**程序说明**

目录

[**排版说明** 2](#_Toc119705749)

[**a)** **读取文件、提取训练集** 3](#_Toc119705750)

[**i.** **data\_information：存储训练集信息** 3](#_Toc119705751)

[**ii.** **MyFunction.get\_event\_json：读取json文件数据** 4](#_Toc119705752)

[**iii.** **MyFunction.get\_row：按行读取openpyxl.load\_workbook** 4](#_Toc119705753)

[**iv.** **save\_dataset\_to\_csv.py：将所需训练集转存为csv** 5](#_Toc119705754)

[**1.** **get\_classify\_dataset：获取用于训练分类模型所需的数据** 5](#_Toc119705755)

[**2.** **get\_no\_event：获取用于训练定位模型所需的部分数据** 7](#_Toc119705756)

[**b)** **数据处理** 10](#_Toc119705757)

[**i.** **MyFunction.** **normalization：归一化处理** 10](#_Toc119705758)

[**ii.** **MyFunction.** **myfft：对单列数据进行快速傅里叶变换** 10](#_Toc119705759)

[**iii.** **data\_information.read\_dataset：读取train\_list/中的数据** 10](#_Toc119705760)

[**iv.** **MyFunction.get\_split\_data：对待分类数据块进行分割处理** 11](#_Toc119705761)

[**c)** **训练定位模型** 13](#_Toc119705762)

[**i.** **MyFunction.get\_YN：导入并处理数据** 13](#_Toc119705763)

[**ii.** **MyModel.location\_model：设计一维卷积神经网络** 14](#_Toc119705764)

[**iii.** **build\_location\_model.py：训练神经网络** 16](#_Toc119705765)

[**d)** **训练分类模型** 19](#_Toc119705766)

[**i.** **MyFunction.get\_XY：导入并处理数据** 19](#_Toc119705767)

[**ii.** **MyFunction.train\_and\_test\_data：分割数据集** 20](#_Toc119705768)

[**iii.** **MyModel.classify\_model：设计一维卷积神经网络** 21](#_Toc119705769)

[**iv.** **build\_classify\_model.py：训练模型** 23](#_Toc119705770)

[**e)** **组装模型** 25](#_Toc119705771)

[**i.** **MyModel.MyModel类:导入以生成模型** 26](#_Toc119705772)

[**ii.** **test\_classify\_model.py：测试分类模型** 27](#_Toc119705773)

[**iii.** **location\_and\_classify\_model：定位-分类模型** 29](#_Toc119705774)

[**f)** **test.py：测试** 32](#_Toc119705775)

**排版说明**

**等宽字体**用于代码清单以及一般的正文中表示代码或之前定义的对象。

代码清单中的**斜体**表示占位符。应该用具体问题中的文本和值来替换它们。

其余部分使用**微软雅黑**字体。

部分代码为了文档排版美观，可能**相比源代码**有一点的**排版差异**，且存在**删除**一些**注释**行避免**无用信息**的可能性，**实际功能并无差异**。

文中**代码缩进**并非无意义，为**源代码拆分形态**，复现时请**注意缩进。**

* 1. **读取文件、提取训练集**
     1. **data\_information：存储训练集信息**

由于题目所给出的**"标注.json"**中的事件对应的文件路径为**Windows**下文件路径格式，如"攀爬[\\220907\_152136.xlsx](file:///\\\\220907_152136.xlsx)"，因此需要将之转为**Linux**下的文件路径，即"攀爬[/220907\_152136.xlsx](file:///\\\\220907_152136.xlsx)"，且同时需要储存json文件中对应的事件标签以及以及事件所在位置。

为了减少读取json文件的次数，故设计一个**data\_information类**：

class data\_information:

    def \_\_init\_\_(self, information) -> None:

        file\_name = information["file\_name"]

        self.data\_path = (

"/home/T02053124/数据/original\_dataset/"

            + file\_name.split("\\")[0]+ "/"

            + file\_name.split("\\")[1])

        self.event\_local = information["boxes"][0]

        self.data\_label = information["labels"]

        self.data\_name = information["file\_name"].split("\\")[1].split(".")[0]

        self.dataset = np.zeros(0)

其获取一个information的字典变量，如：

{

"file\_name":"攀爬[\\220907\_152136.xlsx](file:///\\\\220907_152136.xlsx)",

"labels":[0],

"boxes":[[35,63]]

}

随后，利用split函数对file\_name对应值进行分割组合，最终得到Linux下的对应文件位**"T02053124/数据/original\_dataset/*\*\*(事件名)*/*\*\*.xlsx*"**（题目所给的xlsx文件被储存在**"/home/T02053124/数据/original\_dataset/"**中）。

data\_label存储**事件标签**，event\_local存储**事件对应位置信息**，dataset则用于**存储**后面读出所得的数据，data\_name用于将事件对应的数据集存储为**csv**文件时命名以及读取时使用

* + 1. **MyFunction.get\_event\_json：读取json文件数据**

import json

def get\_event\_json(file\_path):

    with open(file\_path, "r", encoding="utf-8") as json\_file:

        event = json.load(json\_file)

读取file\_path="标注.json"，将**事件分类label**命名为变量event\_label,将**事件信息data**存储为all\_event\_information:

event\_label = event["label"][0]

    all\_event\_information = event["data"]

