RadiologyAI

January 6, 2022

Overview

Chest X-Rays have had a massive role in detecting Covid-19 in an individual. Considering how the disease has affected us, we will create a Deep-Learning model as the first level of self-diagnosis. It will classify the Chest XRay image into COVID, Pneumonia, or a healthy lung (i.e., no diseases found).

This implementation classifies the Chest XRay image as Pneumonia or a healthy lung. Use this as a reference, and create a multiclass classification model, classifying the images into Covid-19, Pneumonia, or normal healthy lung.

We will build this model in Python, using TensorFlow Keras.

Problem Statement

This implementation is ridiculed with errors. The model accuracy is low, and the predictions are incorrect. Moreover, the dataset needs to be updated for three-class classification and trained accordingly. It is recommended that you change the model.

Datasets

This notebook is implemented on Pneumonia and normal images. We've taken the dataset from https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia, and hosted it on dropbox for easy access.Ref: Kermany, Daniel; Zhang, Kang; Goldbaum, Michael (2018), "Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images for Classification", Mendeley Data, V2, doi: 10.17632/rscbjbr9sj.2

For the submission, we are combining the datasets from multiple locations. You can find the individual datasets here:

1. Kermany, Daniel; Zhang, Kang; Goldbaum, Michael (2018), "Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images for Classification", Mendeley Data, V2, doi: 10.17632/rscbjbr9sj.2
2. COVID19_Pneumonia_Normal_Chest_Xray_PA_Dataset
3.
    - M.E.H. Chowdhury, T. Rahman, A. Khandakar, R. Mazhar, M.A. Kadir, Z.B. Mahbub, K.R. Islam, M.S. Khan, A. Iqbal, N. Al-Emadi, M.B.I. Reaz, M. T. Islam, "Can AI help in screening Viral and COVID-19 pneumonia?" IEEE Access, Vol. 8, 2020, pp. 132665 - 132676. Paper link
    - Rahman, T., Khandakar, A., Qiblawey, Y., Tahir, A., Kiranyaz, S., Kashem, S.B.A., Islam, M.T., Maadeed, S.A., Zughaier, S.M., Khan, M.S. and Chowdhury, M.E., 2020. Exploring the Effect of Image Enhancement Techniques on COVID-19 Detection using Chest X-ray Images. Paper Link

We have combined and prepared the dataset already. You can download it from the following dropbox link: https://www.dropbox.com/s/73s9n7nugqrv1h7/Dataset.zip?dl=1

Dataset contains 3 folders, namely: * Covid: This folder contains lung x-ray images with covid-19 disease * Pneumonia: This folder contains lung x-ray images with pneumonia disease * Normal: This folder contains normal functioning and healthy lung x-ray images

Tasks

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.
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.
3. Use ImageDataGenerator's `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.
5. Use transfer learning to train the model. Select and initialise a model from this list. Keep the imagenet weights.
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.
7. Add your desired callbacks which will help you during the training.
8. Now, you can start with the training.
9. This is a medical imaging project; hence 95% accuracy is expected. That is the benchmark for the project.
10. Plot the loss and accuracy graphs using matplotlib or seaborn.
11. Test your model on the test set provided. Generate the classification report and the confusion matrix for the same.

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.

3. Video Explanation

Make a short video explaining the changes you made in the notebook. You can create an interview type of video, screen recording, or send a document for the explanation.

Submission

In the form, you need to fill the following fields: 1. Colab notebook link 2. Screenshot of the external test set evaluation 3. Video/Text explanation

Grading Rubric

1. Train the image classifier 70 Marks
2. Test the model on an external test dataset 10 Marks

3. Code Explanation 20 Marks

[ ]:
```
# Checking the GPU information

!nvidia-smi
```

[1]:
```
# Importing required libraries

import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
tf.keras.backend.clear_session()
from tensorflow.keras.layers import Input, Softmax, Dense, Dropout,␣
↪BatchNormalization
from tensorflow.keras.models import Model
import numpy as np
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

[ ]:
```
# Mounting Google drive to save models, and etc.

