Assignment 2

Materials submitted:

- Question 1
  - Assigment_2_Frozen_Weights.ipynb
  - model_a.h5 (saved model)
  - model_weights_a.h5 (saved weights)
- Question 2
  - Assigment_2_Unfrozen_Weights.ipynb
  - model_c.h5 (saved model)
  - model_weights_c.h5 (saved weights)
- Question 1 & 2
  - check.ipynb
- Question 3
  - A2Q3.ipynb
  - model_c.h5 (saved model)
  - model_weights_c.h5 (saved weights)

Please download the model and the weights with the following link (access given to James):

https://smu.sharepoint.com/:f:/t/Licheng_Assignment/EjOQbluxF5tHk_hN6gJavEUBzuouaYCXSJbGpJpw8jLy3w?email=jameskohby%40smu.edu.sg&e=K1pbLP

Instructions to run the notebooks

For ***Assigment_2_Frozen_Weights.ipynb*** and ***Assigment_2_Unfrozen_Weights.ipynb***

Both notebooks are created in the **google colab environment**. It will run as long as all path are correct 😊. Before running, kindly ensure that all the below paths are set correctly:

- Please set the correct path there the model + weights will be saved and later loaded from.

```
if trainable == 0:
    #path for model
    model_path = '/content/gdrive/My Drive/Colab Notebooks/604_vision/assign_2/model_2/model_a.h5'
    #path for weights
    model_weights = '/content/gdrive/My Drive/Colab Notebooks/604_vision/assign_2/model_2/model_weights_a.h5'
```

- Please also specify path to the image zip folder & extract destination path

```
# load the zip file
zip_ref = zipfile.ZipFile('/content/gdrive/My Drive/Colab Notebooks/604_vision/assign_1/open_images.zip', 'r')
#Extracts the files into the /tmp folder
zip_ref.extractall('/content/tmp')
# close the zip
zip_ref.close()
```

- Please specify the paths to the train and eval set

```
# specify directory to load the unzipped data from
train_directory = '/content/tmp/train'
eval_directory = '/content/tmp/eval'
```

For ***check.ipynb***

- Please set the correct path to the model and weights.

```
#path for model
model_path = '/content/gdrive/My Drive/Colab Notebooks/604_vision/assign_2/model_2/model_a.h5'
#path for weights
model_weights = '/content/gdrive/My Drive/Colab Notebooks/604_vision/assign_2/model_2/model_weights_a.h5'
```

- Please also specify path to the image zip folder & extract destination path

```
# load the zip file
print('loading zip file. . .')
zip_ref = zipfile.ZipFile('/content/gdrive/My Drive/Colab Notebooks/604_vision/assign_1/open_images.zip', 'r')
#Extracts the files into the /tmp folder
zip_ref.extractall('/content/tmp')
```

- Please specify the paths to the eval set

```
# specify directory to load the unzipped data from
eval_directory = '/content/tmp/eval'
```

For ***A2Q3.ipynb***

- Please ensure the path to single image, model & weights are correct.

```
img = '/content/gdrive/My Drive/Colab Notebooks/604_vision/assign_2/img_test.jpg'
model = '/content/gdrive/My Drive/Colab Notebooks/604_vision/assign_2/model_2/model_c.h5'
weights = '/content/gdrive/My Drive/Colab Notebooks/604_vision/assign_2/model_2/model_weights_c.h5'
```

Model setup and design

Model Selection

In our implementation, we have selected the following 2 models:

- InceptionV3
- EfficientNetB2

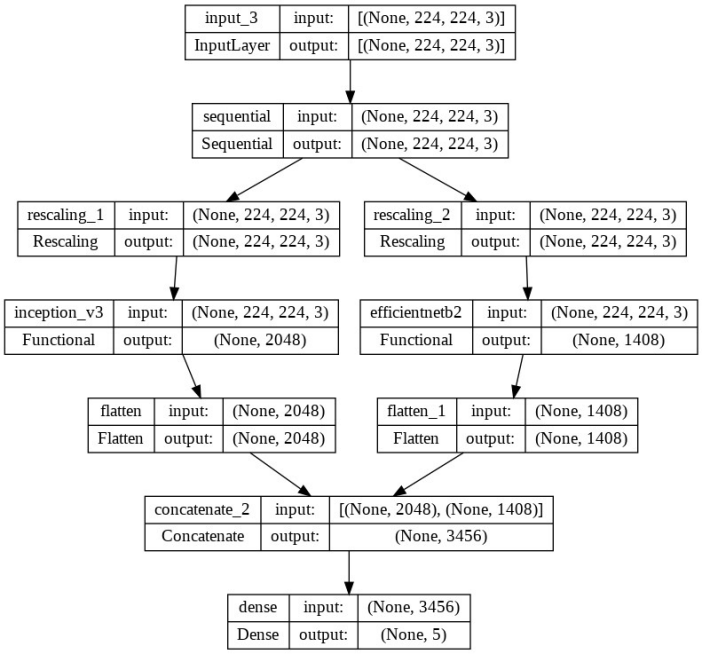


Image Augmentations

Before feeding the images into the entry point, we first perform image augmentation. In general, with image augmentation, we observe a noticeable improvement in accuracy. This is likely due to more data being generated because of the augmentations.

```
# create a data augumentation layer to augument the training data
data_augmentation = tf.keras.Sequential(
  [
    layers.RandomFlip("horizontal", input_shape=(224,224,3)),
    layers.RandomRotation(0.2),
    layers.RandomZoom(0.2),
  ]
)
```

Freezing all layers

All layers are frozen using the following lines:

```
# if all weights are to be frozen
if trainable == 0:
    # set all layers to false
    for layer in a_model.layers:
        layer.trainable = False
```

```
if trainable == 0:
    # set all as false
    for layer in b_model.layers:
        layer.trainable = False
```

Joining output of 2 models together

There are 3 ways to do this:

- Set pooling to “None”
- Set pooling to “Global Average Pooling”
- Set pooing to “Global Max Pooling”

After experimenting with all 3 possibilities:

Using “None” means that all elements of the feature map will need to be flattened and joined with the other model’s flattened output. This results in a larger number of trainable parameters. With all the weights frozen, we are able to get 75% accuracy. However, it the training the becomes less stable since there are much more parameters to optimize.

On the other hand, using “max” pooling will means we will only pick the max value of each feature map and then output them as a flattened vector to combine with the other model. However, we are only able to get 70% accuracy. Taking the max value is a bit tricky since there can be fluctuations in value.

In the end, we settle for the “global average pooling” where the average value of each filter map is being output to the flattened vector. We notice this gives more stable training with the accuracy of 75% despite having less parameters.

Callbacks

We will use 2 kinds of callback:

- Model Checkpoint
  - At the end of each epoch, we will save the model and weights If the accuracy is the higher than all other models
- Learning rate decay
  - Ramp decay after certain epochs
    This is for the training with frozen weights. We will use a larger learning rate of 0.001 for the first 3 epochs to boost the learning following by decay for finer learning.
  - Time based decay
    We use this for training where weights are unfrozen or partially frozen.
    Here there is no need to use a large learning rate or ramping.

Checking the average values of weights before and after training

For this we will use the first convolutional layer of the InceptionNetV3 model. The kernel dimension is 3x3x3 and the number of filters is 32.

