Project 03 RadiologyAI

by Nelson W. Pech-May

1. Train a pre-trained convolution neural network

Steps to approach this task:

1.Save a copy of this notebook to your drive. Make sure to select "GPU" as the Hardware accelerator in the runtime option by going to **Runtime → Change runtime type → Hardware accelerator → GPU**.

I created a copy on my google drive and trained my model in the GPU.

2.Update the wget command to download the new data. Then, unzip the dataset. Set the dataset_path accordingly. Set a fixed **seed** to make splits reproducible.

I did the changes in my notebook as shown in the screenshot:

In [3]:
```
# Downloading the Chest X-ray dataset

!wget https://www.dropbox.com/s/73s9n7nugqrv1h7/Dataset.zip?dl=1 -O 'Dataset.zip'
```

```
--2022-10-03 08:44:31--  https://www.dropbox.com/s/73s9n7nugqrv1h7/Dataset.zip?dl=1
Resolving www.dropbox.com (www.dropbox.com)... 162.125.4.18, 2620:100:6019:18::a27d:412
Connecting to www.dropbox.com (www.dropbox.com)|162.125.4.18|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: /s/dl/73s9n7nugqrv1h7/Dataset.zip [following]
--2022-10-03 08:44:31--  https://www.dropbox.com/s/dl/73s9n7nugqrv1h7/Dataset.zip
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://ucb5c2f7edf961dbcfa0a1905907.dl.dropboxusercontent.com/cd/0/get/BuEMF7
Ua2eVei7SCPVbbjcprV3EJSO7S3WhVKHNIKj-cg9ZXfjd58KmsNPDcw0YmWEtKf2FqcgGFbbg1ch9Y/file?dl=1
--2022-10-03 08:44:31--  https://ucb5c2f7edf961dbcfa0a1905907.dl.dropboxusercontent.com/
pJ15K9JqF8V5kvTUa2eVei7SCPVbbjcprV3EJSO7S3WhVKHNIKj-cg9ZXfjd58KmsNPDcw0YmWEtKf2FqcgGFbbg
Resolving ucb5c2f7edf961dbcfa0a1905907.dl.dropboxusercontent.com (ucb5c2f7edf961dbcfa0a1
Connecting to ucb5c2f7edf961dbcfa0a1905907.dl.dropboxusercontent.com (ucb5c2f7edf961dbcf
HTTP request sent, awaiting response... 200 OK
Length: 2090247044 (1.9G) [application/binary]
Saving to: ‘Dataset.zip’

Dataset.zip         100%[===================>]   1.95G   134MB/s    in 13s

2022-10-03 08:44:45 (152 MB/s) - ‘Dataset.zip’ saved [2090247044/2090247044]
```

In [4]:
```
# Unzipping the dataset and delete the .zip file

!unzip -q '/content/Dataset.zip'
!rm -rf '/content/Dataset.zip'
```

In [5]:
```
# Settting up batch size, random seed, and the dataset path

BATCH_SIZE = 64
SEED = 42
dataset_path = '/content/Dataset'
```

3.Use [ImageDataGenerator's](ImageDataGenerator) flow_from_directory() method for augmentation and loading of the train and validation splits. Data augmentation is optional, but if you use it, make sure you have the proper logic/explanation behind the augmentation that you apply.
4.Normalise the input using the rescale parameter of ImageDataGenerator. Data normalisation is an important step that ensures that each input pixel has a similar data distribution.
For steps 3 and 4, I've created three datagens for training, validation and testing datasets. Additionally, I used data-augmentation by rotation angle, with "rotation_range=15" for the training set. Also, I explored the training set (9 images) to have an idea of the images to be trained, as shown in the screenshot:

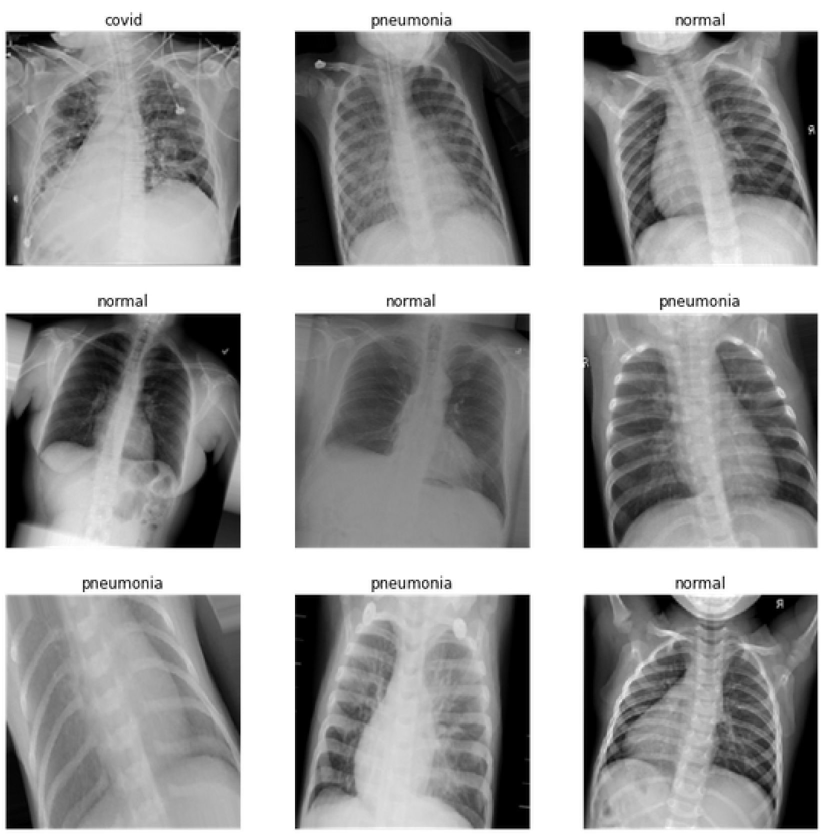
```
Found 11290 images belonging to 3 classes.
Found 3215 images belonging to 3 classes.
Found 1563 images belonging to 3 classes.
```

In [8]:

```
# classes in the train dataset
classes = ['covid', 'normal', 'pneumonia']
```

In [9]:

```
# show some sample images in the dataset
fig = plt.figure(figsize=(12,12))
for u in range(9):
  plt.subplot(330+1+u)
  img, label = next(train_datagen)

  label = label[0]
  label = np.squeeze(label)
  label = np.argmax(label, axis=0)

  plt.axis('off')
  plt.imshow(img[0])
  plt.title(classes[label])

plt.show()
```



5.Use transfer learning to train the model. Select and initialise a model from [this list](). Keep the imagenet weights.

I've choosen the VGG16 model as pretrained model for my transfer learning project.

```
In [10]: # Initialising model and passing the imagenet weights
         # We are specifying classes = 1000 because the model was trained on 1000 classes
         # The classes will be changed afterwards according to our problem

         pretrained_model = tf.keras.applications.VGG16(weights = 'imagenet',
                                                        classes = 1000,
                                                        input_shape = (224, 224, 3),
                                                        include_top = False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 0s 0us/step
58900480/58889256 [==============================] - 0s 0us/step
```

6.Add (at least) one custom Dense layer with softmax activation. After building your model, you will compile it and use accuracy as a metric.

I've frozen except the last six layers from VGG16, then added a global max pooling 2d (to "flatten" the convolutional layers), then passed to a fully-connected layer of 512 nodes with "relu" activation function, then added a dropout layer of 50% (for regularization) and finally add a 3 node dense layer with "softmax" activation for the categorical classification.

```
In [12]: # Adding a prediction layer. It takes input from the last layer (global_max_pooling2d) of the model
         # It has 3 dense units, as it is a 3-class classification problem
         # freeze 6 last layers:
         for layer in pretrained_model.layers[:-6]:
           layer.trainable=False

         #pretrained_model.trainable=True
         x = pretrained_model(inputs=pretrained_model.input, training=True)
         x = tf.keras.layers.GlobalMaxPooling2D()(x)
         x = Dense(512, activation='relu')(x)
         x = Dropout(0.5)(x)
         #x = Dense(64, activation='relu')(x)
         predictions = Dense(3, activation = 'softmax')(x)


         # Defining new model's input and output layers
         # Input layer of the new model will be the same as pretrained_model
         # But the output of the new model will be the output of final dense layer, i.e., 3 units

         model = Model(inputs = pretrained_model.input, outputs = predictions)


         # We use the SGD optimiser, with a very low learning rate, and loss function which is specific to two class classification
         opt_adam = tf.keras.optimizers.Adam(learning_rate=1e-4,
                                             beta_1=0.9,
                                             beta_2=0.99,
                                             epsilon=1e-7,
                                             amsgrad=True)
         model.compile(optimizer = opt_adam,
                       loss = "categorical_crossentropy",
                       metrics = ["accuracy"])
```

