Sleep Apnea detection using Deep Learning Methodologies - Configuration Manual

MSc Research Project
Data Analytics

Gunjit Jain
Student ID: x20251432

School of Computing
National College of Ireland

Supervisor: Mr. Vladimir Milosavljevic

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Gunjit Jain
Student ID:	x20251432
Programme:	Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Mr. Vladimir Milosavljevic
Submission Due Date:	19/09/2022
Project Title:	Sleep Apnea detection using Deep Learning Methodologies - Configuration Manual
Word Count:	XXX
Page Count:	11

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	19th September 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Sleep Apnea detection using Deep Learning Methodologies - Configuration Manual

Gunjit Jain
x20251432

1 Introduction

The environmental setup utilized during the course of this research project ”Sleep Apnea detection using Deep Learning Methodologies” is fully described in this configuration manual. This is followed by the walk-through of the implementation and the results. Section 2 covers hardware and software configurations used for development, Section 3 covers the data sources in the description, and Section 4 covers the implementation of the models.

2 System Specifications

2.1 Hardware specifications

Table 1 shows the details of the hardware used for the development of the model.

Table 1: Details of Hardware used

Features	**Versions**
Operating System	Microsoft Windows 10 Home
Processor	Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz
System Type	64- bit Operating System, x64-based PC
RAM	16 GB
Hard Disk	1 TB

2.2 Software specifications

This research was done on the Google Colabarotory environment for the development and execution of the models. The programming was done in the Python programming language. The latest python version is used in the Google Colab platform. The important libraries used for this research project are mentioned below.

- Matplotlib
- Numpy
- Tensorflow

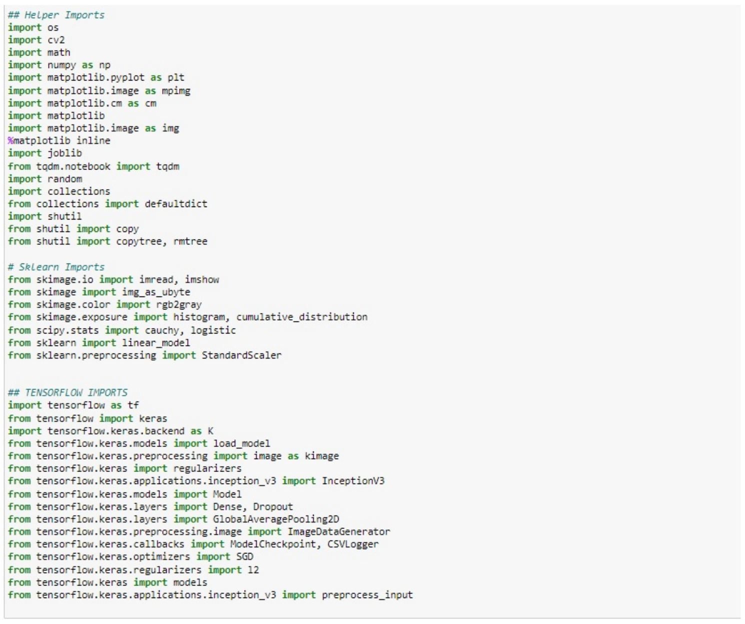
- Keras
- wfdb
- sklearn

3 Data Sources

The Apnea-ECG database used for this research is publically available on the PhysioNet website. This dataset contains overnight polysomnography (PSG) data for 70 people. This data contains readings for heart rate in the form of single-lead ECG and respiratory signals including SpO2 data. Each recording comes with a number of files. Digitized ECGs (100 samples per second, 16 bits per sample) are stored in files with names of the kind rnn, dat. The accompanying signal files' names and formats are specified in the .hea files, which are (text) header files. The .apn files are (binary) annotation files, and each minute of each recording has an annotation for each minute indicating whether or not there was an apnea at that moment.

4 Implementation

The ECG signals were divided into one-minute sections and associated with their relevant annotation. The data for all the 70 persons available in the dataset were used in this research. Figure 1 shows all the libraries used in this research.

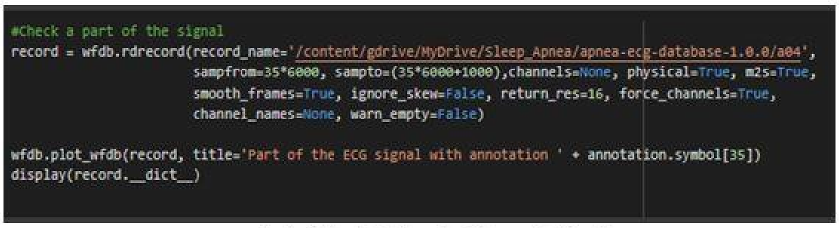


```python
## Helper Imports
import os
import cv2
import math
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import matplotlib.cm as cm
import matplotlib
import matplotlib.image as img
%matplotlib inline
import joblib
from tqdm.notebook import tqdm
import random
import collections
from collections import defaultdict
import shutil
from shutil import copy
from shutil import copytree, rmtree

# Sklearn Imports
from skimage.io import imread, imshow
from skimage import img_as_ubyte
from skimage.color import rgb2gray
from skimage.exposure import histogram, cumulative_distribution
from scipy.stats import cauchy, logistic
from sklearn import linear_model
from sklearn.preprocessing import StandardScaler


## TENSORFLOW IMPORTS
import tensorflow as tf
from tensorflow import keras
import tensorflow.keras.backend as K
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image as kimage
from tensorflow.keras import regularizers
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.layers import GlobalAveragePooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint, CSVLogger
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.regularizers import l2
from tensorflow.keras import models
from tensorflow.keras.applications.inception_v3 import preprocess_input
```

Figure 1: Libraries used in this research

4.1 Data Preprocessing

The dataset was loaded into Google drive and the drive was mounted into Google Colab to use the dataset. The signals present in the dataset are seven to ten hours long. The first minute of each signal is annotated as Normal(N) so the first minute is not considered. The one-minute signal sample is shown in Figure 2 for both annotations.



```
#Check a part of the signal
record = wfdb.rdrecord(record_name='/content/gdrive/MyDrive/Sleep_Apnea/apnea-ecg-database-1.0.0/a04',
                       sampfrom=35*6000, sampto=(35*6000+1000),channels=None, physical=True, m2s=True,
                       smooth_frames=True, ignore_skew=False, return_res=16, force_channels=True,
                       channel_names=None, warn_empty=False)

wfdb.plot_wfdb(record, title='Part of the ECG signal with annotation ' + annotation.symbol[35])
display(record.__dict__)
```

