Python Machine Learning (Part 2)

Image Classifier
Course Assignment

Team 6

Engelbert Teh Fu Chang

Jason Tay Neng Wei

Meng Xianzhu

Xie Wenwen

Xu Bingxin

Paing Zay

Wathanta Aung

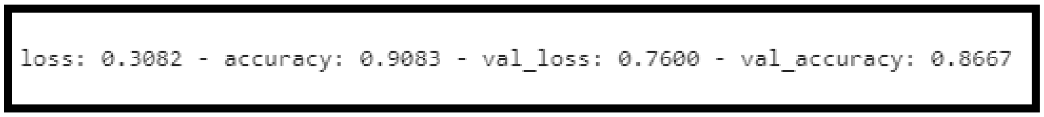
Introduction

In this course assignment, we were required to create an image classifier that could predict accurately on whether the images contained apples, oranges, bananas, or a mix. We imported the Tensorflow-Keras library and created a sequential model. To increase computation speed, images would be resized by our algorithm, on the fly, to a dimension of 224 by 224 pixels, and all features would have their values normalized or standardized.

Methodology

Creating a baseline model

Before making any changes to our dataset, we decided to train a model using the initial dataset provided to get a baseline or reference that could be used for comparison to see how well would the model perform after data pre-processing and augmentation were carried out. After training our model for 50 epochs, we managed to get an accuracy score of 0.8667 (See Figure 1).



```
loss: 0.3082 - accuracy: 0.9083 - val_loss: 0.7600 - val_accuracy: 0.8667
```

Figure 1: Baseline model training history.

Organizing Images

We decided to create subfolders and place our images into their respective subfolders for a number of reasons. Mainly because it was more convenient for us to import these images for training and validation once the images were filed into the labelled subfolders.

Balancing Image Quantities

After placing images into their respective subfolders, their quantities were as follows:

Apple: 75   Banana: 73   Mixed: 20   Orange: 72

We then proceed to add more images, especially to the *Mixed* category, to balance the image quantities.

Correct Mislabelling

According to Muller and Markert (2019), the prediction accuracy of a supervised learning model hinges on two main criteria: the quality of the labelled training data set and the appropriateness of the algorithm. This was one of those steps we paid more attention to. Our approach was to identify and fix any mislabelling of images by either removing, editing, or relabelling them.

For instance, we noticed some images were wrongly labelled, for example, an image of a banana and an apple labelled as *banana* (see Figure 2). We would then relabel and place the image into the *mix* category because it consisted of fruits from 2 different classes.

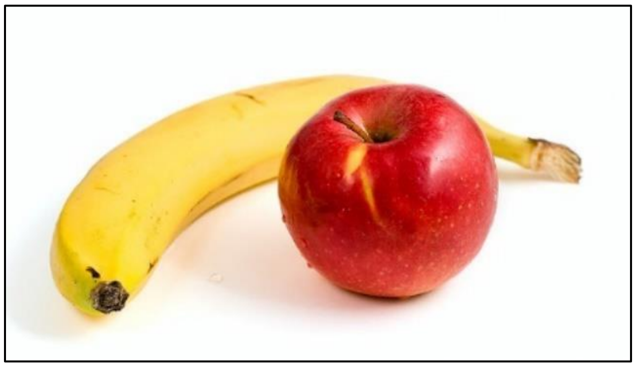


Figure 2: Mislabelled image found in *banana* category.

Scoping

Scoping was the step where we would look through the test data images and identify the common sets of features that we would want our model to learn and to what extent. We would take note of the general features of how would the test set "look like", e.g., what was the dominant colour of respective classes, how were the fruits displayed, did the noises in the image stood out more than the fruit itself, etc. We would then proceed to search for images that were *similar but not the same* with respect to the test set.

A good example to describe our approach would be like the course coordinator of DIPSA gauging the industry on what suitable topics to teach to their students. The coordinator would then get the common and essential topics that the industry would love to have their future employees to know and proceed to package these topics into the course, and then proceed to teach it to their students.

Scoping would be a quick and focused approach to train a model, but such a methodology too has its limitations and disadvantages (See **Limitations** and **Strengths**).

