SA4110
Machine Learning
Course Assignment Project
Report

TEAM 4

Cheng Jun Long

Thune Htet Naing

Shi Xu

Hninn Ei Khaing

Vincent Lee

Contents

Introduction

We were given 4 different groups of fruit pictures (mainly Apples, Bananas, Oranges and Mixed Fruits). We have created the Image Classifier (CNN model) to train these pictures. In this report we have tried various Experiments to train different models to optimize to produce a model with decent accuracy and loss.

In each Experiment, we document the different methods and modifications we have done to improve on the accuracy and loss.

Experiment 1 (Initial tryout):

The main aim of this experiment is to observe how the codes run and how the model is created and tested

Some of the key features that we have utilized in this experiment is as follows:-

1) All the training data are kept, no change to the training data
2) Image Dimension is resize to (250, 250, 3)
3) Of the 240 training data (192 Dataset were used to train the model, 48 Dataset were used to validate the model)

Our model were set to be as follows (snippets code of the model):-

```
#Create an empty neural network
model_s = tf.keras.Sequential()

#Add the first convolution layer indicating the input shape(resized image) and image scale(3 for RGB and 1 for grey scale)
#(3,3) kernel size filter is used for convolution
#activation function 'sigmoid' restricts the input values between 0 and 1
model_s.add(tf.keras.layers.Conv2D(filters=32, kernel_size=(3, 3),
    activation='relu', input_shape=(250, 250, 3)))

#Flatten the data from 2D to 1D before passing to the dense layer
```

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

#Add an output layer with softmax activation function to get probability distributed results.
#4 set of outputs correspond to 4 fruit labels
model_s.add(tf.keras.layers.Dense(units = 4, activation='softmax'))

#Add backpropagation(optimization)
#choose loss function as 'categorical_crossentropy' when softmax is used for output layer
#choose 'accuracy' as metrics to count the number of times neural network has predicted correct.
model_s.compile(loss='categorical_crossentropy',
        optimizer='adam',
        metrics=['accuracy'])

#print model summary
model_s.summary()
```

Model Architecture for Experiment 1

Model: "sequential"

_____

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 248, 248, 32)	896
flatten (Flatten)	(None, 1968128)	0
dense (Dense)	(None, 128)	251920512
dense_1 (Dense)	(None, 4)	516

Total params: 251,921,924

Trainable params: 251,921,924

Non-trainable params: 0

_____

```
hist_n = model_s.fit(x_train, y_train, epochs=10, validation_data= (x_val, y_val))


Evaluation = model_s.evaluate(x_test, y_test)
print("score =", Evaluation)
```

Epoch 1/10

6/6 [==============================] - 22s 3s/step - loss: 225.8932 - accuracy: 0.2812 - val_loss: 82.6253 - val_accuracy: 0.3750

Epoch 2/10

6/6 [==============================] - 13s 2s/step - loss: 36.5347 - accuracy: 0.6042 - val_loss: 23.8730 - val_accuracy: 0.4792

Epoch 3/10

6/6 [==============================] - 14s 2s/step - loss: 12.2459 - accuracy: 0.6562 - val_loss: 10.8815 - val_accuracy: 0.7917

Epoch 4/10

6/6 [==============================] - 14s 2s/step - loss: 7.0763 - accuracy: 0.7708 - val_loss: 7.9246 - val_accuracy: 0.7500

Epoch 5/10

6/6 [==============================] - 14s 2s/step - loss: 3.0876 - accuracy: 0.8594 - val_loss: 7.0530 - val_accuracy: 0.7917

Epoch 6/10

6/6 [==============================] - 14s 2s/step - loss: 1.3492 - accuracy: 0.9271 - val_loss: 6.5511 - val_accuracy: 0.7292

Epoch 7/10

6/6 [==============================] - 14s 2s/step - loss: 0.6956 - accuracy: 0.9323 - val_loss: 7.5030 - val_accuracy: 0.7708

Epoch 8/10

6/6 [==============================] - 14s 2s/step - loss: 0.2728 - accuracy: 0.9792 - val_loss: 6.5584 - val_accuracy: 0.7708

Epoch 9/10

6/6 [==============================] - 14s 2s/step - loss: 0.1107 - accuracy: 0.9792 - val_loss: 6.0151 - val_accuracy: 0.7708

Epoch 10/10

6/6 [==============================] - 14s 2s/step - loss: 0.0524 - accuracy: 0.9844 - val_loss: 5.6136 - val_accuracy: 0.7500

loss: 2.5448 - accuracy: 0.8167

score = [2.544773817062378, 0.8166666626930237]

The training model seems to be increasing in accuracy and the loss is getting better, however, it is noted that the validation accuracy and loss is rather consistent throughout each epoch

(Findings, the model seems to be memorizing the data, causing the validation data to produce constant bad accuracy and loss, due to the memorizing of data, causing the model to be overfitted.)

- Upcoming Experiment, we can try to look at dropping some parts of the image to avoid 'memorizing'.

Experiment 2 (Adding another layer in model: Drop-out layer):

The Data is kept the same at the moment (same as Experiment 1)

Some of the key features that we have utilized in this experiment is as follows:-

1. All the training data are kept, no change to the training data
2. Image Dimension is resize to (250, 250, 3)
3. Of the 240 training data (192 Dataset were used to train the model, 48 Dataset were used to validate the model)