# from google.colab import drive
# drive.mount('/content/drive')
```

[21]:
```
# Downloading the Chest X-ray dataset
!wget https://www.dropbox.com/s/73s9n7nugqrv1h7/Dataset.zip?dl=1 -O 'archive.
↪zip'
```

```
--2022-01-06 20:13:46--
https://www.dropbox.com/s/73s9n7nugqrv1h7/Dataset.zip?dl=1
Resolving www.dropbox.com (www.dropbox.com)… 162.125.64.18,
2620:100:6020:18::a27d:4012
Connecting to www.dropbox.com (www.dropbox.com)|162.125.64.18|:443… connected.
HTTP request sent, awaiting response… 301 Moved Permanently
Location: /s/dl/73s9n7nugqrv1h7/Dataset.zip [following]
--2022-01-06 20:13:47--
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://ucde1882db234ce4e1d73189eab1.dl.dropboxusercontent.com/cd/0/ge
t/BdTtlj0Fdwa75UYHFCddtndwJE_6R84B_BxZokBTdNoGyA5QY5ptavh_iS9f0fJ1YnBVQnEkd0o7Dc
tkm9HgA068bkYhnvS4KOoNTDi7XTsfoWDThziYgzipTfMtwK_SXmFfEht43_itVVixcxiDoHHy/file?
dl=1# [following]
--2022-01-06 20:13:47--  https://ucde1882db234ce4e1d73189eab1.dl.dropboxusercont
ent.com/cd/0/get/BdTtlj0Fdwa75UYHFCddtndwJE_6R84B_BxZokBTdNoGyA5QY5ptavh_iS9f0fJ
1YnBVQnEkd0o7Dctkm9HgA068bkYhnvS4KOoNTDi7XTsfoWDThziYgzipTfMtwK_SXmFfEht43_itVVi
xcxiDoHHy/file?dl=1
Resolving ucde1882db234ce4e1d73189eab1.dl.dropboxusercontent.com
(ucde1882db234ce4e1d73189eab1.dl.dropboxusercontent.com)… 162.125.64.15,
```

```
2620:100:6020:15::a27d:400f
Connecting to ucde1882db234ce4e1d73189eab1.dl.dropboxusercontent.com
(ucde1882db234ce4e1d73189eab1.dl.dropboxusercontent.com)|162.125.64.15|:443…
connected.
HTTP request sent, awaiting response… 200 OK
Length: 2090247044 (1.9G) [application/binary]
Saving to: ‘archive.zip’

archive.zip         100%[===================>]   1.95G  12.8MB/s    in 2m 50s

2022-01-06 20:16:39 (11.7 MB/s) - ‘archive.zip’ saved [2090247044/2090247044]
```

[22]:
```
# Unzipping the dataset and delete the .zip file

!unzip -q './archive.zip'
!rm -rf './archive.zip'
```

[27]:
```
# Settting up batch size, random seed, and the dataset path

BATCH_SIZE = 64
SEED = 21
dataset_path = './Dataset'
```

[28]:
```
# Initialising ImageDataGenerator for data augmentation
# We use random horizontal flip for augmentation
# Pixels will be notmalised between 0 and 1
  # zca_epsilon: Epsilon for ZCA whitening. Default is 1e-6
  # Horizontal_flip: Boolean. Randomly flip inputs horizontally.
  # Rescale: Rescaling factor, defaults to None.
           # If None or 0, no rescaling is applied, otherwise it multiplied the␣
 ↪data by the value provided
           # (after applying all other transformations)

train_val_gen = ImageDataGenerator(zca_epsilon = 0.0,
                                   horizontal_flip = True,
                                   rescale = 1./255)        # Do not change␣
 ↪rescale

test_gen = ImageDataGenerator(zca_epsilon = 0.0,
                              horizontal_flip = False,
                              rescale = 1./255)             # Do not change␣
 ↪rescale