最终返回event\_label，all\_event\_information

return all\_event\_information, event\_label

* + 1. **MyFunction.get\_row：按行读取****openpyxl.load\_workbook**

def get\_row(sht, min\_col: int, max\_col: int, begin\_row: int =1, rows\_end: int = 32):

    row\_datas = []

    for datas in sht.iter\_rows(

        min\_row=begin\_row, max\_row=rows\_end,

        min\_col=min\_col, max\_col=max\_col,

    ):

        temp = []

        for i in range(len(datas)):

            value = datas[i].value

            temp.append(value)

        row\_datas.append(temp)

    return row\_datas

这个函数获取xlsx中的一个表，并通过设定**行与列范围**，默认行的读取范围为1-32，返回一个列表，每个元素为读取范围的一行元素。例如：

data\_file = load\_workbook(dataset\_inf.data\_path, read\_only=True)

sht = data\_file[data\_file.sheetnames[0]]

datalist = get\_row(sht, 0, 500, rows\_end=500)

即读取一个**.xlsx文件**的**第一个表**中的1-500行，0-500列。

* + 1. **save\_dataset\_to\_csv.py：将所需训练集转存为csv**

由于训练神经网络需要的数据量极大，而每次读取.xlsx文件所需的时间长，因此读取到对应数据后，将之转存为csv文件，以便后面调试数据。

首先，导入读取文件时所需的的库和函数，并读取json：

import numpy as np

from openpyxl import load\_workbook

from MyFunction import get\_event\_json, get\_row

from multiprocessing import Pool,cpu\_count

from datetime import datetime

from train\_data\_information import data\_information

**get\_classify\_dataset**与**get\_no\_event**都接受一个名为**data\_informations**的data\_information**列表**

* + - 1. **get\_classify\_dataset：获取用于训练分类模型所需的数据**

首先，我们需要一个函数去通过data\_information类中所包含信息来获取对应.xlsx文件中的数据，并存为np.array对象。因此设计一个函数：

def train\_data(dataset\_inf: data\_information):

    data\_file = load\_workbook(dataset\_inf.data\_path,

                              read\_only=True)

    sht = data\_file[data\_file.sheetnames[0]]

    n = sht.max\_row

*# 读取*

    datalist = get\_row(

        sht, dataset\_inf.event\_local[0],

        dataset\_inf.event\_local[1], rows\_end=n

    )

    dataset = np.array(datalist)

    return dataset

这个函数接受一个data\_information对象，通过其内的信息，依靠**openpyxl.load\_workbook**访问对应.xlsx，并利用**MyFunction.get\_row**将对应列范围内所有行的数据，将之转换成名为dataset的np.array对象后返回。

而后，设计一个函数get\_classify\_dataset来接受data\_information列表，批量获取数据：

def get\_classify\_dataset(data\_informations):

    pool = Pool(cpu\_count())

    print("获取 train\_list 中")

    beg = datetime.now()

首先，应用multiprocessing中的cpu\_count获取计算机的cpu核心数，并将结果导入进程池Pool，设置最大进程数，以期利用并行计算，同时读取多个文件，减少读取时间，并记录程序开始时间。

    results = [

        pool.apply\_async(train\_data, [dataset\_inf])

        for dataset\_inf in data\_informations

    ]

    pool.close()

    pool.join()

    print("读取完毕")

    print("共消耗: " + "{:.2f}".format((datetime.now() - beg).total\_seconds()) + " 秒")

pool.apply\_async会接受函数名和这个函数所需的变量列表，我利用循环，遍历data\_information列表，每一个元素调用一次将之分别作为一个任务加入线程池，设置 pool.close()告诉池不要接受任何新作业，pool.join()告诉池等待所有作业完成然后退出，并阻塞主进程，保证计时的准确性。

读取完毕后，将其按照对应文件的文件名储存为csv文件：

for i in range(len(data\_informations)):

        path\_name = (

            "/home/T02053124/数据/train\_list/"

            + data\_informations[i].data\_name + "\_"

            + str(data\_informations[i].data\_label[0])

            + ".csv"

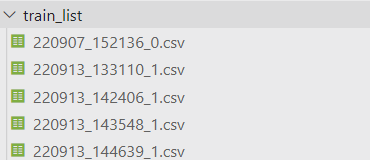
        )

        np.savetxt(path\_name, results[i].get(), fmt="%i", delimiter=",")

    print("train\_list 保存完毕")

具体路径为"/home/T02053124/数据/train\_list/"，利用data\_information中的data\_name和data\_label组合命名。

部分文件目录如下：



* + - 1. **get\_no\_event：获取用于训练定位模型所需的部分数据**

定位问题可以看作二分类问题，因此若把**get\_classify\_dataset**所得的数据看出"1"事件，则需要有"0"事件与之进行对比，故再设置一个get\_no\_event来读取非事件，考虑到反正后续模型应用时读取文件时耗时太长的问题，以及经过测试，选取五百行同时能保证一定的速度和准确率。同样先设置一个函数以接受data\_information对象：

def no\_event(dataset\_inf: data\_information):

    data\_file = load\_workbook(dataset\_inf.data\_path,

                              read\_only=True)

    sht = data\_file[data\_file.sheetnames[0]]

*# 读取*

    datalist = get\_row(sht, 0,

                       dataset\_inf.event\_local[1] + 500,

                       rows\_end=500)

    dataset = np.array(datalist)

    no\_event = np.append(

        dataset[:, 0 : dataset\_inf.event\_local[0]],

        dataset[:, dataset\_inf.event\_local[1] + 1 :],

        axis=1,

    )

    return no\_event

与train\_data函数类似，但它选取范围为第0列到除了事件列以外事件结束后500列（防止提取过多无用数据，以此减少csv文件所占的内存），每列提取500行，将提取的数据转换成np.array返回。get\_no\_event则基本与get\_classify\_dataset相同：

def get\_no\_event(dataset\_informations):

