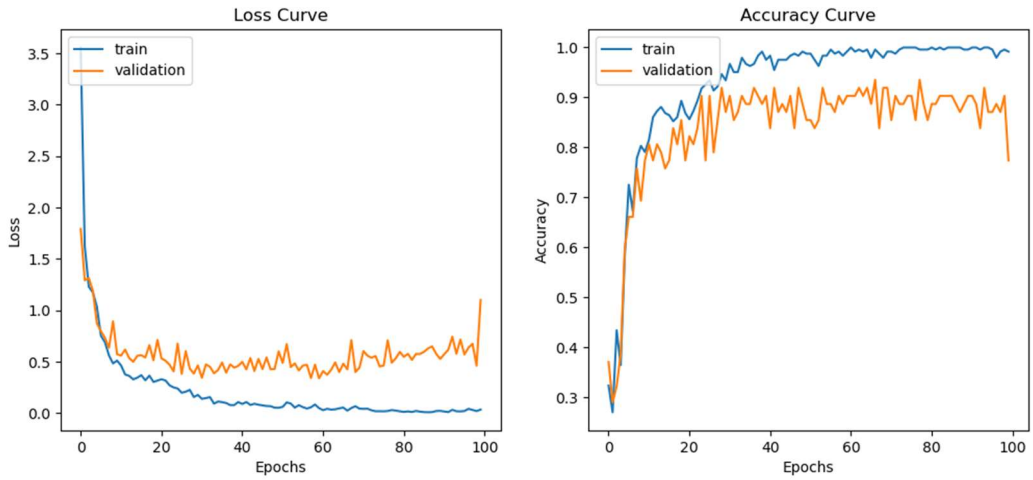
Auto-evaluation of model with test data:
loss = 0.9137647747993469
accuracy = 0.8333333134651184

Manual Evaluation of our model with test data:
correct: 50, wrong: 10
accuracy = 0.8333333333333334

Interpretation of Results:
With the increased in Epoch, it seems that the model has more time for learning and fits much better to the training data, hence improving accuracy at epoch 80 compared to experiment 2 (epoch 50) and in turn, the validation and test accuracy also improved from 0.83 and 0.8 respectively.

Experiment 4:
Actions:

a. **Increased the Epoch to 100**



Epoch 100/100
4/4 [==============================] - 0s 80ms/step
- loss: 0.0344 - accuracy: 0.9918
- validation loss: 1.1011 – validation accuracy: 0.7742

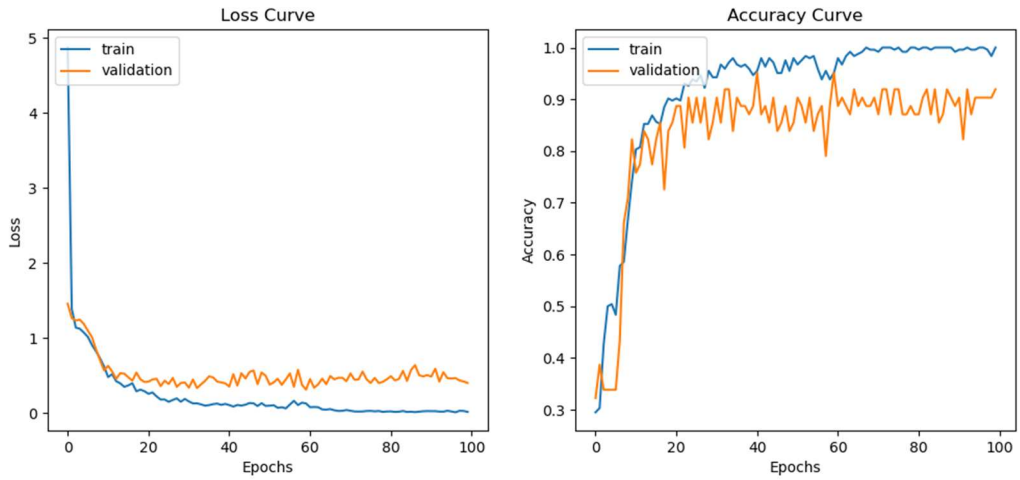
Auto-evaluation of model with test data:
loss = 1.984440565109253
accuracy = 0.7166666388511658

Manual Evaluation of our model with test data:
correct: 43, wrong: 17
accuracy = 0.7166666666666667

Interpretation of Results:
With the increased in Epoch from 80 to 100, we can start seeing the deviation of validation curve from the training curve for both loss and accuracy. This means that the additional epoch is helping the model to fit the training data better with accuracy of 0.99 (improved) at Epoch 100. However, the validation and test accuracy dropped from about 0.8 to 0.7. This shows overfitting of training data with too many epoch cycles and model become too specific and not general enough.

Experiment 5:
Actions:

a. **Keep at Epoch 100.**
b. **Try to fix overfitting by adding a dropout layer (drop 25% neurons at random) for regularization to discourage the model to memorise training data and to discourage fixed training patterns in a neural network.**



Epoch 100/100
4/4 [==============================] - 0s 93ms/step
- loss: 0.0163 - accuracy: 1.0000
- validation loss: 0.4019 – validation accuracy: 0.9194

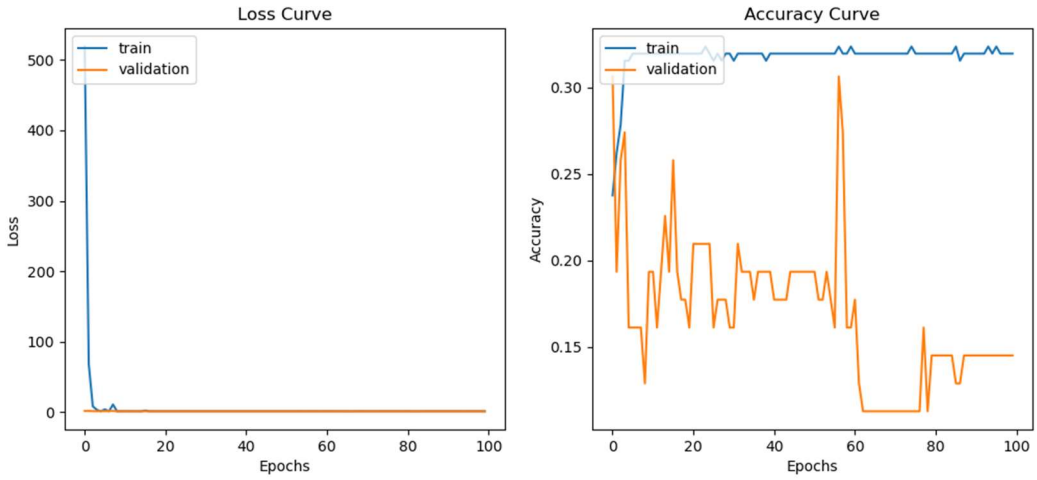
Auto-evaluation of model with test data:
loss = 1.0016264915466309
accuracy = 0.8500000238418579

Manual Evaluation of our model with test data:
correct: 51, wrong: 9
accuracy = 0.85

Interpretation of Results:
With the addition of dropout layer, the validation accuracy and test accuracy are closer to the model accuracy. There is also improvement in the validation accuracy and test accuracy compared to Experiment 3, given more training done with dropout layer to ensure training of the model in a general manner so that model can cope better with test data. Currently the model is still overfitting as its prediction on training data is close to 100% but that of test and validation data is about 85% to 91%.

Experiment 6:
Actions:

a. **Keep at Epoch 100.**
b. **Try to fix overfitting by adding more and better training data. Add random contrast, random zoom, random brightness layers**



Epoch 100/100
4/4 [==========] - 0s 111ms/step
- loss: 1.3726 - accuracy: 0.3197
- validation loss: 1.4030 – validation accuracy: 0.1452

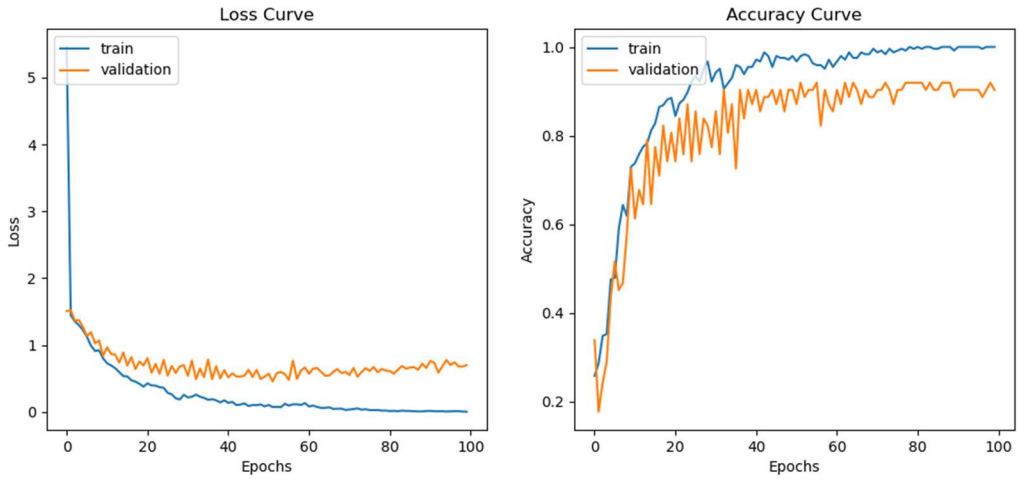
Auto-evaluation of model with test data:
loss = 1.4173662662506104
accuracy = 0.0833333358168602

Manual Evaluation of our model with test data:
correct: 5, wrong: 55
accuracy = 0.08333333333333333

Interpretation of Results:
With the addition of more preprocessing layer for the images, we observe that too much data augmentation leads to underfitting of data in our model and slows down learning as the model is now seeing increased diversity of data which likely contribute as noise and outliers. For example, with the random contrast, brightness and zoom added for preprocessing, it will be difficult for the model to differentiate the fruits based on colour/size/shape/appearance. In this case, our test and validation data are with normal contrast, brightness and zoom as augmentation with these layers are only done for training dataset. Hence, test and validation accuracy drop drastically to 8% to 14% from the previous experiment.

