Отчет по лабораторному заданию №7

ИУ1И-41М Цзинь Сюаньфэн

Апрель 2025

1 Цель задания

Изучение когнитивных особенностей мозга с помощью данных ЭЭГ, освоение вейвлет-преобразования и изучение алгоритмов нейросетевой классификации. Характеристики извлекаются с помощью вейвлет преобразования для создания изображений и передаются для обучения CNN для классификации.

通过脑电图数据研究大脑认知特征，掌握小波变换的方法，研究神经网络分类算法。利用小波变换提取特征，生成图像，并传入到CNN训练，实现分类。

2 Результаты и Обсуждения

2.1 Анализ ээг-сигнала

Непрерывное вейвлет-преобразование - это метод, используемый для анализа сигналов одновременно во временной и частотной областях путем преобразования сигнала с помощью вейвлет-функций в различных масштабах и местах для получения локальных свойств сигнала. Основная идея CWT заключается в разложении сигнала на вейвлеты в различных масштабах (частотах) и местах. Вейвлет Морлета - это одночастотный комплексный синусоидальный модулированный гауссовский вейвлет, наиболее часто используемый комплексно-значный вейвлет с хорошим разрешением во временной и частотной областях. Для

连续小波变换是一种用于在时域和频域上同时分析信号的方法，它通过使用不同尺度和位置的小波函数对信号进行变换，以获取信号的局部特性。CWT的核心思想是在不同尺度（频率）和位置上对信号进行小波分解。 Morlet小波是一种单频复正弦调制高斯波，也是最常用的复值小波，在时频两域均具有良好的分辨率。为了得到数据特征，采用了cmor3-3小波。 训练集大小为（400,3000），通过cwt将每组数据转化为包含时域和频域信息的二维图像（scalograms），作归一化处理，并将其作为CNN的输入。其中一个图像如下。

получения характеристик данных используется вейвлет cmor3-3. Размер обучающего набора составляет (400,3000), и каждый набор данных преобразуется в двумерные изображения (скалограммы), содержащие информацию во временной и частотной областях, с помощью вейвлет преобразования, нормализуется и используется в качестве входных данных для CNN. Одно из изображений выглядит следующим образом.

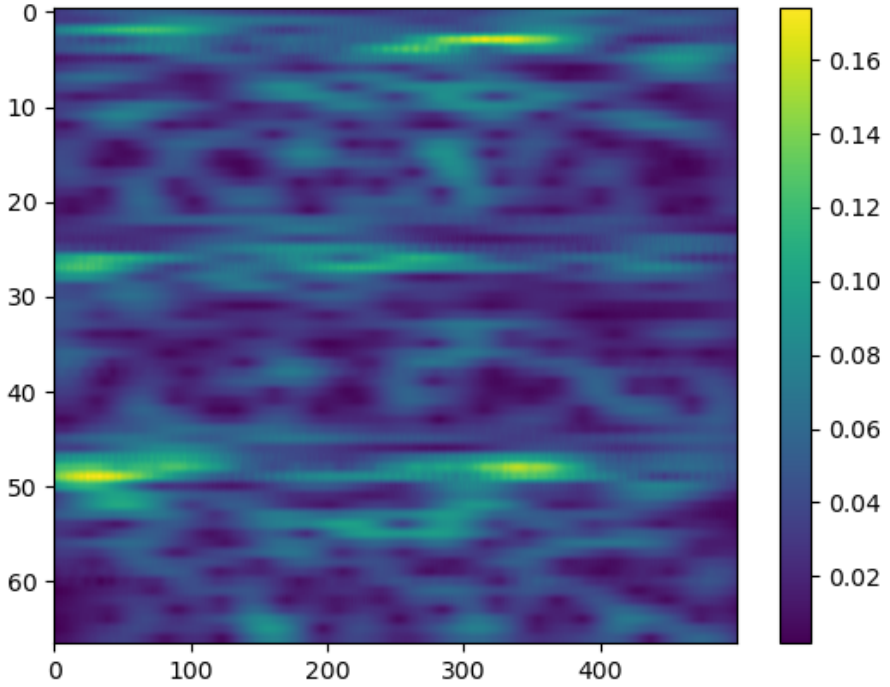

Рис. 1: скейлограмма из обучающего набора

2.2 Обучение классификации нейронных сетей

Размер нового обучающего набора, полученного после обработки, составляет (400, высота, ширина). В то время как в большинстве библиотек глубокого обучения (например, TensorFlow/Keras) формат данных конволюционного слоя - 4 измерения, и количество каналов необходимо увеличивать как новое измерение. Модель CNN была обучена, и были получены следующие результаты.

经处理后得到的新训练集大小为（400，高度，宽度）。而在大多数深度学习库（如 TensorFlow/Keras）中，卷积层的数据格式都是 4 维，需要增加通道数作为新维度。训练的结果如下。

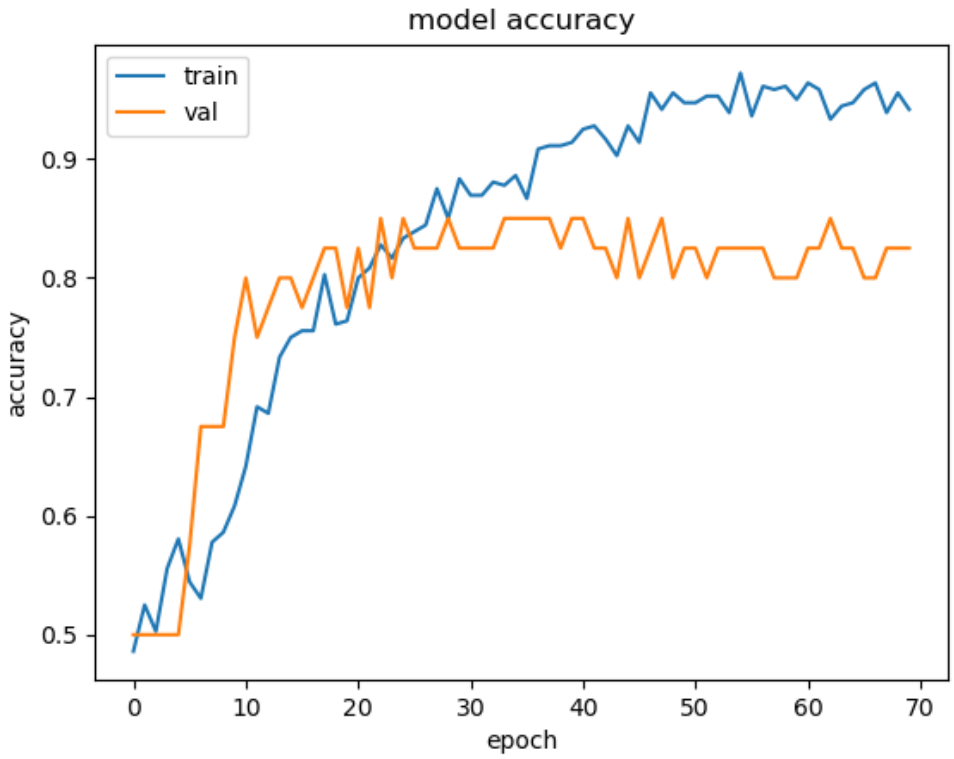

Рис. 2: Кривая изменения точности

Матрица смешения показана на рисунке.

混淆矩阵如图，相关指标计算如表格所示。

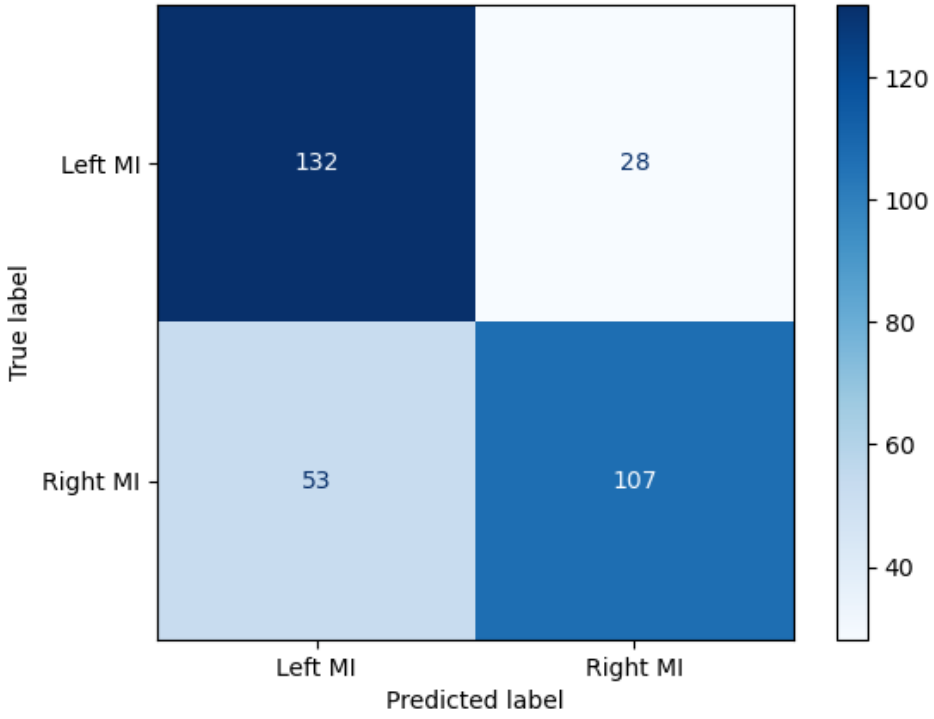

Рис. 3: Матрица смешения

Таблица 1: Расчет ключевых показателей эффективности

Accuracy	Precision	Recall	F1-Score	Cohen's Kappa
74.69%	71.35%	82.50%	76.6%	$\kappa \approx 0.494$(средней консистенции)

3 Ссылки на литературу

https://github.com/TAUforPython/BioMedAI/blob/main/NN%20CNN%20LSTM%20EEG%20DF%20MI%20Class.ipynb
https://cloud.tencent.com/developer/article/2322066

4 Код

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import time
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_v2
# download dataset
x_train = pd.read_csv("./lab7/MI-EEG-B9T.csv",header=None)
x_test = pd.read_csv("./lab7/MI-EEG-B9E.csv",header=None)
y_train = pd.read_csv("./lab7/2class_MI_EEG_train_9.csv",header=None)
y_test = pd.read_csv("./lab7/2class_MI_EEG_test_9.csv",header=None)

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
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)
# count classes
n_classes = 2

# 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)
    # complex morlet wavelet
    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
x_train = scalogram_vertical(x_train, 250, 135, 1000, 3, 1000)
x_test = scalogram_vertical(x_test, 250, 135, 1000, 3, 1000)

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

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
#print(x_train[1].shape)
plt.figure()

plt.imshow(x_train[50], aspect='auto')
plt.colorbar()
plt.show()

x_train = x_train.reshape((x_train.shape[0], x_train.shape[1], x_train.shape[2],
x_test = x_test.reshape((x_test.shape[0], x_test.shape[1], x_test.shape[2], 1))

print(x_train.shape[1:])
```

```
print(x_test.shape)

def CNN_2D(num_filter, size_filter, n_neurons):
    model = Sequential()
    model.add(Conv2D(num_filter, kernel_size=size_filter, activation='relu', pad
                     input_shape=x_train.shape[1:]))
    model.add(MaxPool2D((2, 2)))
    model.add(Conv2D(num_filter, kernel_size=size_filter, activation='relu', pad
    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_v2.Adam(learning_rate=0.001)
    model.compile(optimizer=optimizer,
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model

array_loss = []
array_acc = []
array_kappa = []

for i in range(5):
  print("Iteration:", i+1)
  model = CNN_2D(4, (3,3), 32)
  #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)


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()
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)
from sklearn.metrics import ConfusionMatrixDisplay
labels = ["Left MI", "Right MI"]
disp = ConfusionMatrixDisplay(confusion_matrix=matriz_confusion, display_labels=
disp.plot(cmap=plt.cm.Blues)
plt.show()
model.summary()
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))
```