

步态识别行人分类实验



实验目的

- 综合运用之前课程中学习的各种机器学习算法，完成步行传感器数据的**分类**实验
- 通过完整的数据处理，算法选择，实验设计，结果分析流程掌握并深刻理解各种机器学习算法
- 具备使用机器学习算法解决实际问题的能力
- 培养科学研究的一般思路和方法

实验任务

- 对给定的数据进行预处理
- 将数据处理成为算法可以使用的格式
- 对传感器数据进行分类，判断其属于哪一个人

背景

手机传感器 [1]

为满足手机的日常使用场景，现代智能手机内置了多种传感器，本实验主要介绍两种

- **加速传感器：**加速度传感器，顾名思义就是一种能够测量加速度的电子设备。运用压电效应实现，重力感应模块由一片“重力块”和压电晶体组成，当手机发生动作的时候，重力块会和手机受到同一个加速度，这样重力块作用于不同方向的压电晶体

上的力也会改变，这样输出的电压信号也就发生改变，根据输出电压信号就可以判断手机的方向了。

- 陀螺仪：陀螺仪是一种用于测量角度以及维持方向的设备，原理是基于角动量守恒原理。陀螺仪的用途主要是手机的摇一摇，或者在某些游戏中可以通过移动手机改变视角或 VR 应用。

步态

- 步态是指行人走路时所表现的姿态及走路所有的动作，这是一种复杂的行为特征，每个人都拥有一种与众不同的步态
- 步态唯一性的物理基础是每个人生理结构的差异性，不同的腿骨长度、不同的肌肉强度等共同决定了步态的唯一性
- 步态是一种非接触的生物特征识别技术，在识别的过程中不需要人的行为配合
- 步态识别是一种新兴的生物特征识别技术

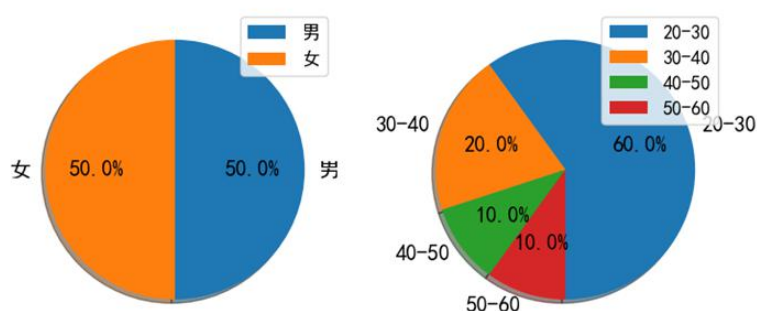
研究步态的意义

- 利用步态进行身份识别
- 利用步态进行健康检查、疾病诊断
- 利用步态进行行为预测、步行状态分析

数据

数据采集信息

- 行人个数：10。行人性别、年龄分布见下图。























- 数据采集条件：行人右手手持手机在自由状态下行走 10 分钟



- 数据格式：时间戳 X 轴数据 Y 轴数据 Z 轴数据
- 采样频率：50HZ，即每 20ms 采样一次
- 数据收集时间：2017-10-26

数据详情

数据在文件夹 **data** 中，**data** 中共有 20 个文件，分别为 **accData0.txt ~ accData9.txt** 和 **gyrData0.txt ~ gyrData9.txt**。10 个 **accData** 对应 10 个人收集的加速度传感器数据，10 个 **gyrData** 对应 10 个人收集的陀螺仪传感器数据。