Experiment 7:
Actions:

a. **Keep at Epoch 100.**
b. **Remove the random contrast, random zoom, random brightness layers which contributes noise to model training**
c. **Adding another dense layer (with units 28) to the neural network. Number of neurons selected as it is between the number of neurons in the input and output layer.**



Epoch 100/100
4/4 [==========] - 0s 103ms/step
- loss: 0.0062 - accuracy: 1.0000
- validation loss: 0.7021 – validation accuracy: 0.9032

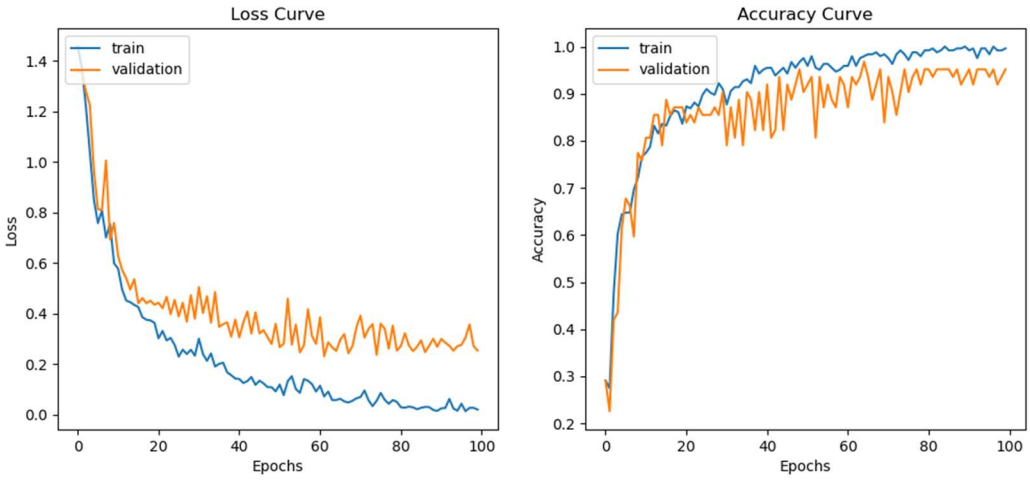
Auto-evaluation of model with test data:
loss = 1.261525273323059
accuracy = 0.8500000238418579

Manual Evaluation of our model with test data:
correct: 51, wrong: 9
accuracy = 0.85

Interpretation of Results:
With the addition of another dense layer, the results are similar to Experiment 5, still indicative of overfitting.

Experiment 8:
Actions:

a. **Added additional max pooling layer (2,2), dropout layer(0.25), dense layer and convoluted layer (Kernel 32) to help with the computation of model and test if it improves accuracy of prediction**



Epoch 100/100
4/4 [==============================] - 0s 111ms/step
- loss: 0.0203 - accuracy: 0.9959
- validation loss: 0.2537 – validation accuracy: 0.9516

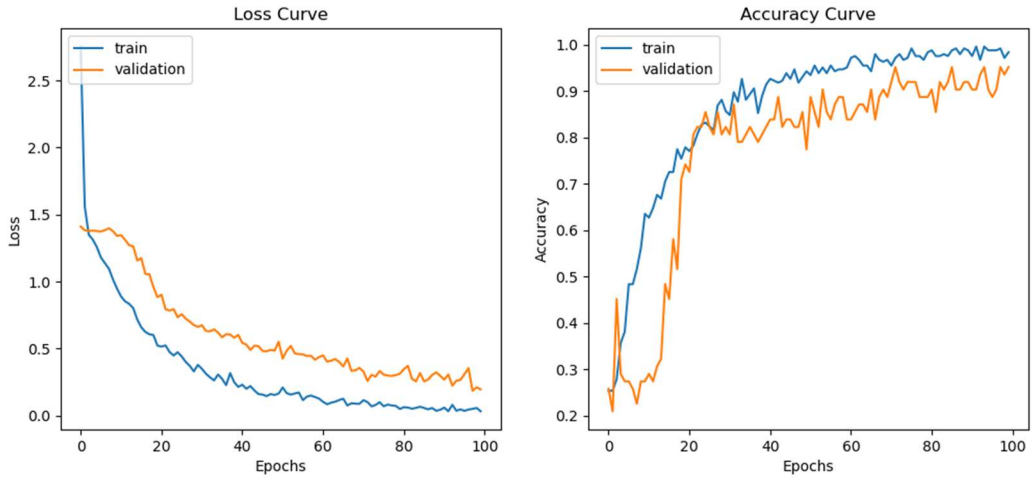
Auto-evaluation of model with test data:
loss = 0.9394766092300415
accuracy = 0.8999999761581421

Manual Evaluation of our model with test data:
correct: 54, wrong: 6
accuracy = 0.9

Interpretation of Results:
With the addition of extra layers, the test and validation accuracy improve to 86% to 91% compared to Experiment 5.

Experiment 9:
Actions:

a. **Added batch normalization layer after the convoluted layer that allows every layer of the network to do learning more independently, by normalizing the output of previous layers. This avoids overfitting of model.**



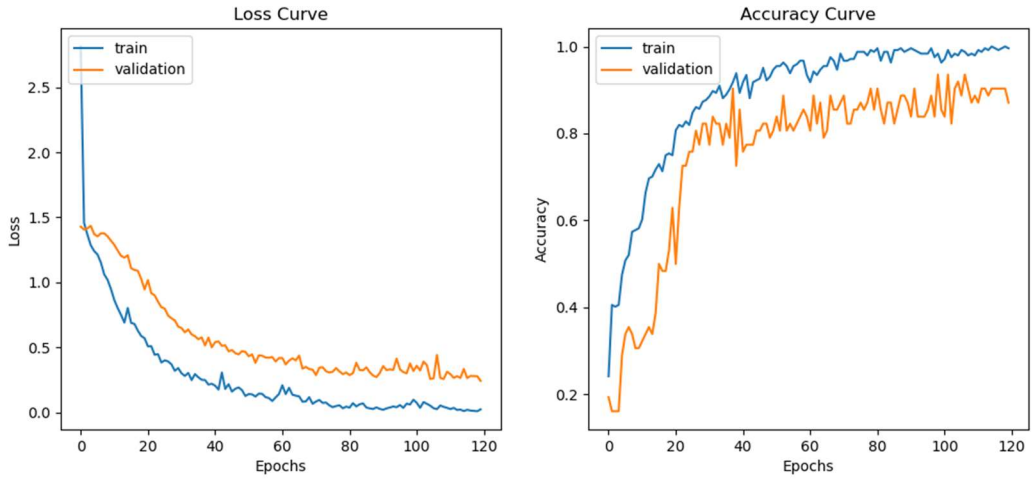
Epoch 100/100
4/4 [==============================] - 1s 159ms/step
- loss: 0.0330 - accuracy: 0.9836
- validation loss: 0.1962 – validation accuracy: 0.9516

Auto-evaluation of model with test data:
loss = 0.6560331583023071
accuracy = 0.8666666746139526
Manual Evaluation of our model with test data:
correct: 52, wrong: 8
accuracy = 0.8666666666666667

Interpretation of Results:
With the addition of the batch normalization layer, it helps to prevent overfitting of model, hence there is slight loss of training accuracy at Epoch 100.

Experiment 10:
Actions:

a. **Increase Epoch to 120 for longer machine learning time.**



Epoch 120/120
4/4 [==============================] - 1s 162ms/step
- loss: 0.0227 - accuracy: 0.9959
- validation loss: 0.2422 – validation accuracy: 0.8710

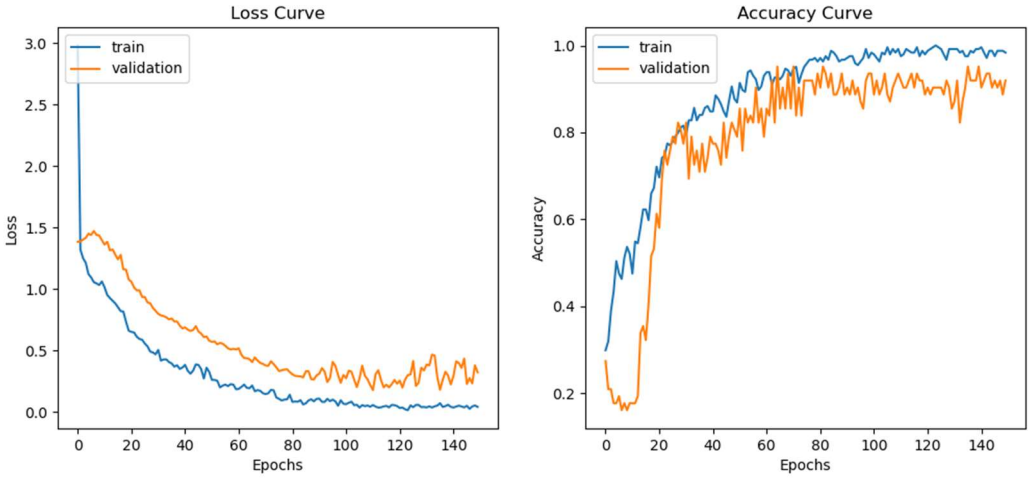
Auto-evaluation of model with test data:
loss = 0.3985848128795624
accuracy = 0.9333333373069763

Manual Evaluation of our model with test data:
correct: 56, wrong: 4
accuracy = 0.9333333333333333

Interpretation of Results:
With the increase in epoch, the curve still looks down trending and can be optimized further.