```
#Create an empty neural network
model_s = tf.keras.Sequential()

#Add the first convolution layer indicating the input shape(resized image) and image scale(3 for RGB and 1 for grey scale)
#(3,3) kernel size filter is used for convolution
#activation function 'sigmoid' restricts the input values between 0 and 1
model_s.add(tf.keras.layers.Conv2D(filters=32, kernel_size=(3, 3),
   activation='relu', input_shape=(250, 250, 3)))

#Flatten the data from 2D to 1D before passing to the dense layer
model_s.add(tf.keras.layers.Flatten())
model_s.add(tf.keras.layers.Dense(units = 128, activation='relu'))

#Dropout fraction of neurons from the network in hidden layers back and forth to avoid strong dependencies(overfitting)
model_s.add(tf.keras.layers.Dropout(0.5))

#Add an output layer with softmax activation function to get probability distributed results.
#4 set of outputs correspond to 4 fruit labels
model_s.add(tf.keras.layers.Dense(units = 4, activation='softmax'))

#Add backpropagation(optimization)
#choose loss function as 'categorical_crossentropy' when softmax is used for output layer
```

```
#choose 'accuracy' as metrics to count the number of times neural network has predicted correct.
model_s.compile(loss='categorical_crossentropy',
        optimizer='adam',
        metrics=['accuracy'])

