Plots a motor imagery paradigm with color-coded signal and rest intervals.

Function to generate confusion matrix based on given accuracy, sensitivity, and specificity

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

# Function to generate confusion matrix based on given accuracy, sensitivity, and specificity

def generate\_confusion\_matrix\_exact(accuracy, sensitivity, specificity, total\_samples=400, num\_classes=4):

    samples\_per\_class = total\_samples // num\_classes

    # Calculate True Positives (TP) based on sensitivity

    TP = int(sensitivity \* samples\_per\_class)

    # Calculate False Positives (FP) and True Negatives (TN) using specificity

    FP = int((1 - specificity) \* (3 \* samples\_per\_class))

    TN = total\_samples - samples\_per\_class - FP

    # Calculate False Negatives (FN) based on accuracy and total samples

    FN = samples\_per\_class - TP

    # Initialize confusion matrix

    confusion\_matrix = np.zeros((num\_classes, num\_classes), dtype=int)

    # Set diagonal (True Positives)

    np.fill\_diagonal(confusion\_matrix, TP)

    # Distribute False Negatives (FN) across non-diagonal elements in rows

    for i in range(num\_classes):

        confusion\_matrix[i, (i + 1) % num\_classes] = FN // (num\_classes - 1)

    # Distribute False Positives (FP) across non-diagonal elements in columns

    for j in range(num\_classes):

        confusion\_matrix[(j + 1) % num\_classes, j] = FP // (num\_classes - 1)

    # Ensure total matches class distribution

    for i in range(num\_classes):

        row\_sum = np.sum(confusion\_matrix[i, :])

        if row\_sum < samples\_per\_class:

            confusion\_matrix[i, i] += (samples\_per\_class - row\_sum)

    return confusion\_matrix

# Function to save confusion matrix as JPG

def save\_confusion\_matrix(cm, classes, title, filename):

    plt.figure(figsize=(8, 6))

    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=classes, yticklabels=classes, cbar=True)

    plt.title(title)

    plt.xlabel('Predicted Labels')

    plt.ylabel('True Labels')

    plt.tight\_layout()

    plt.savefig(filename, format='jpg')

    print(f"Confusion matrix saved as {filename}")

    plt.close()

# Metrics for the proposed method

metrics = {

    "Proposed\_Method": {"accuracy": 0.925, "sensitivity": 0.932, "specificity": 0.918},

}

# Total samples and number of classes

total\_samples = 400

num\_classes = 4

class\_labels = ["Right Hand", "Left Hand", "Foot", "Tongue"]

# Generate and save the confusion matrix

for method, metric in metrics.items():

    # Generate confusion matrix

    cm = generate\_confusion\_matrix\_exact(

        accuracy=metric["accuracy"],

        sensitivity=metric["sensitivity"],

        specificity=metric["specificity"],

        total\_samples=total\_samples,

        num\_classes=num\_classes

    )

    # Save as JPG

    filename = f"{method}\_Confusion\_Matrix.jpg"

    save\_confusion\_matrix(cm, classes=class\_labels, title=f"{method} Confusion Matrix", filename=filename)

import matplotlib.pyplot as plt

import numpy as np

def plot\_mi\_paradigm(signal\_durations, rest\_durations, signal\_colors, action\_labels):

  """

  Plots a motor imagery paradigm with color-coded signal and rest intervals.

  Args:

    signal\_durations: List of durations for each signal interval (in seconds).

    rest\_durations: List of durations for each rest interval (in seconds).

    signal\_colors: List of colors for each signal interval.

    action\_labels: List of labels for each signal (e.g., "Right Hand", "Left Hand").

  Returns:

    A matplotlib figure object.

  """

  fig, ax = plt.subplots(figsize=(12, 2))

  ax.set\_ylim(0, 1)

  ax.set\_xlabel("Time (seconds)")

  ax.set\_ylabel("Task")

  ax.set\_yticks([])

  start\_time = 0

  for i, (signal\_duration, rest\_duration, signal\_color, action\_label) in enumerate(zip(signal\_durations, rest\_durations, signal\_colors, action\_labels)):

    # Signal interval

    ax.fill\_between(

        [start\_time, start\_time + signal\_duration],

        0, 1,

        color=signal\_color,

        alpha=0.7,

        label=action\_label,  # Add label for all signals

    )

    start\_time += signal\_duration

    # Rest interval

    ax.fill\_between(

        [start\_time, start\_time + rest\_duration],

        0, 1,

        color='lightgray',

        alpha=0.7,

        label="Rest" if i == 0 else None,  # Add label only for the first rest

    )

    start\_time += rest\_duration

  # Create a legend

  handles, labels = ax.get\_legend\_handles\_labels()

  ax.legend(handles, labels, loc='upper right')

  # Adjust x-axis ticks for better visibility

  plt.xticks(np.arange(0, start\_time + 1, 2))  # Show ticks every 2 seconds

  return fig

# Example usage with four actions:

signal\_durations = [3, 4, 2, 5]  # Durations of signal intervals in seconds

rest\_durations = [1, 1, 1, 1]    # Durations of rest intervals in seconds

signal\_colors = ['red', 'blue', 'green', 'purple']  # Colors for each signal interval

action\_labels = ["Right Hand", "Left Hand", "Foot", "Tongue"]

fig = plot\_mi\_paradigm(signal\_durations, rest\_durations, signal\_colors, action\_labels)

plt.show()