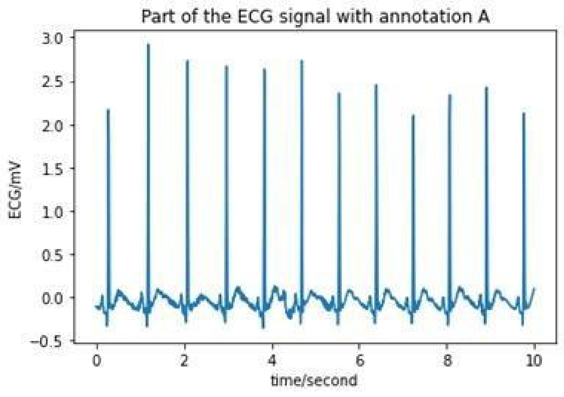
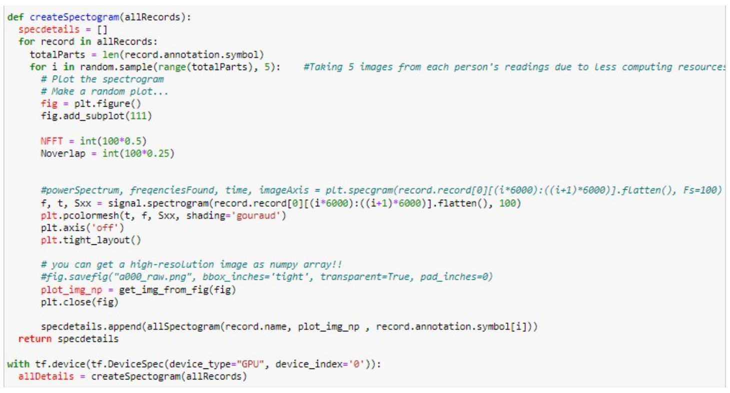


Figure 2: Sample for a part of the ECG signal

4.2 Data Transformation

The one-minute signals have a sampling frequency of 100 Hz was converted into spectogram using Fast Fourier transfer. These spectograms were then used to feed the neural network. The code used to create the spectogram is shown in Figure 3 along with the sample spectogram shown in Figure 4.



```
def createSpectogram(allRecords):
  specdetails = []
  for record in allRecords:
    totalParts = len(record.annotation.symbol)
    for i in random.sample(range(totalParts), 5):    #Taking 5 images from each person's readings due to less computing resource
      # Plot the spectrogram
      # Make a random plot...
      fig = plt.figure()
      fig.add_subplot(111)

      NFFT = int(100*0.5)
      Noverlap = int(100*0.25)


      #powerSpectrum, freqenciesFound, time, imageAxis = plt.specgram(record.record[0][(i*6000):((i+1)*6000)].flatten(), Fs=100)
      f, t, Sxx = signal.spectrogram(record.record[0][(i*6000):((i+1)*6000)].flatten(), 100)
      plt.pcolormesh(t, f, Sxx, shading='gouraud')
      plt.axis('off')
      plt.tight_layout()

      # you can get a high-resolution image as numpy array!!
      #fig.savefig("a000_raw.png", bbox_inches='tight', transparent=True, pad_inches=0)
      plot_img_np = get_img_from_fig(fig)
      plt.close(fig)

      specdetails.append(allSpectogram(record.name, plot_img_np , record.annotation.symbol[i]))
  return specdetails

with tf.device(tf.DeviceSpec(device_type="GPU", device_index='0')):
  allDetails = createSpectogram(allRecords)
```

Figure 3: Code for spectogram

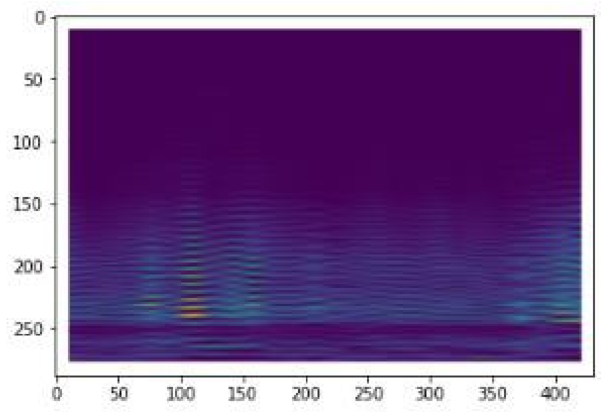
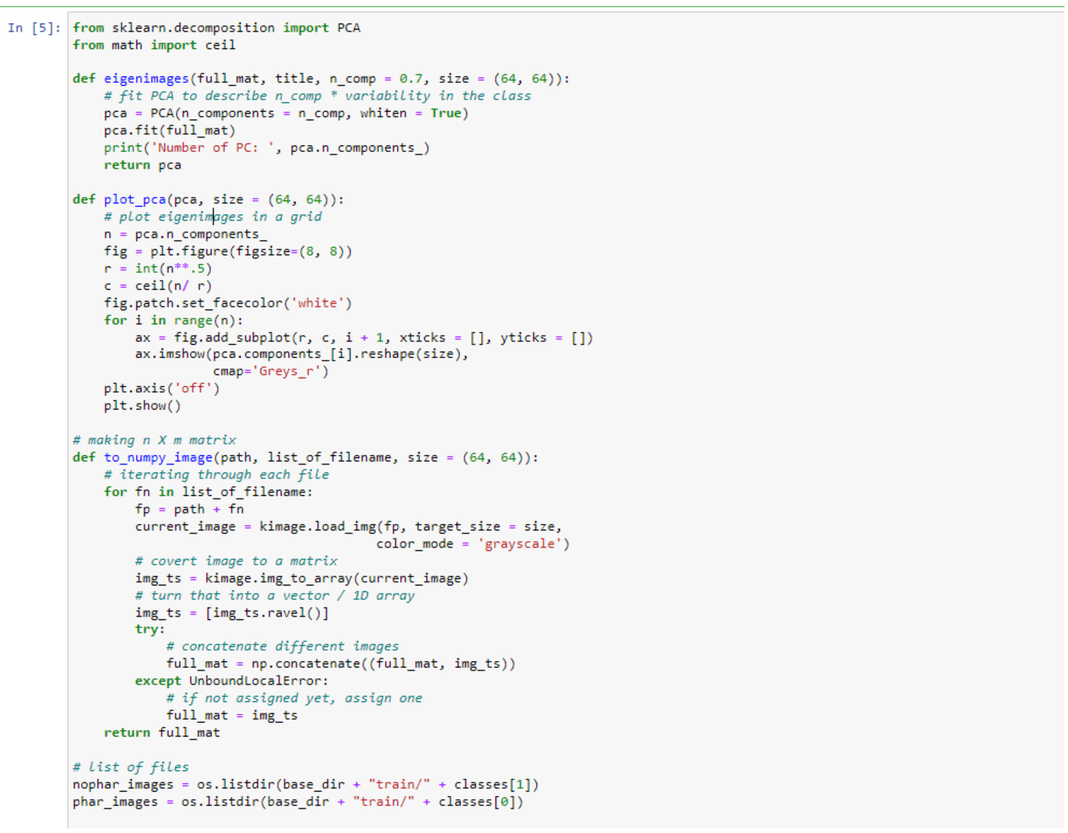