 accData0.txt	2017/10/26 21:25	文本文档	1,336 KB
 accData1.txt	2017/10/26 21:25	文本文档	1,334 KB
 accData2.txt	2017/10/26 21:25	文本文档	1,333 KB
 accData3.txt	2017/10/26 21:25	文本文档	1,329 KB
 accData4.txt	2017/10/26 21:25	文本文档	1,341 KB
 accData5.txt	2017/10/26 21:25	文本文档	1,338 KB
 accData6.txt	2017/10/26 21:25	文本文档	1,335 KB
 accData7.txt	2017/10/26 21:25	文本文档	1,331 KB
 accData8.txt	2017/10/26 21:25	文本文档	1,329 KB
 accData9.txt	2017/10/26 21:25	文本文档	1,337 KB
 gyrData0.txt	2017/10/26 21:25	文本文档	1,394 KB
 gyrData1.txt	2017/10/26 21:25	文本文档	1,364 KB
 gyrData2.txt	2017/10/26 21:25	文本文档	1,402 KB
 gyrData3.txt	2017/10/26 21:25	文本文档	1,383 KB
 gyrData4.txt	2017/10/26 21:25	文本文档	1,378 KB
 gyrData5.txt	2017/10/26 21:25	文本文档	1,377 KB
 gyrData6.txt	2017/10/26 21:25	文本文档	1,398 KB
 gyrData7.txt	2017/10/26 21:25	文本文档	1,377 KB
 gyrData8.txt	2017/10/26 21:25	文本文档	1,409 KB
 gyrData9.txt	2017/10/26 21:25	文本文档	1,382 KB

```
accData0.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
1508939178859 -3.7253778 1.3024458 10.429143
1508939178895 -3.6966474 0.82360536 10.515334
1508939178896 -3.169923 0.92895025 9.950302
1508939178907 -2.6719291 0.90979666 9.3661175
1508939178927 -2.6910827 0.48841715 9.11712
1508939178947 -2.873042 0.2777274 8.839393
1508939178967 -3.0550013 0.41180268 8.714894
1508939178990 -3.1028855 0.5363012 8.66701
1508939179007 -3.0933087 0.80445176 8.590396
1508939179027 -3.064578 1.2545617 8.6095495
1508939179047 -3.0262709 1.7429788 8.7532015
1508939179067 -2.930503 2.0590134 8.465898
1508939179088 -2.8921957 2.2409728 8.159439
1508939179108 -2.873042 2.2505496 7.97748
1508939179127 -2.7389667 2.1164744 8.226477
1508939179147 -2.6623523 1.8195933 8.456321
1508939179168 -2.7102363 1.7238252 8.475474
1508939179190 -2.7868507 1.8770541 8.379706
1508939179208 -2.8634653 2.030283 8.37013
1508939179227 -2.7485435 2.2026656 8.264784
```

对于每一个数据文件（加速度传感器数据和陀螺仪传感器数据），里面有 30500 行，即每一行对应一条数据。每一行有四个数字，以空格分隔，第一个是时间戳，第二个是 X 轴数据，第三个是 Y 轴数据，第四个是 Z 轴数据

请注意：虽然采集数据的时候设定了固定的采样频率 50HZ，但是实际得到的数据并不是严格的间隔 20ms。

你需要做什么

- ☐ 提供的数据是时间连续的序列，你需要自己将其分割成若干组数据，处理成为算法可以使用的格式用于训练和测试
- ☐ 标定数据。10 个文件对应了 10 个人的数据，将处理完的数据的标签设置成为其对应的人
- ☐ 选择合适的机器学习算法，对处理好的数据进行分类（10 分类任务）
- ☐ 不断调整算法的参数或结构进行优化以取得更好的结果
- ☐ 在实验的过程中随时记录你的实验情况，包括但不限于数据的预处理过程及结果、选择的算法、参数的设定、实验的训练过程和测试结果

实验要求

注意：为便于最终的效果对比，请务必遵守以下要求，否则将会影响成绩

1. 请自行将原始数据处理成为算法可用的数据并自行划分训练集、验证集和测试集。但是请保证 **测试集合中的数据量不少于 300 组**
2. 划分训练集和测试集的时候请使用随机划分或者前 N%为训练集，后(100-N)%为测试集的划分方式，不允许间隔划分或选取特殊位置的数据作为测试集
3. 请务必保证你提交的源代码 **可以运行**。因此请将处理数据的所有逻辑也一起提交
4. 对于提交的实验结果，请提供相关的运行 **截图** 或 **实验结果**

