# Biomedical Signal Classification – Full Code with Explanation

Name: Ahmed Ibrahim Elsayed Ahmed  
ID: 120200068  
Course: MIE 322 – Biomedical Signal Processing  
  
This document includes the full Python code with detailed explanation for each part of the biomedical image classification project.

## Part 1

Code:

import os  
import matplotlib.pyplot as plt  
import matplotlib.image as mpimg  
from PIL import Image  
  
dataset\_path = "/mnt/data/extracted\_dataset\_v3/OriginalDataset"  
  
fig, axes = plt.subplots(nrows=4, ncols=5, figsize=(15, 10))  
fig.suptitle("Sample Images from Each Class", fontsize=16)  
  
for row, category in enumerate(os.listdir(dataset\_path)):  
 category\_path = os.path.join(dataset\_path, category)  
 if os.path.isdir(category\_path):  
 images\_shown = 0  
 col = 0  
 for file in os.listdir(category\_path):  
 file\_path = os.path.join(category\_path, file)  
 if os.path.isfile(file\_path):  
 try:  
 img = mpimg.imread(file\_path)  
 axes[row, col].imshow(img, cmap='gray')  
 axes[row, col].axis('off')  
 if col == 0:  
 axes[row, col].set\_title(category)  
 col += 1  
 images\_shown += 1  
 if images\_shown == 5:  
 break  
 except:  
 continue  
 for empty in range(images\_shown, 5):  
 axes[row, empty].axis('off')  
  
plt.tight\_layout()  
plt.show()

Explanation:

This code displays the first 5 grayscale images from each category in the dataset using a 4x5 matplotlib grid. It handles corrupted images and skips them.

## Part 2

Code:

from PIL import Image  
import os  
  
for category in os.listdir(dataset\_path):  
 category\_path = os.path.join(dataset\_path, category)  
 if os.path.isdir(category\_path):  
 files\_in\_category = [f for f in os.listdir(category\_path) if os.path.isfile(os.path.join(category\_path, f))]  
 if files\_in\_category:  
 sample\_image = os.path.join(category\_path, files\_in\_category[0])  
 try:  
 with Image.open(sample\_image) as img:  
 print(f"{category}: {img.size} (W x H)")  
 except Exception as e:  
 print(f"Could not open or read image file {sample\_image}: {e}")  
 else:  
 print(f"No image files found in {category}")

Explanation:

This part loads one sample image from each class and prints its dimensions. It helps decide on a common image size for preprocessing.

## Part 3

Code:

import numpy as np  
from sklearn.model\_selection import train\_test\_split  
from tensorflow.keras.utils import to\_categorical  
  
categories = sorted(os.listdir(dataset\_path))  
img\_size = (128, 128)  
images = []  
labels = []  
  
for label, category in enumerate(categories):  
 category\_path = os.path.join(dataset\_path, category)  
 if os.path.isdir(category\_path):  
 for img\_file in os.listdir(category\_path):  
 img\_path = os.path.join(category\_path, img\_file)  
 try:  
 img = Image.open(img\_path).convert('L')  
 img = img.resize(img\_size)  
 img\_array = np.array(img) / 255.0  
 images.append(img\_array)  
 labels.append(label)  
 except Exception as e:  
 print(f"⚠️ Skipped image {img\_path}: {e}")  
  
X = np.array(images).reshape(-1, 128, 128, 1)  
y = to\_categorical(np.array(labels), num\_classes=len(categories))

Explanation:

This block preprocesses images: resizes them to 128x128, converts to grayscale, normalizes pixel values, and one-hot encodes the labels.

## Part 4

Code:

X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X, y, test\_size=0.3, stratify=y, random\_state=42)  
X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, stratify=y\_temp, random\_state=42)

Explanation:

The data is split into training (70%), validation (15%), and testing (15%) sets using stratified sampling to maintain class distribution.

## Part 5

Code:

import tensorflow as tf  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout  
from sklearn.utils.class\_weight import compute\_class\_weight  
  
num\_classes = y\_train.shape[1]  
  
model = Sequential([  
 Conv2D(32, (3,3), activation='relu', input\_shape=(128, 128, 1)),  
 MaxPooling2D(2,2),  
 Conv2D(64, (3,3), activation='relu'),  
 MaxPooling2D(2,2),  
 Flatten(),  
 Dense(128, activation='relu'),  
 Dropout(0.3),  
 Dense(num\_classes, activation='softmax')  
])  
  
model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

Explanation:

A CNN model with 2 Conv2D layers, max pooling, flattening, a dense hidden layer with dropout, and a softmax output layer is created and compiled.

## Part 6

Code:

y\_train\_labels = np.argmax(y\_train, axis=1)  
class\_weights\_values = compute\_class\_weight(class\_weight='balanced', classes=np.unique(y\_train\_labels), y=y\_train\_labels)  
class\_weights\_dict = {i: weight for i, weight in enumerate(class\_weights\_values)}

Explanation:

This part computes class weights using sklearn to balance the training process and reduce class imbalance issues.

## Part 7

Code:

history = model.fit(X\_train, y\_train, epochs=10, batch\_size=32, validation\_data=(X\_val, y\_val), class\_weight=class\_weights\_dict)

Explanation:

The model is trained for 10 epochs using the training set and validated on the validation set. Class weights are applied.

## Part 8

Code:

import matplotlib.pyplot as plt  
  
plt.figure(figsize=(12, 5))  
plt.subplot(1, 2, 1)  
plt.plot(history.history['accuracy'], label='Train Accuracy')  
plt.plot(history.history['val\_accuracy'], label='Val Accuracy')  
plt.title("Accuracy")  
plt.legend()  
  
plt.subplot(1, 2, 2)  
plt.plot(history.history['loss'], label='Train Loss')  
plt.plot(history.history['val\_loss'], label='Val Loss')  
plt.title("Loss")  
plt.legend()  
plt.savefig("accuracy\_loss\_plot.png")  
plt.show()

Explanation:

This section plots training and validation accuracy and loss curves to visualize the model’s learning progress and identify overfitting.

## Part 9

Code:

from sklearn.metrics import classification\_report, confusion\_matrix  
import seaborn as sns  
  
y\_pred\_probs = model.predict(X\_test)  
y\_pred = np.argmax(y\_pred\_probs, axis=1)  
y\_true = np.argmax(y\_test, axis=1)  
  
print("🧪 Test Accuracy:", np.mean(y\_pred == y\_true))  
print("\n📋 Classification Report:")  
print(classification\_report(y\_true, y\_pred, target\_names=categories))  
  
cm = confusion\_matrix(y\_true, y\_pred)  
plt.figure(figsize=(8,6))  
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=categories, yticklabels=categories)  
plt.xlabel("Predicted")  
plt.ylabel("Actual")  
plt.title("Confusion Matrix")  
plt.savefig("confusion\_matrix.png")  
plt.show()

Explanation:

Finally, the model’s predictions on the test set are evaluated using accuracy, classification report, and a visualized confusion matrix.