*# 获取非事件列*

    pool = Pool(cpu\_count())

    print("获取 no\_event 中")

    beg = datetime.now()

    results = [

        pool.apply\_async(no\_event, [dataset\_inf])

        for dataset\_inf in dataset\_informations

    ]

    pool.close()

    pool.join()

    print("读取完毕")

    print("共消耗: " + "{:.2f}".format((datetime.now() - beg).total\_seconds()) + " 秒")

    n\_temp = np.zeros(500)

    n\_temp = n\_temp.reshape(500, 1)

    for i in range(len(results)):

        temp = results[i].get()

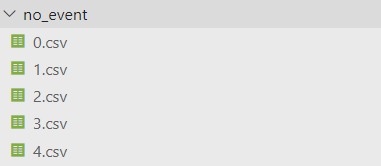
        path = "/home/T02053124/数据/no\_event/" + str(i) + ".csv"

        np.savetxt(path, temp, fmt="%i", delimiter=",")

    print("no\_event 保存完毕")

只是相较于前者，文件名简单按循环编号记录。

部分文件文件目录如下：



至此，储存训练集的前置工作基本已经完成，只需要调用上述的get\_no\_event和get\_classify\_dataset，即可将训练集从原本的.xlsx中提取成csv，以被后续使用，运行代码如下：

if \_\_name\_\_ == "\_\_main\_\_":

    json\_path = "标注.json"

    all\_event\_information, event\_label = get\_event\_json(json\_path)

    dataset\_informations = [

        data\_information(information)

        for information in all\_event\_information

    ]

    print("当前CPU核心数为{}".format(cpu\_count()))

    print("初始化线程池中")

    get\_no\_event(dataset\_informations)

    get\_classify\_dataset(dataset\_informations)

* 1. **数据处理**
     1. **MyFunction.** **normalization：归一化处理**

def normalization(data):  *# 归一化处理*

    \_range = np.max(data) - np.min(data)

    return (data - np.min(data)) / \_range

* + 1. **MyFunction.** **myfft：对单列数据进行快速傅里叶变换**

def myfft(data):  *# 快速傅里叶变换*

    fft\_x = fft(data)

    N = np.size(data)

*# 取复数的绝对值，即复数的模(双边频谱)*

    abs\_y = np.abs(fft\_x)

*# 由于对称性，只取一半区间（单边频谱）*

    normalization\_half\_y = abs\_y[range(int(N / 2))]

    return normalization\_half\_y

* + 1. **data\_information.read\_dataset：读取train\_list/中的数据**

一个data\_information类中的方法：

    def read\_dataset(self):

        path\_name = (

            "/home/T02053124/数据/train\_list/"

            + self.data\_name

            + "\_"

            + str(self.data\_label[0])

            + ".csv"

        )

        temp = np.loadtxt(path\_name, delimiter=",")

        return temp

通过之前**get\_classify\_dataset**储存数据集时的**命名法则**，使用np.loadtxt读取相应的csv文件，并返回一个np.array对象。

* + 1. **MyFunction.get\_split\_data：对待分类数据块进行分割处理**

接受一个数据集original\_dataset，根据step\_number步进：

def get\_split\_data(original\_dataset, step\_number: int = 16):

*# 对每一个数据块元素进行分割处理*

    datasets = []

    max\_row = np.size(original\_dataset, axis=0)

    for i in range(0, max\_row, step\_number):

        if i + 32 > max\_row:

            break

        datasets.append(original\_dataset[i : i + 32])

从i第0行开始，每次步进step\_number，且选取第i行到第i+31行作为一个形状为32\* np.size(original\_dataset,axis=1)的np.array数据块，将其加入列表datasets中。

设置一个results的空列表，以存储datasets中经过处理的数据：

    results = []

开始遍历datasets,对每一列进行一次快速傅里叶变换后，数据块行数减少将一半，变为16行：

    for i in range(len(datasets)):

        X = datasets[i]

        temp = np.zeros(16 \* 22)

        temp = temp.reshape(16, 22)

        X\_fft = []

        for j in range(np.size(X, axis=1)):

            X\_fft.append(myfft(X[:, j]))

        X\_fft = np.array(X\_fft).T

由于每个事件发生时的列数不同，因此需要纵向进行一次切割，且有部分事件的列数太少，因此经过多次测试后，选择X\_fft的第0-21列，总共22列元素作为最终的数据。**为了防止列数过少出现错误，因此若是所需列数超过已有列数，再超出部分由从开头再此选取一部分拼接在后**：

*# 选择0到21列*

        loc = 0

        limit = [0, 21]

        temp[:, limit[0]] = X\_fft[:, loc]

*# 为了防止列数过少出现错误*

*# 因此若是所需列数超过已有列数*

*# 超出部分由从开头再次选取一部分拼接在后*

        l = 0

        if loc + limit[1] > np.size(X\_fft, axis=1) - 1:

            l = (loc + limit[1]) - np.size(X\_fft, axis=1) + 1

            temp[:, -limit[1] :] = np.append(X\_fft[:,loc+1:], X\_fft[:, :l],axis=1)

        else:

            temp[:, -limit[1] :] = X\_fft[:, loc + 1 : loc + limit[1] + 1]

*# 将最终数据块转换成352\*1的矩阵插入results*

        results.append(temp.reshape(16 \* 22, 1))

返回results，用于分类模型

    return results

* 1. **训练定位模型**
     1. **MyFunction.get\_YN：导入并处理数据**

首先仍需要通过**"标注.json"**中的事件信息，来读取**/train\_list**中的文件以获取**"1"事件**：

def get\_YN():

    path = "标注.json"

    all\_event\_information, event\_label = get\_event\_json(path)

    data\_informations = [

        data\_information(information)

        for information in all\_event\_information

    ]

