ARRHYTHMIA DETECTION USING ECG SIGNALS

```
import os

import wfdb

import numpy as np

import matplotlib.pyplot as plt

import pywt

from scipy.signal import butter, filtfilt, resample

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder

from tensorflow.keras.utils import to_categorical

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout


# Configuration

DATA_DIRS = ["mitdb"]

FS_TARGET = 360

WINDOW = 216

SCALES = np.arange(1, 64)

WAVELET = 'morl'


# Bandpass filter

def bandpass_filter(signal, lowcut=0.5, highcut=40.0, fs=360, order=4):

    nyq = 0.5 * fs

    b, a = butter(order, [lowcut / nyq, highcut / nyq], btype='band')

    return filtfilt(b, a, signal)

mport os

import wfdb

import numpy as np

import matplotlib.pyplot as plt

import pywt

from scipy.signal import butter, filtfilt, resample

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

# Configuration
DATA_DIRS = ["mitdb"]
FS_TARGET = 360
WINDOW = 216
SCALES = np.arange(1, 64)
WAVELET = 'morl'

# Bandpass filter
def bandpass_filter(signal, lowcut=0.5, highcut=40.0, fs=360, order=4):
    nyq = 0.5 * fs
    b, a = butter(order, [lowcut / nyq, highcut / nyq], btype='band')
    return filtfilt(b, a, signal)
mport os
import wfdb
import numpy as np
import matplotlib.pyplot as plt
import pywt
from scipy.signal import butter, filtfilt, resample
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

# Configuration
DATA_DIRS = ["mitdb"]
FS_TARGET = 360
```

```
WINDOW = 216
SCALES = np.arange(1, 64)
WAVELET = 'morl'

# Bandpass filter
def bandpass_filter(signal, lowcut=0.5, highcut=40.0, fs=360, order=4):
    nyq = 0.5 * fs
    b, a = butter(order, [lowcut / nyq, highcut / nyq], btype='band')
    return filtfilt(b, a, signal)
import wfdb
import numpy as np
import matplotlib.pyplot as plt
import pywt
from scipy.signal import butter, filtfilt, resample
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

# Configuration
DATA_DIRS = ["mitdb"]
FS_TARGET = 360
WINDOW = 216
SCALES = np.arange(1, 64)
WAVELET = 'morl'

# Bandpass filter
def bandpass_filter(signal, lowcut=0.5, highcut=40.0, fs=360, order=4):
    nyq = 0.5 * fs
    b, a = butter(order, [lowcut / nyq, highcut / nyq], btype='band')
    return filtfilt(b, a, signal)
```

```
# Label mapping
label_map = {
    'N': 'Normal', 'L': 'Normal', 'R': 'Normal',
    'V': 'Ventricular', 'A': 'Atrial', 'F': 'Fusion', 'E': 'Escape'
}

X_scalograms = []
y_labels = []
plotted = False  # Only plot one raw vs filtered and one scalogram

# Load and preprocess
for data_dir in DATA_DIRS:
    if not os.path.isdir(data_dir):
        print(f"Directory not found: {data_dir}")
        continue

    for file in os.listdir(data_dir):
        if file.endswith('.dat'):
            record_name = file.replace('.dat', '')
            print(f"Processing: {record_name}")
            try:
                record = wfdb.rdrecord(os.path.join(data_dir, record_name))
                annotation = wfdb.rdann(os.path.join(data_dir, record_name), 'atr')

                signal = record.p_signal[:, 0]
                fs = record.fs
                if fs != FS_TARGET:
                    signal = resample(signal, int(len(signal) * FS_TARGET / fs))
                    r_peaks = (annotation.sample * FS_TARGET / fs).astype(int)
                else:
                    r_peaks = annotation.sample
```

```
symbols = annotation.symbol
filtered = bandpass_filter(signal, fs=FS_TARGET)

for r, sym in zip(r_peaks, symbols):
    if r > 108 and r + 108 < len(signal) and sym in label_map:
        raw = signal[r - 108:r + 108]
        filt = filtered[r - 108:r + 108]

        if not plotted:
            # Plot raw vs filtered once
            plt.figure(figsize=(10, 4))
            plt.plot(raw, label='Raw')
            plt.plot(filt, label='Filtered')
            plt.title('Raw vs Filtered ECG Beat')
            plt.legend()
            plt.tight_layout()
            plt.show()

        coeffs, _ = pywt.cwt(filt, SCALES, WAVELET, 1 / FS_TARGET)
        scalogram = np.abs(coeffs)
        X_scalograms.append(scalogram)
        y_labels.append(label_map[sym])

        if not plotted:
            # Plot scalogram once
            plt.figure(figsize=(8, 4))
            plt.imshow(scalogram, extent=[0, WINDOW / FS_TARGET, 1, len(SCALES)],
                       aspect='auto', cmap='jet', origin='lower')
            plt.title('Scalogram of ECG Beat')
            plt.xlabel('Time (s)')
            plt.ylabel('Scale')
```

```
                    plt.colorbar(label='Magnitude')
                    plt.tight_layout()
                    plt.show()
                    plotted = True

            except Exception as e:
                print(f"Skipping {record_name}: {e}")

# Ensure data is collected
if not X_scalograms:
    print("No ECG beats were processed. Check your dataset.")
    exit()

# Convert and encode
X = np.array(X_scalograms)[..., np.newaxis]
y = np.array(y_labels)
le = LabelEncoder()
y_encoded = le.fit_transform(y)
y_cat = to_categorical(y_encoded)

# Split
X_train, X_test, y_train, y_test = train_test_split(X, y_cat, test_size=0.2, random_state=42)

# CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=X.shape[1:]),
    MaxPooling2D((2, 2)),
    Dropout(0.25),

    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.25),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(y_cat.shape[1], activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()

# Train
history = model.fit(X_train, y_train, epochs=10, batch_size=64, validation_data=(X_test, y_test))

# Evaluate
loss, acc = model.evaluate(X_test, y_test)
print(f"\nTest Accuracy: {acc*100:.2f}%")

# Plot training curves
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train')
plt.plot(history.history['val_accuracy'], label='Val')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train')
plt.plot(history.history['val_loss'], label='Val')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```