And summary:

```
In [13]: # show final model summary
         model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 global_max_pooling2d (Globa  (None, 512)              0
 lMaxPooling2D)

 dense (Dense)               (None, 512)               262656

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 3)                 1539

=================================================================
Total params: 14,978,883
Trainable params: 9,703,427
Non-trainable params: 5,275,456
_________________________________________________________________
```

7.Add your desired callbacks which will help you during the training.

I've added an additional callback to reduce the learning rate when the validation loss reaches a plateau:

```
In [14]: # You can directly save the model into your Google drive by changing the below path

         model_filepath = '/content/best_model.h5'

         # ModelCheckpoint callback will save models weight if the training accuracy of the model has increased from the previous epoch

         model_save = tf.keras.callbacks.ModelCheckpoint(model_filepath,
                                                         monitor = "val_accuracy",
                                                         verbose = 0,
                                                         save_best_only = True,
                                                         save_weights_only = False,
                                                         mode = "max",
                                                         save_freq = "epoch")

         # Additionally you can add more callbacks, like ReduceLROnPlateau
         reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor="val_loss",
                                                          factor=0.1,
                                                          patience=4,
                                                          verbose=1,
                                                          min_delta=5*1e-3,
                                                          min_lr=3*1e-9)

         # save callbacks:
         callback = [model_save, reduce_lr]
```

8.Now, you can start with the training.

I've trained for 20 Epochs:

In [15]:

```python
# Training the model for 20 epochs
# Shuffle is set to false because the data is already shuffled in flow_from_directory() method

history = model.fit(train_datagen,
                    epochs = 20,
                    callbacks=callback,
                    steps_per_epoch = (len(train_datagen)),
                    validation_data = val_datagen,
                    validation_steps = (len(val_datagen)),
                    shuffle = False)
```