需要提交的材料

实验 9-学号-姓名.[zip | 7z | tar]，其中包含：

1. 实验报告.[doc | docx | txt | pdf | md]，其中包括以下章节（不再使用之前的报告模板，括号内的百分比表示该部分内容的大概篇幅占比，你可以根据实际情况进行调整）：
 - 数据处理（30%）：请说明数据处理的方法，**为什么**要这样处理，并提供相关代码逻辑的截图
 - 实验过程（50%）：请说明实验的整个过程。主要包括 **算法** 的选择，算法相关参数或结构的设定，实验的 **思路** 和具体流程。
 - 结果及结论（10%）：展示实验结果。由于神经网络等算法每次运行的结果可能稍有不同，请提交效果最好的一次，但务必提交相关的截图或者运行结果。
 - 程序说明（10%）：请说明你的代码的运行方式和程序入口
2. 源代码（文件夹），其中包括所有的源码。若有对原始数据的中间处理结果，也请放到这里

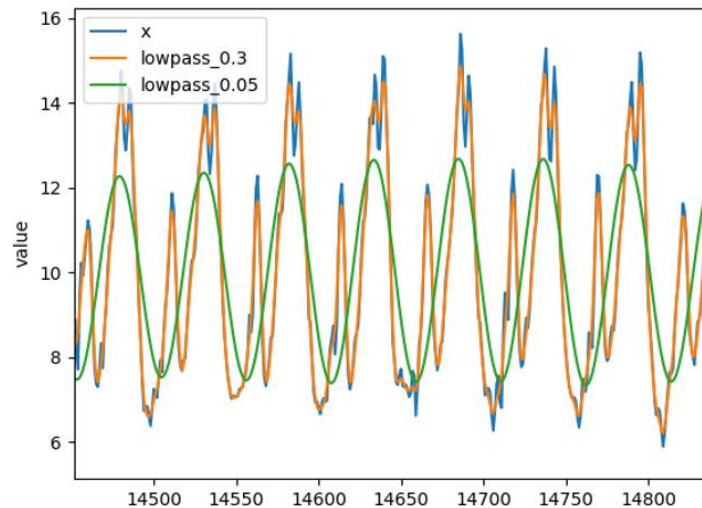
一些有用的提示

数据处理

数据预处理

在划分数据之前可以尝试对数据进行预处理，包括但不限于以下方法：

1. 滤波：对原始数据进行滤波可以有效地去除噪声平滑曲线 [2]。下图为使用巴特沃斯低通滤波器，截断频率分别 0.3hz 和 0.05hz 对原始传感器数据进行滤波

