SA4110 Team Project

Convolutional Neural Network

Image Classifier

Prepared by: SA56 Team 1

26 May 2023

Team Members:

Name	Matriculation ID
Aye Thi Han	A0269553R
Reinardus Channing	A0249486L
Cheong Yi	A0154251N
Muhammad Feroz bin Jamalullah	A0269513Y
Jess Lee Shi Ying	A0190083J
Liu Jiayi	A0269514X
Hla Yamin Aye	A0269548J

Experiment Documentation Report

This document contains a summary of the experiments performed to improve model performance and accuracy as part of the Image Classifier (CNN) developed to classify images of fruits. This report also contains relevant plots and code snippets to summarize the details of the experiments and their impacts on model accuracy.

The task was to develop a CNN model that can accurately classify images of fruits. The provided dataset contains 300 images that fall within 4 classes:

- 'apple' - Apple only
- 'orange' - Orange only
- 'banana' - Banana only
- 'mix' - Any mix of the above 3 fruits

The dataset provided was already split into a train and test folders with an 80/20 split. One point to note is that the image size for each sample is not standardized (the images are of different sizes) and different samples are provided for each class.

The first activity performed was to review the provided dataset to ensure there were no mislabelled images. Several mislabeled images were found in the training and test sets— as follows:

1. An image of mixed fruits which was wrongly labeled 'banana_61'. This was then re-labeled to 'mixed_61' before training the model.
2. A faux orange was wrongly labeled 'mixed_20'. This was re-labeled to 'orange_77' before training the model.
3. An unknown object was wrongly labeled 'banana_67'. This image was removed from the dataset.
4. An image of bananas with peaches was wrongly labeled 'banana_35. This image was cropped to only contain the banana before training the model.
5. A duplicate image was found in the training (orange_62) and test set ('orange_85'). The image was removed from the test set before training the model.

Data preparation

Before going into the machine learning stage, we must perform data preparation to pre-process the data so that it can be used for training the model.

We first go through each image in the training folder and perform a manual one-hot-encoding (OHE) of the image/class labels. We then take the image file, expressly convert to RGB format and re-size it to 28x28 pixels. We experimented with various image sizes (e.g. 28px/32px/50px/150px) and eventually went with 28x28px as there was no significant increase in model accuracy with larger image sizes.

```
'''
Prepare dataset
'''
def prep_data(path):

    y = [] # list to hold one-hot-encoded label

    for img in os.listdir(path): # loop through each image file in directory

        if img[0] == '.':   # skip hidden files
            continue

        y_ohe = label_img(img) # get the ohe label for the image

        # resize and convert to RGB
        img = np.array(Image.open('{}/{}'.format(path,
img)).convert('RGB').resize([IMG_SIZE, IMG_SIZE]))

        # append label and image to lists
        try:
            x = np.concatenate((x, img))
        except:
            x = img

        y.append(y_ohe)

    x = reshape(x) # reshape x array to required format (size, size, channel)
    y = np.array([i for i in y])

    return x , y
```

Model creation

We created a CNN model utilising Keras API and TensorFlow. The model uses an 'Adam' optimizer with the default learning rate and 'categorical cross-entropy' as the loss function, as the labels have already been one hot encoded.

We trialed adding different layers and tuning hyperparameters to get better accuracy results however no significant increase was observed. Therefore the below model summary was to be used as our final model.

Model Summary

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 20)        560

 average_pooling2d (AverageP  (None, 14, 14, 20)       0
 ooling2D)