# The evaluation on streamlit share assumes rescaling takes place,
# and it is 1./255 always
```

[29]:
```
# Taking input of the train, validation, and test images using␣
↪flow_from_directory() function
# Setting the image size to (224, 224) and setting the batch size

train_datagen = train_val_gen.flow_from_directory(directory = dataset_path + '/
↪train',
                                                  target_size = (224, 224),
                                                  color_mode = "rgb",
                                                  classes = None,
                                                  class_mode = "categorical",
                                                  batch_size = BATCH_SIZE,
                                                  shuffle = True,
                                                  seed = SEED,
                                                  interpolation = "nearest")

val_datagen = train_val_gen.flow_from_directory(directory = dataset_path + '/
↪val',
                                                target_size = (224, 224),
                                                color_mode = "rgb",
                                                classes = None,
                                                class_mode = "categorical",
                                                batch_size = BATCH_SIZE,
                                                shuffle = True,
                                                seed = SEED,
                                                interpolation = "nearest")


# For testing, we should take one input at a time. Hence, batch_size = 1

test_datagen = test_gen.flow_from_directory(directory = dataset_path + '/test',
                                            target_size = (224, 224),
                                            color_mode = "rgb",
                                            classes = None,
                                            class_mode = "categorical",
                                            batch_size = 1,
                                            shuffle = False,
                                            seed = SEED,
                                            interpolation = "nearest")
```

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

[30]:
```
# Initialising MobileNet 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.inception_resnet_v2.InceptionResNetV2(
```

```
    include_top=True, weights='imagenet', input_tensor=None,
    input_shape=None, pooling=None, classes=1000,
    classifier_activation='softmax')
```

[47]:
```
# Printing the model summary

#print(pretrained_model.summary())
```

[32]:
```
# Adding a prediction layer. It takes input from the last layer␣
 ↪(global_max_pooling2d) of MobileNet
# It has 2 dense units, as it is a binary classification problem

predictions = Dense(3, activation = 'softmax')(pretrained_model.output)


# Defining new model's input and output layers
# Input layer of the new model will be the same as MobileNet
# But the output of the new model will be the output of final dense layer, i.e.
 ↪, 2 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

model.compile(optimizer = tf.keras.optimizers.RMSprop(learning_rate=0.0001,␣
 ↪decay=1e-6),
              loss = "binary_crossentropy",
              metrics = ["accuracy"])
```

[33]:
```
# 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

callback = [model_save]
```

[48]:
```
# Printing the model summary

#print(model.summary())
```

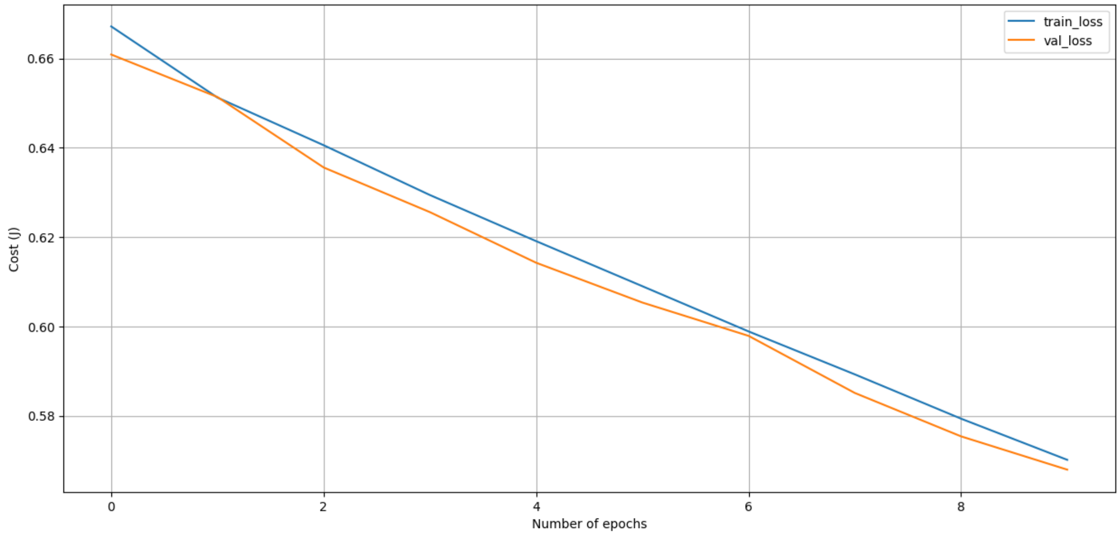
[35]:
```
# Training the model for 5 epochs
# Shuffle is set to false because the data is already shuffled in␣
 ↪flow_from_directory() method