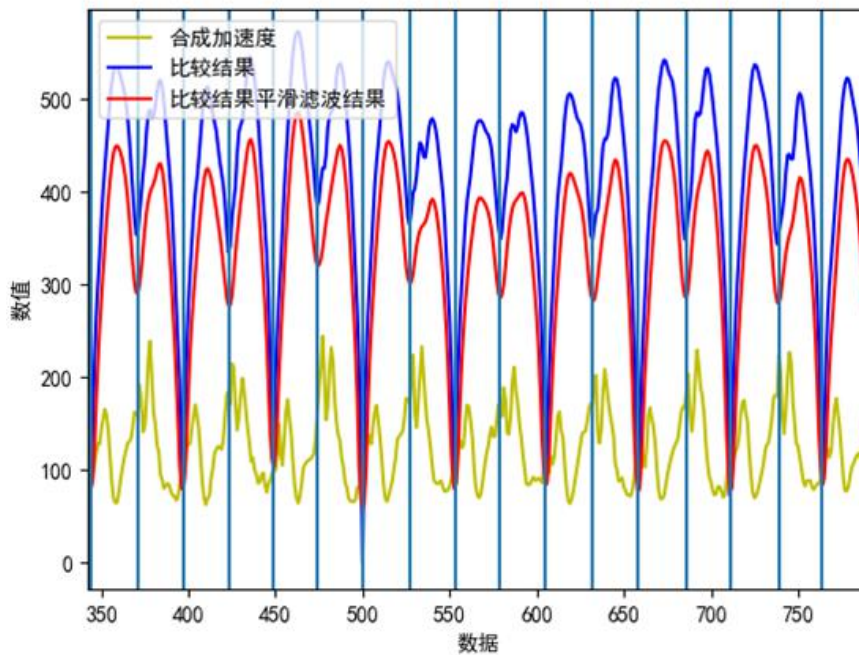


2. `# 低通滤波代码`
3. `from scipy import signal`
4. `b, a = signal.butter(8, 0.02, 'lowpass')`
5. `filteredData = signal.filtfilt(b, a, data)` #data 为要过滤的信号
6. 数据增强：可以使用插值等算法对数据进行增强处理。如使用 RNN 网络进行训练时，输入网络的每一个数据的数据点个数必须相等，但是划分数据之后每一个数据内的数据点个数可能不相等，此时可以通过插值处理得到固定数据点个数的数据

数据划分

需要将连续的时间序列数据划分为若干组数据，包括但不限于以下方法：

1. 滑动窗口分割：选择合适的窗口宽度和滑动步长，对数据进行划分
2. `# 滑动窗口代码`
3. `with`
`open(os.path.join(PATH/TO/DATA/FOLDER, "accData0.txt")) as`
`acc_file:`
4. `acc_datas = acc_file.readlines()`
5. `for i, acc_data in enumerate(acc_datas):`
6. `t, x, y, z = [float(i) for i in acc_data.split(" ")]`
7. `acc_datas[i] = [x, y, z]`
8. `p = 0`
9. `while True:`
10. `if p + WINDOW_LENGTH < len(acc_datas):`
11. `yield acc_datas[p:p+WINDOW_LENGTH]`
12. `p = p + WINDOW_LENGTH//2` # 每两个窗口有50%的重合部分
13. 步态周期分割：由于步态数据呈现明显的周期性，可以据此进行数据划分 [3]。如下图，选取一段数据作为模板，将模板在整个数据上滑动计算模板和数据之间的欧氏距离，得到呈现周期性规律的图像，据此划分步态周期



算法选择

原则上不对使用的算法进行限制。可以使用传统机器学习算法，如 KNN，SVM 等算法，也可以使用深度神经网络，对于网络的选择和结构也不做要求。给定在此数据集上进行分类一些常见算法的 baseline 结果，所以理论上你通过调整参数和网络结构所能取得的最好成绩应该比 baseline 要好。

算法	分类准确率(%)	说明
贝叶斯网络	90	
逻辑回归	92	
多层感知机	94	
KNN	94	K=1
KNN	93	K=2
KNN	94	K=3
J48	92	
随机森林	95	
SVM	92	
神经网络	97	1D 卷积(5)+池化(3)+1D 卷积(5)+池化(3)+全连接(10)
神经网络	95	GRU(128)+全连接(10)
神经网络	97	LSTM(32)+全连接(10)
神经网络	95	LSTM(32)+Dropout(0.01)+全连接(10)

如何获得更好的成绩

实验完成以下中的任意一条都会极大地提高你的成绩

- 获得最高的 分类准确率

- 对于数据中十个人里每一个，都提供了两个数据文件(`accDatai.txt` 和 `gyrDatai.txt`)，分别包含第 *i* 个人的加速度传感器数据和陀螺仪传感器数据。你可以使用单独一个文件中的数据进行分类。如果你同时使用两种数据，请 **探索如何将两种数据有机地结合在一起使用，并探索相较于单独使用一种数据，同时使用两种数据是否会提升分类准确度，请通过详细的实验或者理论进行证明**
- 通过调整网络结构或者改变模型参数，有效地提高分类准确率，并 **从理论和实验两方面证明这种改进的意义**