开启进程池，一次同时读取多个文件，加快数据导入速度：

*# 获取"1"事件*

    yes\_event = []

    pool = Pool(cpu\_count())

    results = [

        pool.apply\_async(data\_inf.read\_dataset)

        for data\_inf in data\_informations

    ]

    pool.close()

    pool.join()

对得到的数据每列中的连续500行分割一次，并对每个数据块进行快速傅里叶变换，最终地250\*1的np.array数据块，加入yes\_event中：

    for result in results:

        y\_temp = result.get()

        max\_row = np.size(y\_temp, axis=0)

        for i in range(0, np.size(y\_temp, axis=1)):

            for j in range(0, max\_row, 500):

                if j + 500 > max\_row:

                    break

                yes\_event.append(

                    np.array(myfft(y\_temp[j : j + 500, i]))

                    .reshape(250, 1)

                )

对于**"0"事件**，则直接读取由**get\_no\_event**所导出的csv文件即可，同样对每列进行快速傅里叶变换，得250\*1的np.array数据块，将之存入no\_event中：

*# 获取"0"事件*

    no\_event = []

    for i in range(28):

        path = "/home/T02053124/数据/no\_event/" + str(i) + ".csv"

        temp = np.loadtxt(path, delimiter=",")

        for i in range(np.size(temp, axis=1)):

            no\_event.append(np.array(myfft(temp[:500, i]))

                            .reshape(250, 1))

将yes\_event和no\_event转换为np.array，使用np.append拼接两个事件集得X，并生成对应元素的便签，即**"1"和"0"**，拼接得Y，最终返回X，Y用于训练定位模型：

*# 设置数据集和对应标签*

    yes\_event = np.array(yes\_event)

    no\_event = np.array(no\_event)

    X = np.append(yes\_event, no\_event, axis=0)

    Y = np.append(np.ones(yes\_event.shape[0]),

                  np.zeros(no\_event.shape[0]))

return X, Y

* + 1. **MyModel.location\_model：设计一维卷积神经网络**

使用**5层Conv1D**来提取特征值，**每2层**Conv1D后添加一层**MaxPooling1D**来保留主要特征，减少计算量。

每层卷积层使用**线性整流函数relu**（Linear rectification function），简化计算过程，加快训练速度

def location\_model():

    clear\_session()

*# 输入层*

    inputs = Input(shape=(250, 1))

*# 卷积层起始*

    conv = Conv1D(16, 3, activation="relu")(inputs)

    conv = Conv1D(16, 3, activation="relu")(conv)

    pool = MaxPooling1D(4)(conv)

    conv = Conv1D(64, 3, activation="relu")(pool)

    conv = Conv1D(64, 3, activation="relu")(conv)

    pool = MaxPooling1D(4)(conv)

*# 卷积层结束*

    flatten = Flatten()(pool)

*# 输出层*

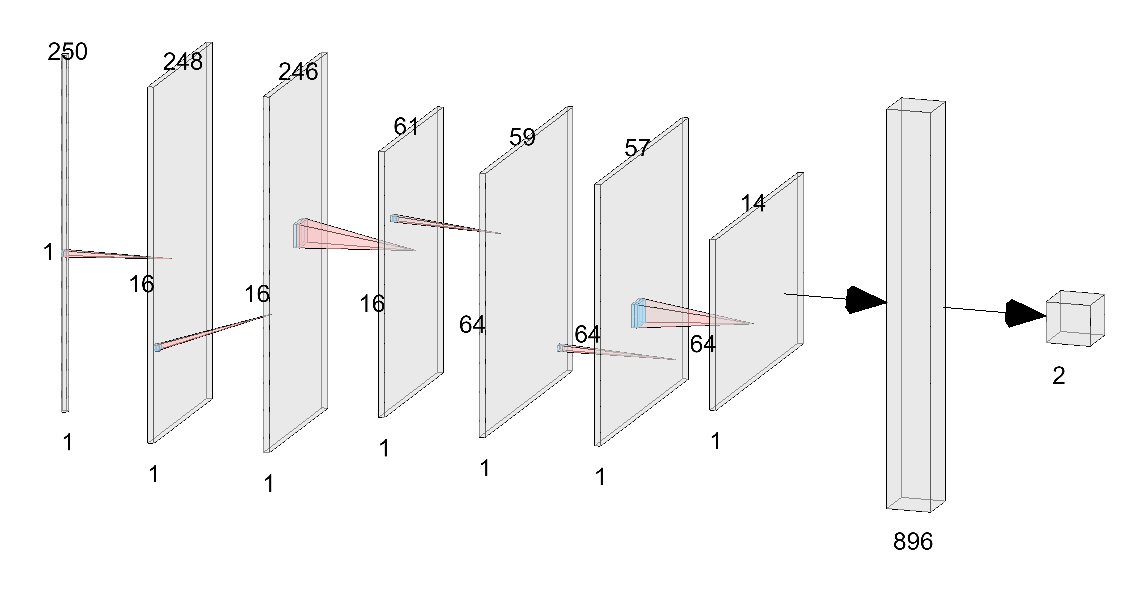
    output = Dense(2, activation="sigmoid")(flatten)

    model = Model(inputs=[inputs], outputs=output)