history = model.fit(train_datagen,
                    epochs = 10,
                    steps_per_epoch = (len(train_datagen)),
                    validation_data = val_datagen,
                    validation_steps = (len(val_datagen)),
                    shuffle = False,
                    callbacks = callback)
```

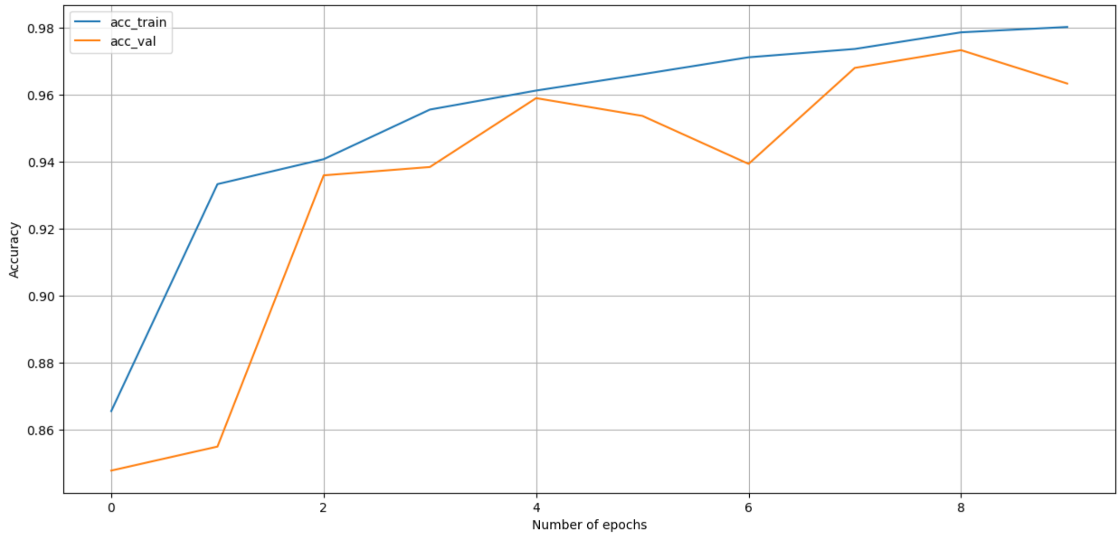
```
Epoch 1/10
177/177 [==============================] - 130s 602ms/step - loss: 0.6672 -
accuracy: 0.8656 - val_loss: 0.6609 - val_accuracy: 0.8479
Epoch 2/10
177/177 [==============================] - 103s 582ms/step - loss: 0.6513 -
accuracy: 0.9333 - val_loss: 0.6514 - val_accuracy: 0.8551
Epoch 3/10
177/177 [==============================] - 104s 587ms/step - loss: 0.6406 -
accuracy: 0.9407 - val_loss: 0.6356 - val_accuracy: 0.9359
Epoch 4/10
177/177 [==============================] - 104s 584ms/step - loss: 0.6294 -
accuracy: 0.9555 - val_loss: 0.6256 - val_accuracy: 0.9384
Epoch 5/10
177/177 [==============================] - 104s 584ms/step - loss: 0.6191 -
accuracy: 0.9612 - val_loss: 0.6143 - val_accuracy: 0.9589
Epoch 6/10
177/177 [==============================] - 102s 574ms/step - loss: 0.6090 -
accuracy: 0.9661 - val_loss: 0.6054 - val_accuracy: 0.9537
Epoch 7/10
177/177 [==============================] - 103s 579ms/step - loss: 0.5989 -
accuracy: 0.9711 - val_loss: 0.5979 - val_accuracy: 0.9393
Epoch 8/10
177/177 [==============================] - 104s 588ms/step - loss: 0.5893 -
accuracy: 0.9736 - val_loss: 0.5851 - val_accuracy: 0.9680
```