#print model summary
model_s.summary()
```

Over here we added in another layer, the dropout layer to remove any over-fitting.

The model architecture has been updated

Model: "sequential"

_____

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 248, 248, 32)	896
flatten (Flatten)	(None, 1968128)	0
dense (Dense)	(None, 128)	251920512
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 4)	516

===

Total params: 251,921,924

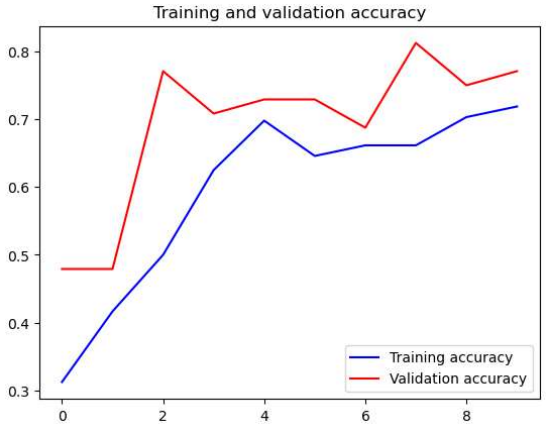
Trainable params: 251,921,924

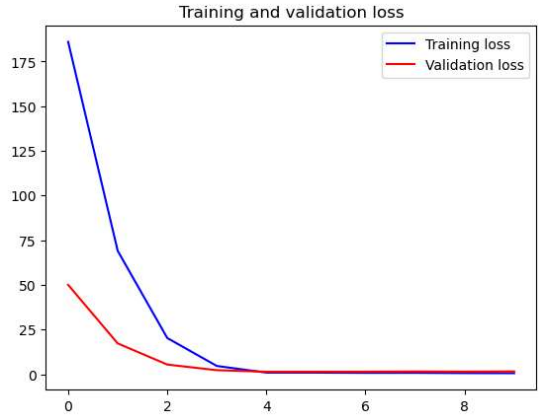
Non-trainable params: 0

_____

loss: 1.3142 - accuracy: 0.8000

score = [1.3142117261886597, 0.800000011920929]





Upon the addition of the extra drop-out layer, it can be seen that the accuracy of the validation set has also improved. This means that the over-fitting data has been improved.

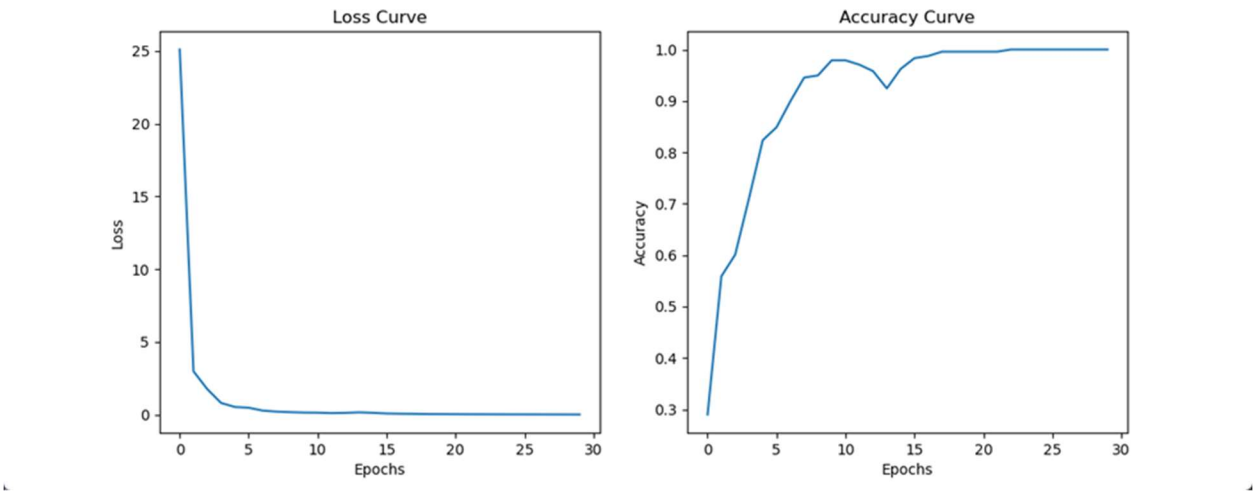
However, further exploration of optimisation is needed to improve on the accuracy and loss of data.

Experiment 3 (Selection of Activation Function):

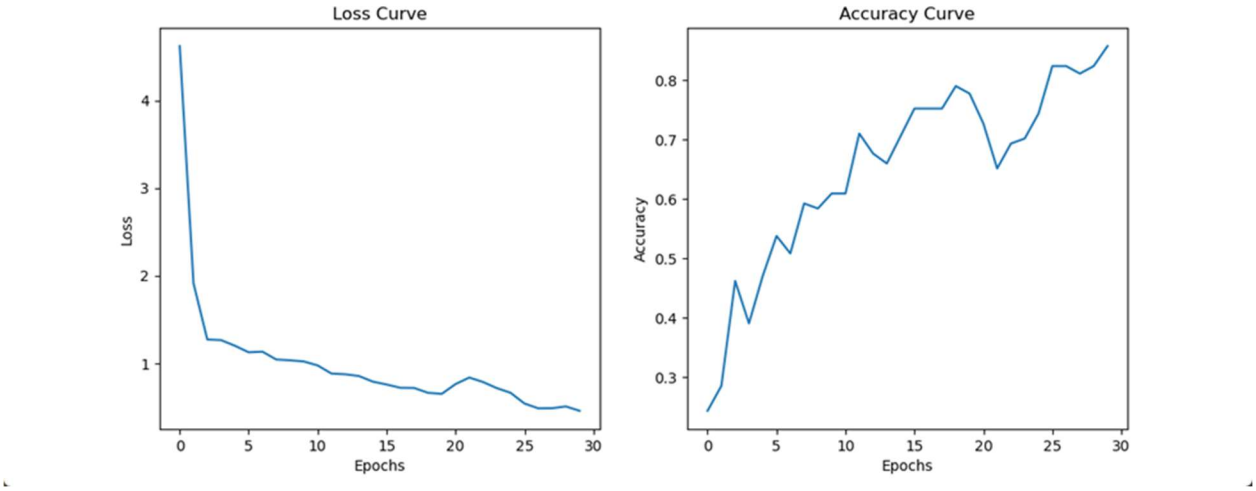
In this experiment, we explore the change of Activation function to sigmoid to investigate if there will be an change to the accuracy and loss of the model trained