Figure 4: Spectogram for a part of signal

5 Model Implementation

5.1 Model 1 Baseline Convolution Neural Network

These components are used to visualize the better representation of each class. For data visualization, different techniques are used like principle component analysis. The spectrograms are converted into the Eigenvalues shown in Figure 6 which are further reshaped into the matrix.



```
In [5]: from sklearn.decomposition import PCA
        from math import ceil

        def eigenimages(full_mat, title, n_comp = 0.7, size = (64, 64)):
            # fit PCA to describe n_comp * variability in the class
            pca = PCA(n_components = n_comp, whiten = True)
            pca.fit(full_mat)
            print('Number of PC: ', pca.n_components_)
            return pca

        def plot_pca(pca, size = (64, 64)):
            # plot eigenimages in a grid
            n = pca.n_components_
            fig = plt.figure(figsize=(8, 8))
            r = int(n**.5)
            c = ceil(n/ r)
            fig.patch.set_facecolor('white')
            for i in range(n):
                ax = fig.add_subplot(r, c, i + 1, xticks = [], yticks = [])
                ax.imshow(pca.components_[i].reshape(size),
                          cmap='Greys_r')
            plt.axis('off')
            plt.show()

        # making n X m matrix
        def to_numpy_image(path, list_of_filename, size = (64, 64)):
            # iterating through each file
            for fn in list_of_filename:
                fp = path + fn
                current_image = kimage.load_img(fp, target_size = size,
                                                color_mode = 'grayscale')
                # covert image to a matrix
                img_ts = kimage.img_to_array(current_image)
                # turn that into a vector / 1D array
                img_ts = [img_ts.ravel()]
                try:
                    # concatenate different images
                    full_mat = np.concatenate((full_mat, img_ts))
                except UnboundLocalError:
                    # if not assigned yet, assign one
                    full_mat = img_ts
            return full_mat

        # list of files
        nophar_images = os.listdir(base_dir + "train/" + classes[1])
        phar_images = os.listdir(base_dir + "train/" + classes[0])
```

Figure 5: Code to form PCA using Eigen

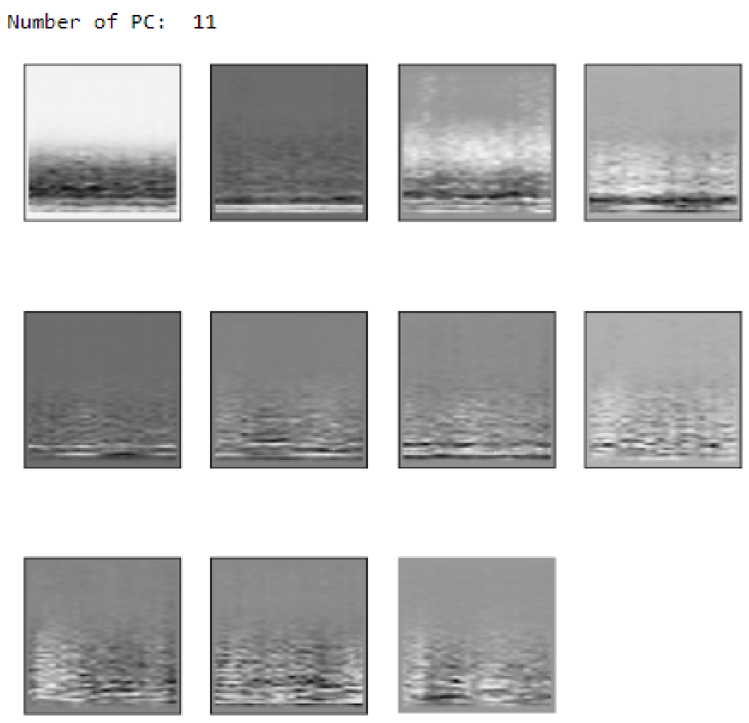
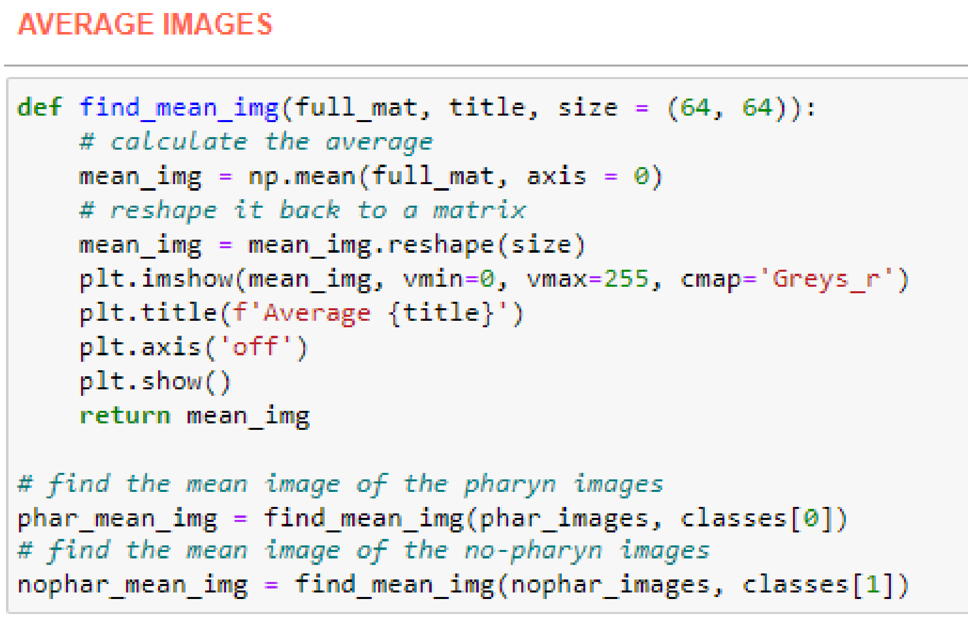


Figure 6: Eigen Images for principle component

These principle components formed using Eigenvalues are shown in Figure 8 shows the average Eigen images for both Apnea and non-apnea type along with the difference between normal and apnea patient's averages.