Most Confusing Image

Using our custom method, the fitted model would identify among the test data images, which image would it consider of having a high probability to be of a certain class, but in reality, be from another class, i.e., wrong prediction with high confidence. The method would then return the respective image (See Figure 3) where we would analyse the reasons for its prediction, make some guesses, and proceed to gather other similar images to feed it back to the model for retraining.

Notice that the model predicted the image to be that of an apple. Most likely the model learnt that red meant apple, which would have been a sign of overfitting since there were little "redness" associated with *banana* in our training dataset.

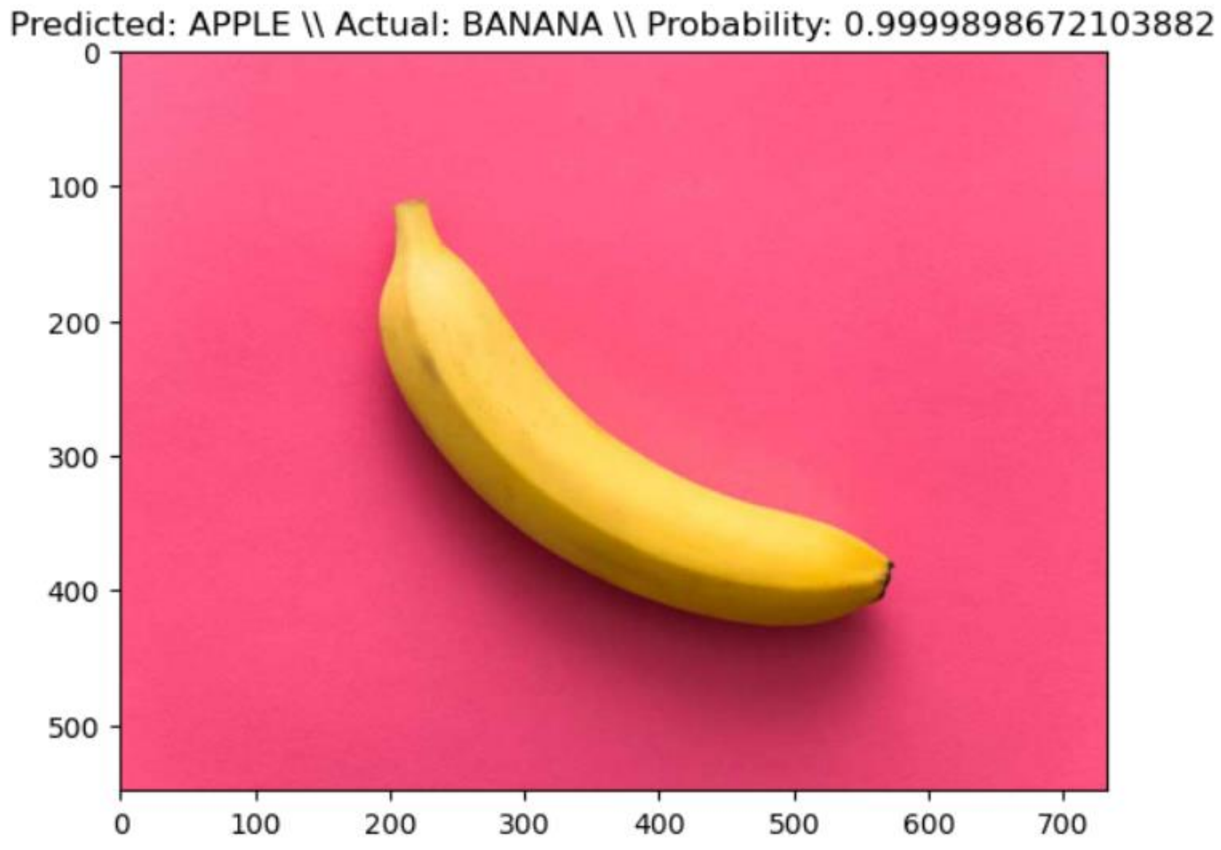

Predicted: APPLE \\ Actual: BANANA \\ Probability: 0.9999898672103882
0
100
200
300
400
500
0
100
200
300
400
500
600
700

Figure 3: Falsely predicted image on test data. (*Data\test\banana\banana_88.jpg)*

Image Cropping

According to research on image cropping and model's prediction accuracy, the experimental results showed that image classification using image cropping applied at the data pre-processing stage churns out model with higher accuracy in comparison to classification without cropping (Mishra, 2020). Hence, we proceeded to do image cropping on some of our images (See Figure 4a & 4b) to remove noises and to focus the fruit more on the centre of the image.