```
def create_model(target_width, target_height):

  model = tf.keras.Sequential()



  model.add(tf.keras.layers.Conv2D(filters=32,   # 32 fiilters means 32 features

                        kernel_size=(3, 3), activation='sigmoid', input_shape=(target_width, target_height, 3)))

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

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

  # add a dropout layer to improve overfitting

  # randomly discarge 50% nodes during trainning

  # model.add(tf.keras.layers.Dropout(0.5))

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

  model.compile(loss='categorical_crossentropy', optimizer='adam',

          metrics=['accuracy'])



  return model
```

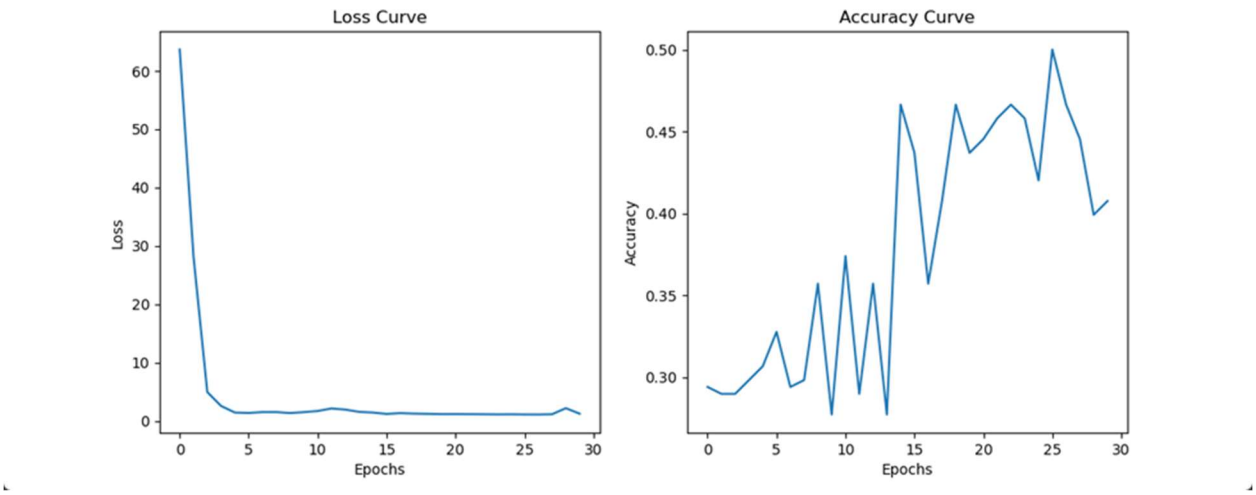
Tanh:



Softmax:



Sigmoid:



Relu: loss_accuracy= [1.3534438610076904, 0.8666666746139526]

Tanh: loss_accuracy= [0.6584991216659546, 0.8333333134651184]

Softmax: loss_accuracy= [0.5929499268531799, 0.800000011920929]

Sigmoid: loss_accuracy= [1.1716198921203613, 0.550000011920929]

Sigmoid is not suitable for CNN model because it shows unstable fluctuations in accuracy. That may cause gradient vanishing problems when we have too many layers.

The accuracy of ReLu and Tanh are highest. The loss of Tanh and Softmax are lower than ReLu may be because those two activation functions reserve negative value while ReLu discharges negative value. Also a main advantage of ReLu is that it will help to remove gradient vanishing problems and is computational cheaper to run as it zero-rised output which is <0.

In our case, it seems to be okay to use ReLu.

Experiment 4 (Resizing the images):

In this experiment, we tried to investigate the effect of resizing the images, whether there is an increase in the accuracy and reduce the loss.

4.1: Control set-up (The size we are using to do our comparison)

```
target_width = 80

target_height = 80
```

Relu: loss_accuracy= [1.1727714538574219, 0.8833333253860474]

4.2: Decrease the size