```
Epoch 9/10
177/177 [==============================] - 104s 585ms/step - loss: 0.5793 -
accuracy: 0.9786 - val_loss: 0.5754 - val_accuracy: 0.9733
Epoch 10/10
177/177 [==============================] - 102s 578ms/step - loss: 0.5701 -
accuracy: 0.9802 - val_loss: 0.5679 - val_accuracy: 0.9633
```

[36]:

```
# Plotting the loss and accuracy graphs

import matplotlib.pyplot as plt

plt.figure(figsize = (15,7))

tr_losses = history.history['loss']
val_losses = history.history['val_loss']

tr_accs = history.history['accuracy']
val_accs = history.history['val_accuracy']

plt.plot(tr_losses, label = "train_loss")
plt.plot(val_losses, label = "val_loss")
plt.xlabel("Number of epochs")
plt.ylabel("Cost (J)")
plt.grid()
plt.legend()
plt.show()

plt.figure(figsize = (15,7))

plt.plot(tr_accs, label = "acc_train")
plt.plot(val_accs, label = "acc_val")
plt.xlabel("Number of epochs")
plt.ylabel("Accuracy")
plt.grid()
plt.legend()
plt.show()
```





0.1 Model evaluation on test set

[37]:
```
# Model prediction on test set

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

```
1563/1563 [==============================] - 60s 37ms/step
```

[38]:
```python
# Printing predicted classes on the test dataset

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

[38]:
```
array([0, 0, 0, …, 2, 2, 2])
```

[39]:
```python
model.save('test.h5')
```

[40]:
```python
# 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       0.97      0.97      0.97       491
           1       0.94      0.97      0.96       545
           2       0.99      0.96      0.97       527