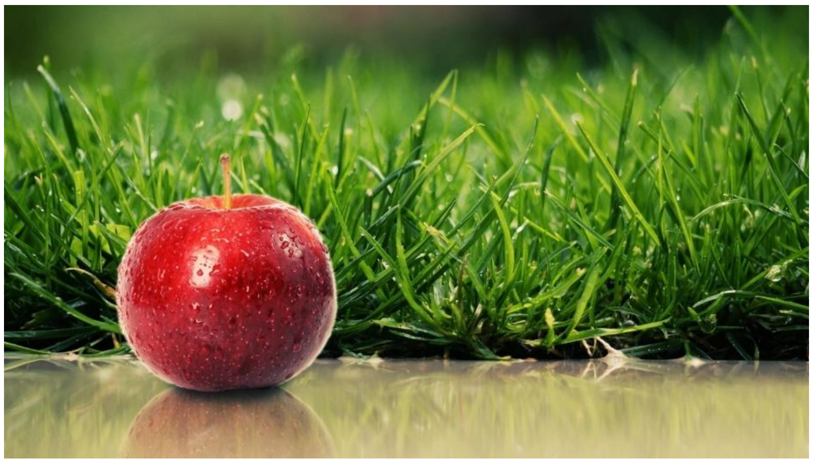


Figure 4a: Uncropped image of an apple.

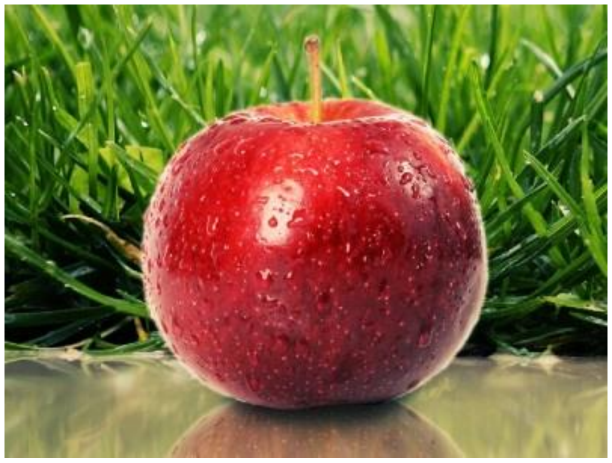
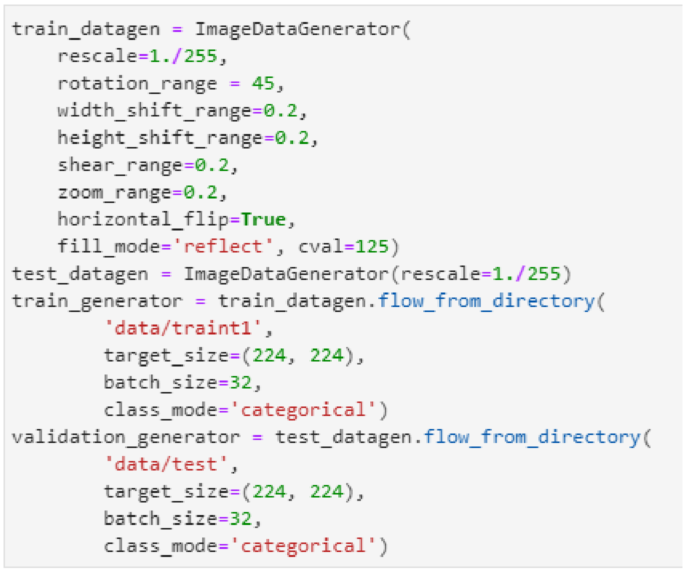


Figure 4b: Cropped image of an apple.