Experiment 11:
Actions:

a. **Increase Epoch to 150 for longer machine learning time.**



Epoch 150/150
4/4 [==============================] - 1s 157ms/step
- loss: 0.0406 - accuracy: 0.9836
- validation loss: 0.3214 – validation accuracy: 0.9194

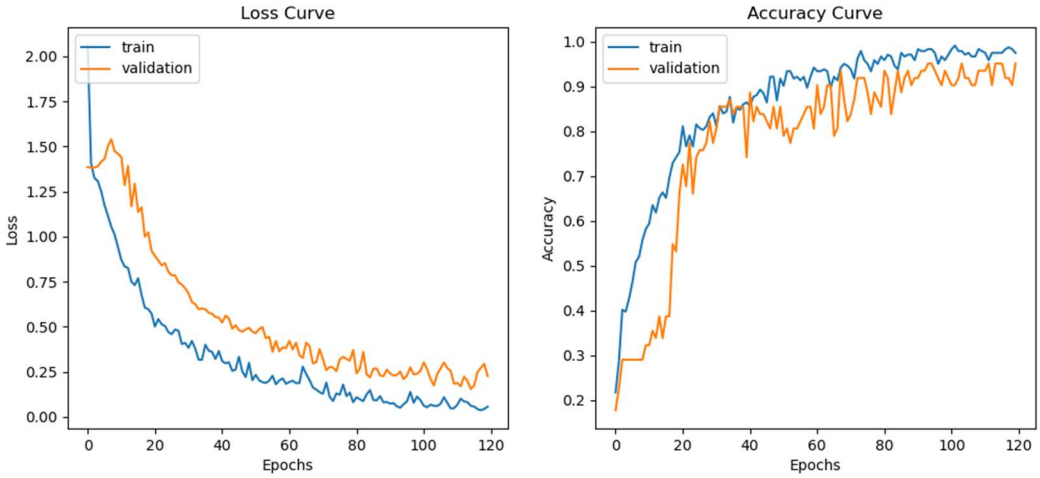
Auto-evaluation of model with test data:
loss = 0.7194293141365051
accuracy = 0.8666666746139526

Manual Evaluation of our model with test data:
correct: 52, wrong: 8
accuracy = 0.8666666666666667

Interpretation of Results:
With this graph, it seems that the curves for validation and train start to deviate a little, hence the minimum losses point looks to be at about epoch 100-120. Based on past experiment data, we can keep to epoch 100-120.

Experiment 12:
Actions:

a. **Modify Epoch to 120.**
b. **Modify one of the dropout layer to 50% neurons drop out at random to prevent overfitting and help the model to be more general**



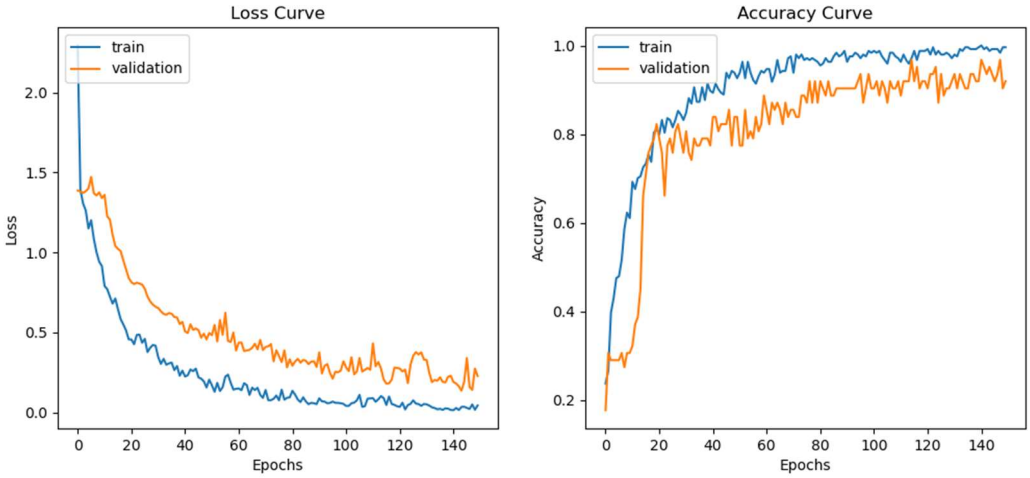
Epoch 120/120
4/4 [==============================] - 1s 165ms/step
- loss: 0.0566 - accuracy: 0.9754
- validation loss: 0.2269 – validation accuracy: 0.9516

Auto-evaluation of model with test data:
loss = 0.7445380687713623
accuracy = 0.8500000238418579
Manual Evaluation of our model with test data:
correct: 51, wrong: 9
accuracy = 0.85

Interpretation of Results:
With the increase in epoch, the curve still looks down trending and can be optimized further.

Experiment 13:
Actions:

a. **Increase Epoch to 150**



Epoch 150/150
4/4 [==============================] - 1s 160ms/step
- loss: 0.0431 - accuracy: 0.9959
- validation loss: 0.2272 – validation accuracy: 0.9194

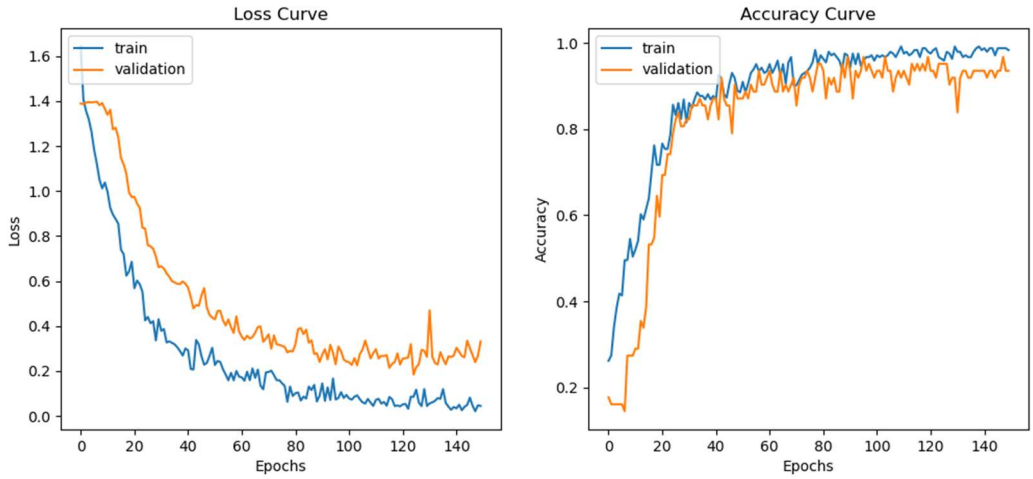
Auto-evaluation of model with test data:
loss = 0.784076452255249
accuracy = 0.8999999761581421

Manual Evaluation of our model with test data:
correct: 54, wrong: 6
accuracy = 0.9

Interpretation of Results:
This seems like the optimal epoch as the curve seems to have reach its minimum point.

Experiment 14:
Actions:

a. **Add extra convoluted and pooling layers into the model**



Epoch 150/150
4/4 [==============================] - 1s 161ms/step
- loss: 0.0444 - accuracy: 0.9836
- validation loss: 0.3318 – validation accuracy: 0.9355

Auto-evaluation of model with test data:
loss = 0.645818829536438
accuracy = 0.8833333253860474

Manual Evaluation of our model with test data:
correct: 53, wrong: 7
accuracy = 0.8833333333333333

Interpretation of Results:
It seems that adding convoluted layers and pooling layers did not help enhance accuracy. In this case, two sets of convoluted layers and pooling layers seem sufficient for model training to attain 80-90% predictive accuracy for test and validation data, and the modification of one dropout layer from 25% to 50% did not help much as well.

Conclusion: Experiment 9 and 10's model seems the most optimal thus far.

