Machine Learning CA-2
Image Classifier

Team 1

Members:

Gao Yumengdie

Kaung Htet Win

Lin Lin Thant

Park Jun Suk

Sharmaine Cheng

Akshaya Hasinee

Classifying the dataset

```python
#for i = 0 to dictionary value
for i in range(img_dict[key]):
```

```python
    #adding to y_train
    if y_set is None:

        #check key for classification of image
        if key == "apple":
            y_set = np.array([[1, 0, 0, 0]])

        if key == "banana":
            y_set = np.array([[0, 1, 0, 0]])

        if key == "orange":
            y_set = np.array([[0, 0, 1, 0]])

        if key == "mixed":
            y_set = np.array([[0, 0, 0, 1]])

    else:
        #check key
        if key == "apple":
            y_set = np.concatenate((y_set, [[1, 0, 0, 0]]))

        if key == "banana":
            y_set = np.concatenate((y_set, [[0, 1, 0, 0]]))

        if key == "orange":
```

Here we use dictionary to upload dataset

Classifying the dataset

- for each key in the dictionary parameter
  Apple, oranges, bananas, mixed becomes key and for values we use one hot encoding.

Step followed in both models

- Upload and classify the dataset for train and test
- Set up a model with required layers
- Compile the model mentioning loss and optimiser type to be used.
- Fit the rain data
- Evaluate the model by using the test dataset
- Plot the accuracy

Model 1

Data Used:
The provided data set

Data augmentation:
no

Number of layers:

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 62, 62, 32)        896

 conv2d_3 (Conv2D)           (None, 58, 58, 32)        25632

 flatten_1 (Flatten)         (None, 107648)            0

 dense_1 (Dense)             (None, 4)                 430596

=================================================================
Total params: 457,124
Trainable params: 457,124
Non-trainable params: 0
```

- 2 Convolutional Layers
- 1 Flatten Layers
- 1 Dense Output Layers

Fitting the model:

```
:
hist = model.fit(x=x_train/255, y=y_train, epochs=15)
```

```
Epoch 1/15
8/8 [==============================] - 3s 72ms/step - loss: 2.9215 - accuracy: 0.2208
Epoch 2/15
8/8 [==============================] - 0s 33ms/step - loss: 1.3353 - accuracy: 0.3333
```

When fitting the model, all the values of the x_train dataset are divided by 255 to normalize the values to [0, 1], making training the model more efficient.

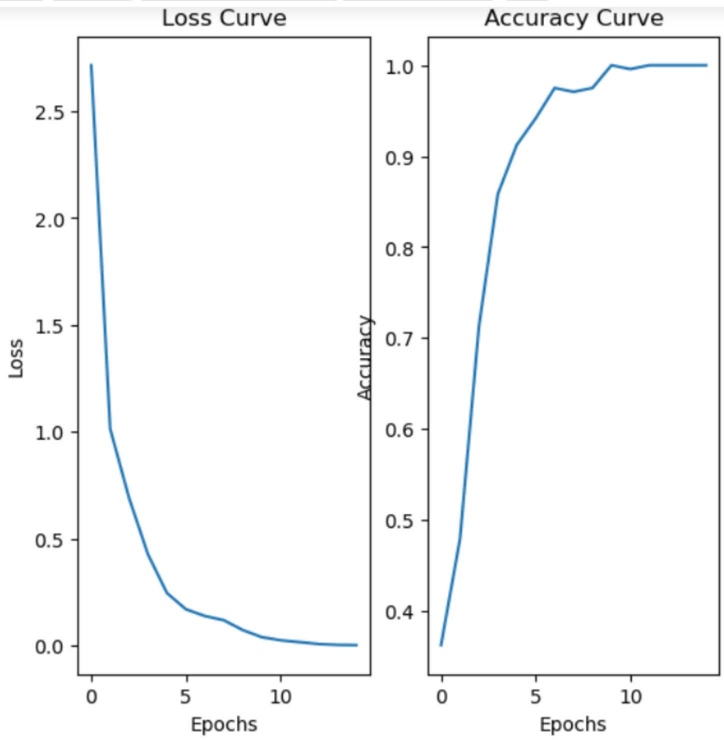
Accuracy:

in Model1- Achieved accuracy :0.866 and Loss:0.651

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
loss, accuracy = model.evaluate(x = x_test/255, y = y_test)
print("Loss: ", loss, "\tAccuracy: ", accuracy)

2/2 [==============================] - 1s 372ms/step - loss: 0.6510 - accuracy: 0.8667
Loss:  0.6510265469551086       Accuracy:  0.8666666746139526
```

Plot:



Need for Model 2

In model 1, the team learnt:

- To process the given image dataset, and add required layers.
- That accuracy varies by changing the number of epochs.
- Our accuracy was 86.6, so we decided to improve our accuracy by creating our new model 2.

Model 2

Data used:

Given data set

Data augmentation:

```
img_tiny = img_rgb.resize((resol, resol))
train_label.append([img_tiny, one_hot[fruit]])
```

Train_label, puts both image object and label into a list so that we can shuffle.

```
random.shuffle(train_label)
```

Random shuffle helps in shuffling image objects.
To check if the shuffle has happened, we compare images at particular position before shuffling, and after shuffling.

Image at position 0,100 before shuffling (Model 1)

```
#viewing images
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(12, 6))

ax[0].imshow(x_train[0])
ax[0].set_title(y_train[0])
ax[1].imshow(x_train[100])
ax[1].set_title(y_train[100])

plt.show()
```

```
C:\Users\aksha\miniconda3\envs\tf\lib\site-packages\matplotlib\text.py:1223: FutureWarning: elementwise comparison failed; retu
rning scalar instead, but in the future will perform elementwise comparison
  if s != self._text:
```

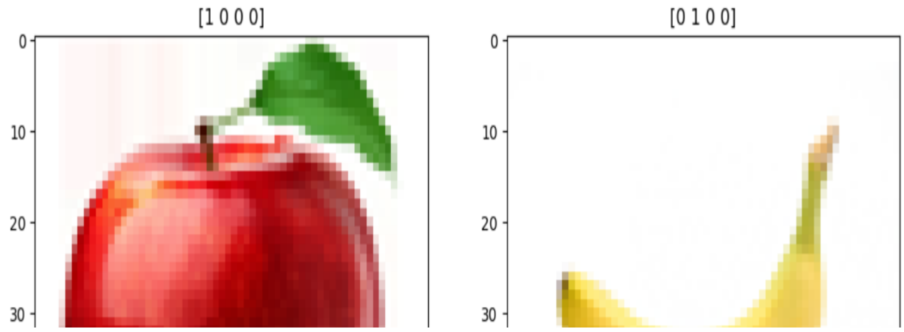
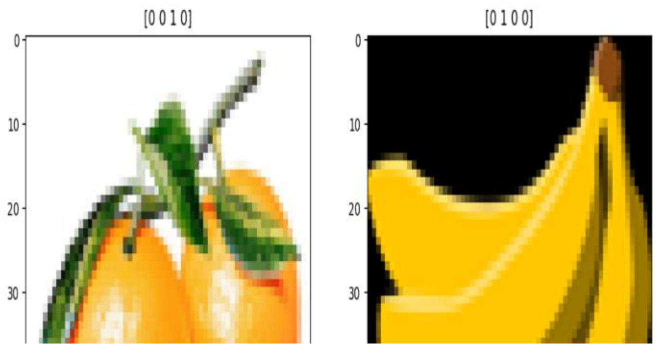


Image at position 0,100 after shuffling (Model 2)

```
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(12, 6))

ax[0].imshow(x_train[0])
ax[0].set_title(y_train[0])
ax[1].imshow(x_train[100])
ax[1].set_title(y_train[100])

plt.show()
```



Number of layers:

Model1, totally had 4 layers all together, here numbers of layers are increased.

- 4Convolutional Layers
- 4 Max Pooling Layers
- 1 Flatten Layers
- 1 Dropout Layer
- 3 Dense Output Layers

```
 ------------------------------------------------------------
 conv2d_135 (Conv2D)         (None, 64, 64, 30)        840

 max_pooling2d_130 (MaxPooli  (None, 32, 32, 30)       0
 ng2D)

 conv2d_136 (Conv2D)         (None, 32, 32, 30)        8130

 max_pooling2d_131 (MaxPooli  (None, 16, 16, 30)       0
 ng2D)

 conv2d_137 (Conv2D)         (None, 16, 16, 30)        8130

 max_pooling2d_132 (MaxPooli  (None, 8, 8, 30)         0
 ng2D)

 conv2d_138 (Conv2D)         (None, 8, 8, 30)          8130

 max_pooling2d_133 (MaxPooli  (None, 4, 4, 30)         0
 ng2D)

 flatten_34 (Flatten)        (None, 480)               0

 dense_95 (Dense)            (None, 200)               96200

 dropout_58 (Dropout)        (None, 200)               0

 dense_96 (Dense)            (None, 100)               20100

 dense_97 (Dense)            (None, 4)                 404

============================================================
Total params: 141,934
```

Fitting the model:

```
hist = modelmod.fit(x_train/255, y_train, batch_size=30, epochs=25, shuffle=True, verbose=2)
```

```
Epoch 1/25
8/8 - 1s - loss: 1.3654 - accuracy: 0.3042 - 681ms/epoch - 85ms/step
Epoch 2/25
8/8 - 0s - loss: 1.2952 - accuracy: 0.3833 - 119ms/epoch - 15ms/step
```

- The epochs are kept at 25, as we are able to achieve 91.6%
- in it.
- Increasing epochs 30 and above, accuracy dropped.
- And decreasing
- Epochs to 20 or below lead to a higher loss.

The dataset was spilt into batches by declaring a batch size

Accuracy:

Thus, in Model2- Achieved accuracy :0.916 and loss:0.680

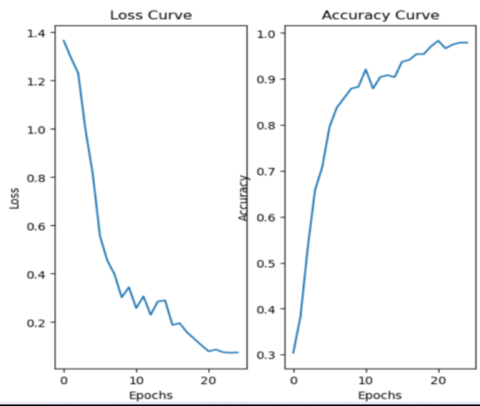
```
loss, accuracy = modelmod.evaluate(x=x_test/255, y=y_test)

print('loss =', loss)
print('accuracy =', accuracy)
```

```
apple count: 19
banana count: 18
orange count: 18
mixed count: 5
2/2 [==============================] - 0s 9ms/step - loss: 0.6809 - accuracy: 0.9167
loss = 0.6808925867080688
```

Plot:

```
plt.show()
```



Conclusion:

Thus, in this CA we learnt to work with image dataset, and have got the idea of improving accuracy.
The team managed to improve accuracy 86.6 to 91.66, by shuffling and adding more layers.