Image Augmentation

We applied image augmentation on our training dataset, on the fly, using Tensorflow-Keras' ImageDataGenerator class. Examples of the effects that were randomly applied on our images during every epoch would be as follows: rotate, width and height translation, shear, zooming, flipping from left to right (See Figure 5).

Image augmentation would virtually increase the total number of training images, prevent model overfitting, and generally improve our model's prediction accuracy.



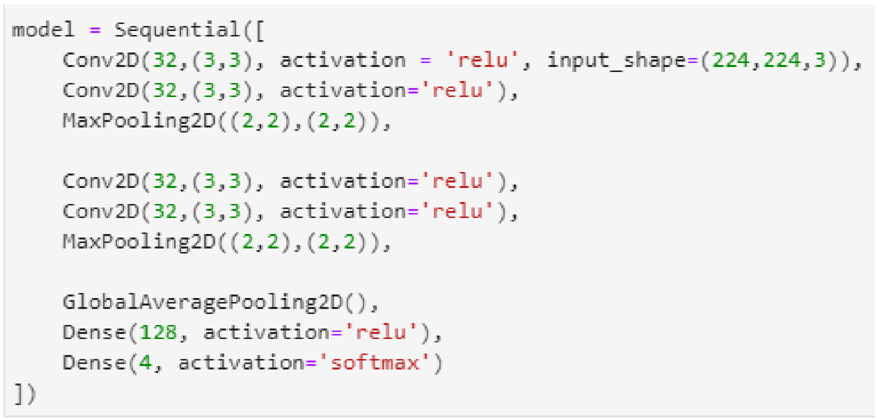
```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range = 45,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='reflect', cval=125)
test_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
        'data/traint1',
        target_size=(224, 224),
        batch_size=32,
        class_mode='categorical')
validation_generator = test_datagen.flow_from_directory(
        'data/test',
        target_size=(224, 224),
        batch_size=32,
        class_mode='categorical')
```

Figure 5: Image augmentation configuration.

Sequential Layers Configuration

After experimenting with different arrangement of the layers in the sequential model, we decided our configuration to be as such (See Figure 6). We also decided to use the *GlobalAveragePooling2D* method instead of *Flatten* because according to Landup (2022), he created a small case study on *Flattening* vs *GlobalAveragePooling* and concluded that the latter was a better choice. To summarize the article in one sentence, with his own words:

" When flattening, you're torturing your network to learn from oddly-shaped vectors in a very inefficient manner." (Landup, 2022).



```
model = Sequential([
    Conv2D(32,(3,3), activation = 'relu', input_shape=(224,224,3)),
    Conv2D(32,(3,3), activation='relu'),
    MaxPooling2D((2,2),(2,2)),

    Conv2D(32,(3,3), activation='relu'),
    Conv2D(32,(3,3), activation='relu'),
    MaxPooling2D((2,2),(2,2)),

    GlobalAveragePooling2D(),
    Dense(128, activation='relu'),
    Dense(4, activation='softmax')
])
```

Figure 6: Sequential layers.

Results and evaluation

The baseline model showed signs of the model overfitting to its training data, and the classical sign would be the model performing well on training data, but performs poorly on the test data (see Figure 7). Notice how the orange line (validation loss) spikes occasionally, and deviates away from the blue line (training loss) – it meant that the model could not predict with consistent accuracy or generalize well, with respect to the images between the two data set. In contrast, after data pre-processing, the training and validation losses started to merge and the losses stabilizes within 30 epochs of training the model with the pre-processed images.