    accuracy                           0.97      1563
   macro avg       0.97      0.97      0.97      1563
weighted avg       0.97      0.97      0.97      1563
```

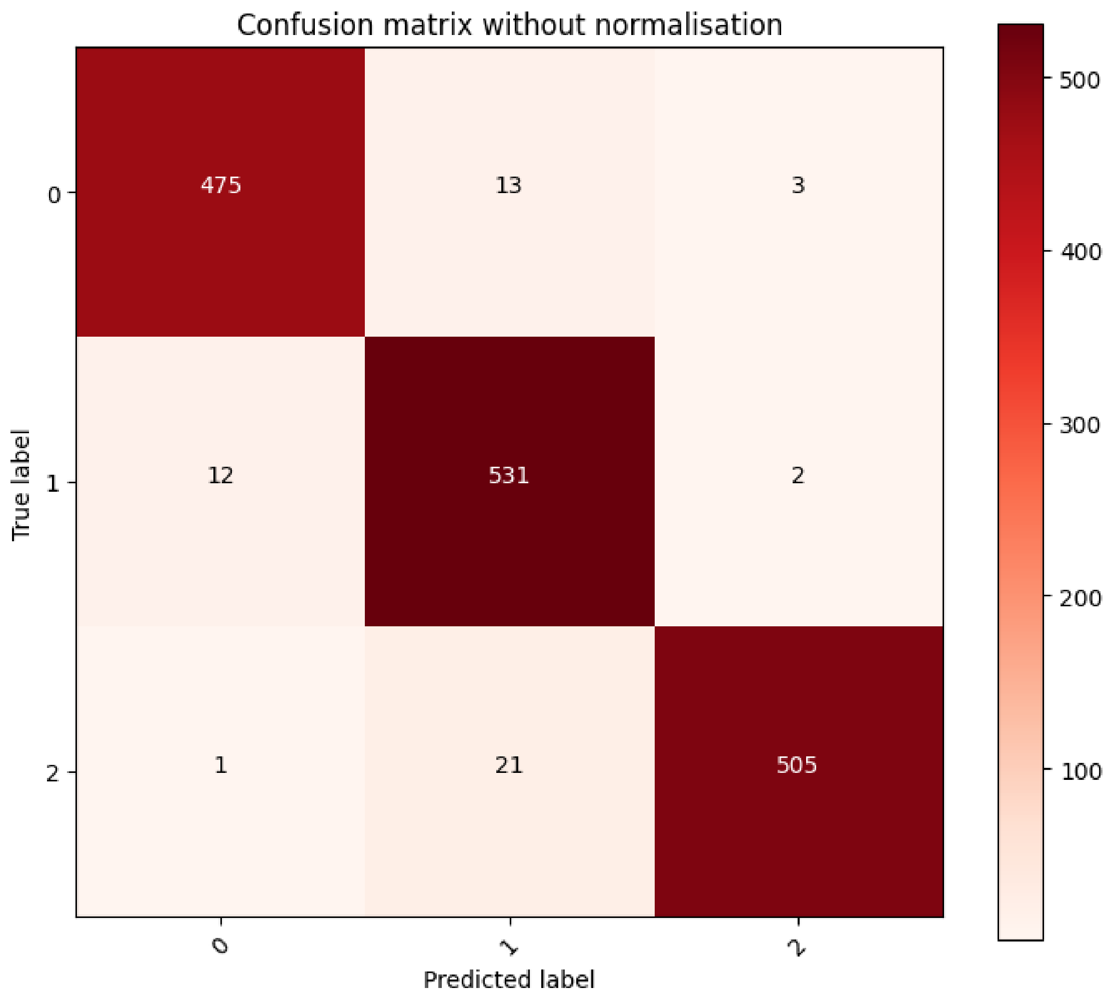
[41]:
```python
# Generating confusion matrix to see where the model is misclassifying

confusion__matrix = confusion_matrix(test_datagen.classes,
                                     predictions.squeeze().argmax(axis = 1))
print(confusion__matrix)
```