3) Experiments with added images for train models

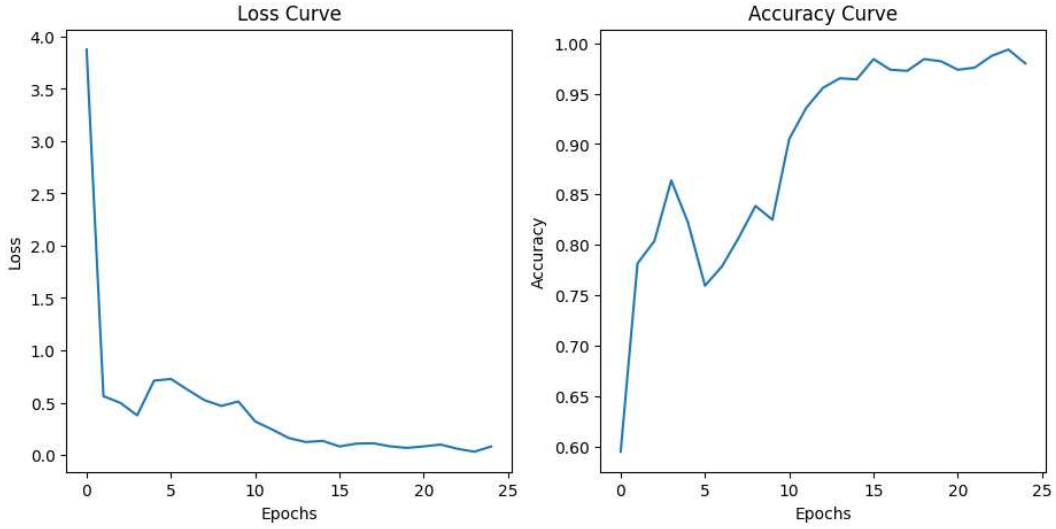
Experiment 1:
Actions:

a. **Added more train images**
   Apples, Banans and Oranges have 300+ images each and 75+ mixed fruits.
b. **Model Sequence**
   Changed image size to 256 by 256.

```
model.add(tf.keras.layers.Conv2D(filters=32,kernel_size=(7, 7), activation='relu',
input_shape=(256,256,3)))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2,2)))
model.add(tf.keras.layers.Dropout(0.20))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(units=128, activation='relu'))
model.add(tf.keras.layers.Dropout(0.20))
model.add(tf.keras.layers.Dense(units=16, activation='relu'))
model.add(tf.keras.layers.Dense(units=4, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

c. **Implementing Callbacks**
   Implemented callback function to cancel the epoch when it reaches to 99.6% accuracy.
d. **Epochs**
   Set epoch to 25

Results:



Epoch 25/25
30/30 [==============================] - 30s 990ms/step
- loss: 0.0802 - accuracy: 0.9800

Auto-evaluation of model with test data:
loss = 0.4851
accuracy = 0.8333

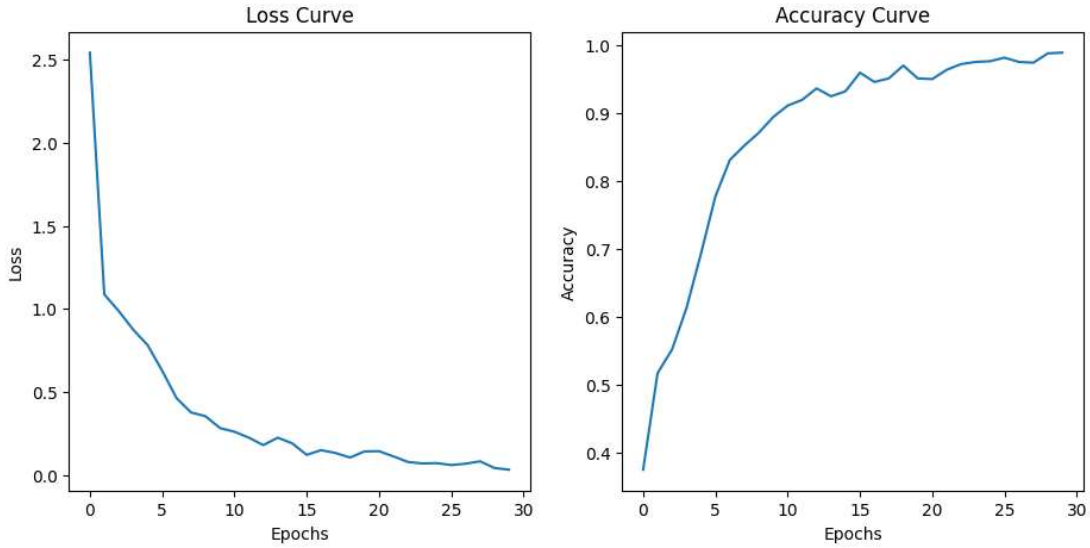
Manual Evaluation of our model with test data:
correct: 50, wrong: 10
accuracy = 0.8333333333333334

Interpretation of Results:
With this initial model, it seems that the 25 epoch is insufficient as the model is still learning at epoch end. Needs room for improvement to further optimize.

Experiment 2:
Actions:

c. **Increase Epoch to 30.**
d. **Add extra convoluted and pooling layers into the model**

Results:



Epoch 30/30
30/30 [==============================] - 35s 1s/step
- loss: 0.0328 - accuracy: 0.9895
Auto-evaluation of model with test data:
loss = 1.1314
accuracy = 0.8667

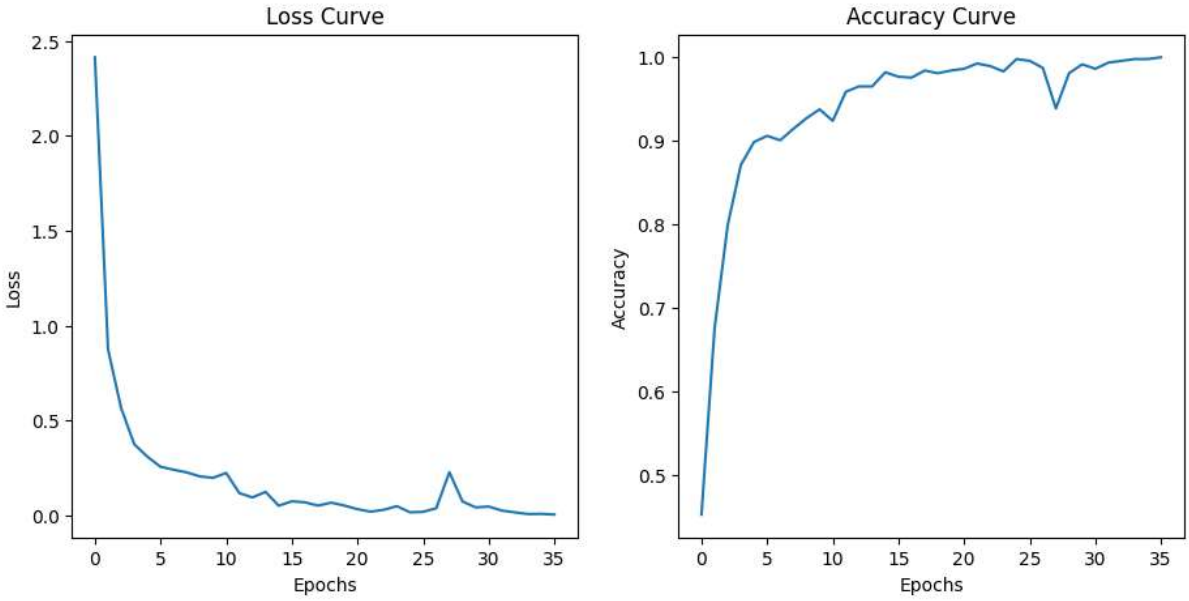
Manual Evaluation of our model with test data:
correct: 52, wrong: 8
accuracy = 0.8333333333333334

Interpretation of Results:
It seems that the 30 epoch is still insufficient and seems to be able to go higher accuracy and yet the accuracy has increased.

Experiment 3:
Actions:

a. **Increase Epoch to 40.**

Results:



Epoch 36/50
30/30 [==============================] - ETA: 0s
- loss: 0.0051 - accuracy: 0.9989
Reached 99.8% accuracy. cancelling training!

Auto-evaluation of model with test data:
loss = 1.3103
accuracy = 0.8500

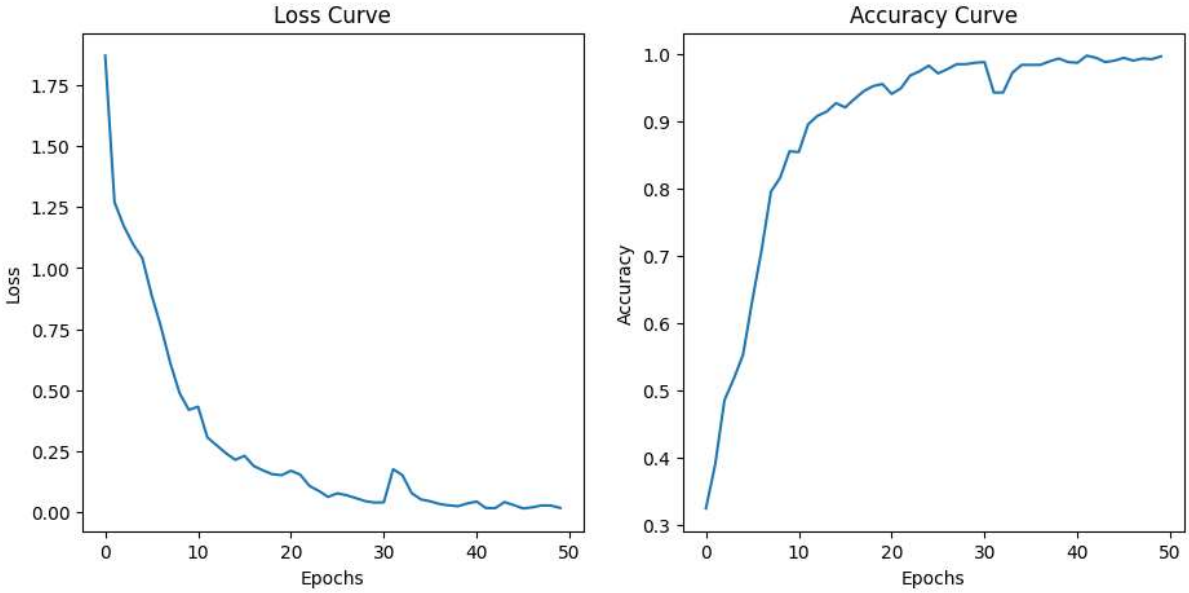
Manual Evaluation of our model with test data:
correct: 51, wrong: 9
accuracy = 0.8500

Interpretation of Results:
The epoch reached 99.8% at the 36th epoch. Increasing epoch didn't seem to improve the accuracy and slightly decreased instead.

Experiment 4:
Actions:

a. Set Epoch to 50.
b. Decrease the first Dense layer neurons from 128 to 64 unit
c. Set the larger batch size for a better gradient

Results:



Epoch 50/50
15/15 [==============================] - 33s 2s/step
- loss: 0.0170 - accuracy: 0.9968

Auto-evaluation of model with test data:
loss = 0.7115
accuracy = 0. 8833

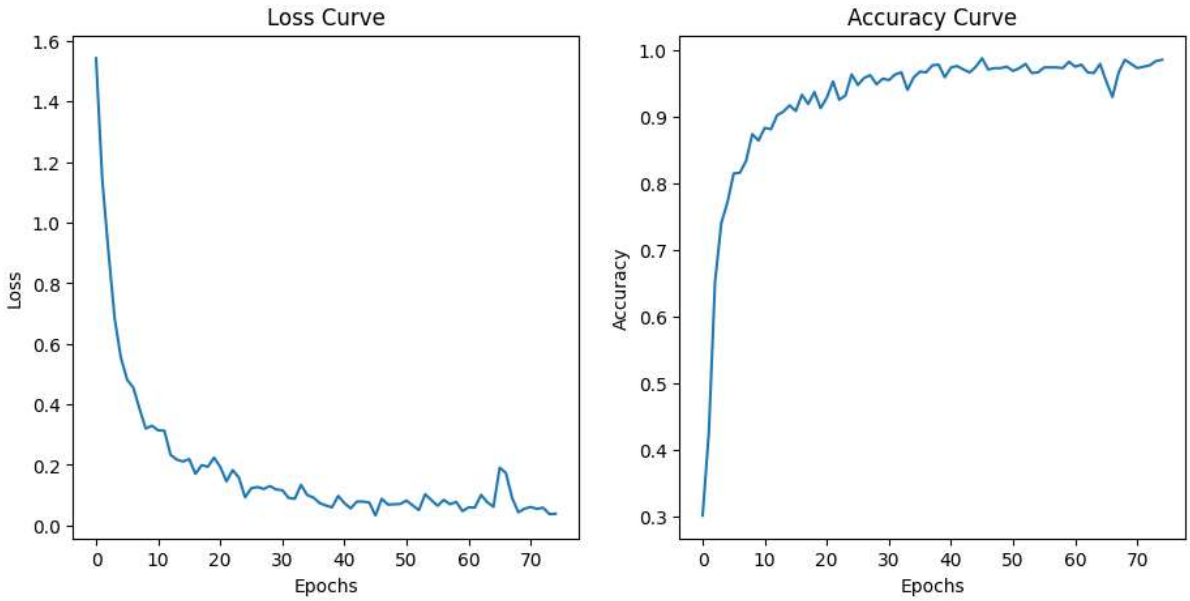
Manual Evaluation of our model with test data:
correct: 53, wrong: 7
accuracy = 0.8833

Interpretation of Results:
The result improved from this setting.

Experiment 5:
Actions:

a. Set Epoch to 75.
b. steps_per_epoch : 30

Results:



Epoch 75/75

30/30 [==============================] - 32s 1s/step

- loss: 0.0388 - accuracy: 0.9863

Auto-evaluation of model with test data:
loss = 1.2008
accuracy = 0.9000

Manual Evaluation of our model with test data:
correct: 54, wrong: 6
accuracy = 0.90

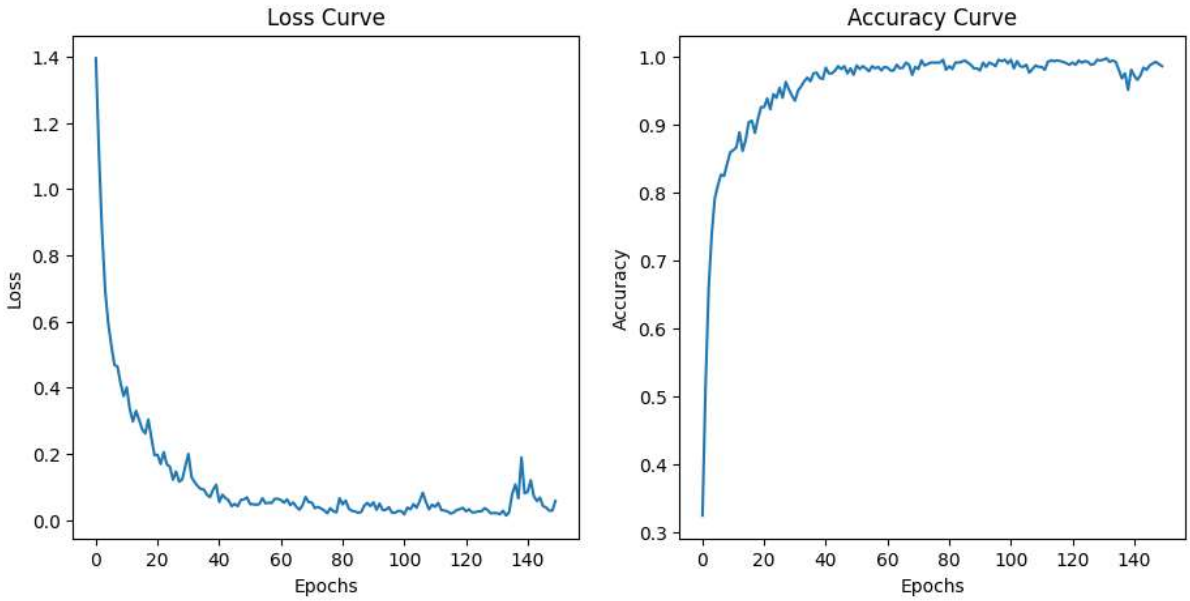
Interpretation of Results:
The result once again improved from increasing the epochs. It seems that higher epochs produce better accuracy even after reducing the neurons.

Experiment 6:

Actions:

a. Set Epoch to 150.
b. Reduce image size from 256x256 to 128x128 in order to run faster.
c. Increase batch size back to 64

Results:



Epoch 150/150
15/15 [==============================] - 6s 422ms/step
- loss: 0.0574 - accuracy: 0.9852

Auto-evaluation of model with test data:
loss = 2.0744
accuracy = 0.8333

Manual Evaluation of our model with test data:
correct: 50, wrong: 10
accuracy = 0.83

Interpretation of Results:
The accuracy went up and down since 60 epochs. Concluding higher epoch alone would not benefit much.

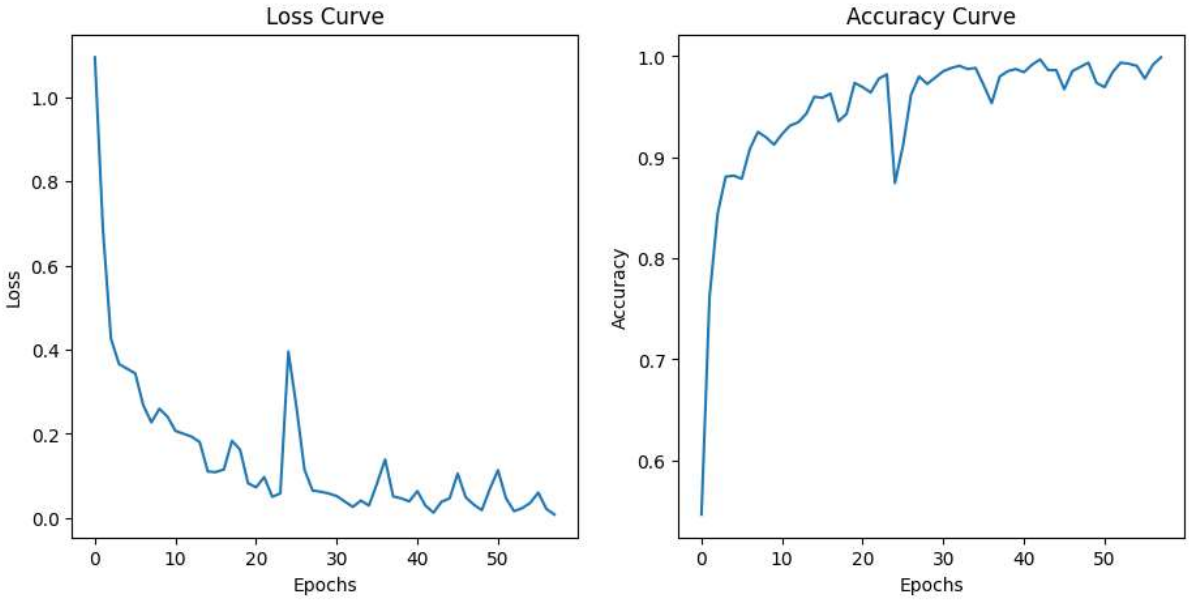
Experiment 7:

Actions: testing to add another convulated layer with kernel size (9,9)

Model Sequence:

```
model.add(tf.keras.layers.Conv2D(filters=32,kernel_size=(9, 9), activation='relu',
input_shape=(64,64,3)))
model.add(tf.keras.layers.Conv2D(filters=32,kernel_size=(7, 7), activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2,2)))
model.add(tf.keras.layers.Dropout(0.20))
model.add(tf.keras.layers.Conv2D(filters=32,kernel_size=(3, 3), activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2,2)))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(units=64, activation='relu'))
model.add(tf.keras.layers.Dropout(0.20))
model.add(tf.keras.layers.Dense(units=4, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Results:



Epoch 58/80

Reached 99.8% accuracy. cancelling training!

30/30 [==============================] - 7s 234ms/step

- loss: 0.0082 - accuracy: 0.9989

Auto-evaluation of model with test data:

loss = 1.2947

accuracy = 0.8833

Manual Evaluation of our model with test data:

correct: 53, wrong: 7

accuracy = 0.8833

Interpretation of Results:

Adding an extra Convoluted Layer did not seem to help with improving model prediction.

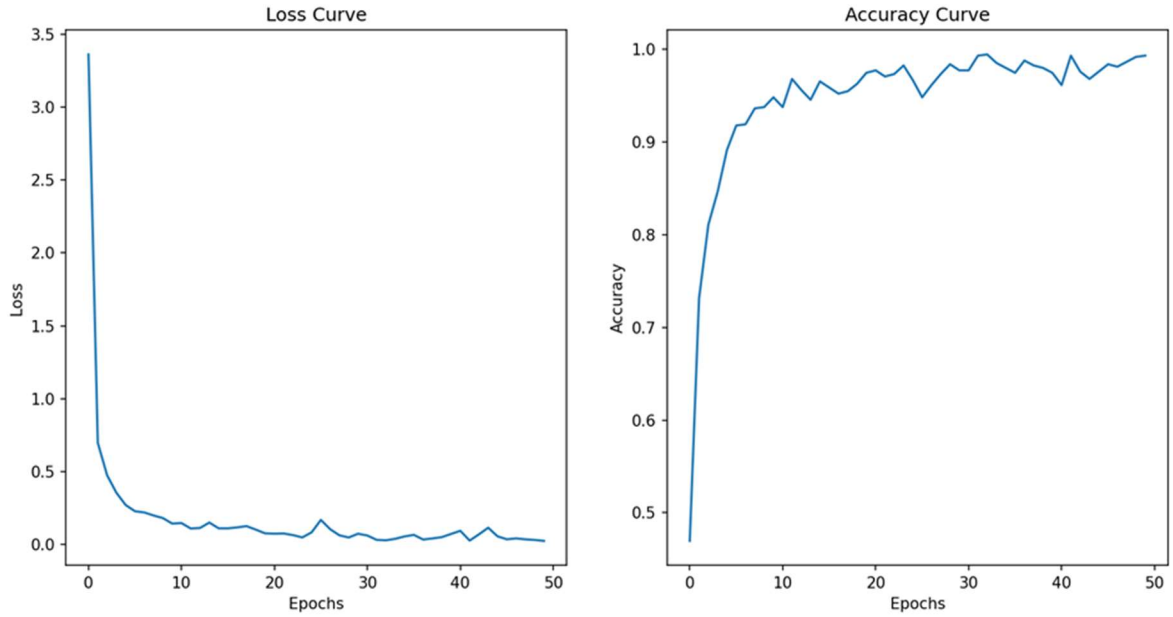
4) Experiments with added images for train models + image augmentation

Experiment 1:
Actions: added built-in augmentation functions (RandomFlip, RandomRotation, RandomZoom). Simple model with 1 Conv2D layer, 2 Dense layers, epoch = 50, image size = 128, callback: accuracy > 0.9999