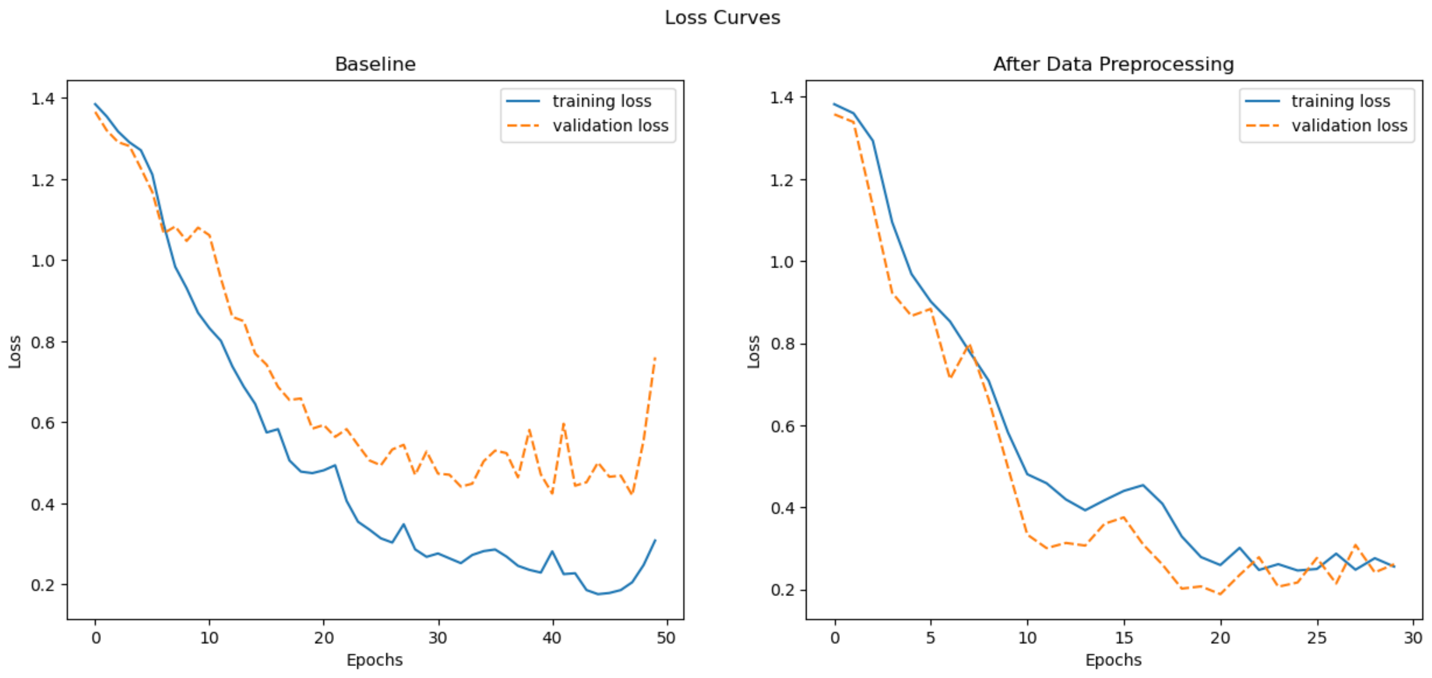


Figure 7: Training and validation loss curves.

By plotting out and looking at the confusion matrix (see Figure 8), we could see the model got confused and wrongly predicted the classes of some images. We were curious what would one of these images looked like, so by calling a custom method, we were able to see the fruit image that our model got confused with (see Figure 9a).

By analysing on the image at Figure 9a, we weren't surprise that the model would label it as *orange*, as the image did indeed appear 'orange-like' to the human eyes, but we believe that such a result could be prevented with a larger training dataset.

