Цель задания:

- исследование когнитивных характеристик мозга по данным ЭЭГ

Ключевой навык:

- алгоритмы классификации в нейронных сетях

```
!pip install pywavelets --quiet
```

Эта команда беззвучно устанавливает библиотеку PyWavelets для поддержки анализа вейвлет-преобразований в последующем коде.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import time
from scipy.signal import spectrogram
import pywt
import cv2
from sklearn.metrics import cohen_kappa_score, confusion_matrix
from keras.models import Sequential
from keras.layers import (Conv2D, MaxPool2D, Flatten, Dense, Dropout,
                          TimeDistributed, LSTM)
from keras.optimizers import Adam
from keras import backend as K
```

Вместе эти импортированные библиотеки выполняют следующие функции:

Поток данных: загрузка сигналов ЭЭГ → преобразование в карты временных частот → импорт моделей.
Поток моделей: CNN извлекает пространственные признаки → LSTM улавливает временные связи → классификация с полным подключением слоев.
Поток оценки: точность/коэффициент Каппа/матрица смешения - многомерная оценка производительности.
Разумно комбинируя эти инструменты, код завершает полный поток от необработанного сигнала ЭЭГ до классификации изображений движения.

作业目的

- 通过脑电图数据研究大脑的认知特征

关键技能

- 神经网络分类算法

```
!pip install pywavelets --quiet
```

该命令会静默安装 PyWavelets 库，以支持后续代码中的小波变换分析。

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import time
from scipy.signal import spectrogram
import pywt
import cv2
from sklearn.metrics import cohen_kappa_score, confusion_matrix
from keras.models import Sequential
from keras.layers import (Conv2D, MaxPool2D, Flatten, Dense, Dropout,
                          TimeDistributed, LSTM)
from keras.optimizers import Adam
from keras import backend as K
```

这些导入的库共同执行以下功能：

数据流：下载脑电信号 → 转换为时频图 → 模型导入。
模型流程：CNN 提取空间特征 → LSTM 捕捉时间关系 → 利用全层连接进行分类。
评估流程：精度/卡帕系数/混合矩阵--多元性能评估。
通过智能组合这些工具，代码完成了从原始脑电信号到运动图像分类的整个流程。

```
# download dataset
x_train = pd.read_csv("https://github.com/TAUforPython/BioMedAI/blob/main/test_datasets/MI-EEG-B9T.csv?raw=true",
                      header=None)
x_test = pd.read_csv("https://github.com/TAUforPython/BioMedAI/blob/main/test_datasets/MI-EEG-B9E.csv?raw=true",
                     header=None)
y_train = pd.read_csv("https://github.com/TAUforPython/BioMedAI/blob/main/test_datasets/2class_MI_EEG_train_9.csv?raw=true",
                      header=None)
y_test = pd.read_csv("https://github.com/TAUforPython/BioMedAI/blob/main/test_datasets/2class_MI_EEG_test_9.csv?raw=true",
                     header=None) 的作用是什么?
```

该代码是数据准备阶段的第一步，负责将远程脑电信号和标签加载到内存中，以便随后进行特征提取（时域和频域分析）和模型训练。

```
# download dataset
x_train = pd.read_csv("https://github.com/TAUforPython/BioMedAI/blob/main/test_datasets/MI-EEG-B9T.csv?raw=true",
                      header=None)
x_test = pd.read_csv("https://github.com/TAUforPython/BioMedAI/blob/main/test_datasets/MI-EEG-B9E.csv?raw=true",
                     header=None)
y_train = pd.read_csv("https://github.com/TAUforPython/BioMedAI/blob/main/test_datasets/2class_MI_EEG_train_9.csv?raw=true",
                      header=None)
y_test = pd.read_csv("https://github.com/TAUforPython/BioMedAI/blob/main/test_datasets/2class_MI_EEG_test_9.csv?raw=true",
                     header=None) 的作用是什么?
```

Этот код является первым шагом на этапе подготовки данных и отвечает за загрузку удаленных сигналов ЭЭГ и меток в память для последующего извлечения признаков (анализ временных и частотных характеристик) и обучения модели.

```
print(x_train.shape)      (400, 3000)
print(x_test.shape)       (320, 3000)
print(y_train.shape)      (400, 1)
print(y_test.shape)       (320, 1)
```

Эти утверждения Print являются фундаментальным шагом проверки на этапе предварительной обработки данных, обеспечивая целостность и непротиворечивость данных и предотвращая последующие ошибки модели из-за ошибок размерности данных.

```
n_samples_train = len(y_train)
n_samples_test = len(y_test)

print("n_samples_train:", n_samples_train)
print("n_samples_test :", n_samples_test)
```

```
n_samples_train: 400
n_samples_test : 320
```

Цель этого кода - получить и вывести на печать количество образцов в обучающем и тестовом наборах, что используется для проверки обоснованности сегментации набораданных и обеспечения ключевых параметров для последующего обучения модели.

```
print(x_train.shape)      (400, 3000)
print(x_test.shape)       (320, 3000)
print(y_train.shape)      (400, 1)
print(y_test.shape)       (320, 1)
```

这些打印语句是数据预处理阶段的基本验证步骤，可确保数据的完整性和一致性，并防止因数据维度错误而导致后续模型错误。

```
n_samples_train = len(y_train)
n_samples_test = len(y_test)

print("n_samples_train:", n_samples_train)
print("n_samples_test :", n_samples_test)
```

```
n_samples_train: 400
n_samples_test : 320
```

这段代码的目的是获取并打印训练集和测试集中的样本数量，用于检查数据集分割的有效性，并为后续的模型训练提供关键参数。

```
# count classes
n_classes = len(np.unique(y_test))

print("n_classes:", n_classes)
```

```
n_classes: 2
```

这段代码是数据验证阶段的关键步骤，用于确保：
测试集包含所有预期类别。
标签格式正确，为后续模型训练提供准确的类别数参数。
输出结果直接影响模型架构设计和评估方法的选择。

```
# calculate STFT

def spectrogram_vertical(data, fs, alto, ancho, n_canales, pts_sig,
                                                pts_superpuestos):
  #fs = fs #frecuencia de muestreo
  datesets = np.zeros((data.shape[0],alto, ancho))

  # crear matriz 2D donde se guardara cada imagen del STFT
  temporal = np.zeros((alto, ancho))

  for i in range(data.shape[0]): # n muestras
    for j in range(n_canales): # n canales

      sig = data.iloc[i, j*pts_sig:(j+1)*pts_sig]

      f, t, Sxx = spectrogram(sig, fs=fs, window='hann', nperseg=fs,
                              noverlap=pts_superpuestos, nfft=fs*2,
                              scaling='spectrum')

      # concatenacion vertical chanels
      temporal[j*45:(j+1)*45, :] = Sxx[16:61, :]

    datesets[i] = temporal
    if i % 100 == 0:
      print(i)
  return datesets
```

```
# count classes
n_classes = len(np.unique(y_test))

print("n_classes:", n_classes)
```

```
n_classes: 2
```

Этот код является ключевым шагом на этапе проверки данных и используется для того, чтобы убедиться, что:
Тестовый набор содержит все ожидаемые категории.
Метки правильно отформатированы, чтобы обеспечить точные параметры подсчета категорий для последующего обучения модели.
Результаты напрямую влияют на дизайн архитектуры модели и выбор методов оценки.

```
# calculate STFT

def spectrogram_vertical(data, fs, alto, ancho, n_canales, pts_sig,
                         pts_superpuestos):
  #fs = fs #frecuencia de muestreo
  datesets = np.zeros((data.shape[0],alto, ancho))

  # crear matriz 2D donde se guardara cada imagen del STFT
  temporal = np.zeros((alto, ancho))

  for i in range(data.shape[0]): # n muestras
    for j in range(n_canales): # n canales

      sig = data.iloc[i, j*pts_sig:(j+1)*pts_sig]

      f, t, Sxx = spectrogram(sig, fs=fs, window='hann', nperseg=fs,
                              noverlap=pts_superpuestos, nfft=fs*2,
                              scaling='spectrum')

      # concatenacion vertical chanels
      temporal[j*45:(j+1)*45, :] = Sxx[16:61, :]

    datesets[i] = temporal
    if i % 100 == 0:
      print(i)
  return datesets
```

Эта функция используется для выполнения кратковременного преобразования Фурье (STFT) для многоканальных сигналов ЭЭГ и вертикального сшивания спектрограмм различных каналов для создания изображения в формате, пригодном для ввода в модели глубокого обучения (например, CNN).

该功能用于对多通道脑电信号进行短时傅立叶变换 (STFT)，并垂直拼接不同通道的频谱图，以创建适合深度学习模型（如 CNN）输入格式的图像。

```
# calculate scalogram CWT

def scalogram_vertical(data, fs, alto, ancho, n_canales, pts_sig):
  dim = (int(np.floor(ancho/2)), int(np.floor(alto/2))) # ancho, alto

  # Wavelet Morlet 3-3
  # frequency 8 - 30 Hz
  scales = pywt.scale2frequency('cmor3-3', np.arange(8,30.5,0.5)) / (1/fs)

  datesets = np.zeros((data.shape[0], int(np.floor(alto/2)),
                       int(np.floor(ancho/2))))

  temporal = np.zeros((alto, ancho))

  for i in range(data.shape[0]):
    for j in range(n_canales):

      sig = data.iloc[i, j*pts_sig:(j+1)*pts_sig]

      coef, freqs = pywt.cwt(sig, scales, 'cmor3-3',
                             sampling_period = (1 / fs))

      temporal[j*45:(j+1)*45, :] = abs(coef)

    resized = cv2.resize(temporal, dim, interpolation=cv2.INTER_AREA)
    datesets[i] = resized
    if i % 100 == 0:
      print(i)
  return datesets
```

此函数通过连续小波变换将多通道 EEG 信号转换为时频图，为深度学习模型提供适合的输入格式，是分析非平稳信号（如运动想象 EEG）的关键预处理步骤。

```
initial = time.time()

# STFT
x_train = spectrogram_vertical(x_train, 250, 135, 31, 3, 1000, 225)
x_test = spectrogram_vertical(x_test, 250, 135, 31, 3, 1000, 225)

# CWT
#x_train = scalogram_vertical(x_train, 250, 135, 1000, 3, 1000)
#x_test = scalogram_vertical(x_test, 250, 135, 1000, 3, 1000)

fin = time.time()
print("time_elapsed:", fin - initial)
```

```
0
100
200
300
0
100
200
300
time_elapsed: 1.48185396194458
```

这段代码是 EEG 信号处理的核心步骤，将原始信号转换为时频图像特征，为后续深度学习模型提供输入。通过计时功能，开发者可以权衡计算效率与特征质量，优化预处理流程。

```
# calculate scalogram CWT

def scalogram_vertical(data, fs, alto, ancho, n_canales, pts_sig):
  dim = (int(np.floor(ancho/2)), int(np.floor(alto/2))) # ancho, alto

  # Wavelet Morlet 3-3
  # frequency 8 - 30 Hz
  scales = pywt.scale2frequency('cmor3-3', np.arange(8,30.5,0.5)) / (1/fs)

  datesets = np.zeros((data.shape[0], int(np.floor(alto/2)),
                       int(np.floor(ancho/2))))

  temporal = np.zeros((alto, ancho))

  for i in range(data.shape[0]):
    for j in range(n_canales):

      sig = data.iloc[i, j*pts_sig:(j+1)*pts_sig]

      coef, freqs = pywt.cwt(sig, scales, 'cmor3-3',
                             sampling_period = (1 / fs))

      temporal[j*45:(j+1)*45, :] = abs(coef)

    resized = cv2.resize(temporal, dim, interpolation=cv2.INTER_AREA)
    datesets[i] = resized
    if i % 100 == 0:
      print(i)
  return datesets
```

Эта функция преобразует многоканальные сигналы ЭЭГ в карты временных частот с помощью непрерывного вейвлет-преобразования, чтобы обеспечить подходящий формат ввода для моделей глубокого обучения, что является ключевым этапом предварительной обработки для анализа негладких сигналов, таких как ЭЭГ с изображениями движения.

```
initial = time.time()

# STFT
x_train = spectrogram_vertical(x_train, 250, 135, 31, 3, 1000, 225)
x_test = spectrogram_vertical(x_test, 250, 135, 31, 3, 1000, 225)

# CWT
#x_train = scalogram_vertical(x_train, 250, 135, 1000, 3, 1000)
#x_test = scalogram_vertical(x_test, 250, 135, 1000, 3, 1000)

fin = time.time()
print("time_elapsed:", fin - initial)
```

```
0
100
200
300
0
100
200
300
time_elapsed: 1.48185396194458
```

Этот код является одним из основных этапов обработки сигналов ЭЭГ, преобразуя необработанные сигналы во временно-частотные характеристики изображений,

```
print(x_train.shape)
print(x_test.shape)
```

```
(400, 135, 31)
(320, 135, 31)
```

print(x_train.shape 和 print(x_test.shape) 的作用是 输出训练集和测试集的特征数据维度，用于验证数据预处理后的形状是否符合预期，并为后续模型构建提供关键信息。

```
x = np.ceil(np.max(x_train))
```

```
# convert to float
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

x_train /= x
x_test /= x
```

```
plt.figure()
plt.imshow(x_train[1],  aspect='auto')
plt.colorbar()
plt.show()
```

归一化：将时频图数据缩放到 [0, 1] 范围内，提升模型训练稳定性。
例如，若原始最大值为 255，则所有值除以 255，变为 0~1。
类型转换：float32 是深度学习框架（如 TensorFlow/Keras）的默认数据类型，可减少内存占用并加速计算。
验证预处理效果：
检查时频图是否正常（如无全黑/全白、噪声异常）。
确认归一化后的值范围是否符合预期（颜色条应在 0~1 之间）。
理解数据分布：
高频（上部）和低频（下部）的能量分布是否合理。
不同通道（垂直排列）的频谱差异是否可见。

которые служат входом для последующих моделей глубокого обучения. Функция синхронизации позволяет разработчикам найти компромисс между вычислительной эффективностью и качеством признаков и оптимизировать процесс предварительной обработки.

```
print(x_train.shape)
print(x_test.shape)
```

```
(400, 135, 31)
(320, 135, 31)
```

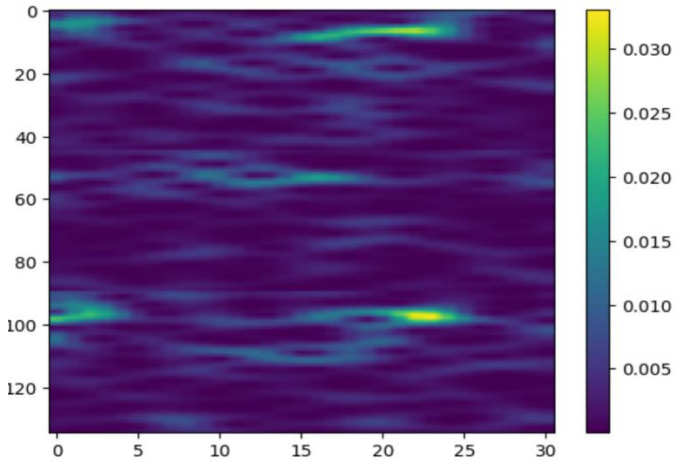
Роль print(x_train.shape и print(x_test.shape)) заключается в выводе размеров данных тренировочного и тестового наборов, которые используются для проверки того, что форма предварительно обработанных данных соответствует ожиданиям и предоставляет ключевую информацию для последующего построения модели.

```
x = np.ceil(np.max(x_train))
```

```
# convert to float
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

x_train /= x
x_test /= x
```

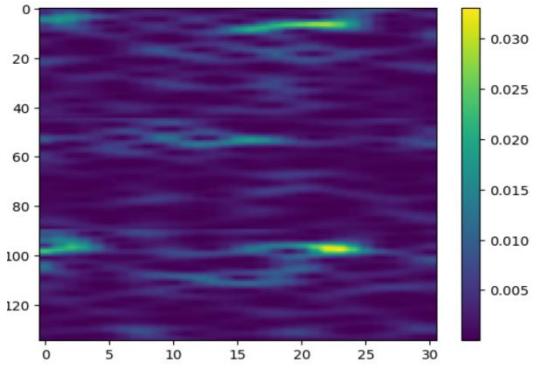
```
plt.figure()
plt.imshow(x_train[1],  aspect='auto')
plt.colorbar()
plt.show()
```

Нормализация: масштабирование данных временно-частотного графика в диапазоне [0, 1] для повышения стабильности обучения модели.
Например, если исходное максимальное значение равно 255, то все значения будут разделены на 255 и станут 0~1.



热力图是 时频分析结果的可视化，具体为脑电信号（EEG）经过 短时傅里叶变换（STFT）或小波变换（CWT）后的频谱能量分布图。以下是关键信息的解读：
1. 坐标轴含义
纵轴（左侧 0~120）：
频率（Hz），覆盖了 EEG 信号中与运动想象相关的频段：
μ 节律（8-12Hz）：想象肢体运动时出现的特征性抑制。
β 节律（13-30Hz）：与运动准备和执行的神经活动相关。
图中范围延伸至更高频率（如 60Hz）：可能包含噪声或高频脑电活动。
横轴（底部 0~30）：
时间（秒），表示 EEG 信号的持续时间（例如一次 4 秒的运动想象试验，截取关键时间窗口）。
2. 颜色映射
颜色条（右侧 0~0.030）：表示 能量强度（或功率谱密度），单位为 $\mu V^2/Hz$。
紫色/蓝色：低能量区域（如背景噪声或静息状态）。
黄色/绿色：高能量区域（如特定时间点的神经活动增强）。
示例解读：若某个区域在 10Hz、时间 5 秒 附近显示黄色，表示此时大脑在 μ 节律频段有显著活动。若在 25Hz、时间 15 秒 附近出现绿色，可能对应 β 节律的短暂增强。
3. 应用场景

Поймите распределение данных:
Является ли распределение энергии на высоких (верхних) и низких (нижних) частотах разумным.
Заметны ли спектральные различия между разными каналами (выровненными по вертикали).



Тепловая карта визуализирует изменения энергии во временном и частотном измерениях сигнала ЭЭГ и является важным инструментом для декодирования нейронной активности в двигательных образах. Распределение цветов и координат непосредственно отражает паттерн активации мозга в данной задаче, предоставляя ключевые характеристики для последующих моделей классификации.

```
#  reshape a 4D  (for CNN-2D)
#x_train = x_train.reshape((x_train.shape[0],  x_train.shape[1],  x_train.shape[2],  1))
#x_test = x_test.reshape((x_test.shape[0],  x_test.shape[1],  x_test.shape[2],  1))

# convert  3D to 5D  (CNN-2D + LSTM)
x_train = x_train.reshape((x_train.shape[0],  1,  x_train.shape[1],  x_train.shape[2],  1))
x_test = x_test.reshape((x_test.shape[0],  1,  x_test.shape[1],  x_test.shape[2],  1))

print(x_train.shape)
print(x_test.shape)
```

```
(400,  1,  135,  31,  1)
(320,  1,  135,  31,  1)
```

LSTM для объединения пространственно-временных признаков (дизайн временного шага должен быть оптимизирован в соответствии с реальными требованиями).

运动想象分类：通过检测 μ/β 节律的抑制或增强，判断用户想象的是左手或右手运动。
例如：想象左手运动时，右脑区域（C4 通道）的 μ 节律能量会降低（紫色区域扩大）。噪声识别：高频区域（如 60Hz 以上）的持续高能量可能由电源干扰（工频噪声）引起。
这张热力图直观展示了 脑电信号在时间和频率维度上的能量变化，是解码运动想象神经活动的重要工具。颜色和坐标的分布直接反映了大脑在特定任务中的激活模式，为后续分类模型提供了关键特征。

```
#  reshape a 4D  (for CNN-2D)
#x_train = x_train.reshape((x_train.shape[0],  x_train.shape[1],  x_train.shape[2],  1))
#x_test = x_test.reshape((x_test.shape[0],  x_test.shape[1],  x_test.shape[2],  1))

# convert  3D to 5D  (CNN-2D + LSTM)
x_train = x_train.reshape((x_train.shape[0],  1,  x_train.shape[1],  x_train.shape[2],  1))
x_test = x_test.reshape((x_test.shape[0],  1,  x_test.shape[1],  x_test.shape[2],  1))

print(x_train.shape)
print(x_test.shape)
```

```
(400,  1,  135,  31,  1)
(320,  1,  135,  31,  1)
```

此代码通过调整数据维度，使时频图能适配不同模型架构：

- **4D 输入**：用于处理静态图像的 2D-CNN。
- **5D 输入**：用于结合时空特征的 CNN-LSTM 混合模型（需根据实际需求优化时间步设计）。

选择哪种方式取决于数据特性和任务目标。

```
# crear red neuronal CNN-2D

def CNN_2D(num_filter,  size_filter,  n_neurons):
  model = Sequential()
  model.add(Conv2D(num_filter,  size_filter,  activation='relu',  padding='same',
                   input_shape=x_train.shape[1:]))
  model.add(MaxPool2D((2,2)))
  model.add(Conv2D(num_filter,  size_filter,  activation='relu',  padding='same'))
  model.add(MaxPool2D((2,2)))
  model.add(Flatten())
  model.add(Dense(n_neurons,  activation='relu'))
  model.add(Dropout(0.5))
  model.add(Dense(n_classes,  activation='softmax'))

  optimizer = Adam(learning_rate=0.001)
  model.compile(optimizer = optimizer,
                loss = 'sparse_categorical_crossentropy',
                metrics = ['accuracy'])
  return model
```

Выбор зависит от характеристик данных и целей задачи.

```
# crear red neuronal CNN-2D

def CNN_2D(num_filter, size_filter, n_neurons):
  model = Sequential()
  model.add(Conv2D(num_filter, size_filter, activation='relu', padding='same',
                   input_shape=x_train.shape[1:]))
  model.add(MaxPool2D((2,2)))
  model.add(Conv2D(num_filter, size_filter, activation='relu', padding='same'))
  model.add(MaxPool2D((2,2)))
  model.add(Flatten())
  model.add(Dense(n_neurons, activation='relu'))
  model.add(Dropout(0.5))
  model.add(Dense(n_classes, activation='softmax'))

  optimizer = Adam(learning_rate=0.001)
  model.compile(optimizer = optimizer,
                loss = 'sparse_categorical_crossentropy',
                metrics = ['accuracy'])
  return model
```

Этот код определяет модель двумерной конволюционной нейронной сети (2D-CNN) для классификации изображений

Этот код определяет базовую модель 2D-CNN, подходящую для небольших и средних задач классификации изображений. Производительность может быть дополнительно оптимизирована для сложных сценариев путем настройки гиперпараметров (например, количество сверточных ядер, количество слоев) и добавления методов регуляризации. На практике архитектура должна гибко адаптироваться к конкретным задачам.

```
# crear red neuronal CNN-2D + LSTM

def CNN_2D_LSTM_TD(num_filter, size_filter, n_neurons, units_LSTM):
  model = Sequential()
  model.add(TimeDistributed(Conv2D(num_filter, size_filter, activation='relu',
                                   padding='same'),
                            input_shape=x_train.shape[1:]))
  model.add(TimeDistributed(MaxPool2D((2,2))))
  model.add(TimeDistributed(Conv2D(num_filter, size_filter, activation='relu',
                                   padding='same')))
  model.add(TimeDistributed(MaxPool2D((2,2))))
  model.add(TimeDistributed(Flatten()))
  model.add(LSTM(units_LSTM, activation='tanh', dropout=0.5))
  model.add(Dense(n_neurons, activation='relu'))
  model.add(Dense(n_classes, activation='softmax'))

  optimizer = Adam(learning_rate=1e-3)
  model.compile(optimizer = optimizer,
                loss = 'sparse_categorical_crossentropy',
                metrics = ['accuracy'])
  return model
```

Этот код определяет гибридную нейронную сеть CNN-LSTM для обработки данных временных рядов изображений (например, видеокадров, непрерывных карт временных частот) для задач классификации, сочетая пространственное извлечение

这段代码定义了一个用于图像分类的二维卷积神经网络（2D-CNN）模型

这段代码定义了一个基础的 2D-CNN 模型，适合于中小型图像分类任务。通过调整超参数（如卷积核数量、层数）和加入正则化技术，可进一步优化性能以适应复杂场景。实际应用中需根据具体任务灵活调整架构。

```
# crear red neuronal CNN-2D + LSTM

def CNN_2D_LSTM_TD(num_filter, size_filter, n_neurons, units_LSTM):
  model = Sequential()
  model.add(TimeDistributed(Conv2D(num_filter, size_filter, activation='relu',
                                   padding='same'),
                            input_shape=x_train.shape[1:]))
  model.add(TimeDistributed(MaxPool2D((2,2))))
  model.add(TimeDistributed(Conv2D(num_filter, size_filter, activation='relu',
                                   padding='same')))
  model.add(TimeDistributed(MaxPool2D((2,2))))
  model.add(TimeDistributed(Flatten()))
  model.add(LSTM(units_LSTM, activation='tanh', dropout=0.5))
  model.add(Dense(n_neurons, activation='relu'))
  model.add(Dense(n_classes, activation='softmax'))

  optimizer = Adam(learning_rate=1e-3)
  model.compile(optimizer = optimizer,
                loss = 'sparse_categorical_crossentropy',
                metrics = ['accuracy'])
  return model
```

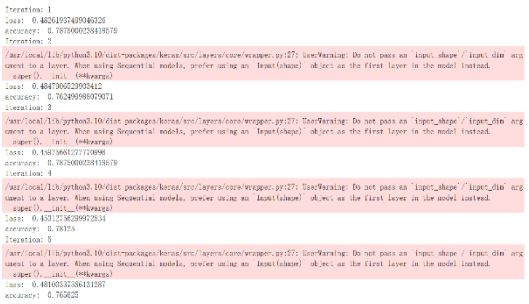
此代码定义了一个 CNN-LSTM 混合神经网络，用于处理 **时序图像数据**（如视频帧、连续时频图），通过结合 CNN 的空间特征提取和 LSTM 的时间序列建模能力进行分类任务。

此模型通过 CNN 提取空间特征 + LSTM 建模时序关系，适用于同时包含空间和时间维度 的数据分类任务（如视频、连续脑电信号）。

```
initial = time.time()
array_loss = []
array_acc = []
array_kappa = []
for i in range(5):
  print("Iteration:", i+1)


  #model = CNN_2D(4, (3,3), 32)
  model = CNN_2D_LSTM_TD(4, (3,3), 32, 4)

 # history = model.fit(x_train, y_train, epochs=40, batch_size=36,
 #                     validation_data=(x_test, y_test), verbose=0)

  history = model.fit(x_train, y_train, epochs=70, batch_size=36,
                      validation_split = 0.1, verbose=0)

  test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)

  array_loss.append(test_loss)
  print("loss: ", test_loss)
  array_acc.append(test_acc)
  print("accuracy: ", test_acc)
```

признаков CNN с возможностью моделирования временных рядов LSTM. Эта модель извлекает пространственные признаки с помощью CNN + моделирует временные связи с помощью LSTM и подходит для задач классификации данных (например, видео, непрерывных сигналов ЭЭГ), которые содержат как пространственные, так и временные измерения.

```
initial = time.time()
array_loss = []
array_acc = []
array_kappa = []
for i in range(5):
  print("Iteration:", i+1)


  #model = CNN_2D(4, (3,3), 32)
  model = CNN_2D_LSTM_TD(4, (3,3), 32, 4)

 # history = model.fit(x_train, y_train, epochs=40, batch_size=36,
 #                     validation_data=(x_test, y_test), verbose=0)

  history = model.fit(x_train, y_train, epochs=70, batch_size=36,
                      validation_split = 0.1, verbose=0)

  test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)

  array_loss.append(test_loss)
  print("loss: ", test_loss)
  array_acc.append(test_acc)
  print("accuracy: ", test_acc)
```



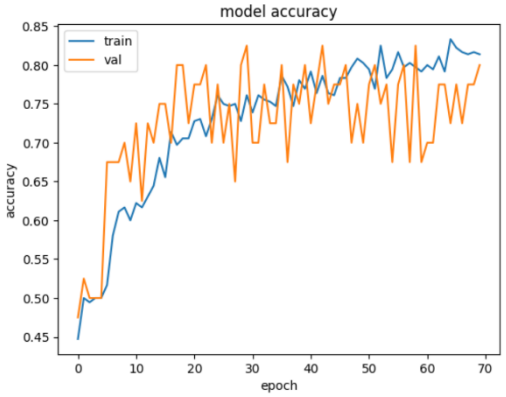
Этот код используется для многократного обучения и оценки производительности модели глубокого обучения, а также для проверки стабильности и надежности модели в ходе многочисленных экспериментов. Эффективность модели проверяется с помощью нескольких независимых экспериментов, сочетающих потери, точность, коэффициент каппа, а также кривые обучения и матрицы путаницы. Проанализируйте поведение модели.

```
Iteration: 1
loss:  0.48261937499046326
accuracy:  0.7875000238418579
Iteration: 2
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/wrapper.py:27: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
loss:  0.48479065299034412
accuracy:  0.762499988079071
Iteration: 3
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/wrapper.py:27: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
loss:  0.45975661277770996
accuracy:  0.7875000238418579
Iteration: 4
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/wrapper.py:27: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
loss:  0.45312756299972534
accuracy:  0.78125
Iteration: 5
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/wrapper.py:27: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
loss:  0.48100337386131287
accuracy:  0.765625
```

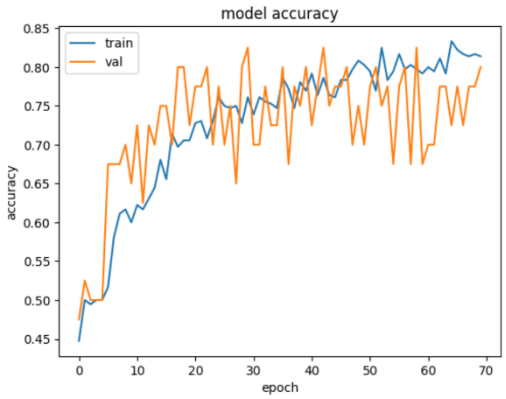
这段代码用于 多次训练并评估深度学习模型的性能，通过多次实验验证模型的稳定性和可靠性。通过 多次独立实验 验证模型性能，结合 损失、准确率、Kappa 系数 多指标评估，并借助 学习曲线和混淆矩阵 分析模型行为。

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```

这段代码的作用是 可视化模型在训练过程中训练集和验证集的准确率（Accuracy）变化曲线，帮助直观分析模型的学习状态和性能表现。



Цель этого кода - визуализировать кривые точности обучающего и валидационного наборов в процессе обучения модели, что помогает наглядно проанализировать состояние обучения и производительность модели.



Обучающий набор (синий): показатель точности неуклонно растет с начального значения около 0,5 до 0,8 и в конце концов снижается, указывая на то, что модель эффективно усвоила особенности обучающих данных.
Валидационный набор (оранжевый): показатель точности постепенно увеличивается с 0,45 до 0,75, что соответствует тенденции обучающего набора, и указывает на то, что модель обладает некоторой способностью к обобщению.

```
probabilidades = model.predict(x_test)

y_pred = np.argmax(probabilidades, 1)

  # calculate kappa cohen
kappa = cohen_kappa_score(y_test, y_pred)
array_kappa.append(kappa)
print("kappa: ", kappa)
matriz_confusion = confusion_matrix(y_test, y_pred)
print("confusion matrix:\n", matriz_confusion)
```

```
kappa:  0.6125
confusion matrix:
 [[132  28]
 [ 34 126]]
```

Код используется для оценки эффективности модели классификации, обеспечивая более полный анализ эффектов

训练集（蓝色）：准确率从初始约 0.5 稳步上升至 0.8 左右，最终趋于平稳，表明模型有效学习到了训练数据的特征。
验证集（橙色）：准确率从 0.45 逐步提升至 0.75，与训练集趋势一致，说明模型具备一定的泛化能力。

```
probabilidades = model.predict(x_test)

y_pred = np.argmax(probabilidades, 1)

  # calculate kappa cohen
kappa = cohen_kappa_score(y_test, y_pred)
array_kappa.append(kappa)
print("kappa: ", kappa)
matriz_confusion = confusion_matrix(y_test, y_pred)
print("confusion matrix:\n", matriz_confusion)
```

```
kappa:  0.6125
confusion matrix:
 [[132  28]
 [ 34 126]]
```

代码用于 评估分类模型的性能，通过 Cohen's Kappa 系数 和 混淆矩阵 提供比准确率更全面的分类效果分析。通过 Kappa 系数 和 混淆矩阵 揭示了模型在类别层面的详细表现，是准确率指标的重要补充。

```
from sklearn.metrics import ConfusionMatrixDisplay

labels = ["Left MI", "Right MI"]

disp = ConfusionMatrixDisplay(confusion_matrix=matriz_confusion, display_labels=labels)

disp.plot(cmap=plt.cm.Blues)
plt.show()
```

这段代码用于 可视化混淆矩阵（Confusion Matrix），将模型分类结果的性能以热力图形式直观展示，帮助快速识别分类错误模式和类别间的混淆情况。通过热力图形式将混淆矩阵可视化，使模型分类性能的分析更加直观。可从中快速识别：模型在哪些类别上表现优异（对角线深色）。哪些类别容易混淆（非对角线浅色区域）。是否需要针对特定类别优化数据或模型。

классификации, чем точность с помощью коэффициентов Каппы Коэна и матриц путаницы.
Коэффициенты Каппа и матрицы путаницы показывают детальную производительность модели на уровне категорий, что является важным дополнением к метрике точности.

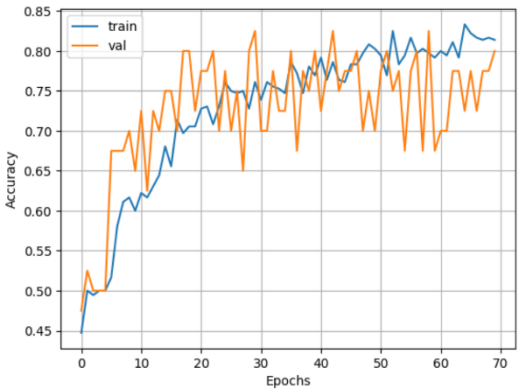
```
from sklearn.metrics import ConfusionMatrixDisplay

labels = ["Left MI", "Right MI"]

disp = ConfusionMatrixDisplay(confusion_matrix=matriz_confusion, display_labels=labels)

disp.plot(cmap=plt.cm.Blues)
plt.show()
```

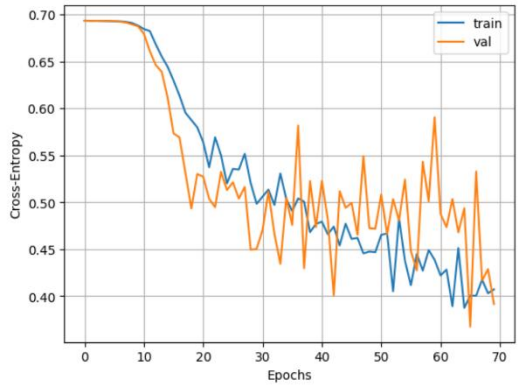
Этот код используется для визуализации матрицы путаницы, которая представляет результаты классификации модели в виде тепловой карты, помогая быстро выявить закономерности неправильной классификации и путаницы между категориями. Визуализация матрицы путаницы в виде тепловой карты делает анализ эффективности классификации модели более интуитивным. С ее помощью можно быстро определить, на каких категориях модель работает хорошо (темные диагональные цвета). Какие категории подвержены путанице (недиагональные области светлого цвета). Нужно ли оптимизировать данные или модель для определенных категорий.

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.grid()
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(['train', 'val'])
plt.show()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.grid()
plt.xlabel('Epochs')
plt.ylabel('Cross-Entropy')
plt.legend(['train', 'val'])
plt.show()

print()
print("Resultados:")
print("loss:", array_loss)
print("accuracy:", array_acc)
print("kappa:", array_kappa)
fin = time.time()
time_elapsed = fin - initial
print("time_elapsed:", time_elapsed)
```

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.grid()
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(['train', 'val'])
plt.show()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.grid()
plt.xlabel('Epochs')
plt.ylabel('Cross-Entropy')
plt.legend(['train', 'val'])
plt.show()

print()
print("Resultados:")
print("loss:", array_loss)
print("accuracy:", array_acc)
print("kappa:", array_kappa)
fin = time.time()
time_elapsed = fin - initial
print("time_elapsed:", time_elapsed)
```

代码用于 可视化模型训练过程 并输出最终评估结果，帮助全面分析模型的性能、训练效率和收敛情况。

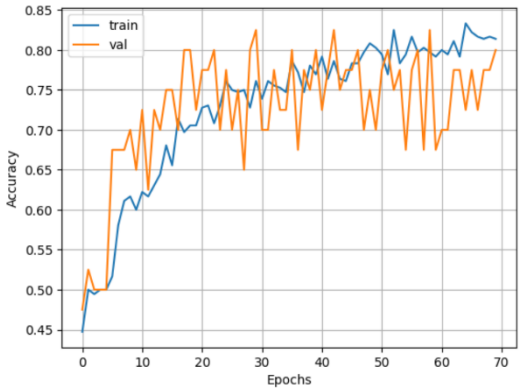


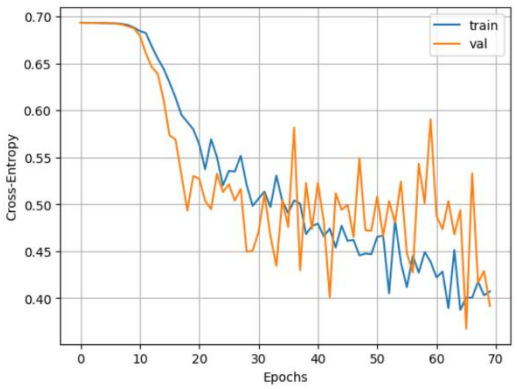



```
Resultados:
loss: [0.48261937499046326, 0.4847906529903412, 0.45975661277770996, 0.45312756299972534, 0.4810033738613128]
accuracy: [0.7875000238418579, 0.762499988079071, 0.7875000238418579, 0.78125, 0.765625]
kappa: [0.6125]
time_elapsed: 122.4216821193695
```

model.summary()
是 Keras/TensorFlow 中用于 快速查看神经网络模型结构 的方法，它会输出模型的层数、每层的参数数量、输出

Код используется для визуализации процесса обучения модели и вывода окончательных результатов оценки, помогая полностью проанализировать производительность модели, эффективность обучения и сходимость.






```
Resultados:
loss: [0.4826193749904632 6, 0.4847906529903412, 0.4597566127777099 6, 0.4531275629997253 4, 0.4810033738613128 7]
accuracy: [0.7875000238418579, 0.7624999880790 71, 0.7875000238418579, 0.78125, 0.765625]
kappa: [0.6125]
time_elapsed: 122.4216821193695
```

model.summary()
это метод в Keras/TensorFlow для быстрого просмотра структуры модели нейронной сети, который выводит такую информацию, как количество слоев модели, количество параметров в каждом слое и форма выходного сигнала.

形状等信息。

Model: "sequential_10"

Layer (type)	Output Shape	Param #
time_distributed_21 (TimeDistributed)	(None, 1, 135, 31, 4)	40
time_distributed_22 (TimeDistributed)	(None, 1, 67, 15, 4)	0
time_distributed_23 (TimeDistributed)	(None, 1, 67, 15, 4)	148
time_distributed_24 (TimeDistributed)	(None, 1, 33, 7, 4)	0
time_distributed_25 (TimeDistributed)	(None, 1, 924)	0
lstm_4 (LSTM)	(None, 4)	14,864
dense_18 (Dense)	(None, 32)	160
dense_19 (Dense)	(None, 2)	66

Total params: 45,836 (179.05 KB)
Trainable params: 15,278 (59.68 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 30,558 (119.37 KB)

```
print("Mean Accuracy: %.4f" % np.mean(array_acc))
print("std: (+/- %.4f)" % np.std(array_acc))
print("Mean Kappa: %.4f" % np.mean(array_kappa))
print("std: (+/- %.4f)" % np.std(array_kappa))
print("Max Accuracy: %.4f" % np.max(array_acc))
print("Max Kappa: %.4f" % np.max(array_kappa))
print("time_elapsed:", int(time_elapsed))
```

这段代码的作用是 汇总并展示模型多次实验的统计结果，通过计算均值、标准差和最大值，全面评估模型的性能稳定性、最佳表现及计算效率。为模型性能的 最终评估报告

```
Mean Accuracy: 0.7875
std: (+/- 0.0163)
Mean Kappa: 0.5563
std: (+/- 0.0000)
Max Accuracy: 0.8188
Max Kappa: 0.5563
time_elapsed: 109
```

平均准确率 78.75%表示模型在所有测试样本中平均有 78.75% 的样本分类正确。
评价：对于二分类任务，此结果属于中等水平（一般 80%以上为较好）。
准确率标准差 ±0.0163 多次实验结果的波动范围很小（约 1.6%），说明模型性能 稳定。
平均 Kappa 系数 0.5563 衡量模型分类结果与真实标签的一致性，超越随机猜测的水平。
评价：0.55 属于中等一致性（Landis & Koch 标准：0.41-0.60 为中等），说明模

Model: "sequential_10"

Layer (type)	Output Shape	Param #
time_distributed_21 (TimeDistributed)	(None, 1, 135, 31, 4)	40
time_distributed_22 (TimeDistributed)	(None, 1, 67, 15, 4)	0
time_distributed_23 (TimeDistributed)	(None, 1, 67, 15, 4)	148
time_distributed_24 (TimeDistributed)	(None, 1, 33, 7, 4)	0
time_distributed_25 (TimeDistributed)	(None, 1, 924)	0
lstm_4 (LSTM)	(None, 4)	14,864
dense_18 (Dense)	(None, 32)	160
dense_19 (Dense)	(None, 2)	66

Total params: 45,836 (179.05 KB)
Trainable params: 15,278 (59.68 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 30,558 (119.37 KB)

```
print("Mean Accuracy: %.4f" % np.mean(array_acc))
print("std: (+/- %.4f)" % np.std(array_acc))
print("Mean Kappa: %.4f" % np.mean(array_kappa))
print("std: (+/- %.4f)" % np.std(array_kappa))
print("Max Accuracy: %.4f" % np.max(array_acc))
print("Max Kappa: %.4f" % np.max(array_kappa))
print("time_elapsed:", int(time_elapsed))
```

Цель данного кода - обобщить и представить статистические результаты многочисленных экспериментов с моделью, а также полностью оценить стабильность работы модели, ее оптимальность и вычислительную эффективность, рассчитав среднее, стандартное отклонение и максимальное значение. Это окончательный отчет об оценке эффективности модели.

```
Mean Accuracy: 0.7875
std: (+/- 0.0163)
Mean Kappa: 0.5563
std: (+/- 0.0000)
Max Accuracy: 0.8188
Max Kappa: 0.5563
time_elapsed: 109
```

Средняя точность 78,75% означает, что модель правильно классифицировала в среднем 78,75% всех тестовых образцов.

ОЦЕНКА: Для задач бинарной классификации этот результат является умеренным (обычно 80 % или выше).

Стандартное отклонение точности ±0,0163 имеет небольшой диапазон колебаний (около 1,6%) в результатах множества экспериментов, что

型有一定可靠性但仍有改进空间。

Kappa 标准差 ±0.0000

多次实验 Kappa 值完全一致，可能因数据分布固定或实验设置未引入随机性（需检查代码）。

最佳单次准确率 81.88%

模型在最优情况下能达到 81.88% 的准确率，显示其潜力。

耗时 109 秒

模型训练或预测的总时间为 109 秒（约 1 分 49 秒），计算效率较高

говорит о стабильности работы модели.

Средний коэффициент Каппа 0,5563 измеряет согласованность результатов классификации модели с истинными метками за пределами уровня случайного угадывания.

Оценка: 0,55 - умеренная согласованность (критерий Лэндиса и Коха: 0,41-0,60 - умеренная), что говорит о том, что модель обладает некоторой надежностью, но все еще имеет возможность для улучшения.

Стандартное отклонение Каппы ±0,0000

Значения Kappa идеально совпадают в нескольких экспериментах, возможно, благодаря фиксированному распределению данных или экспериментальной установке, не вносящей случайности (необходимо проверить код).

Оптимальная точность одиночного выстрела 81,88 %

Модель способна достичь точности 81,88 % в оптимальном случае, демонстрируя свой потенциал.

Затраченное время 109 секунд

Общее время, затраченное на обучение или предсказание модели, составило 109 секунд (около 1 минуты 49 секунд), что является эффективным с вычислительной точки зрения.