Model Sequence:

```
model.add(tf.keras.layers.RandomFlip('horizontal'))
model.add(tf.keras.layers.RandomRotation(0.2))
model.add(tf.keras.layers.RandomZoom(0.2))
model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu'))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(units=32, activation='relu'))
model.add(tf.keras.layers.Dense(units=64, activation='relu'))
model.add(tf.keras.layers.Dense(units=4, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Results:



Epoch 50/50
24/24 [==============================] - 6s 244ms/step - loss: 0.0255 - accuracy: 0.9934

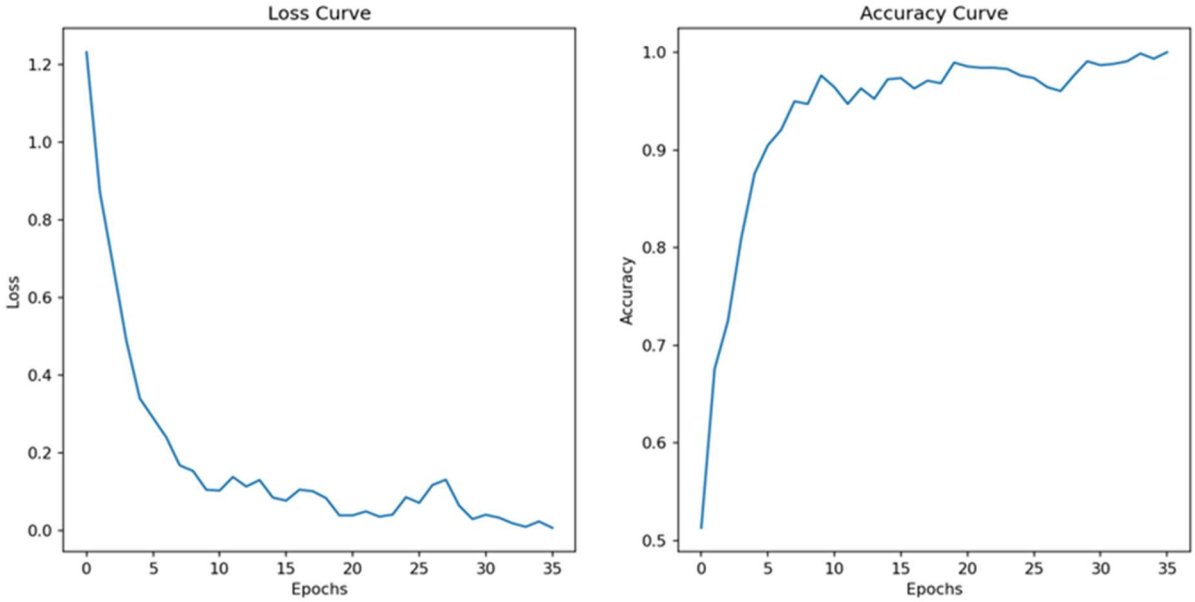
Auto-evaluation of model with test data:
loss = 0.5596
accuracy = 0.8833

Manual Evaluation of our model with test data:
correct: 53, wrong: 7
accuracy = 0.8833

Interpretation of Results:
In terms of testing, this initial model seems to perform well, as seen from a relatively high prediction accuracy of approximately 0.8833. However, as indicated by the accuracy value derived during model training, there is still room for possible improvement and it is worth testing the effects of a larger epoch size.

Experiment 2:
Actions:
a. added more convolutional layers (4 Conv2D layers in total, same parameters)
b. added a pooling layer after each convolutional layer (MaxPooling2D) to reduce training load
c. increased epoch size (100)
d. added dropout layer [Dropout(0.20)]

Results:



Epoch 36/100
24/24 [==============================] - 10s 430ms/step - loss: 0.0060 - accuracy: 1.0000

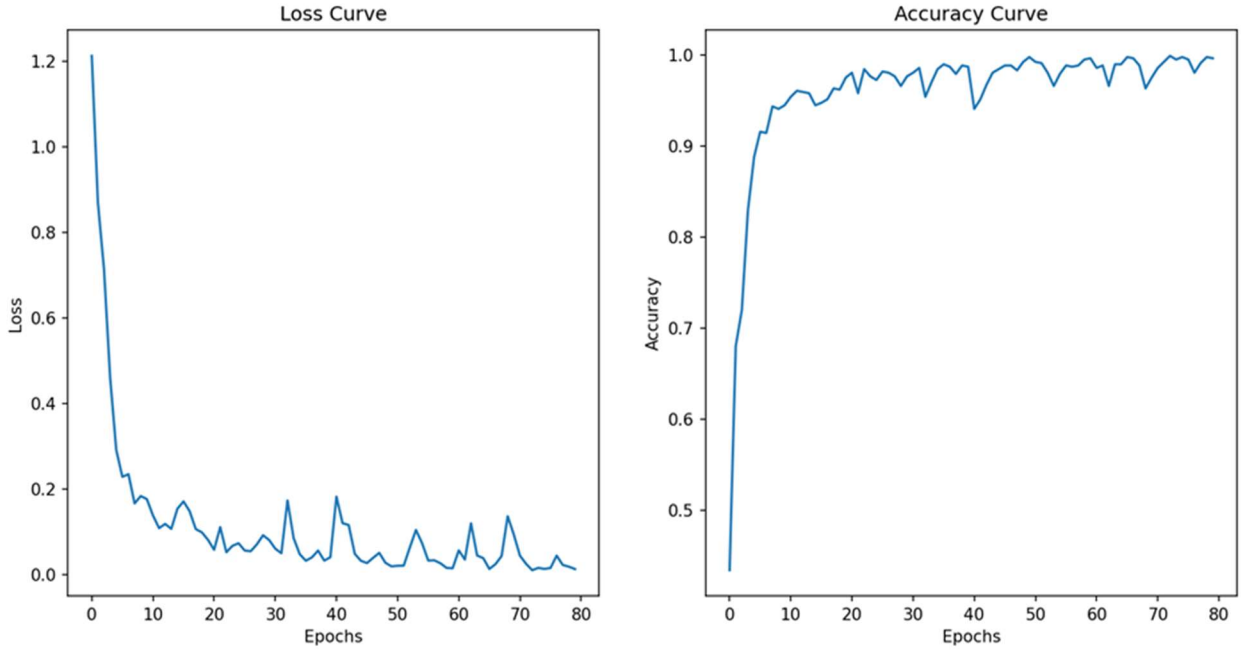
Auto-evaluation of model with test data:
loss = 0.4643
accuracy = 0.9000

Manual Evaluation of our model with test data:
correct: 54, wrong: 6
accuracy = 0.9000

Interpretation of Results:
Testing accuracy has improved with this model. This may be the result of 1) the addition of more convolutional layers, which may have contributed to improved feature extraction; and 2) the addition of a dropout layer, which helped to reduce the possibility of model overfitting.

Experiment 3:
Actions:
a. changed pooling layers from Max to Average pooling
b. reduced epoch size (80)

Results:



Epoch 80/80
24/24 [==============================] - 8s 342ms/step - loss: 0.0124 - accuracy: 0.9960

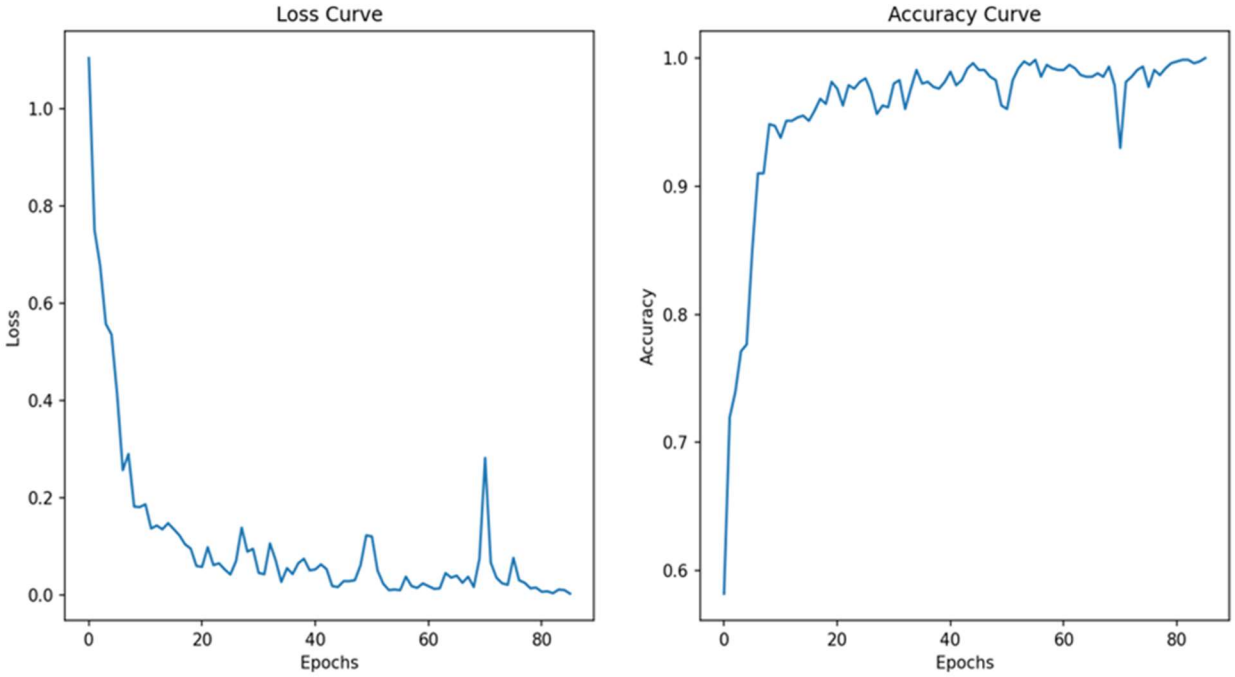
Auto-evaluation of model with test data:
loss = 0.2971
accuracy = 0.8833

Manual Evaluation of our model with test data:
correct: 53, wrong: 7
accuracy = 0.8833

Interpretation of Results:
Using average pooling instead of maximum pooling has caused prediction accuracy to drop this time (i.e. one more wrongly categorized test image as compared to the previous model), but loss value has decreased with the current setting. The drop in accuracy may also be caused by the decreased epoch, which may have prevented the model from reaching an optimal training level.

Experiment 4:
Actions: increased epoch size (150)

Model Sequence:
Results:



Epoch 86/150
24/24 [==============================] - 8s 347ms/step - loss: 0.0035 - accuracy: 1.0000

Auto-evaluation of model with test data:
loss = 0.2533
accuracy = 0.9333

Manual Evaluation of our model with test data:
correct: 56, wrong: 4
accuracy = 0.9333

Interpretation of Results:
We first increased the number of epochs to see what are the possible effects of such a change. This time, the model was able to end training at an accuracy of 1.0, and we observed a very high testing accuracy along with a low loss.

Experiment 5:

Actions:

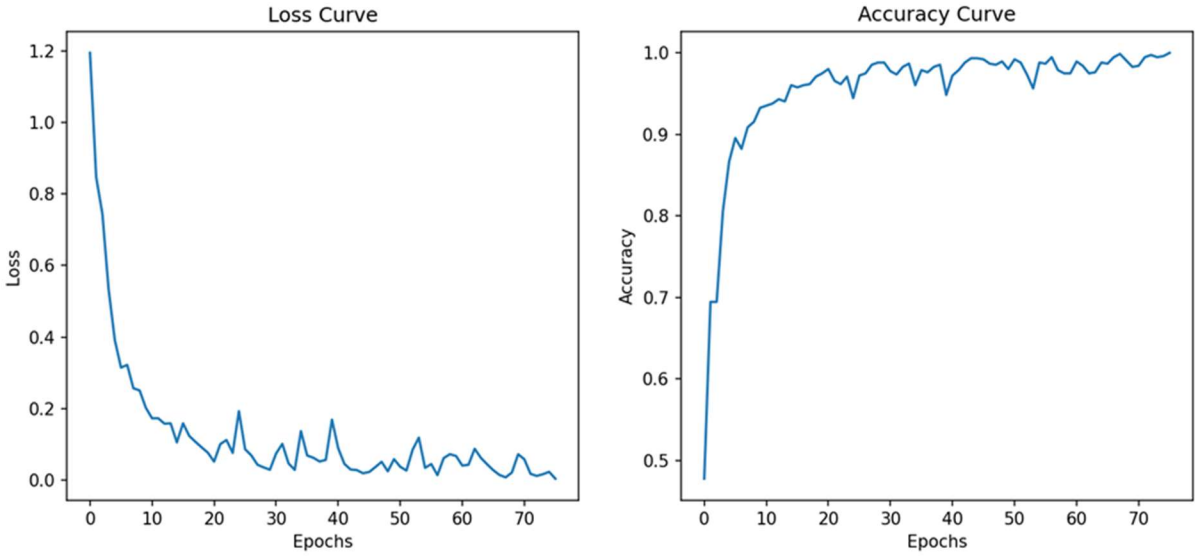
a. changed filter values for convolutional layers

b. changed callback value to accuracy = 1.0

Model Sequence (Changes):