AVERAGE IMAGES

```python
def find_mean_img(full_mat, title, size = (64, 64)):
    # calculate the average
    mean_img = np.mean(full_mat, axis = 0)
    # reshape it back to a matrix
    mean_img = mean_img.reshape(size)
    plt.imshow(mean_img, vmin=0, vmax=255, cmap='Greys_r')
    plt.title(f'Average {title}')
    plt.axis('off')
    plt.show()
    return mean_img

# find the mean image of the pharyn images
phar_mean_img = find_mean_img(phar_images, classes[0])
# find the mean image of the no-pharyn images
nophar_mean_img = find_mean_img(nophar_images, classes[1])
```

Figure 7: Code to generate the average of Eigen images

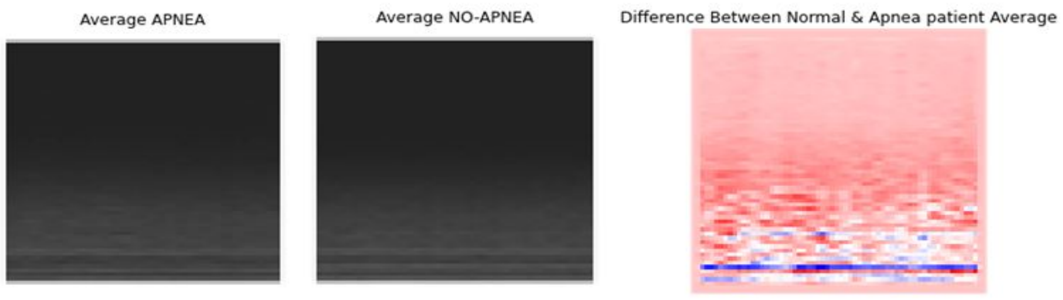
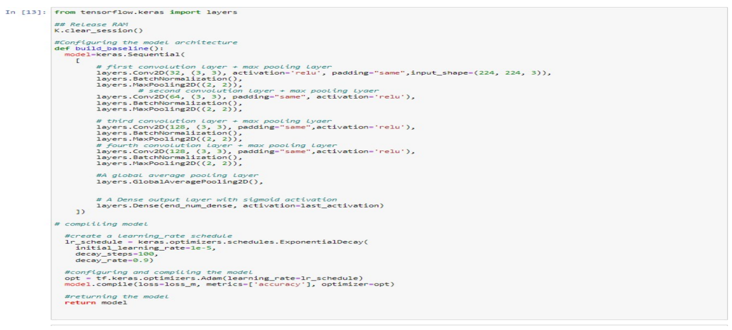


Figure 8: Average of Eigen images and their differences

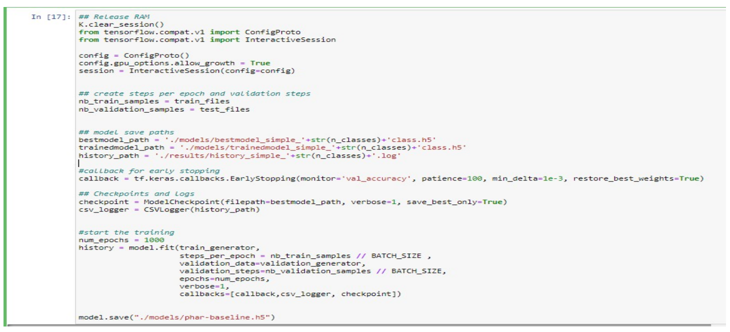
In the base convolution neural network model there is a total of 32 filters and channels. In the model, the output layer has one neuron and the sigmoid activation function because it is a binary classification problem. In the results, the output of the binary classification is one class versus the other class. Batch normalization is used in each layer for fast learning and enhanced generalization. Figure 9 below code is for the model building of baseline convolution neural network.



```
In [13]: from tensorflow.keras import layers

         ## Release RAM
         K.clear_session()

         #Configuring the model architecture
         def build_baseline():
           model=keras.Sequential(
             [
                 # first convolution layer + max pooling layer
                 layers.Conv2D(32, (3, 3), activation='relu', padding="same",input_shape=(224, 224, 3)),
                 layers.BatchNormalization(),
                 layers.MaxPooling2D((2, 2)),
                         # second convolution layer + max pooling lyaer
                 layers.Conv2D(64, (3, 3), padding="same", activation='relu'),
                 layers.BatchNormalization(),
                 layers.MaxPooling2D((2, 2)),

                 # third convolution layer + max pooling lyaer
                 layers.Conv2D(128, (3, 3), padding="same",activation='relu'),
                 layers.BatchNormalization(),
                 layers.MaxPooling2D((2, 2)),
                 # fourth convolution layer + max pooling layer
                 layers.Conv2D(128, (3, 3), padding="same",activation='relu'),
                 layers.BatchNormalization(),
                 layers.MaxPooling2D((2, 2)),

                 #A global average pooling layer
                 layers.GlobalAveragePooling2D(),


                 # A Dense output layer with sigmoid activation
                 layers.Dense(end_num_dense, activation=last_activation)
             ])

         # compliling model

           #create a learning_rate schedule
           lr_schedule = keras.optimizers.schedules.ExponentialDecay(
             initial_learning_rate=1e-5,
             decay_steps=100,
             decay_rate=0.9)

           #configuring and compiling the model
           opt = tf.keras.optimizers.Adam(learning_rate=lr_schedule)
           model.compile(loss=loss_m, metrics=['accuracy'], optimizer=opt)

           #returning the model
           return model
```

