ATRIAL FIBRILLATION DETECTION USING MACHINE LEARNING

PLATFORM used: Google Colab

SOURCE CODE:

Code for Model Training

!pip install wfdb

import pandas as pd

import numpy as np

import wfdb

import random

import seaborn as sns

import tensorflow as tf

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras import layers, losses

from tensorflow.keras.layers import Input, Conv1D, MaxPooling1D, BatchNormalization,concatenate,Conv1DTranspose,Dropout

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.layers import Activation

from tensorflow.keras.losses import binary\_crossentropy

from tensorflow.keras.models import Model

from tensorflow.keras import layers

from tensorflow.keras.models import Model

from wfdb import processing

random.seed(42)

import warnings

warnings.filterwarnings("ignore")

from google.colab import drive

drive.mount('/content/drive')

data = '/content/drive/My Drive/Final/atrial-fib/files/'

def load\_ecg(file,filetype):

    record = wfdb.rdrecord(file)

    annotation = wfdb.rdann(file, filetype)

    p\_signal = record.p\_signal

    atr\_sym = annotation.symbol

    atr\_sample = annotation.sample

    return p\_signal, atr\_sym, atr\_sample

def af\_data(qrs\_sample,atr\_sample):

    AF\_samples\_array = []

    NonAF\_samples\_array = []

    for i in range (len(atr\_sample)):

        if  (i % 2 == 0) & (i < len(atr\_sample)-1) :

            NonAF\_samples = qrs\_sample[( qrs\_sample >= atr\_sample[i] ) & (qrs\_sample < atr\_sample[i+1])  ]

            NonAF\_samples\_array.append(NonAF\_samples)

        if  (i % 2 == 1) & (i < len(atr\_sample)-1) :

            AF\_samples = qrs\_sample[(qrs\_sample >= atr\_sample[i] ) & (qrs\_sample < atr\_sample[i+1])  ]

            AF\_samples\_array.append(AF\_samples)

        if  (i % 2 == 0)  & (i == len(atr\_sample)-1) :

            NonAF\_samples = qrs\_sample [ qrs\_sample > atr\_sample[i] ]

            NonAF\_samples\_array.append(NonAF\_samples)

        if  (i % 2 == 1)  & (i == len(atr\_sample)-1) :

            AF\_samples = qrs\_sample [ qrs\_sample > atr\_sample[i] ]

            AF\_samples\_array.append(AF\_samples)

    return np.array(AF\_samples\_array,dtype='object'),np.array(NonAF\_samples\_array,dtype='object')

patients=['04015','04048','04126','04746','04908','04936','05091','05261','06453','06995']

y\_128 = []

X\_array\_128 = []

for pt in patients:

    file = data + pt

    p\_signal, atr\_sym, atr\_sample = load\_ecg(file,'atr')

    p\_signal, qrs\_sym, qrs\_sample = load\_ecg(file,'qrs')

    AF\_samples\_array, NonAF\_samples\_array = af\_data(qrs\_sample,atr\_sample)

    if not (pt == '07162' or pt == '07859'):

        AF\_samples\_array = np.concatenate(AF\_samples\_array,axis=0)

        NonAF\_samples\_array = np.concatenate(NonAF\_samples\_array,axis=0)

    totalsamples = np.concatenate((AF\_samples\_array,NonAF\_samples\_array),axis=0)

    p\_signal = p\_signal[:,0]

    for i in range (1,np.size(totalsamples)-3,1):

        if totalsamples[i]>= 750:

            if (totalsamples[i] in AF\_samples\_array):

                X  = p\_signal[(totalsamples[i]-64):(totalsamples[i]+64)]

                X\_array\_128.append(X)

                y\_128.append(np.ones((128,)))

        if totalsamples[i]>= 750:

            if (totalsamples[i] in NonAF\_samples\_array):

                X  = p\_signal[(totalsamples[i]-64):(totalsamples[i]+64)]

                X\_array\_128.append(X)

                y\_128.append(np.zeros((128,)))

p\_signal.shape

print(np.array(X\_array\_128).shape)

print(np.array(y\_128).shape)

plt.figure(figsize=(20, 9))

plt.grid()

plt.plot(X\_array\_128[50] ,color='r')

plt.plot(y\_128[50],color='b')

plt.show()