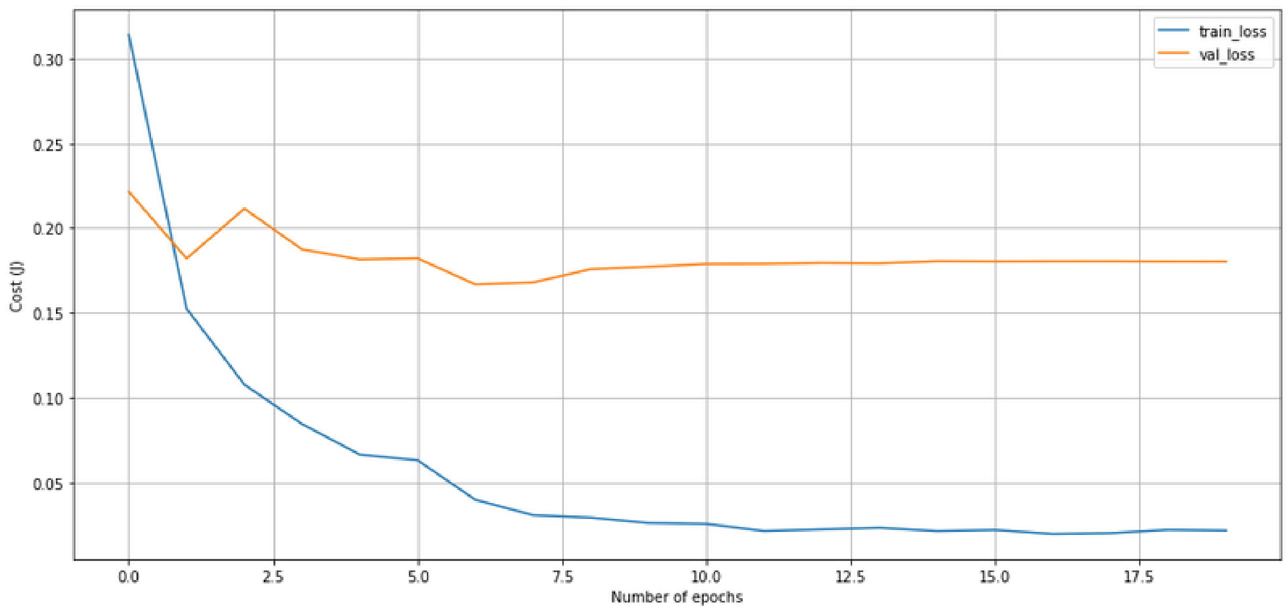
```
Epoch 1/20
177/177 [==============================] - 247s 1s/step - loss: 0.3138 - accuracy: 0.8745 - val_loss: 0.2214 - val_accuracy: 0.9362 - lr: 1.0000e-04
Epoch 2/20
177/177 [==============================] - 236s 1s/step - loss: 0.1525 - accuracy: 0.9431 - val_loss: 0.1820 - val_accuracy: 0.9568 - lr: 1.0000e-04
Epoch 3/20
177/177 [==============================] - 211s 1s/step - loss: 0.1077 - accuracy: 0.9598 - val_loss: 0.2114 - val_accuracy: 0.9549 - lr: 1.0000e-04
Epoch 4/20
177/177 [==============================] - 209s 1s/step - loss: 0.0845 - accuracy: 0.9694 - val_loss: 0.1874 - val_accuracy: 0.9667 - lr: 1.0000e-04
Epoch 5/20
177/177 [==============================] - 209s 1s/step - loss: 0.0665 - accuracy: 0.9774 - val_loss: 0.1815 - val_accuracy: 0.9599 - lr: 1.0000e-04
Epoch 6/20
177/177 [==============================] - ETA: 0s - loss: 0.0631 - accuracy: 0.9771
Epoch 6: ReduceLROnPlateau reducing learning rate to 9.999999747378752e-06.
177/177 [==============================] - 208s 1s/step - loss: 0.0631 - accuracy: 0.9771 - val_loss: 0.1821 - val_accuracy: 0.9621 - lr: 1.0000e-04
Epoch 7/20
177/177 [==============================] - 209s 1s/step - loss: 0.0398 - accuracy: 0.9866 - val_loss: 0.1668 - val_accuracy: 0.9717 - lr: 1.0000e-05
Epoch 8/20
177/177 [==============================] - 209s 1s/step - loss: 0.0308 - accuracy: 0.9893 - val_loss: 0.1679 - val_accuracy: 0.9720 - lr: 1.0000e-05
Epoch 9/20
177/177 [==============================] - 208s 1s/step - loss: 0.0293 - accuracy: 0.9919 - val_loss: 0.1758 - val_accuracy: 0.9723 - lr: 1.0000e-05
Epoch 10/20
177/177 [==============================] - 207s 1s/step - loss: 0.0261 - accuracy: 0.9920 - val_loss: 0.1772 - val_accuracy: 0.9751 - lr: 1.0000e-05
Epoch 11/20
177/177 [==============================] - ETA: 0s - loss: 0.0256 - accuracy: 0.9925
Epoch 11: ReduceLROnPlateau reducing learning rate to 9.999999747378752e-07.
177/177 [==============================] - 207s 1s/step - loss: 0.0256 - accuracy: 0.9925 - val_loss: 0.1789 - val_accuracy: 0.9745 - lr: 1.0000e-05
Epoch 12/20
177/177 [==============================] - 207s 1s/step - loss: 0.0214 - accuracy: 0.9942 - val_loss: 0.1790 - val_accuracy: 0.9748 - lr: 1.0000e-06
Epoch 13/20
177/177 [==============================] - 206s 1s/step - loss: 0.0224 - accuracy: 0.9941 - val_loss: 0.1795 - val_accuracy: 0.9748 - lr: 1.0000e-06
Epoch 14/20
177/177 [==============================] - 206s 1s/step - loss: 0.0233 - accuracy: 0.9942 - val_loss: 0.1792 - val_accuracy: 0.9751 - lr: 1.0000e-06
Epoch 15/20
177/177 [==============================] - ETA: 0s - loss: 0.0214 - accuracy: 0.9934
Epoch 15: ReduceLROnPlateau reducing learning rate to 9.999999974752428e-08.
177/177 [==============================] - 207s 1s/step - loss: 0.0214 - accuracy: 0.9934 - val_loss: 0.1805 - val_accuracy: 0.9748 - lr: 1.0000e-06
Epoch 16/20
177/177 [==============================] - 207s 1s/step - loss: 0.0220 - accuracy: 0.9927 - val_loss: 0.1803 - val_accuracy: 0.9748 - lr: 1.0000e-07
Epoch 17/20
177/177 [==============================] - 207s 1s/step - loss: 0.0197 - accuracy: 0.9939 - val_loss: 0.1804 - val_accuracy: 0.9748 - lr: 1.0000e-07
Epoch 18/20
177/177 [==============================] - 206s 1s/step - loss: 0.0201 - accuracy: 0.9940 - val_loss: 0.1804 - val_accuracy: 0.9748 - lr: 1.0000e-07
Epoch 19/20
177/177 [==============================] - ETA: 0s - loss: 0.0221 - accuracy: 0.9937
Epoch 19: ReduceLROnPlateau reducing learning rate to 1.0000000116860975e-08.
177/177 [==============================] - 205s 1s/step - loss: 0.0221 - accuracy: 0.9937 - val_loss: 0.1803 - val_accuracy: 0.9748 - lr: 1.0000e-07
Epoch 20/20
177/177 [==============================] - 204s 1s/step - loss: 0.0217 - accuracy: 0.9924 - val_loss: 0.1803 - val_accuracy: 0.9748 - lr: 1.0000e-08
```