Figure 9: Model building of Baseline convolution neural network

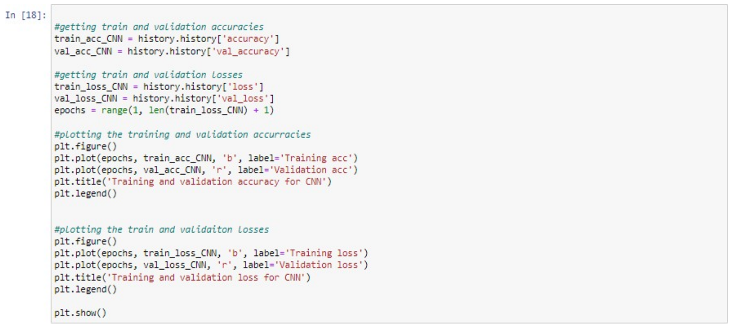
In Figure 9, while model compiling the Adams optimizer is used and the loss function is binary cross-entropy.



```
In [17]: ## Release RAM
         K.clear_session()
         from tensorflow.compat.v1 import ConfigProto
         from tensorflow.compat.v1 import InteractiveSession

         config = ConfigProto()
         config.gpu_options.allow_growth = True
         session = InteractiveSession(config=config)


         ## create steps per epoch and validation steps
         nb_train_samples = train_files
         nb_validation_samples = test_files


         ## model save paths
         bestmodel_path = './models/bestmodel_simple_'+str(n_classes)+'class.h5'
         trainedmodel_path = './models/trainedmodel_simple_'+str(n_classes)+'class.h5'
         history_path = './results/history_simple_'+str(n_classes)+'.log'

         #callback for early stopping
         callback = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=100, min_delta=1e-3, restore_best_weights=True)

         ## Checkpoints and Logs
         checkpoint = ModelCheckpoint(filepath=bestmodel_path, verbose=1, save_best_only=True)
         csv_logger = CSVLogger(history_path)


         #start the training
         num_epochs = 1000
         history = model.fit(train_generator,
                             steps_per_epoch = nb_train_samples // BATCH_SIZE ,
                             validation_data=validation_generator,
                             validation_steps=nb_validation_samples // BATCH_SIZE,
                             epochs=num_epochs,
                             verbose=1,
                             callbacks=[callback,csv_logger, checkpoint])


         model.save("./models/phar-baseline.h5")
```

Figure 10: Code for Model training

In Figure 10 the model is trained for the 1000 epoch for train generator data and among them, the best model is saved. After the model training, the accuracy of the model is increasing over time and it reaches 89%. The accuracy of the validation data is between 72-83%. The training loss continuously decreases.



```
In [18]:
#getting train and validation accuracies
train_acc_CNN = history.history['accuracy']
val_acc_CNN = history.history['val_accuracy']

#getting train and validation losses
train_loss_CNN = history.history['loss']
val_loss_CNN = history.history['val_loss']
epochs = range(1, len(train_loss_CNN) + 1)

#plotting the training and validation accurracies
plt.figure()
plt.plot(epochs, train_acc_CNN, 'b', label='Training acc')
plt.plot(epochs, val_acc_CNN, 'r', label='Validation acc')
plt.title('Training and validation accuracy for CNN')
plt.legend()


#plotting the train and validaiton losses
plt.figure()
plt.plot(epochs, train_loss_CNN, 'b', label='Training loss')
plt.plot(epochs, val_loss_CNN, 'r', label='Validation loss')
plt.title('Training and validation loss for CNN')
plt.legend()

plt.show()
```

Figure 11: Code of Results of baseline CNN

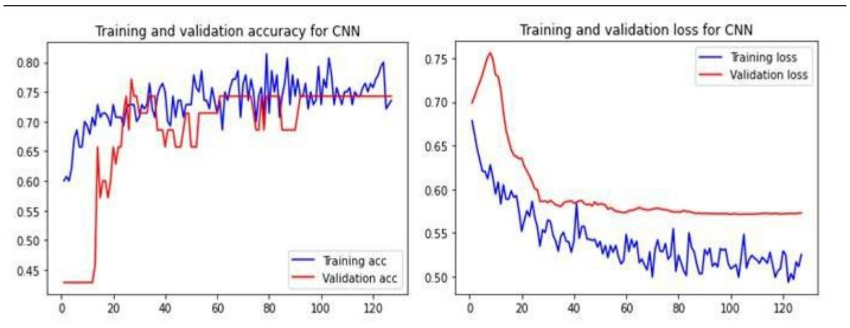


Figure 12: Accuracy and Loss graph of baseline CNN Model

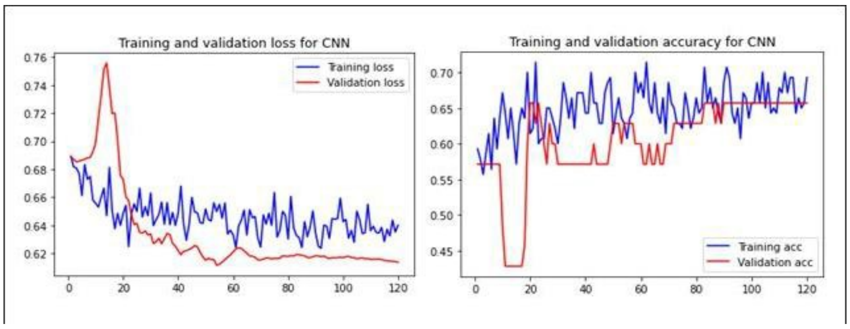
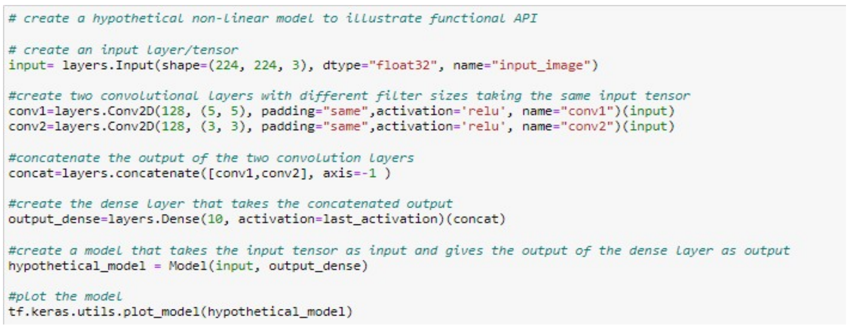