X\_array\_128 = np.array(X\_array\_128).reshape((np.array(X\_array\_128).shape[0],128,1))

y\_128 = np.array(y\_128).reshape((np.array(y\_128).shape[0],128,1))

X\_train, X\_valid, y\_train, y\_valid = train\_test\_split(X\_array\_128, y\_128, test\_size=0.4, random\_state=42)

X\_valid, X\_test, y\_valid, y\_test = train\_test\_split(X\_valid, y\_valid, test\_size=0.5, random\_state=42)

print(X\_train.shape)

print(X\_valid.shape)

print(X\_test.shape)

print(y\_train.shape)

print(y\_valid.shape)

print(y\_test.shape)

def build\_unet\_model():

    inputs = tf.keras.layers.Input((128, 1))

    c1 = tf.keras.layers.Conv1D(16, (3), activation='relu', kernel\_initializer='he\_normal', padding='same')(inputs)

    c1 = tf.keras.layers.Dropout(0.1)(c1)

    c1 = tf.keras.layers.Conv1D(16, (3), activation='relu', kernel\_initializer='he\_normal', padding='same')(c1)

    p1 = tf.keras.layers.MaxPooling1D(pool\_size=(2))(c1)

    c2 = tf.keras.layers.Conv1D(32, (3), activation='relu', kernel\_initializer='he\_normal', padding='same')(p1)

    c2 = tf.keras.layers.Dropout(0.1)(c2)

    c2 = tf.keras.layers.Conv1D(32, (3), activation='relu', kernel\_initializer='he\_normal', padding='same')(c2)

    p2 = tf.keras.layers.MaxPooling1D(pool\_size=(2))(c2)

    c3 = tf.keras.layers.Conv1D(64, (3), activation='relu', kernel\_initializer='he\_normal', padding='same')(p2)

    c3 = tf.keras.layers.Dropout(0.2)(c3)

    c3 = tf.keras.layers.Conv1D(64, (3), activation='relu', kernel\_initializer='he\_normal', padding='same')(c3)

    p3 = tf.keras.layers.MaxPooling1D(pool\_size=(2))(c3)

    c4 = tf.keras.layers.Conv1D(128, (3), activation='relu', kernel\_initializer='he\_normal', padding='same')(p3)

    c4 = tf.keras.layers.Dropout(0.2)(c4)

    c4 = tf.keras.layers.Conv1D(128, (3), activation='relu', kernel\_initializer='he\_normal', padding='same')(c4)

    p4 = tf.keras.layers.MaxPooling1D(pool\_size=(2))(c4)

    c5 = tf.keras.layers.Conv1D(128, (3), activation='relu', kernel\_initializer='he\_normal', padding='same')(p4)

    c5 = tf.keras.layers.Dropout(0.3)(c5)

    c5 = tf.keras.layers.Conv1D(128, (3), activation='relu', kernel\_initializer='he\_normal', padding='same')(c5)

    u6 = tf.keras.layers.Conv1DTranspose(128, (2), strides=(2), padding='same')(c5)

    u6 = tf.keras.layers.concatenate([u6, c4])

    c6 = tf.keras.layers.Conv1D(128, (3), activation='relu', kernel\_initializer='he\_normal', padding='same')(u6)

    c6 = tf.keras.layers.Dropout(0.2)(c6)

    c6 = tf.keras.layers.Conv1D(128, (3), activation='relu', kernel\_initializer='he\_normal', padding='same')(c6)

    u7 = tf.keras.layers.Conv1DTranspose(64, (2), strides=(2), padding='same')(c6)

    u7 = tf.keras.layers.concatenate([u7, c3])

    c7 = tf.keras.layers.Conv1D(64, (3), activation='relu', kernel\_initializer='he\_normal', padding='same')(u7)

    c7 = tf.keras.layers.Dropout(0.2)(c7)

    c7 = tf.keras.layers.Conv1D(64, (3), activation='relu', kernel\_initializer='he\_normal', padding='same')(c7)