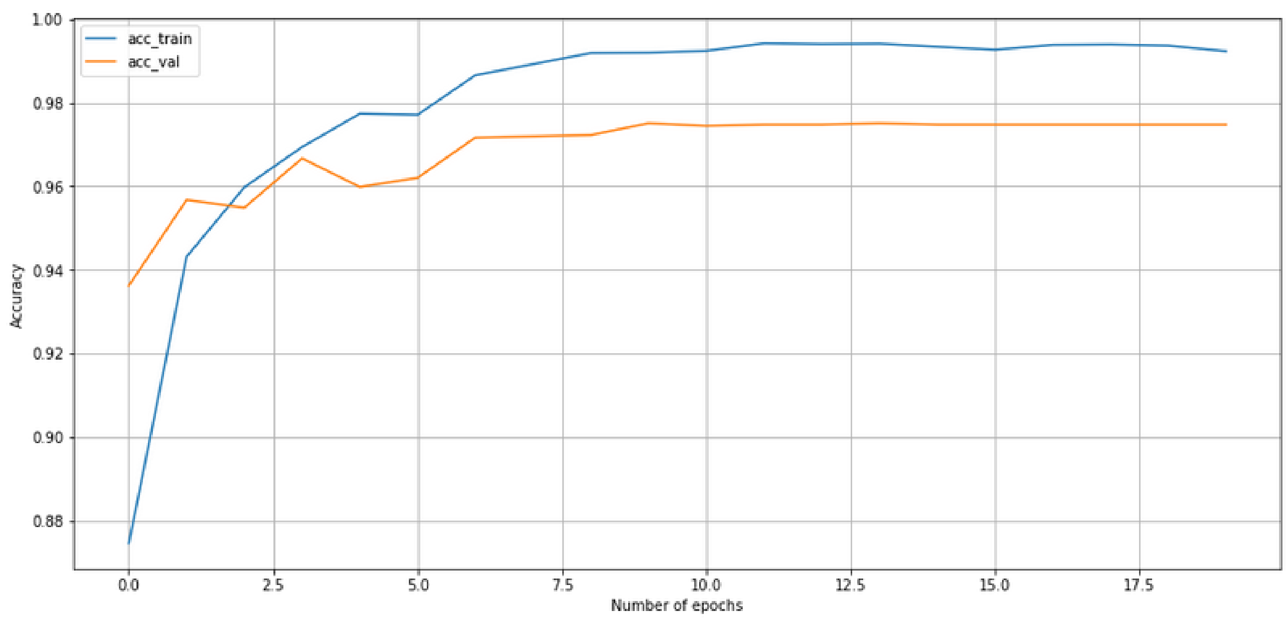
9.This is a medical imaging project; hence 95% accuracy is expected. That is the benchmark for the project.