Figure 13: Accuracy and Loss graph of baseline CNN Model with augmented data

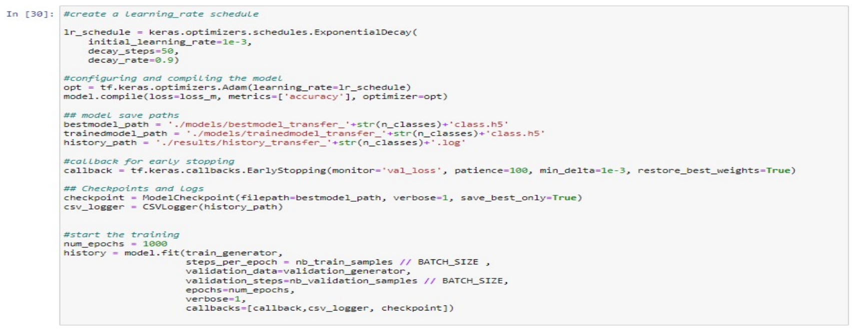
5.2 Model 2 DenseNet121 with CNN

In the hybrid CNN model, a convolution neural network is designed on top of the DenseNet121 network as shown in Figure 14. For better results and accuracy the pre-trained DenseNet121 model is designed with CNN. In the CNN model, the two convolutions 2D layers are used and concatenated in the next layer.



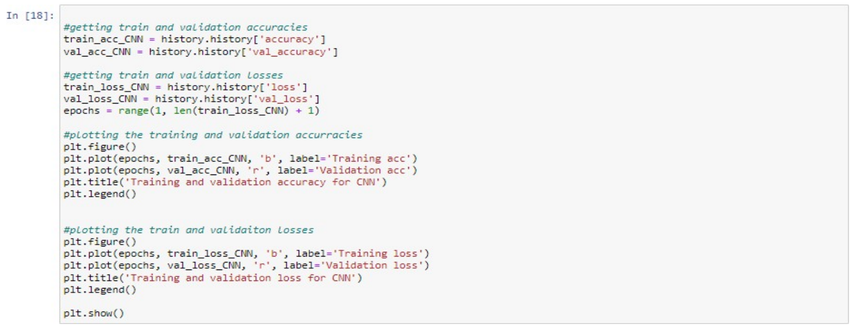
```
# create a hypothetical non-linear model to illustrate functional API

# create an input layer/tensor
input= layers.Input(shape=(224, 224, 3), dtype="float32", name="input_image")

#create two convolutional layers with different filter sizes taking the same input tensor
conv1=layers.Conv2D(128, (5, 5), padding="same",activation='relu', name="conv1")(input)
conv2=layers.Conv2D(128, (3, 3), padding="same",activation='relu', name="conv2")(input)

#concatenate the output of the two convolution layers
concat=layers.concatenate([conv1,conv2], axis=-1 )

#create the dense layer that takes the concatenated output
output_dense=layers.Dense(10, activation=last_activation)(concat)

#create a model that takes the input tensor as input and gives the output of the dense layer as output
hypothetical_model = Model(input, output_dense)

#plot the model
tf.keras.utils.plot_model(hypothetical_model)
```

Figure 14: Model building of hybrid CNN model



```
In [30]: #create a learning_rate schedule

         lr_schedule = keras.optimizers.schedules.ExponentialDecay(
             initial_learning_rate=1e-3,
             decay_steps=50,
             decay_rate=0.9)

         #configuring and compiling the model
         opt = tf.keras.optimizers.Adam(learning_rate=lr_schedule)
         model.compile(loss=loss_m, metrics=['accuracy'], optimizer=opt)

         ## model save paths
         bestmodel_path = './models/bestmodel_transfer_'+str(n_classes)+'class.h5'
         trainedmodel_path = './models/trainedmodel_transfer_'+str(n_classes)+'class.h5'
         history_path = './results/history_transfer_'+str(n_classes)+'.log'

         #callback for early stopping
         callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=100, min_delta=1e-3, restore_best_weights=True)

         ## Checkpoints and logs
         checkpoint = ModelCheckpoint(filepath=bestmodel_path, verbose=1, save_best_only=True)
         csv_logger = CSVLogger(history_path)


         #start the training
         num_epochs = 1000
         history = model.fit(train_generator,
                             steps_per_epoch = nb_train_samples // BATCH_SIZE ,
                             validation_data=validation_generator,
                             validation_steps=nb_validation_samples // BATCH_SIZE,
                             epochs=num_epochs,
                             verbose=1,
                             callbacks=[callback,csv_logger, checkpoint])
```

Figure 15: Code for Model training of hybrid CNN model



```
In [18]:
         #getting train and validation accuracies
         train_acc_CNN = history.history['accuracy']
         val_acc_CNN = history.history['val_accuracy']

         #getting train and validation losses
         train_loss_CNN = history.history['loss']
         val_loss_CNN = history.history['val_loss']
         epochs = range(1, len(train_loss_CNN) + 1)

         #plotting the training and validation accurracies
         plt.figure()
         plt.plot(epochs, train_acc_CNN, 'b', label='Training acc')
         plt.plot(epochs, val_acc_CNN, 'r', label='Validation acc')
         plt.title('Training and validation accuracy for CNN')
         plt.legend()


         #plotting the train and validaiton losses
         plt.figure()
         plt.plot(epochs, train_loss_CNN, 'b', label='Training loss')
         plt.plot(epochs, val_loss_CNN, 'r', label='Validation loss')
         plt.title('Training and validation loss for CNN')
         plt.legend()

         plt.show()
```