```
target_width = 68

target_height = 68
```

Relu: loss_accuracy= [1.3534438610076904, 0.8666666746139526]

Results:

Loss increases! But accuracy also decreases.

As it is fairly obvious that the decrease in size, increases the loss and also decreases the accuracy.

Thus we will be increasing the size for further experiment

4.3: Increase the size to 100

```
target_width = 100

target_height = 100
```

Running time of each epoch : 3s

Relu: loss_accuracy= [0.8991466760635376, 0.8833333253860474]

Loss reduces while accuracy increases.

It seems like our above hypothesis is true as the size increases, the loss has also decreased.

We will investigate to see if the relationship is true for further increased in size

4.4: Increase size to 120

```
target_width = 120

target_height = 120
```

Running time of each epoch : 5s

Relu: loss_accuracy= [1.0991897583007812, 0.8999999761581421]

Compared to the image size of 100, although the accuracy has improved, the loss seemed to be going upwards too. We will try to increase the size again to confirm this.
Also, something worth nothing is the running time is increasing as well

4.5: Increase size to 150

```
target_width = 150

target_height = 150
```

Running time of each epoch : 7s

Relu: loss_accuracy= [1.5020828247070312, 0.9166666865348816]

Similarly to image size of 120, the accuracy has improved, but the loss is also getting larger.

Summary :

Image Size	Loss	Accuracy	Verdict
80	1.17	0.88	Control
68	1.35	0.87	
100	0.90	0.88	Best Results
120	1.10	0.90	
150	1.50	0.92	

After changing size, we understand that changing the size of the images can certainly improve the accuracy and loss, however, do note that there is a 'sweet spot' that yields the best outcome.

Also, the increase in the image size will increase the process time.

Experiment 5 (Image Augmentation):

In this experiment we explore the use of image augmentation

To do this, we import the ImageDataGenerator.

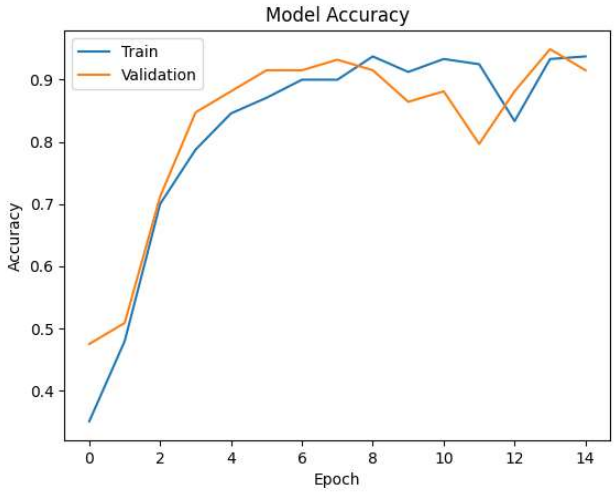
With the ImageDataGenerator, the library allows us to reshape our images (as per the snippet below):-

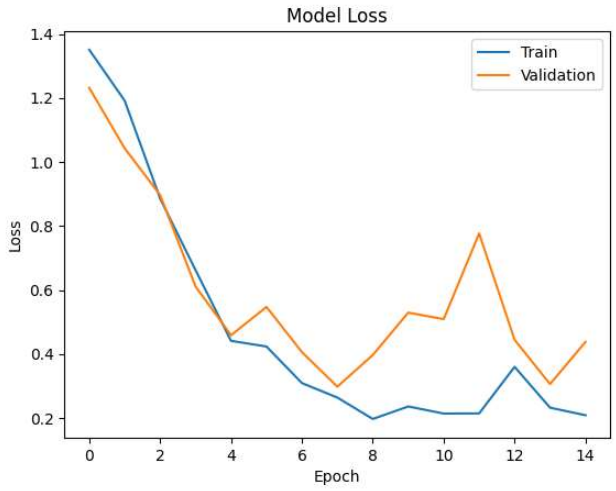
```
from keras.preprocessing.image import ImageDataGenerator
from PIL import Image
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, Activation
```