数据处理

数据的直观理解和可视化

根据任务指导，数据集中包括从手机加速度传感器和角速度传感器中获得的三轴速度数据，同时附带时间戳发现数据为按列用空格分开，考虑直接使用 `pandas.read_csv` 进行读取

```
1 acc_data1 = pd.read_csv(os.path.join(data_path, file_names[0]), sep=' ', header=None)
2 gyro_data1 = pd.read_csv(os.path.join(data_path, file_names[10]), sep=' ', header=None)
```

数据读取代码示例

查看第一个人的加速度和角速度数据

1	acc_data1.head()			
	0	1	2	3
0	1508939178859	-3.725378	1.302446	10.429143
1	1508939178895	-3.696647	0.823605	10.515334
2	1508939178896	-3.169923	0.928950	9.950302
3	1508939178907	-2.671929	0.909797	9.366117
4	1508939178927	-2.691083	0.488417	9.117120

1	gyro_data1.head()			
	0	1	2	3
0	1508939178999	-0.048869	0.095295	0.069639
1	1508939179021	-0.048869	0.095295	0.069639
2	1508939179039	-0.048869	0.095295	0.069639
3	1508939179059	-0.048869	0.095295	0.069639
4	1508939179079	-0.048869	0.095295	0.069639

使用 pandas 读取数据结果示例

这里最需要注意的一点就是时间戳，一则两两之间间隔不同，一则两个表格虽然总行数一样但时间戳并不能很好地对应。数据集中我们已知理论设定的采样间隔是 **20ms**，显然实际时间戳上有很多不均匀的采样点。同时可以预见之后融合两个数据集直接做分类会潜在地影响模型性能，因为二者数据并不逐行匹配，也即并非逐行同时时间点采集出来的。

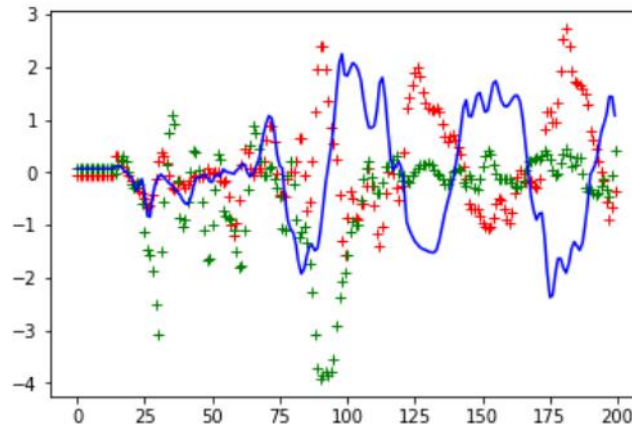
进一步将部分数据绘制出来进行观察。一些代码和截图如下


```

1 plt.figure()
2 plt.plot(gyro_data1.iloc[:200,1], 'k+', color='r')
3 plt.plot(gyro_data1.iloc[:200,2], 'k+', color='g')
4 plt.plot(gyro_data1.iloc[:200,3], color='b')

```

[<matplotlib.lines.Line2D at 0x7f88743242b0>]



三轴角加速度数据可视化

直接选取前 200 个陀螺仪数据，三个维度分别用不同颜色绘制，可以看出每个维度分别呈现出一些独特的周期性，但是一开始 50 个左右的采样点都是处于一个不稳定的“预热阶段”，这块数据极其独特，我们可以理解为是一开始采样实验的噪声，也可以理解为其实是包含了用户初始加速度的重要信息， $50 \times 20\text{ms} = 1\text{s}$ 刚好差不多是被实验者听到指令后开始行动走出第一步到第二步的时间(参考成年人平均步速 1m/s ，步频 1.5步/s ，受试者年龄从中青年到中老年)，因此这一块数据也是一定可以成功分类的，但是得依赖训练集中其他周期性数据中提取出来的时域不变性特征(如果希望性能尽可能高的话，这里就基本确定了我们的出发点应该是神经网络)。