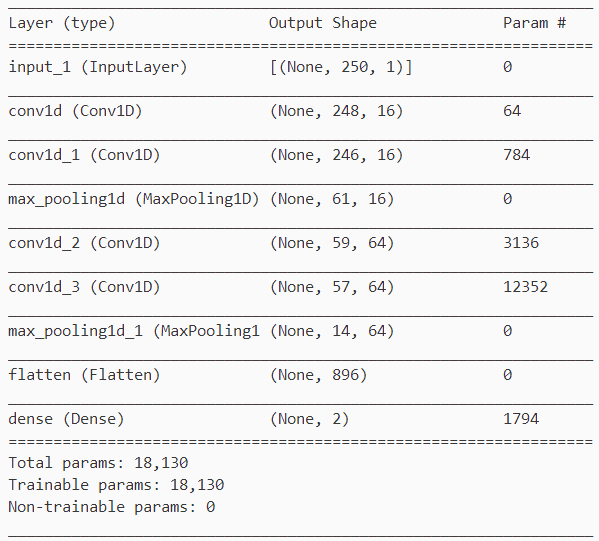
    return model

Flatten()层作为中间层来链接卷积神经网络和全连接层，神经网络被训练来最小化一个损失函数（sigmoid loss）。

可视化图像如下：

****

结构如下：



* + 1. **build\_location\_model.py：训练神经网络**

前期工作准备完毕，可以开始训练神经网络。

首先，导入所需的函数：

from keras.utils.np\_utils import to\_categorical

from sklearn.model\_selection import train\_test\_split

from MyFunction import normalization, get\_YN

from MyModel import location\_model

调用get\_YN，获取数据集，train\_test\_split分割数据集，对于X，使用normalization进行归一化处理，对于Y，利用to\_categorical转化标签形式为one-hots格式：

if \_\_name\_\_ == "\_\_main\_\_":

*# 获取数据集*

    X,Y = get\_YN()

    X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(

        X, Y, test\_size=0.25, random\_state=16

    )

    X\_train = normalization(X\_train)

    X\_test = normalization(X\_test)

    Y\_train = to\_categorical(Y\_train)

    Y\_test = to\_categorical(Y\_test)

导入数据训练location\_model，

*# 训练分类器*

    m = location\_model()

    m.compile(loss="categorical\_crossentropy",

              optimizer="adam", metrics=["accuracy"])

    m.summary()

    m.fit(

        X\_train,

        Y\_train,

        validation\_data=(X\_test, Y\_test),

        epochs=5,

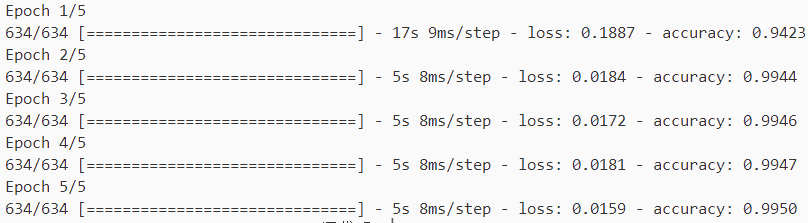
        batch\_size=64, *# 设置较大batch\_size，加快训练速度*

        use\_multiprocessing=True,

    )

设定**交叉熵损失函数（categorical crossentropy）**作为模型训练的损失函数，它刻画的是当前学习到的概率分布与实际概率分布的距离，也就是损失函数越小，两个概率分布越相似，此时损失函数接近于0。

训练后发现：分类训练仅需要五轮就能达到**99.5%** 准确率：



因此Epoch直接设置为5，减少训练时间。

最终将结果保存，留待后续组合模型使用：

    model\_json = m.to\_json()

*# 保存神经网络结构*

    with open("MyModel/location.json", "w") as json\_file:

        json\_file.write(model\_json)

*# 保存权重*

    m.save\_weights("MyModel/location.h5")

* 1. **训练分类模型**
     1. **MyFunction.get\_XY：导入并处理数据**

前面步骤基本与get\_YN相同，即多线程读取数据：

def get\_XY(step\_number: int = 16):

    path = "标注.json"

    all\_event\_information, event\_label = get\_event\_json(path)

    data\_informations = [

        data\_information(information)

        for information in all\_event\_information

    ]

    pool = Pool(cpu\_count())

    results = [

        pool.apply\_async(data\_inf.read\_dataset)

        for data\_inf in data\_informations

    ]

    pool.close()

    pool.join()

将读取的数据结果储存在对应的data\_information.dataset中，设置空列表X和Y，用于防止随后处理所得的结果：

    for i in range(len(data\_informations)):

        data\_informations[i].dataset = results[i].get()

    X, Y = [], []

遍历data\_information类组成的列表，将每一个元素中的dataset加入get\_split\_data，并设置分割步数step\_number，得到分割结果datasets后，生成与datasets所含元素相同数量的标签列表labels，二者分别加入X和Y中：

    for data\_inf in data\_informations:

        datasets = get\_split\_data(data\_inf.dataset,

                                  step\_number)

        labels = [x for x in data\_inf.data\_label

                  for i in range(len(datasets))]

        for block in datasets:

            X.append(block)

        for label in labels:

            Y.append(label)

将X，Y转换为np.array后，返回结果和事件种类：

    return np.array(X), np.array(Y), event\_label

* + 1. **MyFunction.train\_and\_test\_data：分割数据集**

由于之前遍历data\_information时数据集中数据是按顺序排列，因此若是直接使用sklearn.model\_selection中的train\_test\_split来分割，可能会导致部分事件分割不均进而导致训练不充分的问题，因此让每个事件对应部分的数据都进行一次分割，来保证均匀分割是比较稳妥的办法：

def train\_and\_test\_data(X, Y, event\_label):

    X\_train, X\_test, Y\_train, Y\_test = [], [], [], []

    i = 0

    for event in event\_label:

        X\_temp, Y\_temp = [], []

        while Y[i] == event\_label[event]:

            X\_temp.append(X[i])

            Y\_temp.append(Y[i])

            i += 1

            if i == np.size(Y):

                break

        X\_train\_temp,X\_test\_temp,Y\_train\_temp,Y\_test\_temp = train\_test\_split(

            X\_temp, Y\_temp, test\_size=0.2, random\_state=16

        )

        for j in X\_train\_temp:

            X\_train.append(j)

        for j in X\_test\_temp:

            X\_test.append(j)

        Y\_train.extend(Y\_train\_temp)

        Y\_test.extend(Y\_test\_temp)

    return np.array(X\_train), np.array(X\_test), np.array(Y\_train), np.array(Y\_test)

* + 1. **MyModel.classify\_model：设计一维卷积神经网络**

相比location\_model，增加了**2层Conv1D**，其余结构基本相同：

def classify\_model():

    clear\_session()

    inputs = Input(shape=(16 \* 22, 1))  *# 输入层*

*# 卷积层起始*

    conv = Conv1D(16, 3, activation="relu")(inputs)

    conv = Conv1D(16, 3, activation="relu")(conv)

    pool = MaxPooling1D(3)(conv)

    conv = Conv1D(128, 3, activation="relu")(pool)

    conv = Conv1D(128, 3, activation="relu")(conv)

    pool = MaxPooling1D(3)(conv)

    conv = Conv1D(128, 3, activation="relu")(pool)

    conv = Conv1D(128, 3, activation="relu")(conv)

    pool = MaxPooling1D(3)(conv)

*# 卷积层结束*

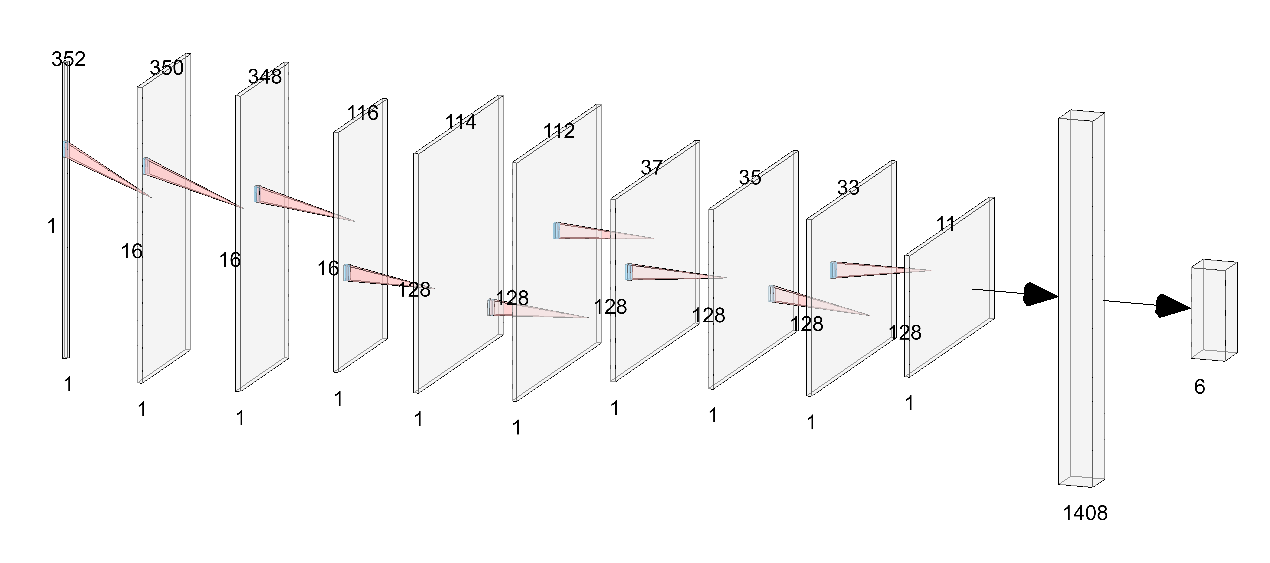
    flatten = Flatten()(pool)

    output = Dense(6, activation="softmax")(flatten)

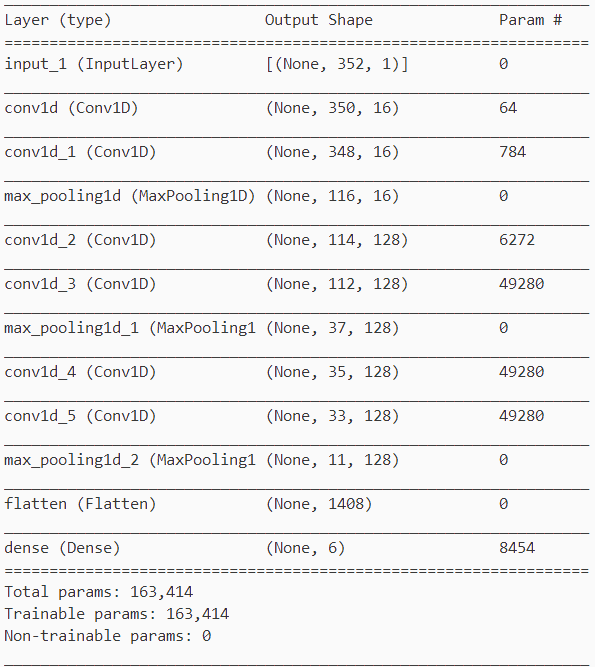
    model = Model(inputs=[inputs], outputs=output)

    return model

可视化结构如下：



具体结构如下：



* + 1. **build\_classify\_model.py：训练模型**

前期工作完成，开始训练模型，同样先导入所需的函数：

from MyFunction import get\_XY,normalization, train\_and\_test\_data

from keras.utils.np\_utils import to\_categorical

from MyModel import classify\_model

然后使用**get\_XY**获取数据集，设置切割步数**step\_number**为8以求在**有限的数据集**中获取更多训练数据：

if \_\_name\_\_ == "\_\_main\_\_":