```
train_path= 'C:/Users/65972/Desktop/TERM 2 COURSE MATERIAL/09 Team Project/train'
test_path = 'C:/Users/65972/Desktop/TERM 2 COURSE MATERIAL/09 Team Project/test'

train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range = 0.3,
                                   horizontal_flip = True,
                                   vertical_flip = False,
                                   zoom_range = 0.3)
test_datagen = ImageDataGenerator(rescale= 1./255)
train_generator = train_datagen.flow_from_directory(train_path,
                                                    target_size = (100,100),
                                                    batch_size = 32,
                                                    color_mode = 'rgb',
                                                    class_mode = 'categorical')
test_generator = test_datagen.flow_from_directory(test_path,
                                                  target_size = (100,100),
                                                  batch_size = 32,
                                                  color_mode = 'rgb',
                                                  class_mode = 'categorical')
```

```
test_accuracy = model.evaluate(test_generator)
```

loss: 0.4378 - accuracy: 0.9153





We trial with the use of Data Augmentation and the loss is about 0.44 and accuracy is about 0.915 which we feel is quite good improvement from our previous experiments.

Experiment 6 (adding More data in training set):

In this experiment, we have increased our training data by increasing more 'Mixed fruits' pictures for training.

Furthermore we have also explored various layers to run (the code snippets shown below is one that we have achieve a better accuracy and the least loss)

```
def create_model():
    model = Sequential()
    model.add(Conv2D(128,3, activation='relu', input_shape=(100, 100, 3)))
    model.add(MaxPooling2D())
    model.add(Conv2D(64,3, activation='relu'))
    model.add(MaxPooling2D())
    model.add(Conv2D(32,3, activation='relu'))
    model.add(MaxPooling2D())
    model.add(Flatten())
    model.add(Dense(256, activation='relu'))
    model.add(Dense(4, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model

model = create_model()
```

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    vertical_flip=False
)
```

```
test_datagen = ImageDataGenerator(rescale= 1./255)
train_generator = train_datagen.flow_from_directory(train_path,
                                    target_size = (100,100),
                                    batch_size = 32,
                                    color_mode = 'rgb',
                                    class_mode = 'categorical')
test_generator = test_datagen.flow_from_directory(test_path,
                                    target_size = (100,100),
                                    batch_size = 32,
                                    color_mode = 'rgb',
                                    class_mode = 'categorical')
```

Found 293 images belonging to 4 classes.

Found 60 images belonging to 4 classes.

From results above, 293 images were used for training the model as compared to 240 previously. The extra 53 images are mainly for mixed fruits.

```
# Apply learning rate reduction
learning_rate_reduction = ReduceLROnPlateau(monitor='val_loss', patience=5, verbose=1, factor=0.5,
min_lr=0.00001)

# Fit the model with training data
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.n // train_generator.batch_size,
    epochs=20,
    validation_data=test_generator,
    validation_steps=test_generator.n // test_generator.batch_size,
    callbacks=[learning_rate_reduction]
)
```

```
# Train the model
# history = model.fit(train_generator,epochs=15 ,validation_data=(test_generator))
```

```
Epoch 1/20
9/9 [==============================] - 4s 336ms/step - loss: 1.4289 - accuracy: 0.2567 - val_loss: 1.3464
- val_accuracy: 0.3750 - lr: 0.0010
Epoch 2/20
9/9 [==============================] - 3s 324ms/step - loss: 1.3726 - accuracy: 0.2759 - val_loss: 1.2427
- val_accuracy: 0.5938 - lr: 0.0010
...
Epoch 19/20
9/9 [==============================] - 4s 448ms/step - loss: 0.4121 - accuracy: 0.8429 - val_loss: 0.6483
- val_accuracy: 0.8750 - lr: 0.0010
Epoch 20/20
9/9 [==============================] - 4s 468ms/step - loss: 0.4351 - accuracy: 0.8314 - val_loss: 0.2724
- val_accuracy: 0.9062 - lr: 0.0010
```