The **accuracy** has reached **99.24%**, but the **validation accuracy** only **97.48%**, which is still higher than the excepted value.

10.Plot the loss and accuracy graphs using matplotlib or seaborn.

The plots show that there's slight overfitting of the model, however, the performance is pretty good, since the first 5 epochs. More efforts can be done to improve the overfitting issue.





11.Test your model on the test set provided. Generate the classification report and the confusion matrix for the same.

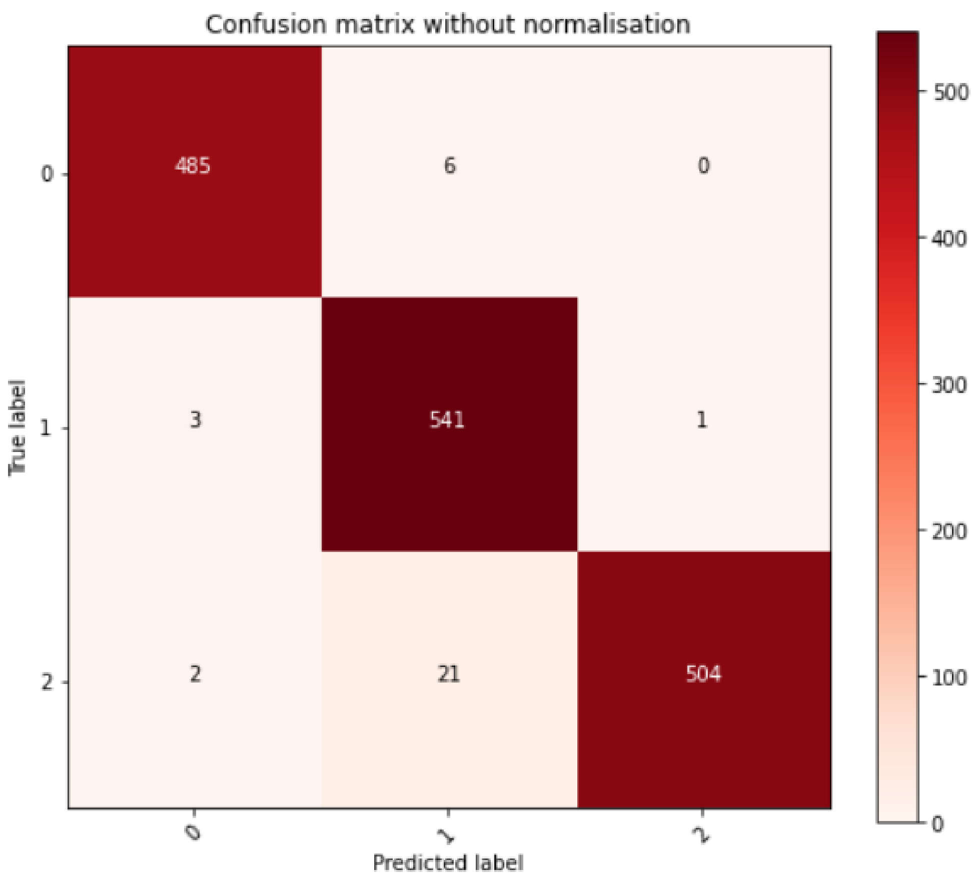
Here is the report obtained: (the results look very promising for precision, recall and F1-score)

```
In [17]: # Model prediction on test set

         predictions = model.predict(test_datagen,
                                     verbose = 1,
                                     steps = (len(test_datagen)))

1563/1563 [==============================] - 19s 12ms/step
```