For the frozen layer training we have the following:
The average weights remain the same.

```
Name of layer: conv2d <keras.layers.convolutional.conv2d.Conv2D object at 0x7f754567cad0>
filter width dimension = 3
filter height dimension = 3
filter depth dimension = 3
number of filters = 32
total_elements= 864
total weights= -0.67274463
avg weights= -0.000779
```

```
Name of layer: conv2d <keras.layers.convolutional.conv2d.Conv2D object at 0x7f754567cad0>
filter width dimension = 3
filter height dimension = 3
filter depth dimension = 3
number of filters = 32
total_elements= 864
total weights= -0.67274463
avg weights= -0.000779
avg weights before = -0.000779

Before = After ?
True
```

For the training with all layers unfrozen, we have the following:
We can see that once the layers are unfrozen, the average weights are not the same anymore.

```
Name of layer: conv2d <keras.layers.convolutional.conv2d.Conv2D object at 0x7f155a508ed0>
filter width dimension = 3
filter height dimension = 3
filter depth dimension = 3
number of filters = 32
total_elements= 864
total weights= -0.67274463
avg weights= -0.000779
```

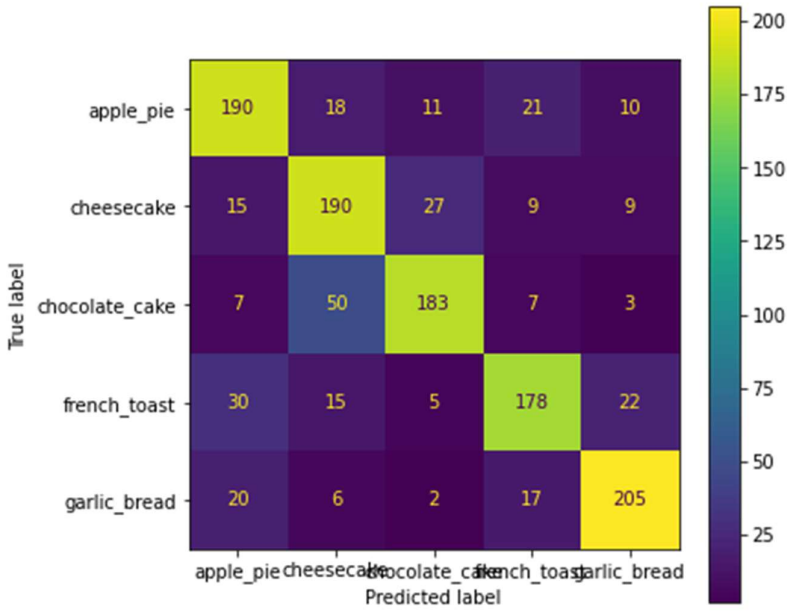
```
Name of layer: conv2d <keras.layers.convolutional.conv2d.Conv2D object at 0x7f155a508ed0>
filter width dimension = 3
filter height dimension = 3
filter depth dimension = 3
number of filters = 32
total_elements= 864
total weights= -0.5528536
avg weights= -0.000640
avg weights before = -0.000779

Before = After ?
False
```

Discussion on the results

F1-Score, Precision, Recall & Accuracy

Frozen:

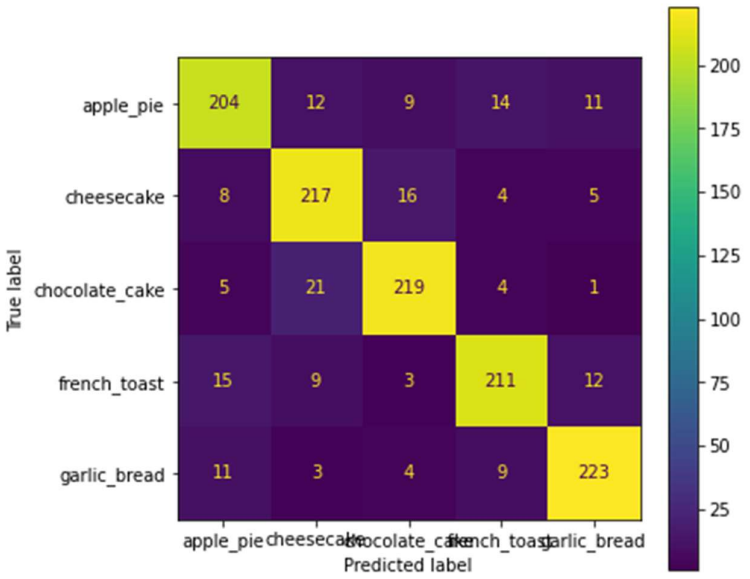


	precision	recall	f1-score	support
apple_pie	0.73	0.76	0.74	250
cheesecake	0.68	0.76	0.72	250
chocolate_cake	0.80	0.73	0.77	250
french_toast	0.77	0.71	0.74	250
garlic_bread	0.82	0.82	0.82	250
accuracy			0.76	1250
macro avg	0.76	0.76	0.76	1250
weighted avg	0.76	0.76	0.76	1250

```
micro precision score: 0.7568
micro recall score: 0.7568
micro f1 score: 0.7568
macro precision score: 0.7598721109745047
macro recall score: 0.7567999999999999
macro f1 score: 0.7572893717384777
weighted precision score: 0.7598721109745046
weighted recall score: 0.7568
weighted f1 score: 0.7572893717384777
```

```
apple_pie : 0.76
cheesecake : 0.76
chocolate_cake : 0.732
french_toast : 0.712
garlic_bread : 0.82
```

Fully Unfrozen:



	precision	recall	f1-score	support
apple_pie	0.84	0.82	0.83	250
cheesecake	0.83	0.87	0.85	250
chocolate_cake	0.87	0.88	0.87	250
french_toast	0.87	0.84	0.86	250
garlic_bread	0.88	0.89	0.89	250
accuracy			0.86	1250
macro avg	0.86	0.86	0.86	1250
weighted avg	0.86	0.86	0.86	1250

```
micro precision score: 0.8592
micro recall score: 0.8592
micro f1 score: 0.8592000000000001
macro precision score: 0.8594163738349889
macro recall score: 0.8592000000000001
macro f1 score: 0.8591327492555509
weighted precision score: 0.8594163738349889
weighted recall score: 0.8592
weighted f1 score: 0.8591327492555508
```

```
apple_pie : 0.816
cheesecake : 0.868
chocolate_cake : 0.876
french_toast : 0.844
garlic_bread : 0.892
```

Findings & Discussion

Findings:

Based on the 2 experiments in question 1 and questions 2 where the weights are frozen and unfrozen, we see that by unfreezing all the layers of the model in question 2, we are able to get a much better performance on the Accuracy, Precision, Recall as well as the F1-Score. This behaviour is consistent across the macro-micro-weighted scenarios. There is an estimated 10% improvement in the performance across all the metrics for classification.

Discussion: Is the new performance better & why?

In the first experiment, where we freeze all the inner layers except for the output of the classification head, all the filters are not being trained, instead they are using the pre-trained weights. In class we learn that the initial layers of the CNN are responsible for high level details. The deeper we go into the neural network, the finer the details are. Hence with all the weights frozen, the output of both models will respond more to the features that they were trained to recognise on, hence the lower performance of 75%. Also since there is no fully connected layer between the joined model and the classification dense layer, there is now less ability for the model to capture of the complexity of the 5 classes. The output layer needs to figure out which class it is just based on the activations from the pre-trained filters alone.

In the second experiment where we unfroze all the layers of the model, we were able to get much better performance of 85%. This is likely due to the reason that the final few layers can now get trained and recognise the finer details of the 5 classes of different foods. This means that the final layers of convolutions are now fine-tuned to the features of the new classes which gives it better recognition ability even without the fully connected layers.

For question 3 it says that we may have the option to further customise by just unfreezing some layers instead of all the layers. I have also attempted to only unfreeze the few top layers but found that the performance has decreased to 80%. Hence I opt to submit the fully unfreeze model instead for question 3.

However, one benefit of partial unfreeze is that we then have less parameters to train. And since our models are already pre-trained very well on a large dataset, it may not be to helpful to train the first few layers since they recognise some general high level features. Besides, for deeper models, the gradient propagation backwards is very tiring due to gradient loss. It could be that that front layer doesn't get affected much even if it is unfrozen. I am sure that eventually if we unfreeze enough layers, we should get to 85% accuracy performance.