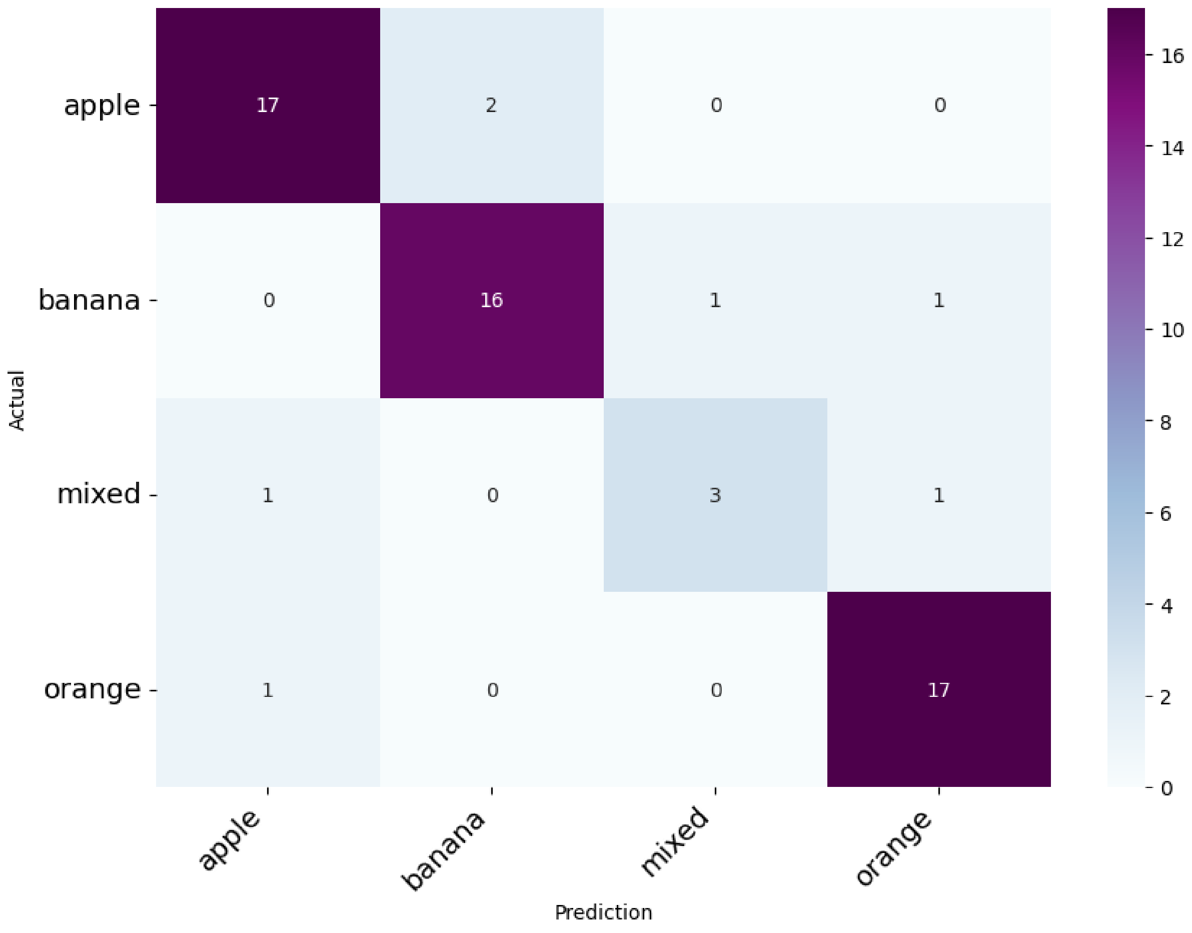


Figure 8: Confusion matrix on test data.

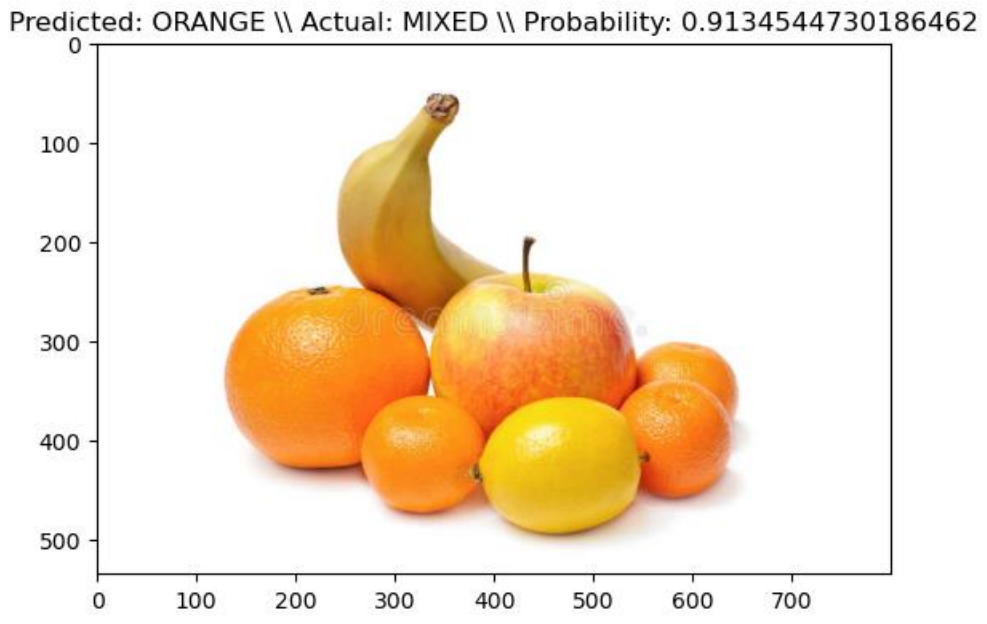
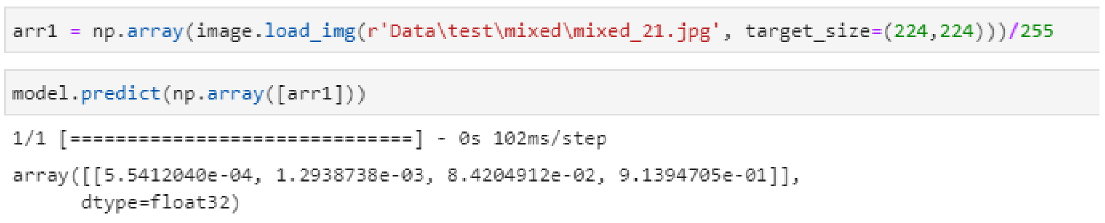