    u8 = tf.keras.layers.Conv1DTranspose(32, (2), strides=(2), padding='same')(c7)

    u8 = tf.keras.layers.concatenate([u8, c2])

    c8 = tf.keras.layers.Conv1D(32, (3), activation='relu', kernel\_initializer='he\_normal', padding='same')(u8)

    c8 = tf.keras.layers.Dropout(0.1)(c8)

    c8 = tf.keras.layers.Conv1D(32, (3), activation='relu', kernel\_initializer='he\_normal', padding='same')(c8)

    u9 = tf.keras.layers.Conv1DTranspose(16, (2), strides=(2), padding='same')(c8)

    u9 = tf.keras.layers.concatenate([u9, c1])

    c9 = tf.keras.layers.Conv1D(16, (3), activation='relu', kernel\_initializer='he\_normal', padding='same')(u9)

    c9 = tf.keras.layers.Dropout(0.1)(c9)

    c9 = tf.keras.layers.Conv1D(16, (3), activation='relu', kernel\_initializer='he\_normal', padding='same')(c9)

    outputs = tf.keras.layers.Conv1D(1, (1), activation='sigmoid')(c9)

    unet\_model = tf.keras.Model(inputs, outputs, name="U-Net")

    return unet\_model

unet\_model = build\_unet\_model()

unet\_model.summary()

unet\_model.compile(loss = 'mean\_squared\_error',optimizer = 'adam',metrics='binary\_accuracy')

callback = tf.keras.callbacks.EarlyStopping(monitor='val\_loss', patience=10)

history = unet\_model.fit(X\_train, y\_train,

          epochs=10,

          batch\_size=64,

          validation\_data=(X\_valid,y\_valid),

          shuffle=True,verbose=1,

          callbacks=[callback])

unet\_model.save("atrialfib.h5")

from pydrive.auth import GoogleAuth

from pydrive.drive import GoogleDrive

from google.colab import auth

from oauth2client.client import GoogleCredentials

auth.authenticate\_user()

gauth = GoogleAuth()

gauth.credentials = GoogleCredentials.get\_application\_default()

drive = GoogleDrive(gauth)

folder\_id = None

file\_list = drive.ListFile({'q': "'root' in parents and trashed=false"}).GetList()

for file in file\_list:

    if file['title'] == 'unet\_model' and file['mimeType'] == 'application/vnd.google-apps.folder':

        folder\_id = file['id']

        break

if folder\_id:

    model\_file = drive.CreateFile({'title': 'atrialfib.h5', 'parents': [{'id': folder\_id}]})

    model\_file.SetContentFile('atrialfib.h5')

    model\_file.Upload()

    print("Model saved to 'unet\_model' folder in Google Drive.")

else:

    print("Folder 'unet\_model' not found in Google Drive.")

plt.plot(history.history['binary\_accuracy'], 'r', label='Training accuracy')

plt.plot(history.history['val\_binary\_accuracy'], 'b', label='Training accuracy')

plt.title('Training Vs Validation Accuracy')

plt.xlabel('No. of Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

plt.plot(history.history['loss'], 'r', label='Training loss')

plt.plot(history.history['val\_loss'], 'b', label='Validation loss')

plt.title('Training vs Validation Loss')

plt.xlabel('No. of Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

pred = unet\_model.predict(X\_test)

pred = (pred > 0.5).astype(np.uint8).astype(int)

preds = []

test = []

for i in pred:

    if 0 in i:

        preds.append(np.zeros((1)))

    else:

        preds.append(np.ones((1)))

for i in y\_test:

    if 0 in i:

        test.append(np.zeros((1)))

    else:

        test.append(np.ones((1)))

TN = sum(1 for pred, actual in zip(preds, test) if pred == 0 and actual == 0)