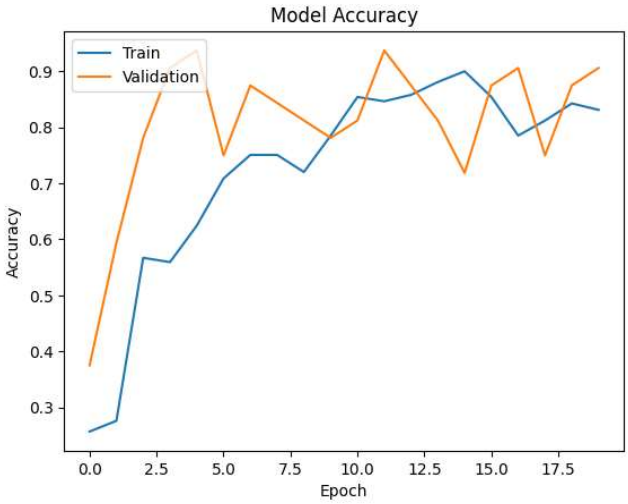
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings…

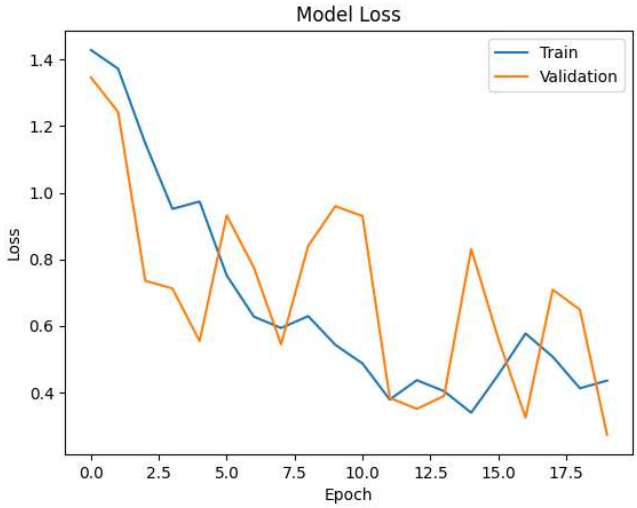
In the code snippet above, we have explored ideas to reduce the learning rate.

In a nutshell, these codes help to dynamically alter the learning rate during training. The model will train and the parameters are being monitored to measure if there are any improvements. The learning rates are altered to help to improve on the model coverage which can also help to reduce the loss.

```
test_accuracy = model.evaluate(test_generator)
```

```
2/2 [==============================] - 0s 122ms/step - loss: 0.2997 - accuracy: 0.8833
```





From our results, we manage to reduce the loss to about 0.3, with the accuracy of about 0.88 which we think is quite alright.