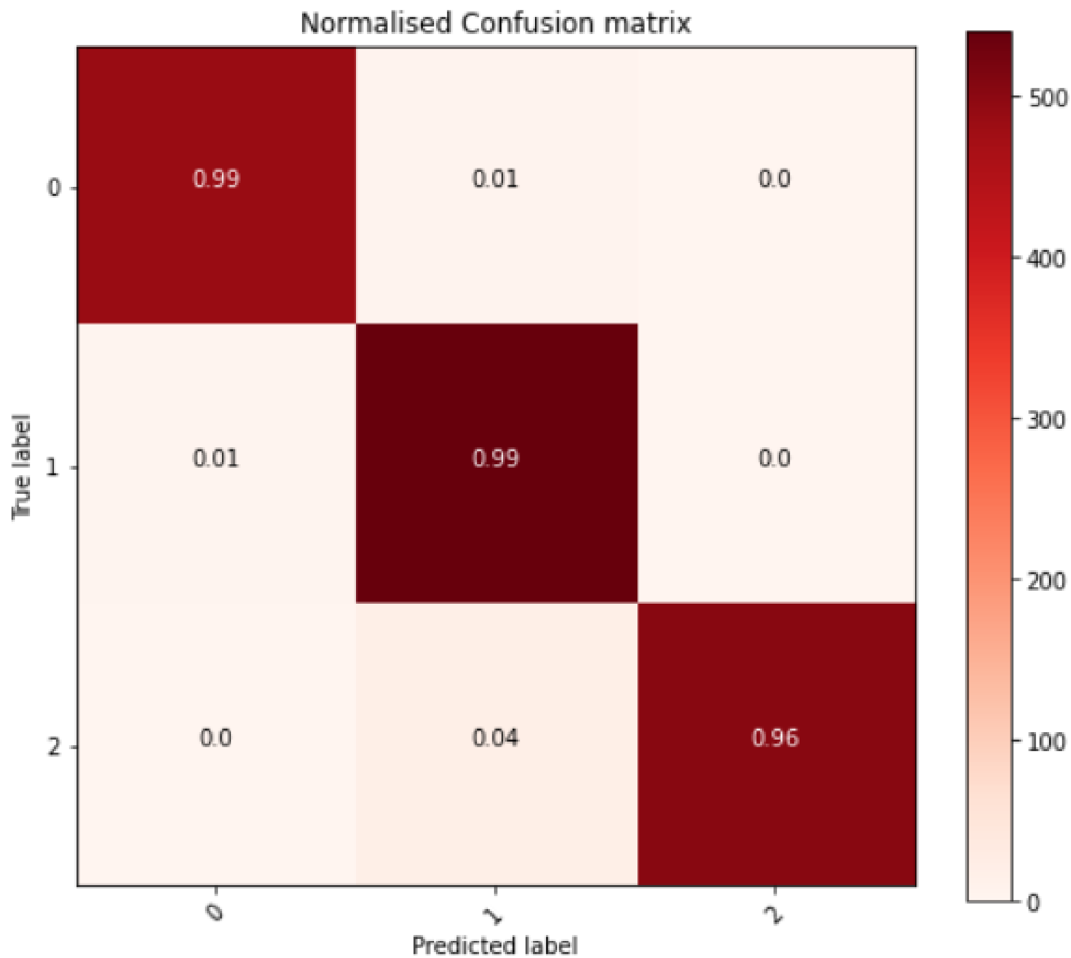
```
In [18]: # Printing predicted classes on the test dataset

         predictions.squeeze().argmax(axis = -1)

Out[18]: array([0, 0, 0, ..., 2, 2, 2])
```

```
In [19]: # Generating the classification report for checking the model's performance on the test set of the same dataset

         classification__report = classification_report(test_datagen.classes,
                                                        predictions.squeeze().argmax(axis = 1))
         print(classification__report)

              precision    recall  f1-score   support

           0       0.99      0.99      0.99       491
           1       0.95      0.99      0.97       545
           2       1.00      0.96      0.98       527

    accuracy                           0.98      1563
   macro avg       0.98      0.98      0.98      1563
weighted avg       0.98      0.98      0.98      1563
```

And the confusion matrix (without normalization):



And with normalization: (the results on the test set look better than I expected)



2. Test the model on an external test dataset

After you are satisfied with your model, head on to [this link](). Upload your best model here, and get the evaluation results. The size limit to upload the model is 700MB.