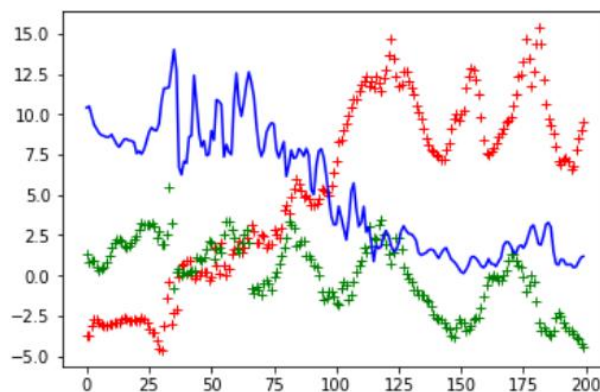
进一步看一下加速度数据，

```

1 plt.figure()
2 plt.plot(acc_data1.iloc[:200,1], 'k+', color='r')
3 plt.plot(acc_data1.iloc[:200,2], 'k+', color='g')
4 plt.plot(acc_data1.iloc[:200,3], color='b')

```

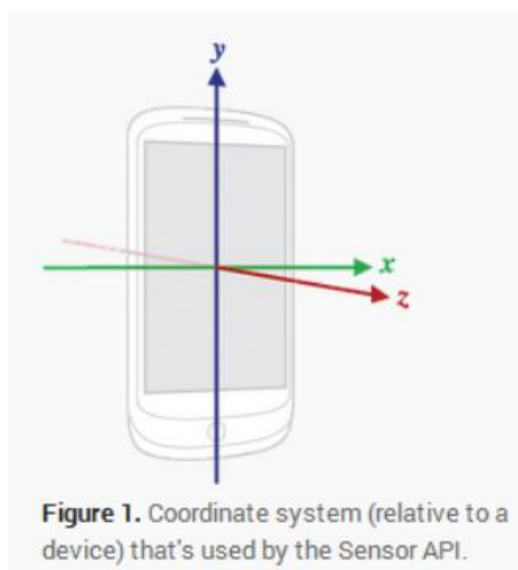
[<matplotlib.lines.Line2D at 0x7f887424abe0>]



三轴加速度数据可视化

可以看到加速度数据倒是很早就呈现出了周期性，相较于角速度那边“预热”阶段只用了 25ms 左右，不过也情有可原，一般第一步迈出去腿的初速度会比手臂大（大腿肌肉力量远大于大臂和肩膀而且是先有迈步为了维持身体平衡，才有摆手动作）。在初期加速度受腿部动作支配，当摆手动作规律之后，整个信号的周期性将主要受到手部动作支配。同时还注意到加速度周期性比角速度更好。这个图里有一个神奇的地方就是前 $200 \times 20 = 4\text{s}$ 内， z 轴速度平均一直在下降，为了理解手机的世界坐标

系，请看如下图示，



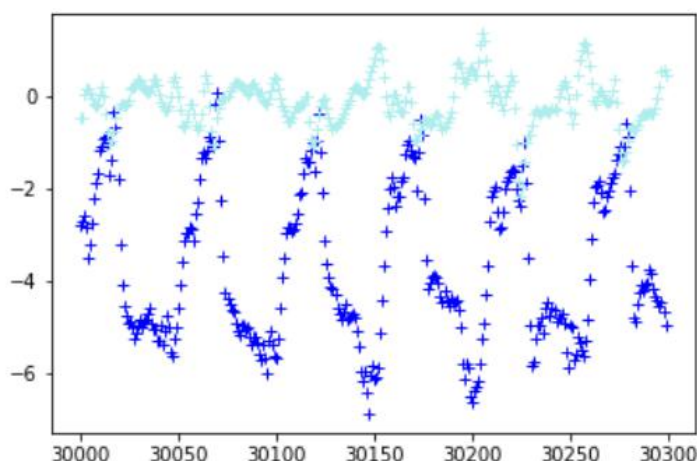
手机坐标系示例

手机屏幕朝向为 z ，和屏幕同平面的分别是 x ， y 方向。结合人们走路时拿手机的方式（虽然会有一定变动，但大部分人不会横着拿手机走路，硌手也不好抓；也可以看数据集采集示例图）由此可知此处的 z 轴大概率指的是人手左右摆动的加速度，这就可以理解为何加速度变慢了，因为一开始手甩出去然后逐渐回到一个平均的摆动幅度和周期，包括左右晃动的幅度也就逐渐稳定下来。

现在来将两个数据结合起来看一看比较靠后的 300 个采样点(基于受实验者不会专门注意实验的时间，快结束时实验者提醒还剩多少秒他们会提起精神，最后认真地规律走一小段路，所以取了靠后的点来看)

```
1 plt.figure()
2 plt.plot(acc_data1.iloc[-500:-200,2], 'k+', color='b')
3 plt.plot(gyro_data1.iloc[-500:-200,2], 'k+', color='paleturquoise')
```

[<matplotlib.lines.Line2D at 0x7f8873217240>]