*# 获取数据集*

    X, Y, event\_label = get\_XY(step\_number=8)

先导入模型，并查看模型的整体结构：

    m = classify\_model()

    m.summary()

由于分类模型与定位模型相比，情况更复杂，因此需要多次训练模型后，再从效果较好的模型中挑选一个作为最终的结果，我选择进行循环十次，生成十个模型，再经过测试，选择效果最好的模型留待后续使用：

    for i in range(10):

        print("i={}".format(i))

        X\_train, X\_test, Y\_train, Y\_test = train\_and\_test\_data(X, Y, event\_label)

        X\_train = normalization(X\_train)

        X\_test = normalization(X\_test)

        Y\_train = to\_categorical(Y\_train, len(event\_label))

        Y\_test = to\_categorical(Y\_test, len(event\_label))

*# 训练分类器*

        m = classify\_model()

        m.compile(

            loss="categorical\_crossentropy",

            optimizer="adam", metrics=["accuracy"]

        )

        m.fit(

            X\_train,

            Y\_train,

            epochs=60,

            batch\_size=64,

            use\_multiprocessing=True,

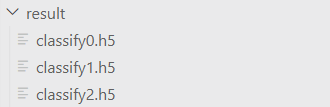
        )

每轮保存一次权重，将最终结果都保存在MyModel/result中：

        name = "classify" + str(i)

        m.save\_weights("MyModel/result/" + name + ".h5")

部分效果如下：

****

循环结束，将模型保存为json文件：

*# 保存卷积神经网络模型*

    model\_json = m.to\_json()

    with open("MyModel/classify.json", "w") as json\_file:

        json\_file.write(model\_json)

* 1. **组装模型**
     1. **MyModel.MyModel类:导入以生成模型**

首先导入所需的库函数（部分函数用于MyModel.location\_model以及MyModel.classify\_model）：

import numpy as np

from MyFunction import normalization

from keras.utils.np\_utils import to\_categorical

from keras.models import model\_from\_json, Model

from keras.backend import clear\_session

from keras.layers import Conv1D, Dense, Flatten, Input, MaxPooling1D

上述两个模型分别生成了一个json文件保存神经网络结构，h5文件保存训练结果对应的参数，因此，要使用以生成模型，需要将之读取，而为了使用方便，设置一个类来保证模型的易用性：

class MyModel:

    def \_\_init\_\_(self, model\_name, h5\_name) -> None:

        with open("MyModel/" + model\_name, "r") as model:

            json\_str = model.read()

        self.model = model\_from\_json(json\_str)

        self.model.load\_weights("MyModel/" + h5\_name)

该类接受一个json文件名和一个h5文件名，之后在/Model文件夹中查找文件是否存在，然后使用tensorflow.Keras所带函数导入模型以及参数到self.model中。

其具有两个函数，第一个函数为**evaluate**，接受X、Y，用于**测试**数据集的准确度和损失值，在其内设置的函数与前面训练模型时完全相同：

    def evaluate(self, X, Y):

        X = normalization(X)

        Y = to\_categorical(Y)

        self.model.compile(

            loss="categorical\_crossentropy",

            optimizer="adam",

            metrics=["accuracy"]

        )

*# 分类准确率*

        loss, accuracy = self.model.evaluate(X, Y, verbose=0)

        print("accuracy: {:.2f}%".format(accuracy \* 100),

              end=" - ")

        print("loss: {:.3f}".format(loss))

第二个函数为**predict**，只需要接受X，

    def predict(self, X):

        X = normalization(X)

        p = np.argmax(self.model.predict(X, batch\_size=32,

                                         verbose=0),

                      axis=1)

        return p

由于预测结果为one-hots格式，故利用np.argmax函数，定位结果最大值的位置，即为原本的预测标签形式，返回结果列表

* + 1. **test\_classify\_model.py：测试分类模型**

由于**build\_classify\_model.py**中产生了多个h5文件，我们需要测试，找到效果最好的参数结果，且需要知道模型对于每个事件的识别率各自为多少，首先导入所需函数：

import numpy as np

from MyModel import MyModel

from MyFunction import get\_XY

然后提取所以事件数据X及其对应标签Y：从每个事件各取一个数据块和对应标签，设为列表X\_extend以及Y\_extend，留在后续测试单一事件时使用：

if \_\_name\_\_=='\_\_main\_\_':

    X, Y, event\_label = get\_XY()

    X\_extend, Y\_extend = [], []

    i = 0

    for event in event\_label:

        while Y[i] == event\_label[event]:

            i += 1

            if i == np.size(Y):

                break

        X\_extend.append(X[i - 1])

        Y\_extend.append(Y[i - 1])

开始循环，将模型导入MyModel类，每轮循环导入一个h5文件：

    for i in range(10):

        print("i={}".format(i))

        name = "classify" + str(i) + ".h5"

        my\_model = MyModel("classify.json", "result/" + name)

测试整体准确率：

        my\_model.evaluate(X, Y)

分事件测试，由于事件排列有序，直接按照事件标签遍历，存为temp：

        j = 0

        for event in event\_label:

            X\_temp = []

            Y\_temp = []

            print(event, end=":")

            while Y[j] == event\_label[event]:

                X\_temp.append(X[j])

                Y\_temp.append(Y[j])

                j += 1

                if j == np.size(Y):

                    break

            X\_temp = np.array(X\_temp)

将前面的extend接到单一事件集结尾，否则无法将Y\_temp转为one\_\_hots形式：

            X\_temp[-6:] = np.array(X\_extend)

            Y\_temp[-6:] = np.array(Y\_extend)

测试并查看预测结果：

            my\_model.evaluate(X\_temp, Y\_temp)

            p\_temp = my\_model.predict(X\_temp)

            print(p\_temp)