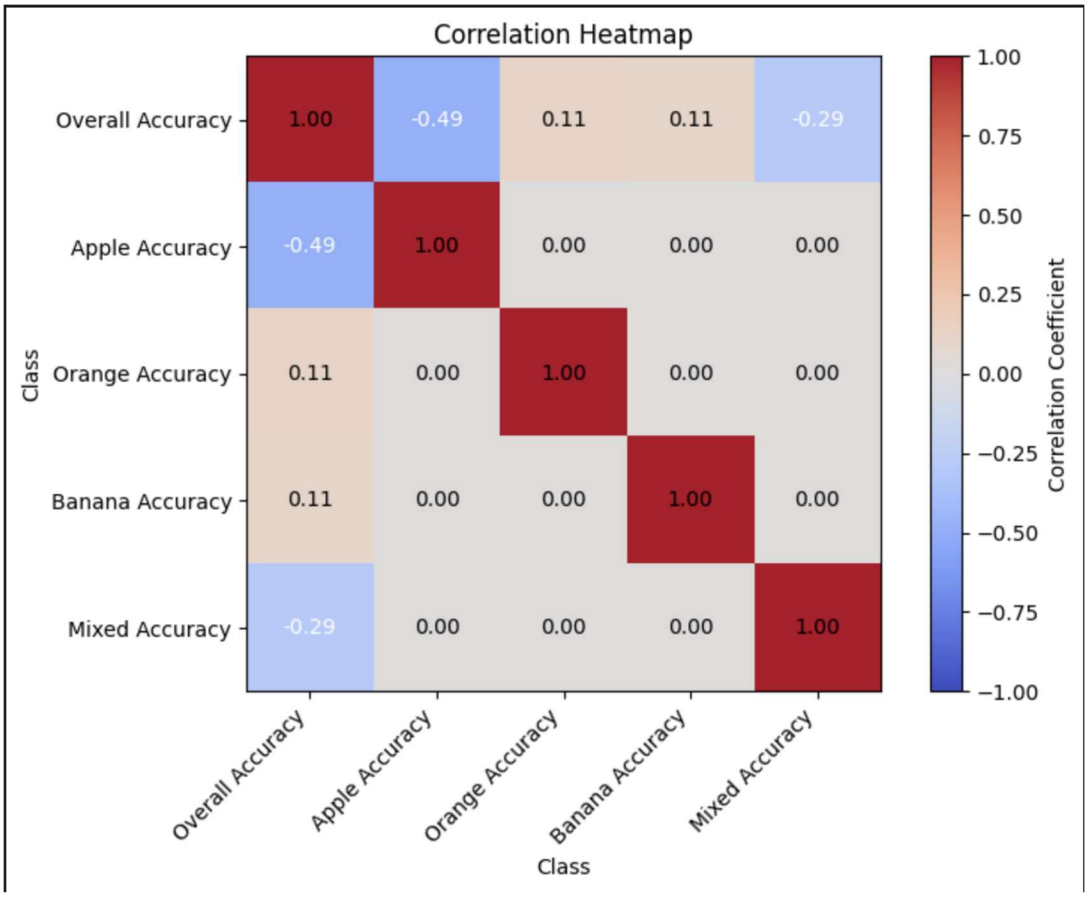
Additional Experiment

We have also explored the use of various diagrams to analyze our data.

In this analysis we looked in to the use of Heat Map to generate the accuracy against each trained data.

```
labels = ['Overall ', 'Apple ', 'Orange ', 'Banana ', 'Mixed ']

# Create the correlation matrix
correlation_matrix = np.array([[1.0, correlation_apple, correlation_orange, correlation_banana,
correlation_mixed],
                               [correlation_apple, 1.0, 0.0, 0.0, 0.0],
                               [correlation_orange, 0.0, 1.0, 0.0, 0.0],
                               [correlation_banana, 0.0, 0.0, 1.0, 0.0],
                               [correlation_mixed, 0.0, 0.0, 0.0, 1.0]])

# Create the heatmap
plt.figure(figsize=(10, 12))
plt.imshow(correlation_matrix, cmap='GnBu', vmin=-1, vmax=1)

# Add colorbar
cbar = plt.colorbar()
cbar.set_label('Correlation Coefficient')

# Set the tick labels and axis labels
plt.xticks(range(len(labels)), labels, rotation=45, ha='right')
plt.yticks(range(len(labels)), labels)
plt.xlabel('Class')
plt.ylabel('Class')

# Add text annotations in each cell
for i in range(len(labels)):
    for j in range(len(labels)):
        plt.text(j, i, f'{correlation_matrix[i, j]:.2f}', ha='center', va='center', color='white' if correlation_matrix[i, j]
< 0 else 'black')
```

```
# Set the title
plt.title('Correlation Heatmap')

# Show the plot
plt.tight_layout()
plt.show()
```



Conclusion

In conclusion, the training dataset seems to be a little too small to produce accurate readings, thus we utilized Data augmentation to increase the training data. Also, we found that the dataset was not well spread (ie the number of training data for Apples, Bananas, Oranges and Mixed fruits were not of similar number), thus we added in some Mixed fruits pictures to enhance the training. Also, we have relabelled the wrong dataset.

With the dataset prepared, we then move on to creating the model where we set to the following layers:

1. Convolution Layer with 128 filters, activation relu
2. MaxPooling Layer
3. Convolution Layer with 64 filters, activation relu
4. MaxPooling Layer
5. Convolution Layer with 32 filters, activation relu
6. MaxPooling Layer
7. Flatten Layer
8. Dense Layer with 256 filters, activation relu
9. Output Layer (Dense Layer = 4 filters, Activation = softmax)

The loss were measured using the Categorical crossentropy and
Optimiser = Adam
Metrics = Accuracy

Furthermore, during our training of the data, we have included the code to alter the learning rate to help to reduce the loss dynamically, all in all we feel that Experiment 6 produces a decent model for testing.
However, we feel that, even when the model is developed, we should test out the model and try to optimize it even further with various datasets. Lastly, it is worth noting that every machine trains the model differently and it seems like optimisation may differ from machine to machine, thus for the optimisation of the model, multiple PCs could be used to ensure the results are similar before rolling out for use.