```
model.add(tf.keras.layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu',
i          nput_shape=(length,length,3)))
model.add(tf.keras.layers.AveragePooling2D())
model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu'))
model.add(tf.keras.layers.AveragePooling2D())
model.add(tf.keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(tf.keras.layers.AveragePooling2D())
model.add(tf.keras.layers.Conv2D(filters=128, kernel_size=(3, 3), activation='relu'))
model.add(tf.keras.layers.AveragePooling2D())
```

Results:



Epoch 76/150

24/24 [==============================] - 7s 291ms/step - loss: 0.0032 - accuracy: 1.0000

Auto-evaluation of model with test data:

loss = 0.1709

accuracy = 0.9167

Manual Evaluation of our model with test data:

correct: 55, wrong: 5

accuracy = 0.9167

Interpretation of Results:
The change in filter values did not help with improving prediction accuracy. Instead, accuracy dropped but we observed a further decrease in loss value.

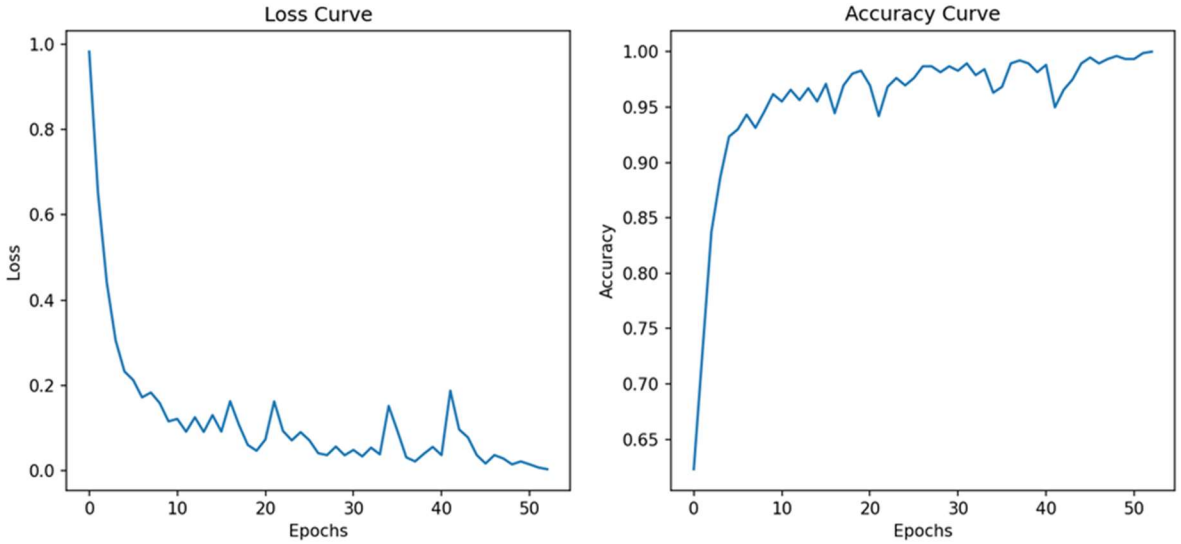
Experiment 6:

Actions: changed number of nodes in Dense layer

Model sequence (changes):

```
model.add(tf.keras.layers.Dense(units=128, activation='relu'))
model.add(tf.keras.layers.Dense(units=64, activation='relu'))
```

Results:



Epoch 53/150

24/24 [==============================] - 7s 296ms/step - loss: 0.0029 - accuracy: 1.0000

Auto-evaluation of model with test data:

loss = 0.1814

accuracy = 0.9333

Manual Evaluation of our model with test data:

correct: 56, wrong: 4

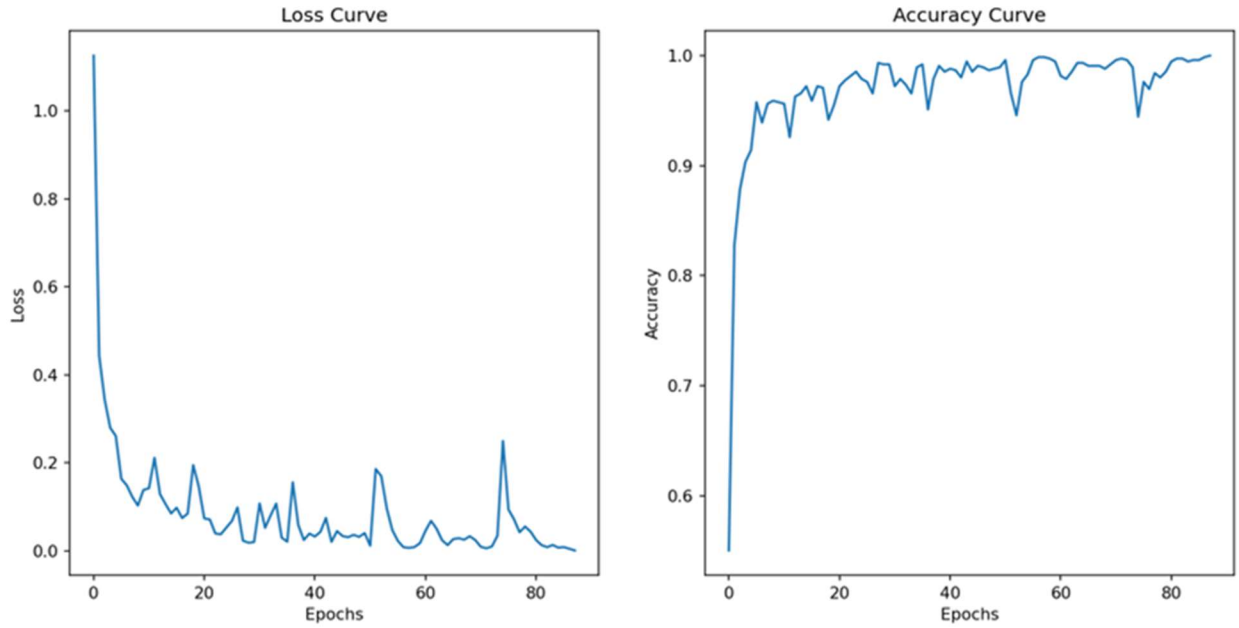
accuracy = 0.9333

Interpretation of Results:

Adding more nodes to the first Dense layer had a beneficial effect on prediction accuracy. The resulting loss value is also very low.

Experiment 7:
Actions: added normalization layer [BatchNormalization(axis=1)]

Results:



Epoch 88/150
24/24 [==============================] - 7s 273ms/step - loss: 0.0011 - accuracy: 1.0000

Auto-evaluation of model with test data:
loss = 0.2894
accuracy = 0.9333

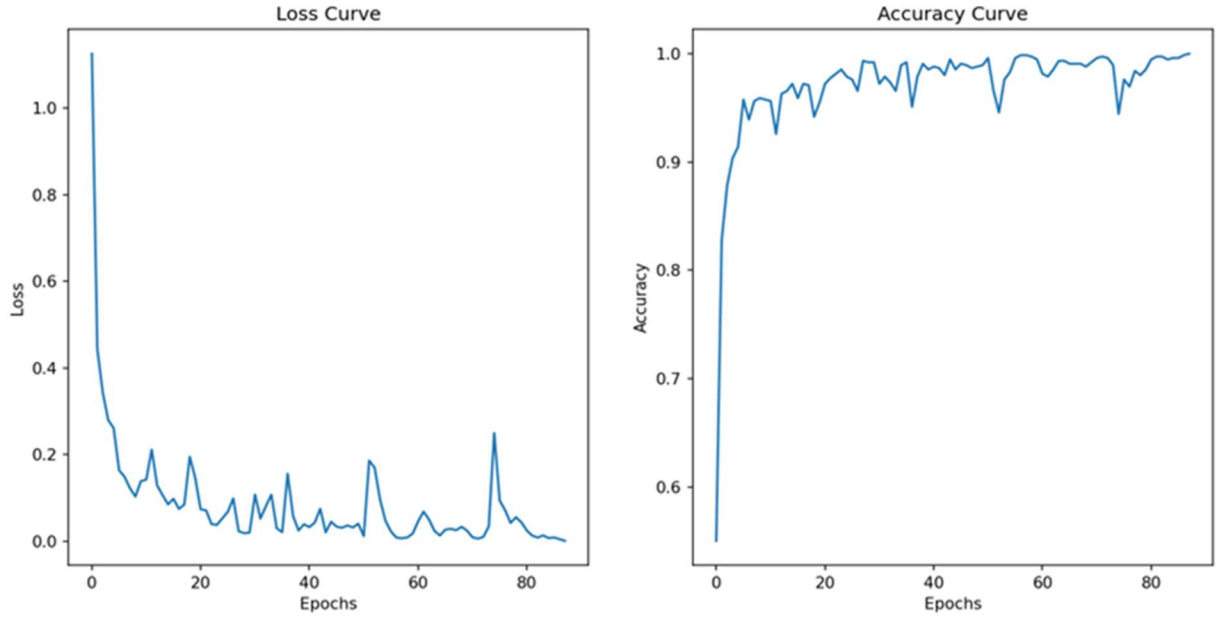
Manual Evaluation of our model with test data:
correct: 56, wrong: 4
accuracy = 0.9333

Interpretation of Results:
Adding a normalization layer does not seem to help improve the model's prediction ability.

Experiment 8:
Actions:

a. removed normalization layer
b. increased image size to 256 x 256

Results:



Epoch 72/150
24/24 [==============================] - 27s 1s/step - loss: 0.0033 - accuracy: 1.0000

Auto-evaluation of model with test data:
loss = 0.2654
accuracy = 0.9167

Manual Evaluation of our model with test data:
correct: 55, wrong: 5
accuracy = 0.9167

Interpretation of Results:
Increasing image size also did not seem to help with improving model prediction. This may suggest that a smaller image size is sufficient for the extraction of useful features for image categorization. In this case, the usage of increased image size is much more resource intensive with little benefits.