```
[[475  13   3]
 [ 12 531   2]
 [  1  21 505]]
```

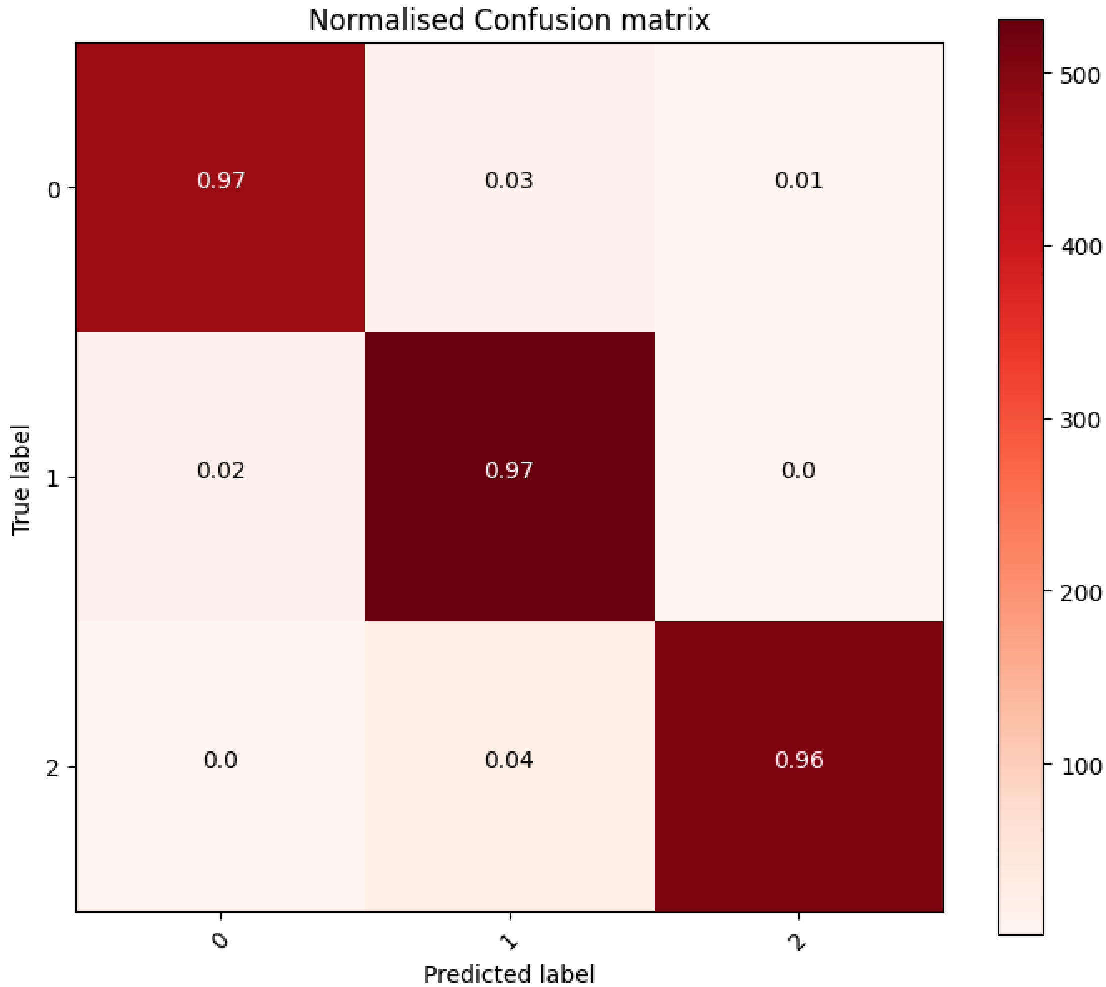
[42]:
```python
# Defining a function to print a confusion matrix
# Code snippet referenced from: https://scikit-learn.org/0.18/auto_examples/
↪model_selection/plot_confusion_matrix.html

import itertools
def plot_confusion_matrix(cm,
                          classes,
                          normalise = False,
                          title = 'Confusion matrix',
                          cmap = plt.cm.Reds):
```

```
    plt.imshow(cm, interpolation = 'nearest', cmap = cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation = 45)
    plt.yticks(tick_marks, classes)

    if normalise:
        cm = cm.astype('float') / cm.sum(axis = 1)[:, np.newaxis]
        cm = cm.round(2)

    thresh = cm.max() / 2.

    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment = "center",
                 color = "white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

[45]:
```
# Printing the confusion matrix (not normalised)

np.set_printoptions(precision = 2)
fig1 = plt.figure(figsize = (7, 6))
plot_confusion_matrix(confusion__matrix,
                      classes = np.unique(test_datagen.classes),
                      title = 'Confusion matrix without normalisation')
fig1.savefig('./cm_wo_norm.jpg')
plt.show()
```



[46]:
```python
# Printing the confusion matrix (normalised)

np.set_printoptions(precision = 2)
fig2 = plt.figure(figsize = (7,6))
plot_confusion_matrix(confusion__matrix,
                      classes = np.unique(test_datagen.classes),
                      normalise = True,
                      title = 'Normalised Confusion matrix')
fig2.savefig('./cm_norm.jpg')
plt.show()
```


Normalised Confusion matrix
0.97
0.03
0.01
0.02
0.97
0.0
0.0
0.04
0.96
0
1
2
True label
0
1
2
Predicted label
500
400
300
200
100

[ ]:

[ ]: