一种基于脑电信号诊断癫痫的 CNN 算法

20 数据科学　王婧怡

202000810044

摘　要： 脑电信号是一种常用的辅助诊断癫痫的手段。传统上，神经病学家采用直接的视觉检查来识别癫痫异常，这样的做法不仅非常耗时，同时受到设备硬件条件、学者的专业水平差异的限制, 导致癫痫检测存在不可避免的错误。传统机器学习对癫痫检测有较深研究，但是对癫痫的发作期和发作间期的预测仍然存在一定难度。本文利用深度神经网络，根据脑电波的频率将脑电信号分解成 δ、θ、α、β、γ 五大类脑波，逐一提取时域特征、频域特征。利用卷积神经网络模型提取深度特征，自动诊断癫痫发作间期和发作期，消除对手工特征的依赖，降低人工的成本，并且利用了十折交叉验证法保证了模型的鲁棒性，达到了 100% 的准确率，相较于传统算法有明显的优势。

关键词：癫痫、EEG、特征提取、CNN

1　背景

癫痫 (Epilepsy) 是影响全年龄人群的一种由脑部神经元阵发性异常超同步电活动 (如图1) 导致的慢性非传染性疾病也是全球最常见的神经系统疾病之一。癫痫的反复发作会对患者的精神与认知功能造成持续性的负面影响，甚至危及生命。因此，癫痫诊断和治疗的研究具有非常重要的临床意义。

脑电图 (electroencephalogram, EEG) 是放置于头皮特定位置的电极采集获得的大脑内同步神经元活动产生的微伏级电信号，是诊断癫痫相关疾病最有效的方法。

尽管传统的机器学习方法对癫痫发作的检测目前能够达到较高水平，但是对癫痫的发作期和发作间期的预测仍然存在一定难度，并且对提取出的特征有极高的要求。本文利用了一个高效简洁的卷积神经网络模型，对五个不同频率的脑电信号 δ、θ、α、β、γ 提取时域、频域特征，在较低的特征成本情况下，准确地预测出发作间期、发作期。

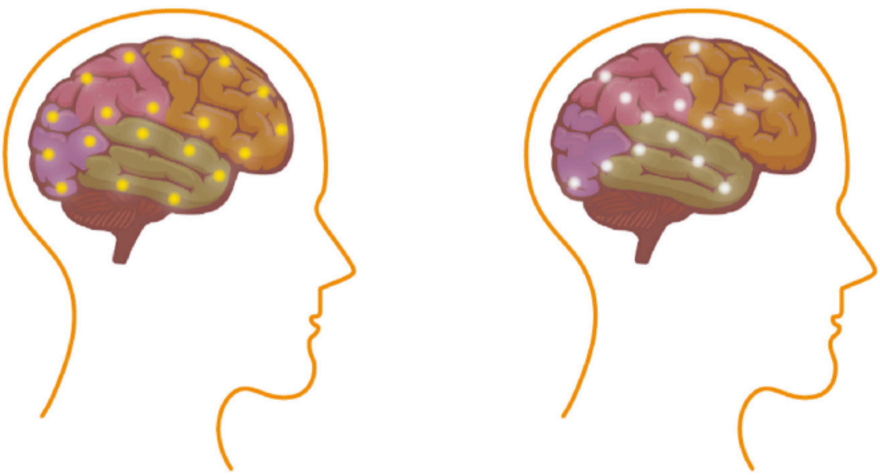


图 1: 大脑中正常活动和癫痫活动的图示

2 项目的实现

2.1 数据集介绍

本研究中使用的 EEG 片段是由 Andrzejak 等人 [1] 在德国波恩大学收集的数据集 (http://epilepsy.uni-freiburg.de/database)。

该数据集是由 5 个健康人和 5 个癫痫患者的脑电数据构成 (表1所示)，包含三类单通道数据，即健康（B_O）组，发作间期（D_F 组），发作期（E_S 组）。每一组都包含 100 个数据片段，每个数据片段的时间长度为 23.6 秒，数据点为 4097 个。该脑电图信号都是用同一个 128 通道的放大器系统记录。经过 12 位模数转换后，数据被连续写到数据采集的计算机磁盘上，采样率为 173.61hz，并进行了 0.5-70hz 的带通滤波。

表 1: 波恩数据集描述

	健康人	癫痫患者	
状态	闭眼	发作间期	发作期
数据集	B_O	D_F	E_S
数据类型	头皮 EEG	颅内 EEG	颅内 EEG
电极位置	头皮	病灶区	病灶区

2.2 项目流程概览

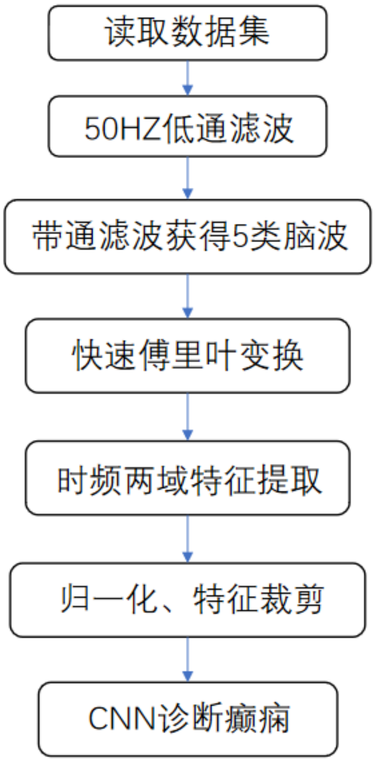


图 2: 项目流程图

2.3 项目实现具体步骤

由于每个组都包含 100 个数据片段，本文先将同组的 100 个数据（4097×1）进行合并，得到正常组（4097×1×100）、发作间期组（4097×1×100）、发作组（4097×1×100）的原时域数据，再按照正常-发作间期-发作期的顺序合并成 4097×1×300 大小的数组，方便后续操作。

2.3.1 时域滤波

该数据集已进行了 0.5-70hz 的带通滤波，采样频率大于被采样信号最高频率的两倍，符合奈奎斯特采样定理，利用离散采样点重建出连续的信号，后续不会出现信号混叠的现象。

脑电的有效信息大多集中于 0.5-50hz 之间，对于该数据集，还需 50hz 的低通滤波器，消除采集过程中受到的 50hz 工频信号影响，同时本文利用 matlab 设计了一个 4 阶的巴特沃斯带通滤波器，实现零相位滤除干扰信号，将该滤波器系数代入 matlab 自带的 filtfilt 函数来提取 δ、θ、α、β、γ 的时域波。该滤波器系数 a(1×9)，b（1×9）如表2所示（以 Delta 波为例）：

表 2: 滤波器系数

a	1.000	-7.659	25.6795	-49.229	59.0214	-45.315	21.7587	-5.974	0.7180
b	1.3e-05	0.000	-5.4e-05	0.000	8.2e-05	0.000	-5.4e-05	0.000	1.3e-05

得到系数 a,b 后，带入原始信号时域信号 X(n) 解常系数线性差分方程 (如下) 得滤波后的时域信号 Y(n)：

$$\sum_{k=0}^{9} a_k Y(n-k) = \sum_{m=0}^{9} b_m X(n-m)$$

我们知道,EEG 信号可以根据频谱划分成 δ、θ、α、β、γ 五大类波。脑波种类和对应的频率范围如下所示:

表 3: 五类波介绍

成分波	频率范围	大脑区域	对应人体活动
δ 波	0.5-4HZ	额部	睡眠状态
θ 波	4-9HZ	额区、颞区	困倦、生病状态
α 波	8-13HZ	顶枕区	清醒、放松状态
β 波	13-32HZ	大脑前半球	激动、紧张、兴奋状态
γ 波	>30HZ	全脑	高度兴奋状态

本次实验中，本文依照上表，设计了 5 个 IIR 滤波器，依次得到 5 个相应频段的 δ、θ、α、β、γ 脑波，将原始信号从单通道拓展为 5 个通道的时域信号

2.3.2 快速傅里叶变换

傅里叶原理表明：任何连续测量的信号都可以表示为不同频率的正弦波信号的无限叠加。事实上，有些信号难以在时域上提取有效特征，而傅里叶变换的作用就是将波从时域转到频域，帮助我们换个角度分析信号。

但一般的离散傅里叶变换（DFT）的算法复杂度为 $o(N^2)$，在实际使用中将会耗费大量时间在计算上。为减少计算花费时间，本文使用的是快速傅里叶变换（FFT），它本身就是 DFT 的快速算法，复杂度为 $o(NlogN)$。

通过上述时域滤波步骤，我们得到 5 个通道是时域信号，分别对其进行快速傅里叶变换（FFT）, 得到 5 个通道的频域信号，再提取单侧幅值频谱, 方便后续频域分析。此时每个样本拥有 4097×5 的时域信号，2049×5 的频域信号.

2.3.3 特征提取

通过对信号进行分析, 提取出有效的特征作为分类依据, 是实现癫痫自动检测的重要步骤. 对每个样本的信号进行特征提取, 时频两域信号的特征有:

时域特征: 最大值、最小值、极差、均值、标准差、方差、中位数、绝对中值误差、四分位差、绝对平均值、峭度、偏度、波形因子、脉冲因子、裕度因子、峰度因子.

频域特征: 重心频率、均方频率、频率方差、均值、变异系数、极大值、四分位数、功率、绝对中值误差、信息熵、绝对平均值、峰度、偏度、波形因子、峰值因子、脉冲因子、裕度因子.

本文对每个样本, 从时频两域 (共 10 个通道) 依次提取 16 个特征, 则 300 个样本一共得到 300×[16×(5+5)] 个特征值.

2.3.4 归一化

在进行分类之前, 为减少神经网络的计算量, 本文采用归一化的操作, 将每个样本的 160 个特征映射到 [0,1] 之间, 使不同表征的数据规约到相同的尺度内, 同时保留原始分布信息, 减少运算时间.

2.3.5 特征裁剪

在实际应用中, 我们当然希望在较短的时间里得到准确的癫痫诊断结果, 提高医生工作效率, 节省患者看病时间. 为了验证利用卷积神经网络在特征较少的情况下仍然具有较好的表现能力, 本文将每个样本的 1×(4097+160) 个特征（其中 1×4097 是时域信号）, 分别裁剪成 1×4096 和 1×169 大小的特征, 再调整成 64×64 和 13×13 作为卷积的输入进行对比试验.

表 4: 网络输入的组成

组成		裁剪后	输入网络的尺寸
特征 (1×160)	样本信号 (1×4097)		
1×160	1×9	1×169	13×13
1×160	1×3936	1×4096	64×64

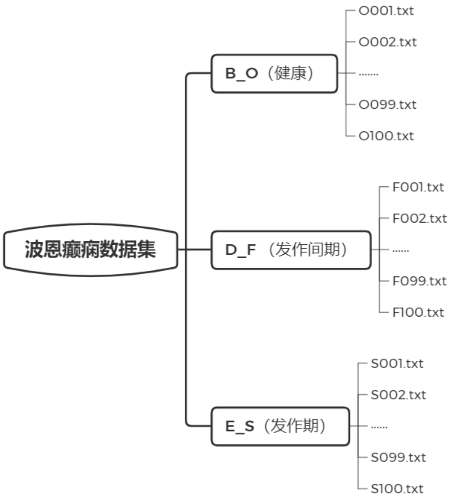
2.3.6 CNN 诊断癫痫

本文对三类特征采用当下比较流行的卷积网络 Swin-transformer 来进行分类. 将结果与传统机器学习算法结果进行对比研究。每种算法实验均是将 300 个样本按照 7:3 划分成训练集和测试集。

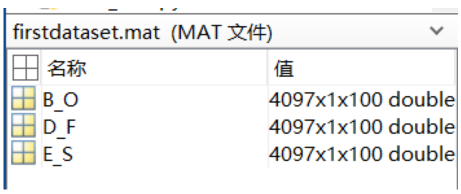
3 实验结果及可视化

3.1 加载数据集

该数据集一共有三类 (正常, 发作间期, 发作期) 文件, 每个文件下包含 100 个数据片段的 txt. 首先利用 matlab 读取每个 txt, 再将相同类型进行合并, 得到三个 4097×1×100 的矩阵



(a) 数据集结构

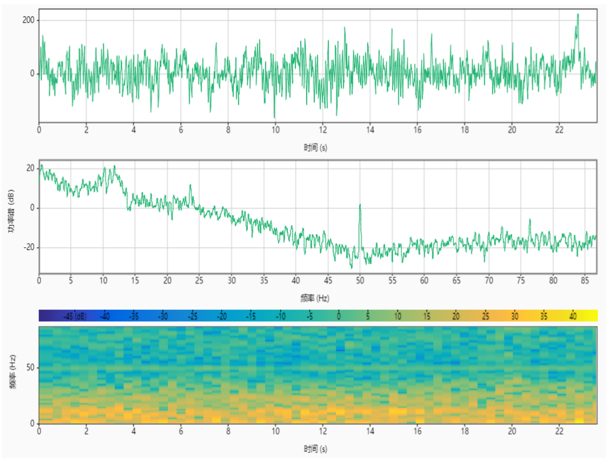


(b) 合并后的矩阵

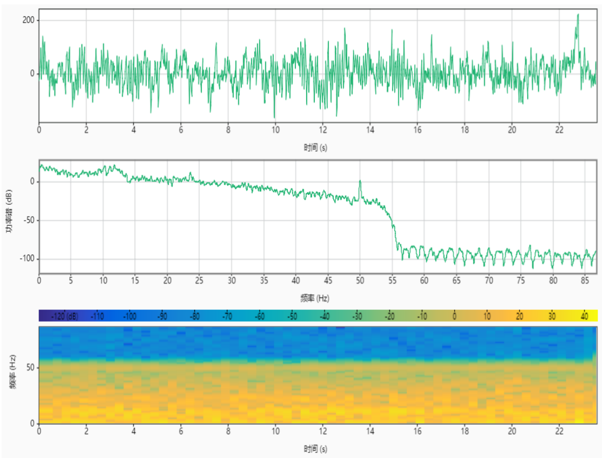
图 3: 数据集介绍

3.2 滤波前后对比

为了可视化更直观方便, 本文分别从三个类别中抽取一个样本出来, 进行 50hz 低通滤波, 消除工频信号带来的影响. 依次做三个样本滤波前后的**时域图**, **频谱图**, **时频图**对比，如图4,5,6

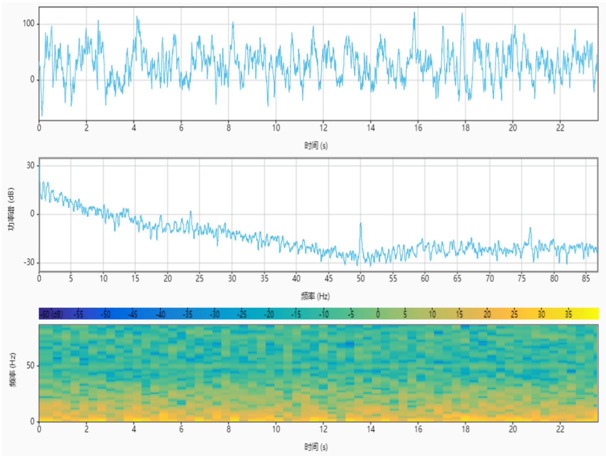


(a) 滤波前

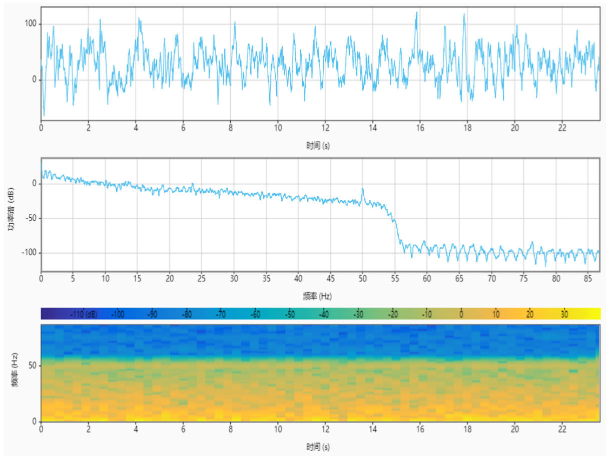


(b) 滤波后

图 4: 健康人

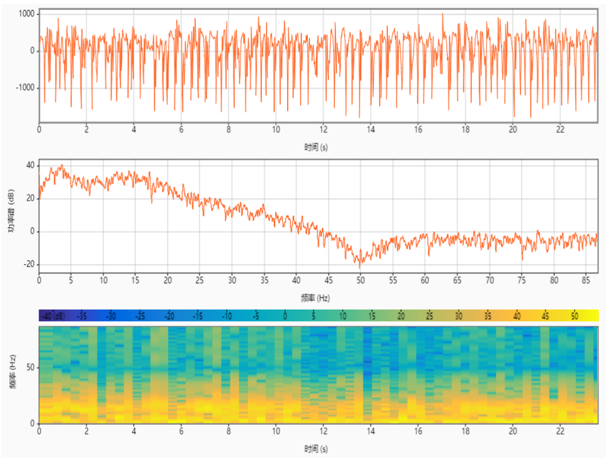


(a) 滤波前

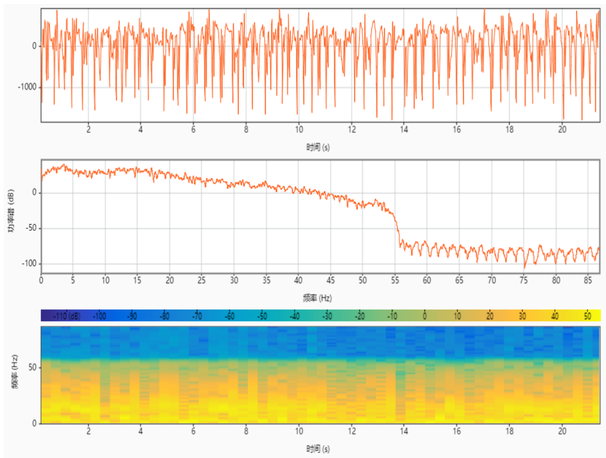


(b) 滤波后

图 5: 发作间期的癫痫患者



(a) 滤波前

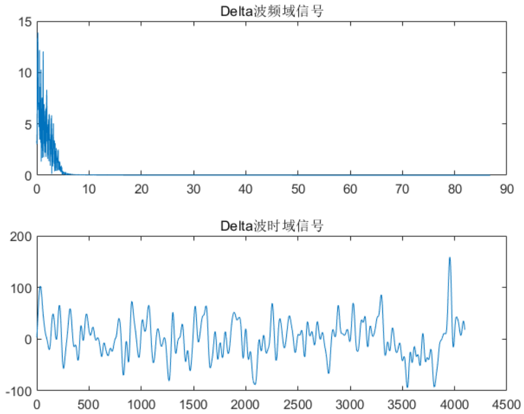


(b) 滤波后

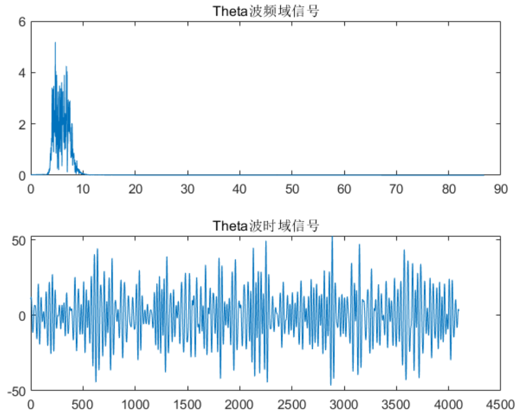
图 6: 发作期的癫痫患者

3.3 五类脑波提取及快速傅里叶变换

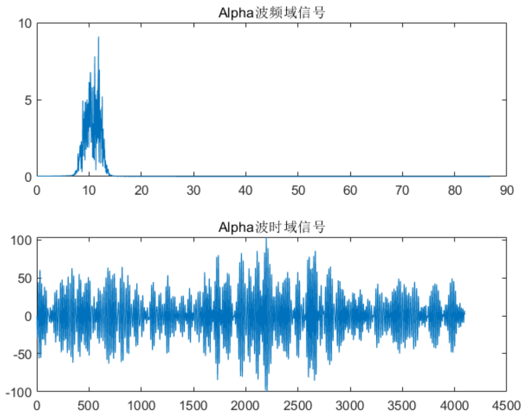
根据表3列出的脑波频率范围，我们利用带通滤波和快速傅里叶变换依次得到五种脑波的时域、频域信号。这里以随机抽取一个样本为例，可视化该样本的五个时域、频域信号。



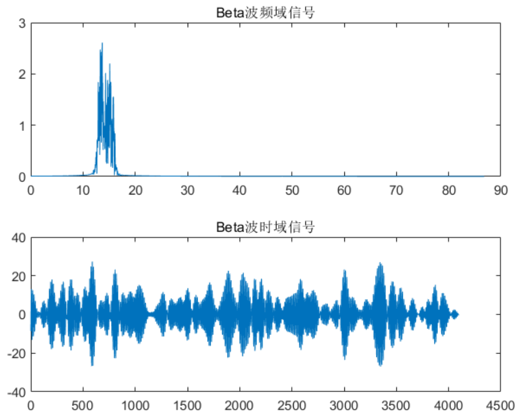
(a) Delta 时频域信号



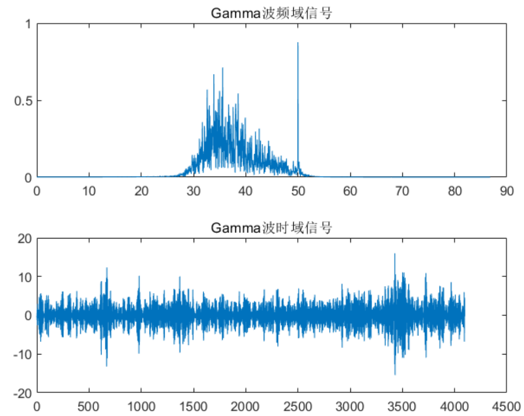
(b) Theta 时频域信号



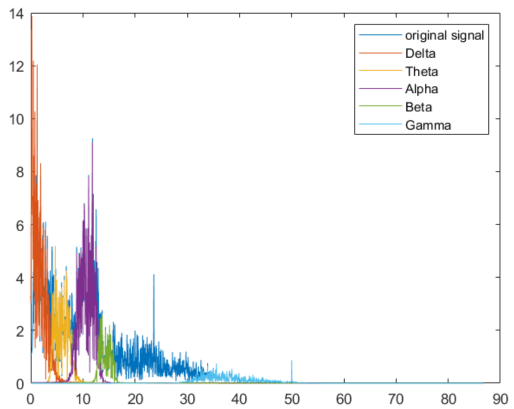
(c) Alpha 时频域信号



(d) Beta 时频域信号



(e) Gamma 时频域信号



(f) 总频谱图

图 7: 五波时频域信号可视化

3.4 特征提取、归一化

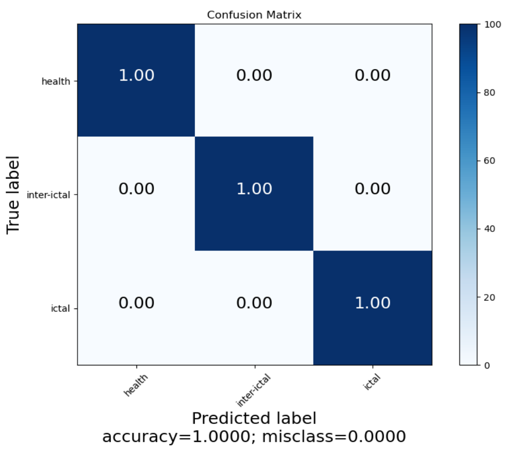
我们将得到的五个通道的时域和频域信号分别提取 16 个特征，300 个样本将得到 300×16×（5+5）个特征，再对每一个样本的 160 个特征进行归一化。

3.5 算法分类结果

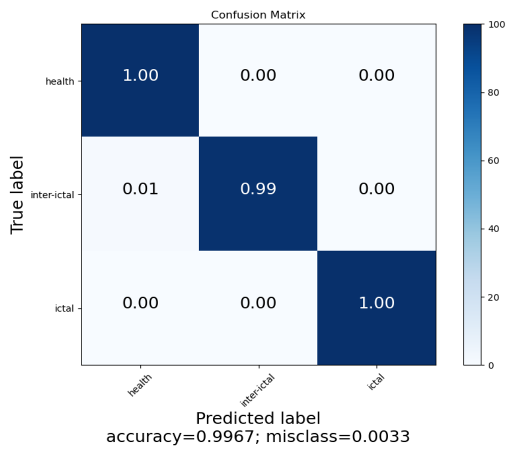
深度学习往往依赖大量的数据集，但受实际研究限制，很难获得大量训练样本，为了降低模型过拟合的可能性，提高模型的可靠性和稳定性，本文使用**十折交叉验证**的方法训练网络，将 300 个样本随机分成 10 个相等的成分，其中 9 份用来训练 CNN，1 份用来作为测试集，重复实验 10 次，保证 10 份子数据都分别做过测试数据，最后把得到的 10 个实验结果进行平分。本文将两个不同大小的特征（13×13，64×64）作为网络输入，经过 10 折交叉验证后，相应结果如表5，健康-发作间期-发作期的预测分类混淆矩阵如图8：

表 5: 不同尺寸下的实验结果

Input size	**Precision%**	**Recall%**	**F1-score%**	**Accuracy%**
13×13	100	100	100	100
64×64	99.6	99.7	99.7	99.7



(a) input:13×13



(b) input:64×64

图 8: 混淆矩阵

不难看出，在输入尺寸较小，特征少的情况下进行 10 折交叉验证，CNN 仍然可以达到 100% 的准确率。这意味着在实际应用中，卷积神经网络可以利用较少的特征，提取深度特征，达到可观的效果。

为了证明本文所提出的模型的优势，我们将该模型与使用相同数据集的癫痫检测方法进行比较。结果如下所示：

表 6: 方法对比

作者	创新点	分类器	Accuracy%	Sensitivity%	Specify%
Chua et al.[2]	HOS feature.	GMM	93.1	89.7	94.8
Faust et al. [3]	功率谱密度计算	SVM	93.1	98.3	96.7
Acharya et al.[4]	离散小波变换	SVM	96.3	100	97.9
Bhattavharyya et al.[5]	经验模态分解	Random forest	99.4	97.9	99.5
Acharya et al.[6]	13 层深度卷积	CNN	88.7	95.0	90.0
Our work	十折交叉验证、Swin transformer	**CNN**	**100**	**100**	**100**

准确率（Accuracy）、灵敏度（Sensitivity）、特异性（Specify）都是评价分类模型好坏的指标，越高则说明模型好，泛化能力强。

从上表可以看出，即使我们使用较少的特征，也能在众多方法中脱颖而出，达到非常高的准确率。这样不仅可以减少人工提取的特征量，在日后医学上实际应用中也有较高的可靠性。

4 总结

本文提出了基于深度卷积神经网络检测癫痫的方法，提高了利用 EEG 检测癫痫发作期、发作间期的准确率。相比于其他需要提取较多特征的方法模型，本文利用十折交叉验证方法保证模型在较少的特征下，仍能表现出色。

尽管深度学习在识别癫痫中表现良好，但该类方法也存在许多挑战，首先是深度学习依赖于大量的数据集，这对模型的鲁棒性和准确性会造成较大影响。其次，现有的公开数据集几乎都是稍处理过的 EEG 信号片段，与实际场景中连续实时信号存在差异。如何使深度神经网络模型真正应用于现实场景，还有很长的路要走。

5 源码介绍

5.1 程序运行流程

1. 下载数据集：https://www.upf.edu/web/ntsa/downloads 选择‘The Bonn EEG time series download page’，下载 SET A、B、C、D、E

2. 将五个文件与 writeindata.m 放在同一个路径下，运行该.m 文件，得到 5 个 4097×1×100 的矩阵，本文仅需要 B、D、E 这三个数据

3. 运行 data_process.m 文件，得到时频两域的信号矩阵

4. 运行 extracted_features.m，得到时频两域特征个 16 个

5. 运行 combine_fea_ori.m，得到提取的特征 + 原时域信号作为输入的数据集的 feature.csv 文件

6. 制作 label.csv，因为数据只有 300 个，第 1-100 行为健康人，设为 0；101-200 行为发作间期，设为 1；201-300 行为发作期，设为 2. 将 feature.csv，label.csv 文件放在指定文件夹内。

7. 运行 main.py 进行模型训练。

5.2 具体源码

5.2.1 matlab 部分

writeindata.m（将 100 个 txt 文件->4097×1×100 的矩阵）

```
path = '.\';
str={'A_Z','B_O','C_N','D_F','E_S'};%设定5个字符串
for class=1:5
[FileNames] = GetFileNames([path,str{class},'\'],'*.txt');
    switch(class)
        case 1
            [A_Z]= WriteInTxtdata([path,str{class},'\'],FileNames);
        case 2
            [B_O]= WriteInTxtdata([path,str{class},'\'],FileNames);
        case 3
            [C_N]= WriteInTxtdata([path,str{class},'\'],FileNames);
        case 4
            [D_F]= WriteInTxtdata([path,str{class},'\'],FileNames);
        case 5
            [E_S]= WriteInTxtdata([path,str{class},'\'],FileNames);
    end
end
save firstdataset.mat  B_O D_F E_S

function [FileNames] = GetFileNames(Path,Format)
% GetFileNames
% 函数的功能为获得某一路径下，某种格式所有文件名
% 函数的输入1为Path,要获取的路径。eg: 'D:\Program Files\FileZilla FTP Client\docs\'
% 函数的输入2为Format，要获取路径的文件格式。eg: '*.txt','*.docx','*.png'

 fileFolder = fullfile (Path);
dirOutput=dir( fullfile ( fileFolder ,Format));
FileNames={dirOutput.name};

end
function [curve_data]= WriteInTxtdata(Path,FileNames)
% 函数的返回curve_data是一个3d矩阵，第三位为文件个数表示个文件
```

```
% Path：写入的文件地址目录字符串.eg: 'D:\Program Files\FileZilla FTP Client\docs\'
% FileNames:所有的文件名字胞组矩阵{'S001.txt','S002.txt','S003.txt',...}
 files  = dir(Path);
number_files = length(files)-2;
for i=1:number_files

 fileID  = fopen([Path,FileNames{i}],'r');
formatSpec = '%f';
curve_data(:,:,i)= fscanf(fileID,formatSpec);
 fclose(fileID);
end
end
```

data_process.m(滤波 +FFT)

```
load database
% 将健康闭眼、发作间期、发作期的三个subdataset合并在一起->4097，1，300
ori_dataset = cat(3,B_O, D_F, E_S);
Fs=173.61; % 采样率
N = 4097; % 采样点
f = (0:N-1)*Fs/N ;% 采样点tensor
% 对每一行分别进行特征提取
five_data = [];
five_fft_data = [];
for i=1:length(ori_dataset(1,1,:))
    data = ori_dataset(:,:,i); % (1,1,100)
    % 低通滤波，去除50hz以上
    data = low_pass(data,Fs);
    % ---提取5种波-- 滤波+快速傅里叶变换-----------
    % Delta 0.5-4hz
    W_del = [0.5*2 4*2] /Fs;
    [del, del_fft] = filterandfft (W_del,4,data);
   % Theta 4-9hz
   W_the = [4*2 8*2] /Fs;
   [the, the_fft] = filterandfft (W_the,4,data);
   % Alpha 8-13hz
   W_alp = [8*2 13*2] / Fs;
   [alp, alp_fft] = filterandfft (W_alp,4,data);
   % Beta 13-32hz
   W_be = [13*2 32*1] / Fs;
   [be, be_fft] = filterandfft (W_be,4,data);
   % Gamma >32hz
   W_ga = [32*2 50*2] / Fs;
```

```
    [ga, ga_fft] = filterandfft (W_ga,4,data);
    sub_data = [del,the,alp,be,ga]; % 横向拼接 size(4097,5)
    sub_fft_data = [del_fft,the_fft,alp_fft,be_fft,ga_fft];% size(2049,5)
   % 在第三维进行拼接 用cat即可
   five_data = cat(3,five_data,sub_data);
   five_fft_data = cat(3,five_fft_data,sub_fft_data);
end
```

extracted_features.m(提取时频两域信号)

```

% 特征提取
load load_data
all_features = [];
for i=1:300 % 有300个sample
    sample_data = five_data(:,:,i); % 4097*5
    sample_fft_data = five_fft_data(:,:,i);%2049*5
    sample_features = [];
    % 特征提取 sample_data
    for j=1:5 % 对4097*1进行特征分析
        % 时域
        sub_data = sample_data(:,j); % 4097*1
        sub_fft_data = sample_fft_data(:,j); % 16
        sample_features = [sample_features,max(sub_data)];
        sample_features = [sample_features,min(sub_data)];
        sample_features = [sample_features,(max(sub_data)-min(sub_data))];
        sample_features = [sample_features,mean(sub_data)];
        sample_features = [sample_features,std(sub_data)];
        sample_features = [sample_features,var(sub_data)];
        sample_features = [sample_features,mad(sub_data)];
        sample_features = [sample_features,iqr(sub_data)]; % 四分位差
        sample_features = [sample_features,mean(abs(sub_data))];
        sample_features = [sample_features,kurtosis(sub_data)]; % 峭度
        sample_features = [sample_features,skewness(sub_data)]; % 偏度
        sample_features = [sample_features,rms(sub_data) / mean(abs(sub_data))]; % 波形因子
        sample_features = ...
            [sample_features,(max(sub_data)-min(sub_data))/rms(sub_data)];% 峰值因子
        sample_features = ...
            [sample_features,(max(sub_data)-min(sub_data))/mean(abs(sub_data))];% 脉冲因子
        sample_features = [sample_features,(max(sub_data)-min(sub_data))/ ...
            mean(sqrt(abs(sub_data)))^2]; % 裕度因子
        sample_features = [sample_features,entropy(sub_data)]; %绝对中值误差
        % 频域
```

```
        sample_features = [sample_features,max(sub_fft_data)];
        sample_features = [sample_features,min(sub_fft_data)];
        sample_features = [sample_features,(max(sub_fft_data)-min(sub_fft_data))];
        sample_features = [sample_features,mean(sub_fft_data)];
        sample_features = [sample_features,std(sub_fft_data)];
        sample_features = [sample_features,var(sub_fft_data)];
        sample_features = [sample_features,mad(sub_fft_data)];
        sample_features = [sample_features,iqr(sub_fft_data)]; % 四分位差
        sample_features = [sample_features,mean(abs(sub_fft_data))];
        sample_features = [sample_features,kurtosis(sub_fft_data)]; % 峭度
        sample_features = [sample_features,skewness(sub_fft_data)]; % 偏度
        sample_features = [sample_features,rms(sub_fft_data) / mean(abs(sub_fft_data))]; ...
            % 波形因子
        sample_features = ...
            [sample_features,(max(sub_fft_data)-min(sub_fft_data))/rms(sub_fft_data)];% ...
            峰值因子
        sample_features = ...
            [sample_features,(max(sub_fft_data)-min(sub_fft_data))/mean(abs(sub_fft_data))];% ...
            脉冲因子
        sample_features = [sample_features,(max(sub_fft_data)-min(sub_fft_data))/ ...
            mean(sqrt(abs(sub_fft_data)))^2]; % 裕度因子
        sample_features = [sample_features,entropy(sub_fft_data)]; %绝对中值误差

    end

    all_features = [all_features; sample_features]; % 1sample -> 160 features
end
```

combine_fea_ori.m(将提取到的特征和原时域信号合并，存为 csv)

```
% 300*160 4097*1*300
% 先分别z-scores，再合并到一起
load ori_dataset
load all_features
% ori_data zscores
new_dataset = ori_dataset;
Fs = 173.61;
% for i=1:300
% data = ori_dataset(:,:,i);
% data = (data-min(data))/(max(data)-min(data));
% new_dataset(:,:,i) = data;
% end
% all_features zscore
```

```
new_all_features = zeros(300,160);
for j=1:300
    feature = all_features(j,:);
    % 归一化
    feature = (feature-min(feature))/(max(feature)-min(feature));
    new_all_features(j,:) = feature;
end

Mydataset = [];
% 合并到一起，成300*（4097+160）
for m=1:300
    sample_fea = [];
    data_m = new_dataset(:,:,m); % 4097*1
    % 带通滤波 把0.5hz以及50hz以上的过滤掉
    W = [0.5*2 50*2] / Fs
    [data,data_fft] = filterandfft(W,4,data_m);
% data = low_pass(data_m,Fs);
    data = (data-min(data))/(max(data)-min(data));
    data = data'; % 1*4097
    feature = new_all_features(m,:); % 1*300
    sample_fea = [data,feature];
    Mydataset = [Mydataset;sample_fea]; % 1*4397
end
% 保存至feature.csv
train_feature = [];
train_feature = [train_feature;Mydataset(1:90,:)];
train_feature = [train_feature;Mydataset(101:190,:)];
train_feature = [train_feature;Mydataset(201:290,:)];
%
test_feature = [];
test_feature = [test_feature;Mydataset(91:100,:)];
test_feature = [test_feature;Mydataset(191:200,:)];
test_feature = [test_feature;Mydataset(291:300,:)];

% csvwrite('train_feature.csv', Mydataset(1:270,:));
% csvwrite('test_feature.csv', Mydataset(271:300,:));

csvwrite('train_feature_daitong.csv',train_feature);
csvwrite('test_feature_daitong.csv',test_feature);
```

matlab 处理信号过程中的使用到的自定义函数

1. filterandfft.m (带通滤波 +fft)

```matlab
function [newdata,fftdata] = filterandfft (Wn,n,signal)
    [b,a] = butter(n,Wn,'bandpass');
    N = length(signal);
    data = filtfilt (b,a,signal);
    fftdata = fft(data);
    newdata = ifft(fftdata);
    fftdata = abs(fftdata(1:round(N/2))*2/N);
end
```

2. low_pass.m (低通滤波)

```matlab
function y = low_pass(x,Fs)
% Fs = 1/mean(diff(tx)); % 平均采样率
y = lowpass(x,50,Fs,'Steepness',0.85,'StopbandAttenuation',60);
```

5.2.2 python 部分

main.py baseline: swin_transformer methond：十折交叉验证

```python
import os
import argparse
import torch
import torch.optim as optim
from my_dataset import MyDataSet
from model import swin_tiny_patch4_window7_224 as create_model
from utils import read_split_data, train_one_epoch, evaluate
from torch.utils.data import DataLoader,ConcatDataset,SubsetRandomSampler
from sklearn.model_selection import KFold
import numpy as np
from confusion import confusion_matrix,plot_confusion_matrix
from sklearn.metrics import classification_report


train_fea = 'dataset/train/train_feature_lowpass.csv'
train_label = 'dataset/train/train_label.csv'
test_fea = 'dataset/test/test_feature_lowpass.csv'
test_label = 'dataset/test/test_label.csv'


def main(args):
    batch_size = args.batch_size
```

```
device = torch.device(args.device if torch.cuda.is_available() else ”cpu”)
if os.path.exists(”./weights”) is False:
    os.makedirs(”./weights”)

# 实例化训练数据集
train_dataset = MyDataSet(train_fea, train_label)
val_dataset = MyDataSet(test_fea, test_label)
# 进行kflod
dataset = ConcatDataset([train_dataset,val_dataset])
k = 10
splits = KFold(n_splits=k,shuffle=True,random_state=42)
foldperf = {}

model = create_model(num_classes=args.num_classes).to(device)

if args.weights != ””:
    assert os.path.exists(args.weights), ”weights file: '{}' not exist.”.format(args.weights)
    weights_dict = torch.load(args.weights, map_location=device)[”model”]
    # 删除有关分类类别的权重
    for k in list(weights_dict.keys()):
        if ”head” in k:
            del weights_dict[k]
    print(model.load_state_dict(weights_dict, strict=False))

if args.freeze_layers:
    for name, para in model.named_parameters():
        # 除head外，其他权重全部冻结
        if ”head” not in name:
            para.requires_grad_(False)
        else:
            print(”training {}”.format(name))

pg = [p for p in model.parameters() if p.requires_grad]
optimizer = optim.AdamW(pg, lr=args.lr, weight_decay=5E-2)

for fold, (train_idx,val_idx) in enumerate(splits.split(np.arange(len(dataset)))):
    # 一共10个flod 每个flod有300*0.1*9个trainset 有 300*0.1*1个testset
    print('Fold {}'.format(fold + 1))
    train_sampler = SubsetRandomSampler(train_idx)
    test_sampler = SubsetRandomSampler(val_idx)
    train_loader = DataLoader(dataset, batch_size=batch_size, sampler=train_sampler)
    test_loader = DataLoader(dataset, batch_size=batch_size, sampler=test_sampler)
    history = {'train_loss': [], 'test_loss': [], 'train_acc': [], 'test_acc': []}
    for epoch in range(args.epochs):
        train_loss, train_correct = train_one_epoch(model=model,
```

```
                                                    optimizer=optimizer,
                                                    data_loader=train_loader,
                                                    device=device,
                                                    epoch=epoch)
        test_loss, test_correct,label_list, pre_list = evaluate(model=model,
                                        data_loader=test_loader,
                                        device=device,
                                        epoch=epoch)
        train_loss = train_loss / len(train_loader.sampler)
        train_acc = train_correct / len(train_loader.sampler) * 100
        test_loss = test_loss / len(test_loader.sampler)
        test_acc = test_correct / len(test_loader.sampler) * 100

        #### confusion matrix
        conf_mat = confusion_matrix(y_true=label_list,y_pred=pre_list)
        # plot_confusion_matrix(conf_mat,normalize=True,
                          target_names=['health', 'inter-ictal', 'ictal'],title='Confusion ...
                             Matrix')
        if test_acc>93:
            print(conf_mat)
            print(classification_report(y_true=label_list,y_pred=pre_list,
                target_names=['health', 'inter-ictal', 'ictal']))

        print(
            "Epoch:{}/{} AVG Training Loss:{:.3f} AVG Test Loss:{:.3f} AVG Training ...
                Acc {:.2f} % AVG Test Acc {:.2f} %".format(
                epoch + 1,
                args.epochs,
                train_loss,
                test_loss,
                train_acc,
                test_acc))

        history['train_loss'].append(train_loss)
        history['test_loss'].append(test_loss)
        history['train_acc'].append(train_acc)
        history['test_acc'].append(test_acc)

    foldperf['fold{}'.format(fold + 1)] = history

    torch.save(model, 'k_cross_CNN.pt')

testl_f, tl_f, testa_f, ta_f = [], [], [], []
for f in range(1, k + 1):
    tl_f.append(np.max(foldperf['fold{}'.format(f)]['train_loss']))
```

```
        testl_f.append(np.max(foldperf['fold{}'.format(f)]['test_loss']))

        ta_f.append(np.max(foldperf['fold{}'.format(f)]['train_acc']))
        testa_f.append(np.max(foldperf['fold{}'.format(f)]['test_acc']))

    print('Performance of {} fold cross validation'.format(k))
    print(
        "Average Training Loss: {:.3f} \t Average Test Loss: {:.3f} \t Average Training Acc: ...
            {:.2f} \t Average Test Acc: {:.2f}".format(
            np.mean(tl_f), np.mean(testl_f), np.mean(ta_f), np.mean(testa_f)))



if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--num_classes', type=int, default=3)
    parser.add_argument('--epochs', type=int, default=30) # 十折交叉验证
    parser.add_argument('--batch-size', type=int, default=8)
    parser.add_argument('--lr', type=float, default=0.00001)
    # 预训练权重路径，如果不想载入就设置为空字符
    parser.add_argument('--weights', type=str, default='',
                        help='initial weights path')
    # 是否冻结权重
    parser.add_argument('--freeze-layers', type=bool, default=False)
    parser.add_argument('--device', default='cuda:0', help='device id (i.e. 0 or 0,1 or cpu)')

    opt = parser.parse_args()

    main(opt)
```

my_dataset.py

（输入网络前的预处理将行向量 resize 成 13×13 或者 64×64 的 tensor）

```
from torch.utils.data import Dataset
import numpy as np
import torch
import random


class MyDataSet(Dataset):
    def __init__(self, feature_dir, label_dir):
        self.feature_dir = feature_dir
        self.label_dir = label_dir
```

```
        with open(self.label_dir,encoding='utf-8') as f:
            self.label_data = np.loadtxt(f,delimiter=',')
        self.ids = self.label_data[:,0] # 300*2
        with open(self.feature_dir,encoding='utf-8') as f:
            self.feature_data = np.loadtxt(f,delimiter=',')
        # print(self.feature_data.shape) # 300*4257


    def __len__(self):
        return len(self.ids)

    def __getitem__(self, i):
        #-----------------13*13的输入------------------
        idx = self.ids[i]
        sample_fea = self.feature_data[i,:4258] # (4257,)
        # 将(4257,) -> (64,64)
        # cut strategy: 4097 + 160 -> 3936+160 = 4096 -> 64*64
        cut_slice = random.randint(0,161) # 0-161任意取一个
        fea = np.hstack((sample_fea[cut_slice:cut_slice+9], sample_fea[4097:4257]))
        # fea = np.hstack(sample_fea[0:4096])
        # print(fea.shape) # shape (4096,)
        fea = fea.reshape(13,13) # 变成size = 64*64
        if len(fea.shape) == 2:
            fea = np.expand_dims(fea, axis=0) # 1*64*64

        # -----------如果要改成64*64输入----------------
        # idx = self.ids[i]
        # sample_fea = self.feature_data[i, :4258] # (4257,)
        # # 将(4257,) -> (64,64)
        # # cut strategy: 4097 + 160 -> 3936+160 = 4096 -> 64*64
        # cut_slice = random.randint(0, 161) # 0-161任意取一个
        # fea = np.hstack((sample_fea[cut_slice:cut_slice + 3936], sample_fea[4097:4257]))
        # # fea = np.hstack(sample_fea[0:4096])
        # # print(fea.shape) # shape (4096,)
        # fea = fea.reshape(64, 64) # 变成size = 64*64
        # if len(fea.shape) == 2:
        # fea = np.expand_dims(fea, axis=0) # 1*64*64


        # get label
        # print(self.label_data)
        label = self.label_data[:,1][i] # 前100是0 100-200是1 200-300是2
        new_label = label
        fea = np.array(fea)
        label = np.array(new_label)
```

```
        label = torch.from_numpy(label)
        fea = torch.from_numpy(fea)

        return {'feature':fea, 'label': label, 'id':idx}

    def collate_fn(batch):
        images, labels = tuple(zip(*batch))
        images = torch.stack(images, dim=0)
        labels = torch.as_tensor(labels)
        return images, labels
```

model.py swin transformer 的框架

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.utils.checkpoint as checkpoint
import numpy as np
from typing import Optional

def drop_path_f(x, drop_prob: float = 0., training: bool = False):

    if drop_prob == 0. or not training:
        return x
    keep_prob = 1 - drop_prob
    shape = (x.shape[0],) + (1,) * (x.ndim - 1) # work with diff dim tensors, not just 2D ConvNets
    random_tensor = keep_prob + torch.rand(shape, dtype=x.dtype, device=x.device)
    random_tensor.floor_() # binarize
    output = x.div(keep_prob) * random_tensor
    return output


class DropPath(nn.Module):
    """Drop paths (Stochastic Depth) per sample (when applied in main path of residual blocks).
    """
    def __init__(self, drop_prob=None):
        super(DropPath, self).__init__()
        self.drop_prob = drop_prob

    def forward(self, x):
        return drop_path_f(x, self.drop_prob, self.training)

```

```python

def window_partition(x, window_size: int):
    """
    将feature map按照window_size划分成一个个没有重叠的window
    Args:
        x: (B, H, W, C)
        window_size (int): window size(M)

    Returns:
        windows: (num_windows*B, window_size, window_size, C)
    """
    B, H, W, C = x.shape
    x = x.view(B, H // window_size, window_size, W // window_size, window_size, C)
    # permute: [B, H//Mh, Mh, W//Mw, Mw, C] -> [B, H//Mh, W//Mh, Mw, Mw, C]
    # view: [B, H//Mh, W//Mw, Mh, Mw, C] -> [B*num_windows, Mh, Mw, C]
    windows = x.permute(0, 1, 3, 2, 4, 5).contiguous().view(-1, window_size, window_size, C)
    return windows


def window_reverse(windows, window_size: int, H: int, W: int):
    """
    将一个个window还原成一个feature map
    Args:
        windows: (num_windows*B, window_size, window_size, C)
        window_size (int): Window size(M)
        H (int): Height of image
        W (int): Width of image

    Returns:
        x: (B, H, W, C)
    """
    B = int(windows.shape[0] / (H * W / window_size / window_size))
    # view: [B*num_windows, Mh, Mw, C] -> [B, H//Mh, W//Mw, Mh, Mw, C]
    x = windows.view(B, H // window_size, W // window_size, window_size, window_size, -1)
    # permute: [B, H//Mh, W//Mw, Mh, Mw, C] -> [B, H//Mh, Mh, W//Mw, Mw, C]
    # view: [B, H//Mh, Mh, W//Mw, Mw, C] -> [B, H, W, C]
    x = x.permute(0, 1, 3, 2, 4, 5).contiguous().view(B, H, W, -1)
    return x


class PatchEmbed(nn.Module):
    """
    2D Image to Patch Embedding
    """
    def __init__(self, patch_size=4, in_c=3, embed_dim=96, norm_layer=None):
```

```
        super().__init__()
        patch_size = (patch_size, patch_size)
        self.patch_size = patch_size
        self.in_chans = in_c
        self.embed_dim = embed_dim
        self.proj = nn.Conv2d(in_c, embed_dim, kernel_size=patch_size, stride=patch_size)
        self.norm = norm_layer(embed_dim) if norm_layer else nn.Identity()

    def forward(self, x):
        _, _, H, W = x.shape

        # padding
        # 如果输入图片的H，W不是patch_size的整数倍，需要进行padding
        pad_input = (H % self.patch_size[0] != 0) or (W % self.patch_size[1] != 0)
        if pad_input:
            # to pad the last 3 dimensions,
            # (W_left, W_right, H_top,H_bottom, C_front, C_back)
            x = F.pad(x, (0, self.patch_size[1] - W % self.patch_size[1],
                          0, self.patch_size[0] - H % self.patch_size[0],
                          0, 0))

        # 下采样patch_size倍
        x = self.proj(x)
        _, _, H, W = x.shape
        # flatten: [B, C, H, W] -> [B, C, HW]
        # transpose: [B, C, HW] -> [B, HW, C]
        x = x.flatten(2).transpose(1, 2)
        x = self.norm(x)
        return x, H, W


class PatchMerging(nn.Module):
    r""" Patch Merging Layer.

    Args:
        dim (int): Number of input channels.
        norm_layer (nn.Module, optional): Normalization layer. Default: nn.LayerNorm
    """

    def __init__(self, dim, norm_layer=nn.LayerNorm):
        super().__init__()
        self.dim = dim
        self.reduction = nn.Linear(4 * dim, 2 * dim, bias=False)
        self.norm = norm_layer(4 * dim)

```

```
    def forward(self, x, H, W):
        """
        x: B, H*W, C
        """
        B, L, C = x.shape
        assert L == H * W, "input feature has wrong size"

        x = x.view(B, H, W, C)

        # padding
        # 如果输入feature map的H，W不是2的整数倍，需要进行padding
        pad_input = (H % 2 == 1) or (W % 2 == 1)
        if pad_input:
            # to pad the last 3 dimensions, starting from the last dimension and moving forward.
            # (C_front, C_back, W_left, W_right, H_top, H_bottom)
            # 注意这里的Tensor通道是[B, H, W, C]，所以会和官方文档有些不同
            x = F.pad(x, (0, 0, 0, W % 2, 0, H % 2))

        x0 = x[:, 0::2, 0::2, :] # [B, H/2, W/2, C]
        x1 = x[:, 1::2, 0::2, :] # [B, H/2, W/2, C]
        x2 = x[:, 0::2, 1::2, :] # [B, H/2, W/2, C]
        x3 = x[:, 1::2, 1::2, :] # [B, H/2, W/2, C]
        x = torch.cat([x0, x1, x2, x3], -1) # [B, H/2, W/2, 4*C]
        x = x.view(B, -1, 4 * C) # [B, H/2*W/2, 4*C]

        x = self.norm(x)
        x = self.reduction(x) # [B, H/2*W/2, 2*C]

        return x


class Mlp(nn.Module):
    """ MLP as used in Vision Transformer, MLP-Mixer and related networks
    """
    def __init__(self, in_features, hidden_features=None, out_features=None, ...
         act_layer=nn.GELU, drop=0.):
        super().__init__()
        out_features = out_features or in_features
        hidden_features = hidden_features or in_features

        self.fc1 = nn.Linear(in_features, hidden_features)
        self.act = act_layer()
        self.drop1 = nn.Dropout(drop)
        self.fc2 = nn.Linear(hidden_features, out_features)
        self.drop2 = nn.Dropout(drop)
```

```
    def forward(self, x):
        x = self.fc1(x)
        x = self.act(x)
        x = self.drop1(x)
        x = self.fc2(x)
        x = self.drop2(x)
        return x


class WindowAttention(nn.Module):


    def __init__(self, dim, window_size, num_heads, qkv_bias=True, attn_drop=0., ...
            proj_drop=0.):

        super().__init__()
        self.dim = dim
        self.window_size = window_size # [Mh, Mw]
        self.num_heads = num_heads
        head_dim = dim // num_heads
        self.scale = head_dim ** -0.5

        # define a parameter table of relative position bias
        self.relative_position_bias_table = nn.Parameter(
            torch.zeros((2 * window_size[0] - 1) * (2 * window_size[1] - 1), num_heads)) # ...
                [2*Mh-1 * 2*Mw-1, nH]

        # get pair-wise relative position index for each token inside the window
        coords_h = torch.arange(self.window_size[0])
        coords_w = torch.arange(self.window_size[1])
        coords = torch.stack(torch.meshgrid([coords_h, coords_w])) # [2, Mh, Mw]
        coords_flatten = torch.flatten(coords, 1) # [2, Mh*Mw]
        # [2, Mh*Mw, 1] - [2, 1, Mh*Mw]
        relative_coords = coords_flatten[:, :, None] - coords_flatten[:, None, :] # [2, Mh*Mw, ...
            Mh*Mw]
        relative_coords = relative_coords.permute(1, 2, 0).contiguous() # [Mh*Mw, Mh*Mw, 2]
        relative_coords[:, :, 0] += self.window_size[0] - 1 # shift to start from 0
        relative_coords[:, :, 1] += self.window_size[1] - 1
        relative_coords[:, :, 0] *= 2 * self.window_size[1] - 1
        relative_position_index = relative_coords.sum(-1) # [Mh*Mw, Mh*Mw]
        self.register_buffer("relative_position_index", relative_position_index)

        self.qkv = nn.Linear(dim, dim * 3, bias=qkv_bias)
        self.attn_drop = nn.Dropout(attn_drop)
```

```
        self.proj = nn.Linear(dim, dim)
        self.proj_drop = nn.Dropout(proj_drop)

        nn.init.trunc_normal_(self.relative_position_bias_table, std=.02)
        self.softmax = nn.Softmax(dim=-1)

    def forward(self, x, mask: Optional[torch.Tensor] = None):
        """
        Args:
            x: input features with shape of (num_windows*B, Mh*Mw, C)
            mask: (0/-inf) mask with shape of (num_windows, Wh*Ww, Wh*Ww) or None
        """
        # [batch_size*num_windows, Mh*Mw, total_embed_dim]
        B_, N, C = x.shape
        # qkv(): -> [batch_size*num_windows, Mh*Mw, 3 * total_embed_dim]
        # reshape: -> [batch_size*num_windows, Mh*Mw, 3, num_heads, ...
            embed_dim_per_head]
        # permute: -> [3, batch_size*num_windows, num_heads, Mh*Mw, ...
            embed_dim_per_head]
        qkv = self.qkv(x).reshape(B_, N, 3, self.num_heads, C // ...
            self.num_heads).permute(2, 0, 3, 1, 4)
        # [batch_size*num_windows, num_heads, Mh*Mw, embed_dim_per_head]
        q, k, v = qkv.unbind(0) # make torchscript happy (cannot use tensor as tuple)
        q = q * self.scale
        attn = (q @ k.transpose(-2, -1))

        # relative_position_bias_table.view: [Mh*Mw*Mh*Mw,nH] -> [Mh*Mw,Mh*Mw,nH]
        relative_position_bias = ...
            self.relative_position_bias_table[self.relative_position_index.view(-1)].view(
            self.window_size[0] * self.window_size[1], self.window_size[0] * ...
                self.window_size[1], -1)
        relative_position_bias = relative_position_bias.permute(2, 0, 1).contiguous() # [nH, ...
            Mh*Mw, Mh*Mw]
        attn = attn + relative_position_bias.unsqueeze(0)

        if mask is not None:
            # mask: [nW, Mh*Mw, Mh*Mw]
            nW = mask.shape[0] # num_windows
            # attn.view: [batch_size, num_windows, num_heads, Mh*Mw, Mh*Mw]
            # mask.unsqueeze: [1, nW, 1, Mh*Mw, Mh*Mw]
            attn = attn.view(B_ // nW, nW, self.num_heads, N, N) + ...
                mask.unsqueeze(1).unsqueeze(0)
            attn = attn.view(-1, self.num_heads, N, N)
            attn = self.softmax(attn)
        else:
```

```
            attn = self.softmax(attn)

        attn = self.attn_drop(attn)

        # @: multiply -> [batch_size*num_windows, num_heads, Mh*Mw, ...
            embed_dim_per_head]
        # transpose: -> [batch_size*num_windows, Mh*Mw, num_heads, ...
            embed_dim_per_head]
        # reshape: -> [batch_size*num_windows, Mh*Mw, total_embed_dim]
        x = (attn @ v).transpose(1, 2).reshape(B_, N, C)
        x = self.proj(x)
        x = self.proj_drop(x)
        return x


class SwinTransformerBlock(nn.Module):
    def __init__(self, dim, num_heads, window_size=7, shift_size=0,
                 mlp_ratio=4., qkv_bias=True, drop=0., attn_drop=0., drop_path=0.,
                 act_layer=nn.GELU, norm_layer=nn.LayerNorm):
        super().__init__()
        self.dim = dim
        self.num_heads = num_heads
        self.window_size = window_size
        self.shift_size = shift_size
        self.mlp_ratio = mlp_ratio
        assert 0 <=self.shift_size < self.window_size, "shift_size must in 0-window_size"

        self.norm1 = norm_layer(dim)
        self.attn = WindowAttention(
            dim, window_size=(self.window_size, self.window_size), ...
                num_heads=num_heads, qkv_bias=qkv_bias,
            attn_drop=attn_drop, proj_drop=drop)

        self.drop_path = DropPath(drop_path) if drop_path > 0. else nn.Identity()
        self.norm2 = norm_layer(dim)
        mlp_hidden_dim = int(dim * mlp_ratio)
        self.mlp = Mlp(in_features=dim, hidden_features=mlp_hidden_dim, ...
            act_layer=act_layer, drop=drop)

    def forward(self, x, attn_mask):
        H, W = self.H, self.W
        B, L, C = x.shape
        assert L == H * W, "input feature has wrong size"

        shortcut = x
```

```
        x = self.norm1(x)
        x = x.view(B, H, W, C)

        # pad feature maps to multiples of window size
        # 把feature map给pad到window size的整数倍
        pad_l = pad_t = 0
        pad_r = (self.window_size - W % self.window_size) % self.window_size
        pad_b = (self.window_size - H % self.window_size) % self.window_size
        x = F.pad(x, (0, 0, pad_l, pad_r, pad_t, pad_b))
        _, Hp, Wp, _ = x.shape

        # cyclic shift
        if self.shift_size > 0:
            shifted_x = torch.roll(x, shifts=(-self.shift_size, -self.shift_size), dims=(1, 2))
        else:
            shifted_x = x
            attn_mask = None

        # partition windows
        x_windows = window_partition(shifted_x, self.window_size) # [nW*B, Mh, Mw, C]
        x_windows = x_windows.view(-1, self.window_size * self.window_size, C) # [nW*B, Mh*Mw, C]

        # W-MSA/SW-MSA
        attn_windows = self.attn(x_windows, mask=attn_mask) # [nW*B, Mh*Mw, C]

        # merge windows
        attn_windows = attn_windows.view(-1, self.window_size, self.window_size, C) # [nW*B, Mh, Mw, C]
        shifted_x = window_reverse(attn_windows, self.window_size, Hp, Wp) # [B, H', W', C]