FP = sum(1 for pred, actual in zip(preds, test) if pred == 1 and actual == 0)

specificity = TN / (TN + FP)

print("Specificity:", specificity)

from sklearn import metrics

from sklearn.metrics import confusion\_matrix

CLASS\_LABELS = ['Normal', 'Atrial Fibrillation']

cm\_data = confusion\_matrix(test, preds)

cm = pd.DataFrame(cm\_data, columns=CLASS\_LABELS, index = CLASS\_LABELS)

cm.index.name = 'Actual'

cm.columns.name = 'Predicted'

plt.figure(figsize = (5,5))

plt.title('Confusion Matrix', fontsize = 20)

sns.set(font\_scale=1.2)

ax = sns.heatmap(cm, cbar=False, cmap="Blues", annot=True, annot\_kws={"size": 16}, fmt='g')

print('Accuracy:', np.round(metrics.accuracy\_score(test, preds),3)\*100,'%')

print('Precision:', np.round(metrics.precision\_score(test, preds, average='weighted'),3))

print('Recall:', np.round(metrics.recall\_score(test, preds, average='weighted'),3))

print('F1 Score:', np.round(metrics.f1\_score(test, preds, average='weighted'),3))

Code for Detection

!pip install wfdb

import os

import cv2

import numpy as np

import matplotlib.pyplot as plt

import tensorflow.keras

import pandas as pd

import numpy as np

import wfdb

import random

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

from wfdb import processing

random.seed(42)

import warnings

warnings.filterwarnings("ignore")

import matplotlib.pyplot as plt

import numpy as np

from scipy.signal import find\_peaks

from tensorflow.keras.models import load\_model

from tensorflow.keras.preprocessing import image

from tensorflow.keras.preprocessing.image import load\_img, img\_to\_array

from google.colab import drive

drive.mount('/content/drive')

data = '/content/drive/My Drive/pred/files/'

!pip install -U -q PyDrive

from pydrive.auth import GoogleAuth

from pydrive.drive import GoogleDrive

from google.colab import auth

from oauth2client.client import GoogleCredentials

auth.authenticate\_user()

gauth = GoogleAuth()

gauth.credentials = GoogleCredentials.get\_application\_default()

drive = GoogleDrive(gauth)

file\_id = '1bDGZ75f5HcPR-382TtxZ0MloE8P5GbQZ'

import os

downloaded\_file = drive.CreateFile({'id': file\_id})

downloaded\_file.GetContentFile('atrialfib.h5')

from keras.models import load\_model

model = load\_model('atrialfib.h5')

def load\_ecg(file,filetype):

    record = wfdb.rdrecord(file)

    annotation = wfdb.rdann(file, filetype)

    p\_signal = record.p\_signal

    atr\_sym = annotation.symbol

    atr\_sample = annotation.sample

    return p\_signal, atr\_sym, atr\_sample

def af\_data(qrs\_sample,atr\_sample):

    AF\_samples\_array = []

    NonAF\_samples\_array = []

    for i in range (len(atr\_sample)):

        if  (i % 2 == 0) & (i < len(atr\_sample)-1) :

            NonAF\_samples = qrs\_sample[( qrs\_sample >= atr\_sample[i] ) & (qrs\_sample < atr\_sample[i+1])  ]

            NonAF\_samples\_array.append(NonAF\_samples)

        if  (i % 2 == 1) & (i < len(atr\_sample)-1) :

            AF\_samples = qrs\_sample[(qrs\_sample >= atr\_sample[i] ) & (qrs\_sample < atr\_sample[i+1])  ]

            AF\_samples\_array.append(AF\_samples)

        if  (i % 2 == 0)  & (i == len(atr\_sample)-1) :

            NonAF\_samples = qrs\_sample [ qrs\_sample > atr\_sample[i] ]

            NonAF\_samples\_array.append(NonAF\_samples)

        if  (i % 2 == 1)  & (i == len(atr\_sample)-1) :

            AF\_samples = qrs\_sample [ qrs\_sample > atr\_sample[i] ]

            AF\_samples\_array.append(AF\_samples)

  return np.array(AF\_samples\_array,dtype='object'),np.array(NonAF\_samples\_array,dtype='object')

patients = ['00']

y\_128 = []

X\_array\_128 = []

for pt in patients:

    file = data + pt

    p\_signal, atr\_sym, atr\_sample = load\_ecg(file, 'atr')

    p\_signal, qrs\_sym, qrs\_sample = load\_ecg(file, 'qrs')