 conv2d_1 (Conv2D)           (None, 12, 12, 40)        7240

 flatten (Flatten)           (None, 5760)              0

 dense (Dense)               (None, 100)               576100

 dropout (Dropout)           (None, 100)               0

 dense_1 (Dense)             (None, 4)                 404

=================================================================
Total params: 584,304
Trainable params: 584,304
Non-trainable params: 0
_________________________________________________________________
```

First Iteration – Baseline model

Before performing any additional processing on the data, we first evaluated the baseline model performance using the provided training and test dataset.

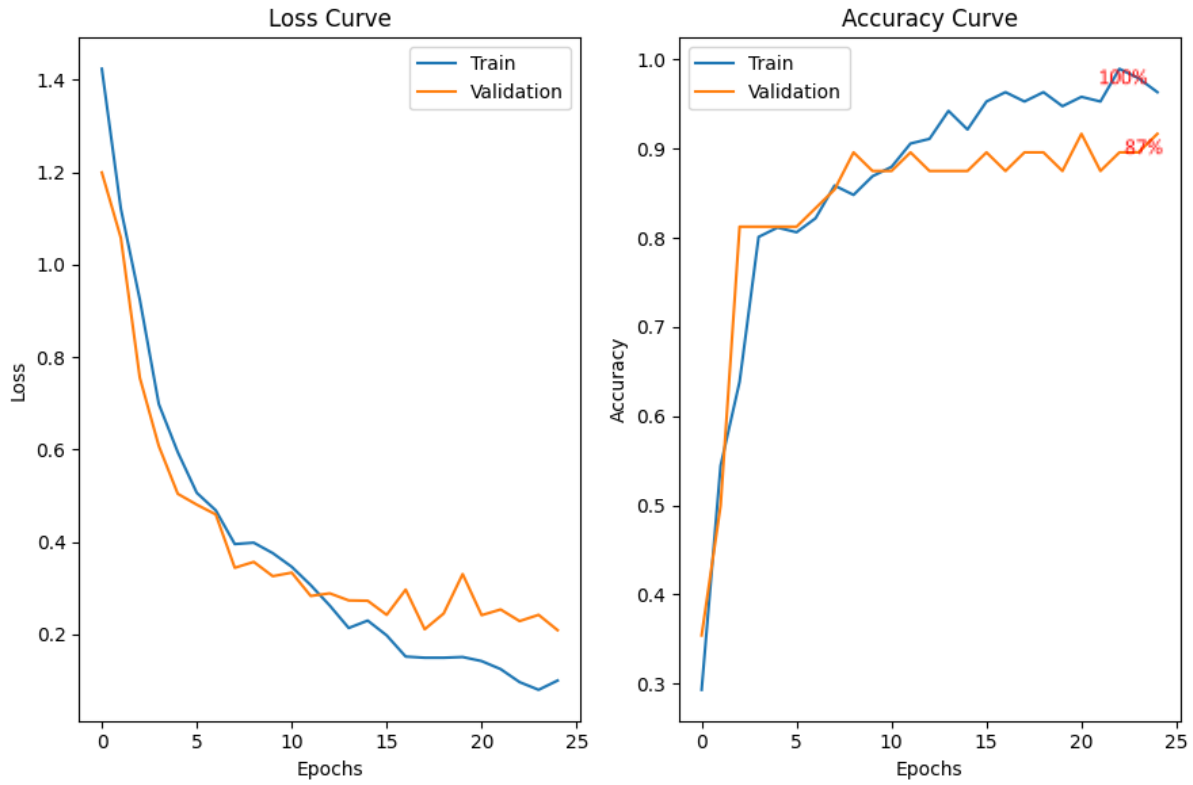


Figure 1: Baseline Model - Loss and Accuracy Plots

The baseline model produced an accuracy score of 1.0 and 0.03 loss after 25 epochs with the training data. This can be considered very good considering the limited number of samples. However, the lower accuracy score (0.87) against validation data suggests that the model is over-fitting to the training and therefore cannot be generalized to unseen data causing the lower validation score.

Another point to note is that while accuracy and loss normally have an inverse relationship (i.e. loss decreases with increasing accuracy), the validation data for this model shows a different behavior where the accuracy score plateaus at 0.87 after 6 epochs but loss continues to increase. This suggests that while some images are being correctly predicted, the softmax probability for the wrong prediction is higher thus increasing the overall loss value. Lastly, the test accuracy score of 0.86, with loss value of 0.73 is similar to the results from the validation data and suggests that the high training accuracy is likely due to model over-fitting to the training data.

To try and improve the model results and test accuracy, we perform a second iteration with data augmentation to increase the number of features for the model to be trained on.

Second iteration - Data Augmented (DA) Model

For the second iteration, we used an image data generator class to generate batches of augmented data which is fed into the model for training.

We apply different augmentations such as image rotation, width and height shift, and horizontal/vertical flip which will be randomly applied to the batches of data from the dataset and passed into the model during training. Several of the data generator hyperparameters were trialed and the below code snippet shows the final set of parameters.

After fitting and training the model, the results were plotted in the same manner as the baseline model to understand the difference and if any improvements in results were seen.