Figure 16: Code for Results of baseline CNN

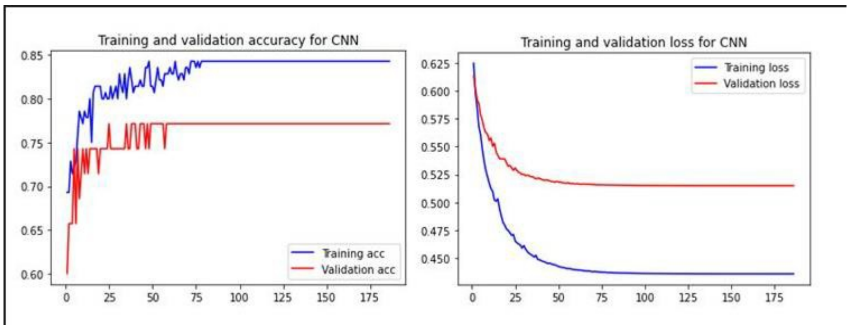


Figure 17: Code for Model training of hybrid CNN model

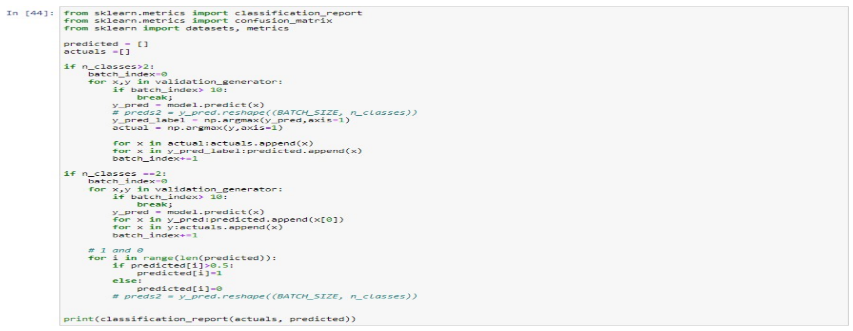
In the results of the model shown in Figure 17, the overfitting problem has been addressed to a great extent. After that, the fine-tuning of the model by freezing and unfreezing some of the layers, the accuracy on the train data came to a little less than 1.00 while the test accuracy came out to be 80%.

6 Evaluation

Accuracy is the ability of a classification model to accurately classify the data. In medical situations, it is crucial that an individual who is unwell be accurately classified; but, if some individuals who are not ill are also recorded as positive, there may be a problem. Therefore, it may be claimed that accuracy is more significant in medical instances. As many researchers in the past have used accuracy to show the effectiveness of various models

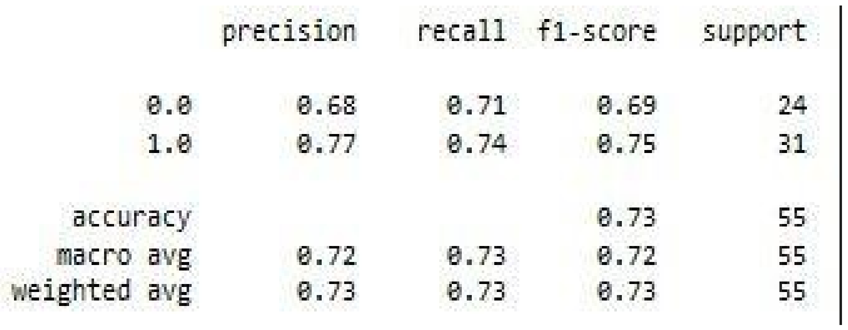
6.1 Baseline Convolution Neural Network

In the baseline convolution, neural network model the accuracy of the overall research is 73%, F1-score was 0.75. The Evaluation matrix and confusion matrix are shown in the Figure 19



```
In [44]: from sklearn.metrics import classification_report
         from sklearn.metrics import confusion_matrix
         from sklearn import datasets, metrics

         predicted = []
         actuals =[]

         if n_classes>2:
             batch_index=0
             for x,y in validation_generator:
                 if batch_index> 10:
                     break;
                 y_pred = model.predict(x)
                 # preds2 = y_pred.reshape((BATCH_SIZE, n_classes))
                 y_pred_label = np.argmax(y_pred,axis=1)
                 actual = np.argmax(y,axis=1)

                 for x in actual:actuals.append(x)
                 for x in y_pred_label:predicted.append(x)
                 batch_index+=1

         if n_classes ==2:
             batch_index=0
             for x,y in validation_generator:
                 if batch_index> 10:
                     break;
                 y_pred = model.predict(x)
                 for x in y_pred:predicted.append(x[0])
                 for x in y:actuals.append(x)
                 batch_index+=1

             # 1 and 0
             for i in range(len(predicted)):
                 if predicted[i]>0.5:
                     predicted[i]=1
                 else:
                     predicted[i]=0
                 # preds2 = y_pred.reshape((BATCH_SIZE, n_classes))

         print(classification_report(actuals, predicted))
```

Figure 18: Code to Generate the Classification Report of Model 1

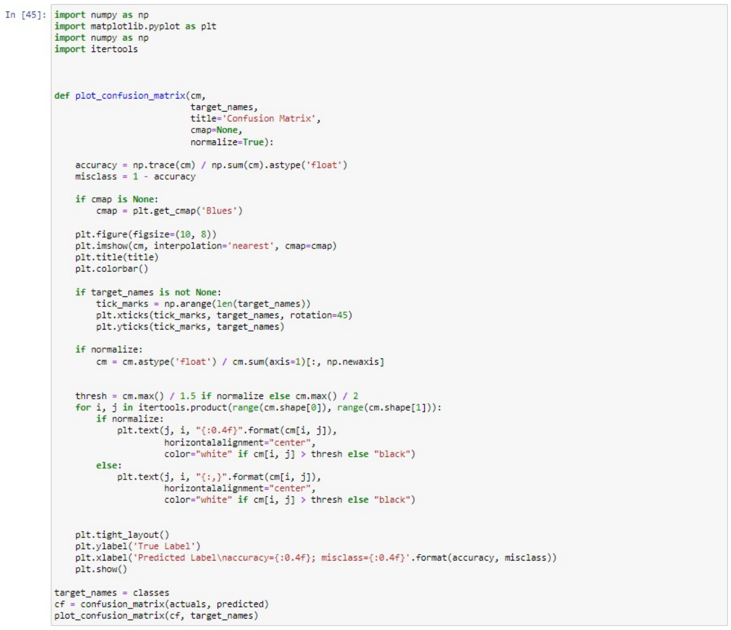


```
              precision    recall  f1-score   support

         0.0       0.68      0.71      0.69        24
         1.0       0.77      0.74      0.75        31

    accuracy                           0.73        55
   macro avg       0.72      0.73      0.72        55
weighted avg       0.73      0.73      0.73        55
```

Figure 19: Classification Report of the Baseline Convolution Neural Network Model.