    AF\_samples\_array, NonAF\_samples\_array = af\_data(qrs\_sample, atr\_sample)

    AF\_samples\_array = np.concatenate(AF\_samples\_array, axis=0)

    NonAF\_samples\_array = np.concatenate(NonAF\_samples\_array, axis=0)

    totalsamples = AF\_samples\_array

    p\_signal = p\_signal[:, 0]

    for i in range(1, np.size(totalsamples) - 3, 1):

        if totalsamples[i] in AF\_samples\_array:

            X = p\_signal[(totalsamples[i] - 64):(totalsamples[i] + 64)]

            X\_array\_128.append(X)

            y\_128.append(np.ones((128,)))

        if totalsamples[i] in NonAF\_samples\_array:

            X = p\_signal[(totalsamples[i] - 64):(totalsamples[i] + 64)]

            X\_array\_128.append(X)

            y\_128.append(np.zeros((128,)))

    X\_array\_128 = np.array(X\_array\_128).reshape((np.array(X\_array\_128).shape[0], 128, 1))

    y\_128 = np.array(y\_128).reshape((np.array(y\_128).shape[0], 128, 1))

    pred = model.predict(X\_array\_128)

    pred = (pred > 0.5).astype(np.uint8).astype(int)

    plt.figure(figsize=(20, 9))

    plt.grid()

    plt.plot(X\_array\_128.reshape(-1)[:600], color='r', label='X\_array\_128')

    plt.plot(pred.reshape(-1)[:600], color='b', label='Prediction')

    plt.plot(y\_128.reshape(-1)[:600], color='g', label='True Label')

    peaks, \_ = find\_peaks(X\_array\_128.reshape(-1)[:600], height=0.1)

    plt.scatter(peaks, X\_array\_128.reshape(-1)[:600][peaks], color='green', label='Peaks (Height > 0.1)')

    plt.legend()

    plt.show()

def is\_array\_of\_zeros(arr):

  return np.all(arr == 1)

from IPython.display import HTML, display

html\_code1 = """

<div class="alert">

  <p>AF DETECTED.</p>

</div>

<style>

.alert {

  padding: 20px;

  align= center;

  background-color: #f44336;

  color: white;

  margin-bottom: 15px;

}

</style>

"""

html\_code2 = """

<div class="alert">

  <p>AF NOT DETECTED.</p>

</div>

<style>

.alert {

  padding: 20px;

  align= center;

  background-color: #48D426;

  color: white;

  margin-bottom: 15px;

}

</style>

"""

if is\_array\_of\_zeros(y\_128):

    display(HTML(html\_code1))

else:

    display(HTML(html\_code2))

Code for Extracting Real-Time Signal

#include <ThingSpeak.h>

#include <WiFi.h>

const char \*ssid = "Wifi Name"; // Our Wifi name is to be given

const char \*password = "Password";

const char \*thingSpeakAddress = "api.thingspeak.com";

const String apiKey = "O2JUIN8J6CPPX1AW";

void setup() {

Serial.begin(9600);

pinMode(10, INPUT);

pinMode(11, INPUT);

connectToWiFi();

}

void loop() {

if ((digitalRead(10) == 1) || (digitalRead(11) == 1)) {

Serial.println("!");

} else {

int ecgValue = analogRead(A1);

Serial.println(ecgValue);

uploadToThingSpeak(ecgValue);

}

delay(100);

}

void connectToWiFi() {

Serial.print("Connecting to Wi-Fi");

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("WiFi connected");

}

void uploadToThingSpeak(int value) {

WiFiClient client;

if (client.connect(thingSpeakAddress, 80)) {

String postStr = apiKey;

postStr += "&field1=";

postStr += String(value);

postStr += "\r\n\r\n";

client.print("POST /update HTTP/1.1\n");

client.print("Host: api.thingspeak.com\n");

client.print("Connection: close\n");

client.print("X-THINGSPEAKAPIKEY: " + apiKey + "\n");

client.print("Content-Type: application/x-www-form-urlencoded\n");

client.print("Content-Length: ");

client.print(postStr.length());

client.print("\n\n");

client.print(postStr);

}

client.stop();

}