部分效果如下：



选取整体效果最好的模型，改为**/MyModel/classify.h5**，用于最终模型。

* + 1. **location\_and\_classify\_model：定位-分类模型**

导入所需函数：

import numpy as np

from collections import Counter

from openpyxl import load\_workbook

from MyFunction import get\_row, myfft, get\_split\_data

from MyModel import MyModel

正式开始组装模型，设定一个**location\_and\_classify\_model**类，其内存在两个**MyModel**类 ，分别为定位模型**location\_model**以及分类模型**classify\_model，**同时，将事件及其标签作为一个dict单独存储为labels：

class location\_and\_classify\_model:

    def \_\_init\_\_(self) -> None:

        self.location\_model = MyModel("location.json",

                                      "location.h5")

        self.classify\_model = MyModel("classify.json",

                                      "classify.h5")

        self.labels = {"攀爬": 0, "大风": 1, "敲击": 2,

                       "锯磨": 3, "摇晃": 4, "电钻": 5}

设计一个方法函数，用于读取.xlsx文件，**定位模型**仅需要每列的500行来进行判断和处理数据，因此接受一个start\_row，默认从第0行开始，取start\_row后500行数据，然后将读取的数据转化为np.array后返回：

    def read\_data(self, file\_path, start\_row:int =0):

        data\_file = load\_workbook(file\_path, read\_only=True)

        sht = data\_file[data\_file.sheetnames[0]]

        event = np.array(get\_row(sht, 0, sht.max\_column,

                                 rows\_end=start\_row+500))

        return event

接下来开始定位及预测，接受np.array类型的事件数据event，将event按列分割为500\*1的数据块，通过MyFunction.myfft对每个数据块进行快速傅里叶变换处理后只取对半，经过reshape为250\*1的np.array对象，插入待定位数据集to\_location，最后调用**location\_model.predict**，对每个数据块进行“0”与“1”分类，得到结果列表p：

    def location\_and\_classify(self, event):

        to\_location = []

        for i in range(np.size(event, axis=1)):

            temp = event[:, i]

            to\_location.append(np.array(myfft(temp))

                               .reshape(250, 1))

        to\_location = np.array(to\_location)

        p = self.location\_model.predict(to\_location)

对于得到的结果p，有即是一个全是“0”与“1”的列表，而“1”就是判断为有事件发生，因此其所对应的列便是事件发生位置：

*# 找到预测中为1的位置*

        loc = np.where(p == 1)[0]

但是由于可能产生离散点，故需要将之进行分割为数个元素连续的列表：

        temp = []

        for i in range(len(loc)):

            if not temp:

                temp.append([loc[i]])

            elif loc[i - 1] + 1 == loc[i]:

                temp[-1].append(loc[i])

            else:

                temp.append([loc[i]])

对‘大风/220913\_142406.xlsx’定位结果分割前：



分割后：



对于分割结果，选择所有长度超过12的列表，将之判断为事件发生位置，分别记录首位和末尾作为事件发生范围，加入location\_results：

*# 定位结果*

        location\_results = []

        for l in temp:

            if len(l) >= 12:

location\_results.append([l[0], l[-1]])

加下了进行事件分类，遍历定位结果，根据其内所记录范围**按列**切割事件数据集event，再使用函数get\_split\_data分割处理后使用分类模型classify\_model.predict进行预测，得到预测结果：每个数据块所产生一个标签组成的列表，使用collections.Counter找到出现次数最多的标签，记为最终分类结果，加入classify\_results：

        classify\_results = []

        for i in range(len(location\_results)):

            location\_result = location\_results[i]

            total\_dataset = event[:, location\_result[0] : location\_result[1] + 1]

            X = []

            to\_classify = get\_split\_data(total\_dataset, step\_number=8)

            for block in to\_classify:

                X.append(block)

            X = np.array(X)

            p = self.classify\_model.predict(X)

*# 分类结果*

            classify\_results.append(Counter(p).

                                    most\_common(1)[0][0])

将最终结果存为dict形式并返回：

        result = {"labels": classify\_results,

                  "boxes": location\_results}

        return result

对‘大风/220913\_142406.xlsx’定位-预测结果如下



设置一个函数，将结果保存，文件名默认为result.npy：

    def save\_result(self, result: dict, file\_path, npy\_name="result.npy"):

        result['file\_path']=file\_path

        out={"result":result,"labels":self.labels}

        np.save(npy\_name,out)

* 1. **test.py：测试**

依次读取**"标注.json"**的文件，利用并行计算快速读取文件后：

if \_\_name\_\_ == "\_\_main\_\_":

    json\_path = "标注.json"

    pool = Pool(cpu\_count())

    all\_event\_information, event\_label = get\_event\_json(json\_path)

    dataset\_informations = [

        data\_information(information) for information in all\_event\_information

    ]

    beg\_time = datetime.now()

    model = location\_and\_classify\_model()

    print("获取预测数据")

    results = [

        pool.apply\_async(read\_data, [dataset\_inf.data\_path])

        for dataset\_inf in dataset\_informations

    ]

    pool.close()

    pool.join()

    print("读取完毕")

    print("共消耗: " + "{:.2f}".format((datetime.now() - beg\_time).total\_seconds()) + " 秒")

对比每一个文件的预期和结果：

    for n in range(len(results)):

        ans = {

            "labels": dataset\_informations[n].data\_label,

            "boxes": [dataset\_informations[n].event\_local],

        }

        res = model.location\_and\_classify(results[n].get())

        print("---------")

        print("预期:", end="\t")

        print(ans)

        print("结果:", end="\t")

        print(res)

部分结果如下：