6.2 Baseline Convolution Neural Network after Data Augmentation

In this instance, the data was enhanced using a data augmentation technique before being fed to the CNN model. However, due to data augmentation's slight addition of noise to the signals data, accuracy fell to 69%. So, for the purpose of the final evaluation, the aforesaid model was used. The code for the generation of the confusion matrix is shown in Figure 20. The confusion matrix is shown in Figure 21.



```
In [45]: import numpy as np
         import matplotlib.pyplot as plt
         import numpy as np
         import itertools



         def plot_confusion_matrix(cm,
                                   target_names,
                                   title='Confusion Matrix',
                                   cmap=None,
                                   normalize=True):

             accuracy = np.trace(cm) / np.sum(cm).astype('float')
             misclass = 1 - accuracy

             if cmap is None:
                 cmap = plt.get_cmap('Blues')

             plt.figure(figsize=(10, 8))
             plt.imshow(cm, interpolation='nearest', cmap=cmap)
             plt.title(title)
             plt.colorbar()

             if target_names is not None:
                 tick_marks = np.arange(len(target_names))
                 plt.xticks(tick_marks, target_names, rotation=45)
                 plt.yticks(tick_marks, target_names)

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


             thresh = cm.max() / 1.5 if normalize else cm.max() / 2
             for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
                 if normalize:
                     plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                              horizontalalignment="center",
                              color="white" if cm[i, j] > thresh else "black")
                 else:
                     plt.text(j, i, "{:,}".format(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\naccuracy={:0.4f}; misclass={:0.4f}'.format(accuracy, misclass))
             plt.show()

         target_names = classes
         cf = confusion_matrix(actuals, predicted)
         plot_confusion_matrix(cf, target_names)
```

Figure 20: Code to Generate the Classification Report of Model 1 with Augmentation

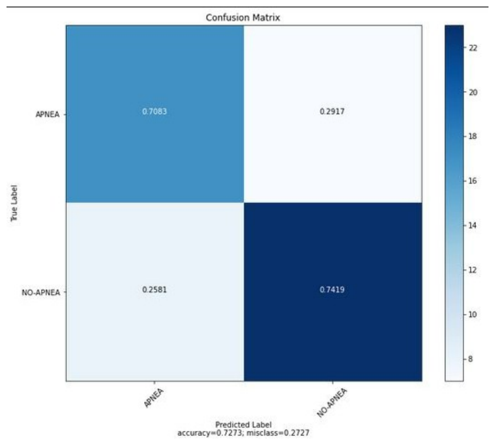
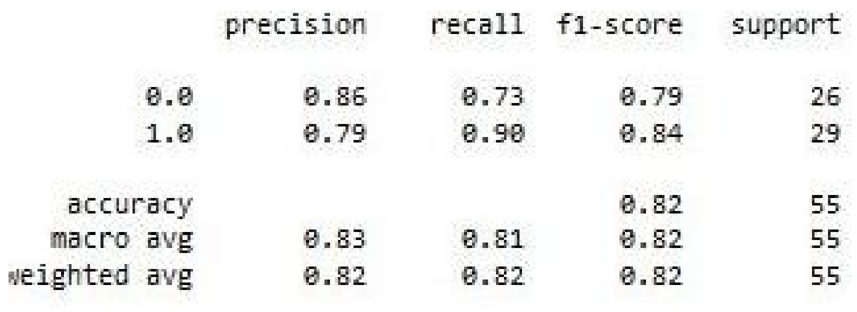


Figure 21: Confusion Matrix of Model 1

6.3 DenseNet121 + CNN

The Hybrid Convolutional Neural Network with DenseNet121+CNN that was created for this research had a training accuracy of about 85% as shown in Figure 22, but after some fine-tuning, it performed remarkably well. So, the refined model was taken into consideration for the initial assessment.



```
              precision    recall  f1-score   support

         0.0       0.86      0.73      0.79        26
         1.0       0.79      0.90      0.84        29

    accuracy                           0.82        55
   macro avg       0.83      0.81      0.82        55
weighted avg       0.82      0.82      0.82        55
```

Figure 22: Confusion Matrix of Model 2

6.4 DenseNet121+ CNN after fine-tuning

By briefly freezing and unfreezing some layers, the Hybrid CNN was adjusted. The final results were superior to the initial ones. Here, the validation accuracy was about 77%, and the training accuracy was over 99%. After adjustments, this model's total accuracy was 81.82% and its F1 score was 0.84. Figure 23, the additional evaluation metrics are displayed. According to the confusion matrix in Figure 22, 73.08% of people are actually classified as having apnea, and 89.66% of people are actually classed as not having apnea.

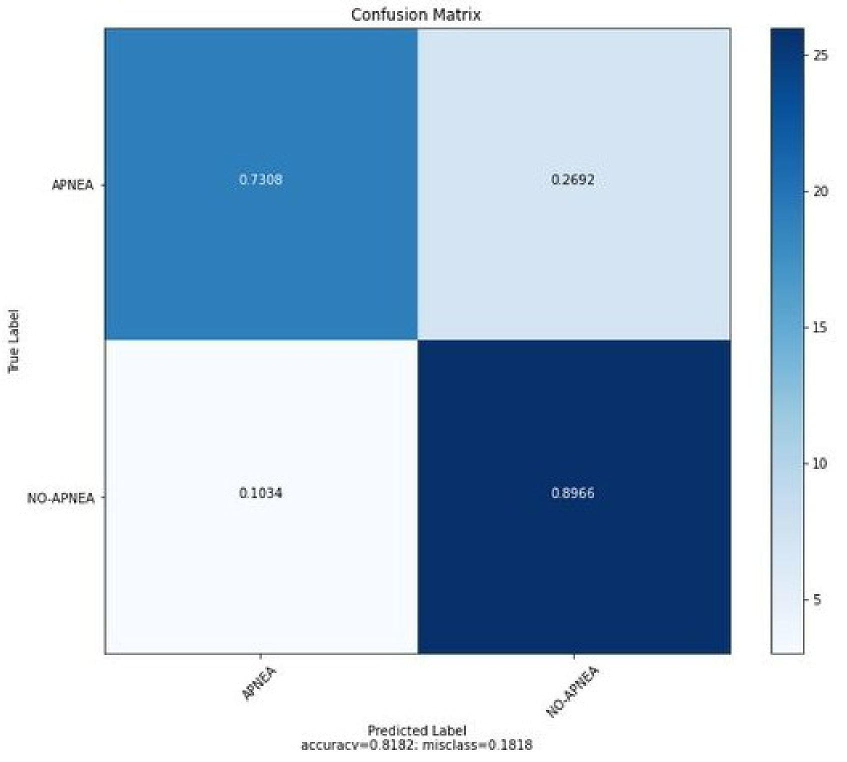


Figure 23: Confusion Matrix of Model 2