```
# Initialize image generator object
from keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
            rotation_range=0.5, # rotate 5 degree
            width_shift_range=0.5, # shift 5%
            height_shift_range=0.5,
            horizontal_flip=True,  # randomly flip images
            vertical_flip=True)

datagen.fit(X_train) # fit training data to image generator
```

```
# fit training data to model
batch_size = 18

dataAug_history = dataAug_model.fit(datagen.flow(X_train, y_train,
batch_size=batch_size),
                                    validation_data=(X_val, y_val),
epochs=NO_EPOCHS, shuffle=True)
```

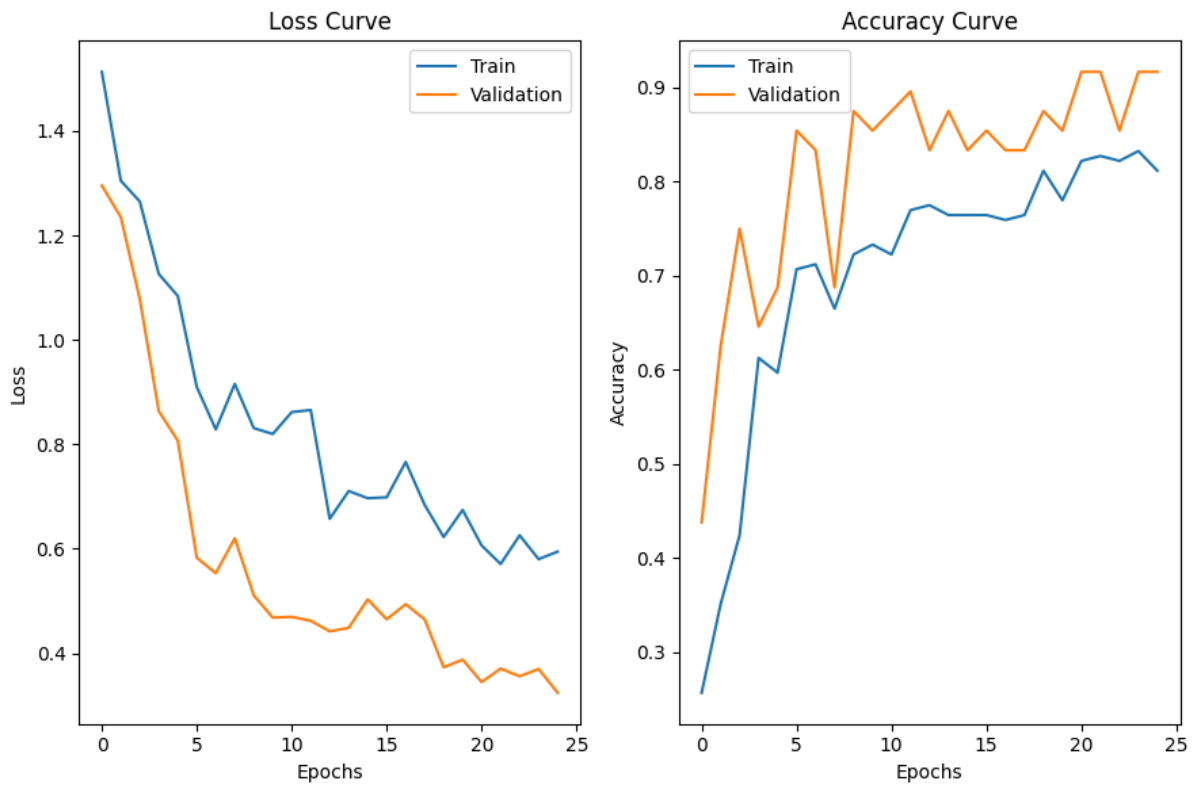


Figure 2: Data Augmented Model - Loss and Accuracy Plot

In addition to the loss and accuracy plots, a confusion matrix was also plotted to visualize the breakdown of true and predicted outcomes from model evaluation with test data.

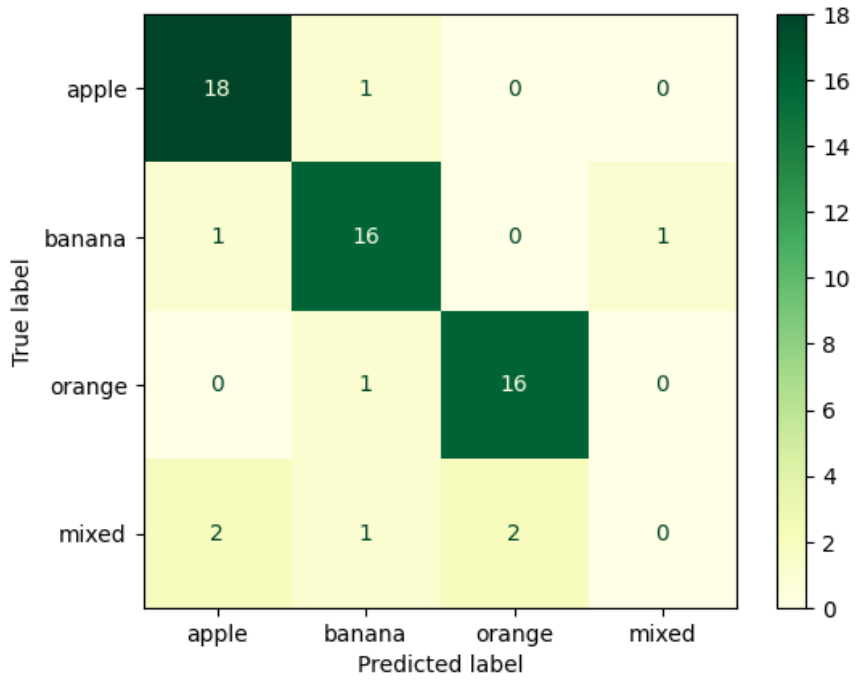


Figure 3: Data Augmented Model - Confusion Matrix

Results from the DA model were overall slightly better than those of the baseline model. The DA model achieved an accuracy score of 87% with a loss of 0.3 using the test data. Comparing this with the baseline model's results of 87% and a loss of 0.73, even though the DA model had the same accuracy score, it achieved a lower loss value which suggests that it is less likely to be overfitting to the training set. This can also be verified by looking at the validation and loss curves for the DA model which closely follows the curve plotted for training data. This shows that the model is able to correctly classify the images on unseen data (validation set) with the same accuracy and loss as on the training data.

Even though the DA model performed better than the baseline model, after plotting the confusion matrix, we noted that the model is poor at predicting images of mixed fruits, as none of the predictions class were correctly classified under this label. After review of the training data, we then attributed the poor model results in classifying this label due to the lower number of samples usable for training in the dataset.

To further improve the DA model accuracy in detecting these classes, we focused the next iteration on resolving class imbalance before model training.

Third iteration - Resolving Class Imbalance

After understanding that one class of images, 'mixed', is always wrongly classified, the next step was to resolve the class imbalance issue. We utilized the Random Under Sampler module of the Imbalance Learn (imblearn) library to re-sample images to equalize the number of classes in the training data. Below is a code snippet of the re-balancing after which, we plotted the pie plot to show the before and after plots of the classes in the training set.