Figure 9a: Image the model predicted wrongly but with high probability.

(*Data\test\mixed\mixed_21.jpg)*



```
arr1 = np.array(image.load_img(r'Data\test\mixed\mixed_21.jpg', target_size=(224,224)))/255
```

```
model.predict(np.array([arr1]))
```

```
1/1 [==============================] - 0s 102ms/step
array([[5.5412040e-04, 1.2938738e-03, 8.4204912e-02, 9.1394705e-01]],
      dtype=float32)
```

Figure 9b: Probability distribution of model's prediction on *mixed_21.jpg* image.

Limitations

Our methodology in getting samples from the internet was to use a custom method to identify the image that the model got really confused with, search for similar images according to the theme that we identified (through browsing the test data set), and feed it back to the model for training. The steps would then be repeated until our model's accuracy score increased to an acceptable level. We understood the drawbacks and limitations of such a methodology as it would, undoubtedly, overfits our model to the *test dataset*, in other words, if our model would be deployed and be used to predict a different test set, it would perform poorly.

Strengths

We chose such a methodology because it greatly helped us understand what the model was "thinking" and how it was "behaving". Because by cherry picking our images, and perform training and evaluation right after some minor image additions, it changed some of our perspectives, believes, and ideas on how a model would learn given a set of data.

For instance, the image at Figure 3 changed our assumptions on how a model would learn: we initially thought a model would learn to identify a fruit mainly by its shape, but the model predicted wrongly about that image. It was clearly an image of a banana, and not an apple. So, it begged the question, "Why such a prediction?". That would then lead us to select images according to our newfound perception on how a model learns.

Despite having some drawbacks in our approach, e.g., overfitting model, we believe it would be advantageous for our learning.

Conclusion

Building a fruit image classifier was indeed a challenging but rewarding experience for us. To sum it up, we have learned a great amount on how a convolutional neural network works, how to build one, and how to analyse it. Our choice of methodology in building our model may not be ideal as we could have fed our model with more images, probably by the thousands, in the hopes that our model could generalize better, but we recognised the limitations of our physical computing devices and the limitations of available time, hence the choice of our approach in training the model.

References

Landup, D. (2022, August 31). *Don't Use Flatten() - Global Pooling for CNNs with TensorFlow and Keras*. Stack Abuse. https://stackabuse.com/dont-use-flatten-global-pooling-for-cnns-with-tensorflow-and-keras/

Mishra, B. K. (2020, February 24). *A novel application of deep learning with image cropping: a smart city use case for flood monitoring*. SpringerLink. https://link.springer.com/article/10.1007/s40860-020-00099-x?error=cookies_not_supported&code=6b95ab39-047c-4522-824d-de04ccbff560

Muller, N. M., & Markert, K. (2019, December 11). *Identifying mislabeled instances in classification*. arxiv.org. https://arxiv.org/pdf/1912.05283.pdf