        # reverse cyclic shift
        if self.shift_size > 0:
            x = torch.roll(shifted_x, shifts=(self.shift_size, self.shift_size), dims=(1, 2))
        else:
            x = shifted_x

        if pad_r > 0 or pad_b > 0:
            # 把前面pad的数据移除掉
            x = x[:, :H, :W, :].contiguous()

        x = x.view(B, H * W, C)

        # FFN
        x = shortcut + self.drop_path(x)
```

```
        x = x + self.drop_path(self.mlp(self.norm2(x)))

        return x


class BasicLayer(nn.Module):

    def __init__(self, dim, depth, num_heads, window_size,
                 mlp_ratio=4., qkv_bias=True, drop=0., attn_drop=0.,
                 drop_path=0., norm_layer=nn.LayerNorm, downsample=None, use_checkpoint=False):
        super().__init__()
        self.dim = dim
        self.depth = depth
        self.window_size = window_size
        self.use_checkpoint = use_checkpoint
        self.shift_size = window_size // 2

        # build blocks
        self.blocks = nn.ModuleList([
            SwinTransformerBlock(
                dim=dim,
                num_heads=num_heads,
                window_size=window_size,
                shift_size=0 if (i % 2 == 0) else self.shift_size,
                mlp_ratio=mlp_ratio,
                qkv_bias=qkv_bias,
                drop=drop,
                attn_drop=attn_drop,
                drop_path=drop_path[i] if isinstance(drop_path, list) else drop_path,
                norm_layer=norm_layer)
            for i in range(depth)])

        # patch merging layer
        if downsample is not None:
            self.downsample = downsample(dim=dim, norm_layer=norm_layer)
        else:
            self.downsample = None

    def create_mask(self, x, H, W):
        # calculate attention mask for SW-MSA
        # 保证Hp和Wp是window_size的整数倍
        Hp = int(np.ceil(H / self.window_size)) * self.window_size
        Wp = int(np.ceil(W / self.window_size)) * self.window_size
        # 拥有和feature map一样的通道排列顺序，方便后续window_partition
```

```
        img_mask = torch.zeros((1, Hp, Wp, 1), device=x.device) # [1, Hp, Wp, 1]
        h_slices = (slice(0, -self.window_size),
                    slice(-self.window_size, -self.shift_size),
                    slice(-self.shift_size, None))
        w_slices = (slice(0, -self.window_size),
                    slice(-self.window_size, -self.shift_size),
                    slice(-self.shift_size, None))
        cnt = 0
        for h in h_slices:
            for w in w_slices:
                img_mask[:, h, w, :] = cnt
                cnt += 1

        mask_windows = window_partition(img_mask, self.window_size) # [nW, Mh, Mw, 1]
        mask_windows = mask_windows.view(-1, self.window_size * self.window_size) # [nW, Mh*Mw]
        attn_mask = mask_windows.unsqueeze(1) - mask_windows.unsqueeze(2) # [nW, 1, Mh*Mw] - [nW, Mh*Mw, 1]
        # [nW, Mh*Mw, Mh*Mw]
        attn_mask = attn_mask.masked_fill(attn_mask != 0, float(-100.0)).masked_fill(attn_mask == 0, float(0.0))
        return attn_mask

    def forward(self, x, H, W):
        attn_mask = self.create_mask(x, H, W) # [nW, Mh*Mw, Mh*Mw]
        for blk in self.blocks:
            blk.H, blk.W = H, W
            if not torch.jit.is_scripting() and self.use_checkpoint:
                x = checkpoint.checkpoint(blk, x, attn_mask)
            else:
                x = blk(x, attn_mask)
        if self.downsample is not None:
            x = self.downsample(x, H, W)
            H, W = (H + 1) // 2, (W + 1) // 2

        return x, H, W


class SwinTransformer(nn.Module):
    def __init__(self, patch_size=4, in_chans=3, num_classes=1000,
                 embed_dim=96, depths=(2, 2, 6, 2), num_heads=(3, 6, 12, 24),
                 window_size=7, mlp_ratio=4., qkv_bias=True,
                 drop_rate=0., attn_drop_rate=0., drop_path_rate=0.1,
                 norm_layer=nn.LayerNorm, patch_norm=True,
                 use_checkpoint=False, **kwargs):
```

```
        super().__init__()

        self.num_classes = num_classes
        self.num_layers = len(depths)
        self.embed_dim = embed_dim
        self.patch_norm = patch_norm
        # stage4输出特征矩阵的channels
        self.num_features = int(embed_dim * 2 ** (self.num_layers - 1))
        self.mlp_ratio = mlp_ratio

        # split image into non-overlapping patches
        self.patch_embed = PatchEmbed(
            patch_size=patch_size, in_c=in_chans, embed_dim=embed_dim,
            norm_layer=norm_layer if self.patch_norm else None)
        self.pos_drop = nn.Dropout(p=drop_rate)

        # stochastic depth
        dpr = [x.item() for x in torch.linspace(0, drop_path_rate, sum(depths))]

        # build layers
        self.layers = nn.ModuleList()
        for i_layer in range(self.num_layers):
            # 注意这里构建的stage和论文图中有些差异
            # 这里的stage不包含该stage的patch_merging层，包含的是下个stage的
            layers = BasicLayer(dim=int(embed_dim * 2 ** i_layer),
                                depth=depths[i_layer],
                                num_heads=num_heads[i_layer],
                                window_size=window_size,
                                mlp_ratio=self.mlp_ratio,
                                qkv_bias=qkv_bias,
                                drop=drop_rate,
                                attn_drop=attn_drop_rate,
                                drop_path=dpr[sum(depths[:i_layer]):sum(depths[:i_layer + 1])],
                                norm_layer=norm_layer,
                                downsample=PatchMerging if (i_layer < self.num_layers - 1) ...
                                    else None,
                                use_checkpoint=use_checkpoint)
            self.layers.append(layers)

        self.norm = norm_layer(self.num_features)
        self.avgpool = nn.AdaptiveAvgPool1d(1)
        self.head = nn.Linear(self.num_features, num_classes) if num_classes > 0 else ...
            nn.Identity()

        self.apply(self._init_weights)
```

```python

    def _init_weights(self, m):
        if isinstance(m, nn.Linear):
            nn.init.trunc_normal_(m.weight, std=.02)
            if isinstance(m, nn.Linear) and m.bias is not None:
                nn.init.constant_(m.bias, 0)
        elif isinstance(m, nn.LayerNorm):
            nn.init.constant_(m.bias, 0)
            nn.init.constant_(m.weight, 1.0)

    def forward(self, x):
        # x: [B, L, C]
        x, H, W = self.patch_embed(x)
        x = self.pos_drop(x)

        for layer in self.layers:
            x, H, W = layer(x, H, W)

        x = self.norm(x) # [B, L, C]
        x = self.avgpool(x.transpose(1, 2)) # [B, C, 1]
        x = torch.flatten(x, 1)
        x = self.head(x)
        return x


def swin_tiny_patch4_window7_224(num_classes: int = 1000, **kwargs):
    # trained ImageNet-1K
    model = SwinTransformer(in_chans=1,
                            patch_size=4,
                            window_size=7,
                            embed_dim=96,
                            depths=(2, 2, 6, 2),
                            num_heads=(3, 6, 12, 24),
                            num_classes=num_classes,
                            **kwargs)
    return model
```

参考文献

[1] R. G. Andrzejak, K. Lehnertz, F. Mormann, C. Rieke, P. David, and C. E. Elger, “Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state,” *Physical Review E*, vol. 64, no. 6, p. 061907, 2001.

[2] K. Chua, V. Chandran, U. R. Acharya, and C. Lim, “Automatic identification of epileptic electroencephalography signals using higher-order spectra,” *Proceedings of the Institution of Mechanical Engineers, Part H: Journal of Engineering in Medicine*, vol. 223, no. 4, pp. 485–495, 2009.

[3] O. Faust, U. R. Acharya, L. C. Min, and B. H. Sputh, “Automatic identification of epileptic and background eeg signals using frequency domain parameters,” *International journal of neural systems*, vol. 20, no. 02, pp. 159–176, 2010.

[4] U. R. Acharya, S. V. Sree, and J. S. Suri, “Automatic detection of epileptic eeg signals using higher order cumulant features,” *International journal of neural systems*, vol. 21, no. 05, pp. 403–414, 2011.

[5] A. Bhattacharyya and R. B. Pachori, “A multivariate approach for patient-specific eeg seizure detection using empirical wavelet transform,” *IEEE Transactions on Biomedical Engineering*, vol. 64, no. 9, pp. 2003–2015, 2017.

[6] U. R. Acharya, S. L. Oh, Y. Hagiwara, J. H. Tan, and H. Adeli, “Deep convolutional neural network for the automated detection and diagnosis of seizure using eeg signals,” *Computers in biology and medicine*, vol. 100, pp. 270–278, 2018.