```
# apply sampling strategy to equalize number of samples
from imblearn.under_sampling import RandomUnderSampler

# create random sampler
sampling_strategy = 'not minority'
rus = RandomUnderSampler(sampling_strategy=sampling_strategy)

# apply sampling strategy
X_rus, y_rus = rus.fit_resample(X=X, y=y_train)

# reshape back to original dimensions
X_rus = reshape(X_rus)
```

As seen in Figure 4, the original dataset had a very low percentage (8%) of mixed fruit samples as part of the training data. After sample re-balancing was done, the distribution of samples for each class was equalized. This new sampled training dataset was then used for training the models to check the impact of this change on model accuracy and loss.

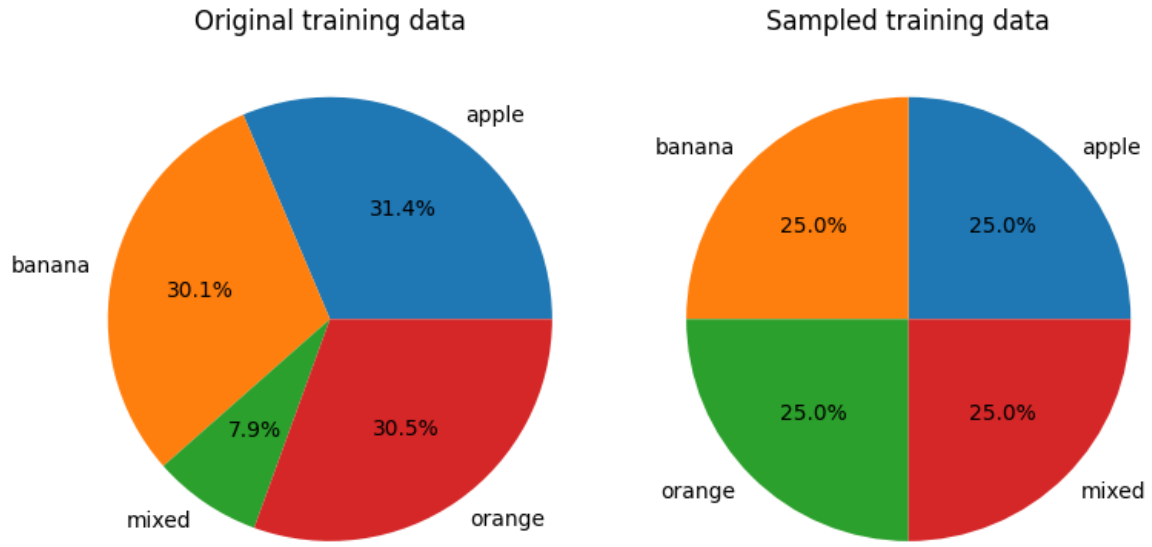
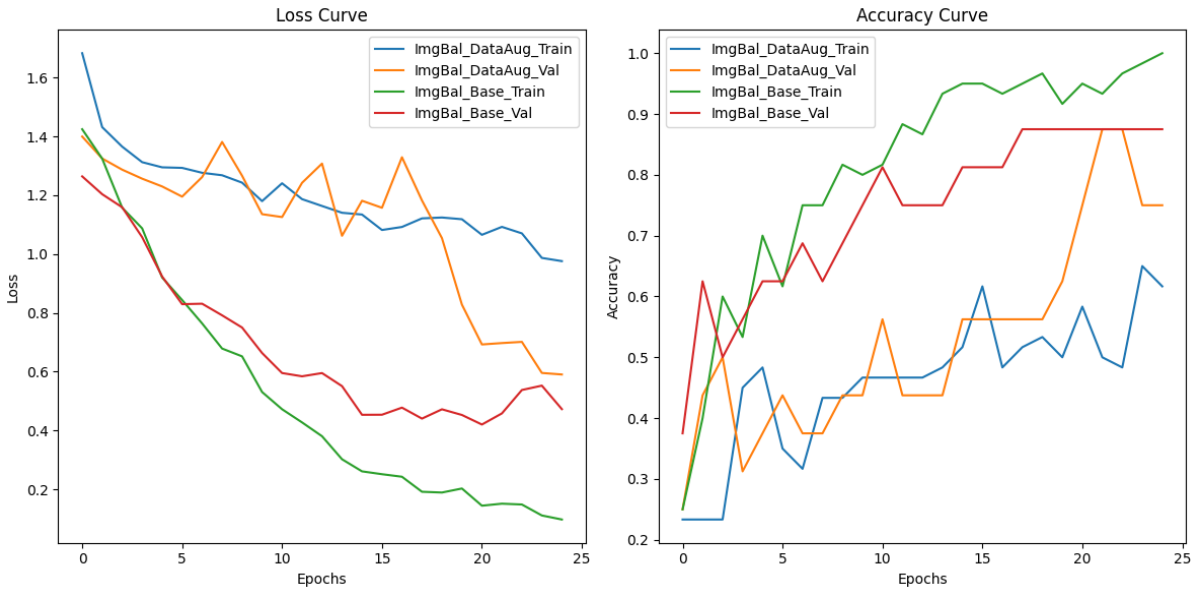
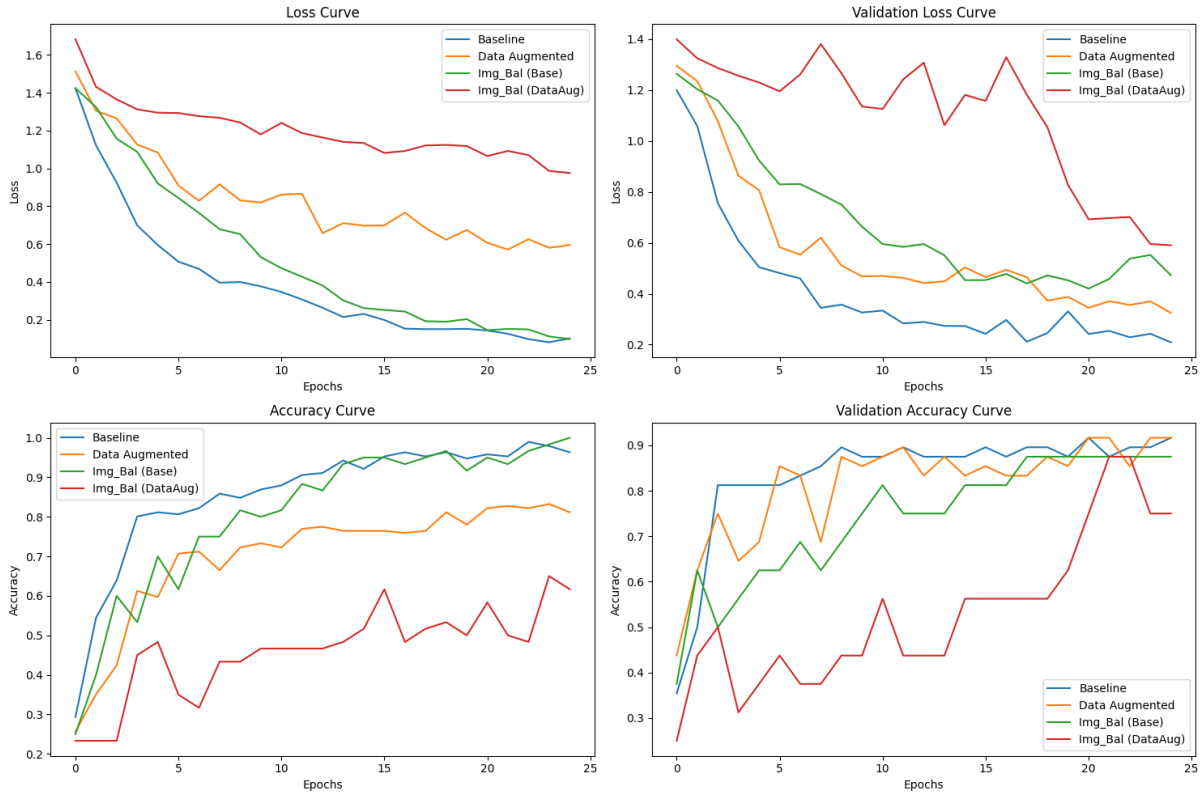


Figure 4: Pie Plot of Training Set Class Distribution Before (Left) and After (Right) Re-balancing

Both baseline (Img_Bal_Base) and image augmented (Img_Bal_DataAug) models were fit with the training data and accuracy and loss results are plotted below.



Conclusion



After evaluating several different approaches as well as several attempts adjusting the model variables, we found that a simple approach— a baseline model with simple Image Augmentation, seemed to yield the best results.