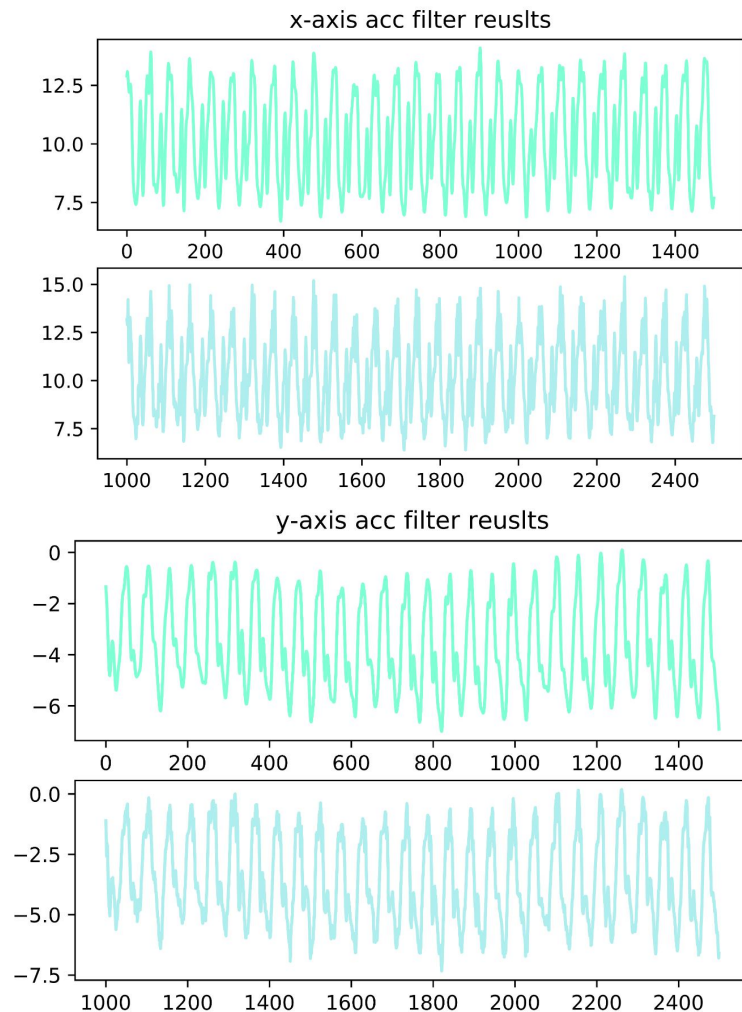


加速度和角加速度数据可视化对比

果然这里呈现出了很强的周期性，我们取了 y 轴方向也即人前进的方向的加速度和角速度都很有规律，符合我们之前的推测和直觉。

数据的预处理

接下来我们需要对数据进行预处理。数据处理可以分为四部分：降噪，重采样，时间序列划分，选取不同的数据来源(acc, gry, both 二者都使用)。降噪我们采用了 Butterworth 低通滤波器，经过经验性地简单测试和绘图，在保证信号信息并尽可能减少噪声干扰的条件下，我分别确定了对加速度数据和角速度数据应该使用的滤波器参数，如下图(经验性测试还表明不同轴的同种类数据可以使用同一种滤波器参数影响不大)



x, y 轴降噪前后数据变化示例(上:降噪后, 下:原始数据)

用同样的方式使用小波变换对信号进行了降噪，结果如下，

```
def wavelet_denoising(data):  
    # 小波函数取db4  
    db4 = pywt.Wavelet('db4')  
    if type(data) is not None:  
        # 分解  
        coeffs = pywt.wavedec(data, db4)  
        # 高频系数置零  
        coeffs[len(coeffs)-1] *= 0  
        coeffs[len(coeffs)-2] *= 0  
        # 重构  
        meta = pywt.waverec(coeffs, db4)  
    return meta
```

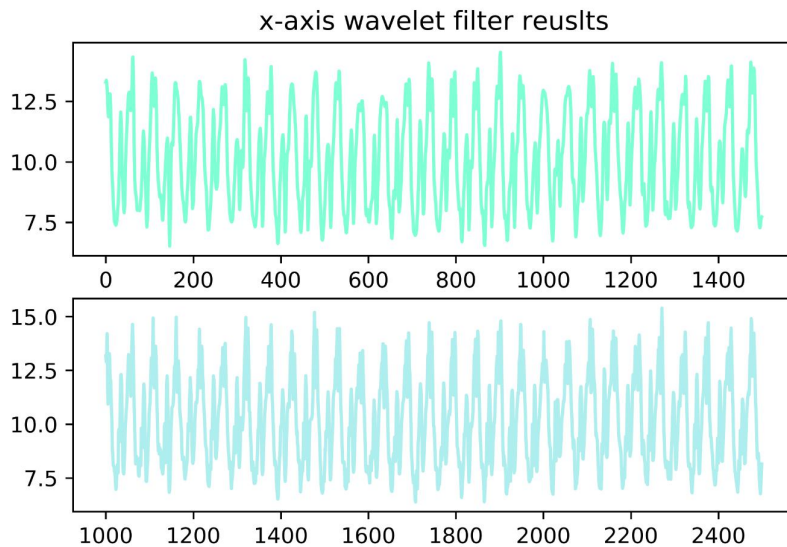


图 10 小波降噪函数和结果示例

可以看到通过高频系数置零，我们比较好地去除了噪声并保留了信号的尖峰信息，相比较 **Butterworth** 低通滤波，在尖峰和尖峰前一个副高峰信号的保留上，小波变换做的更好，这也说明这种看似不平滑的锯齿对于这个信号来说其实属于相对平稳变化的低频部分，这样才没有被滤波过滤掉。

接下来在以上基础上，我们可以进一步对数据进行重采样并进行插值，以缓解之前时间戳不均匀带来的潜在问题。首先我们可以直观感受一下数据偏差有多少，如下图我统计了时间差在 20ms 以上/21ms/22ms/25ms 等的点数量，发现阈值为 21-22 时基本离群点就很少了，相比较严格 20 时有将近 4000 个点不符合要求来看，采样还是值得的。

```
1 sum(np.abs(acc_data1.iloc[:-1,0].reset_index(drop=True) - acc_data1.iloc[1:,0].reset_index(drop=True)) > 22)
237

1 plt.figure(dpi=1000)
2 plt.plot(np.abs(acc_data1.iloc[:-1,0].reset_index(drop=True) - acc_data1.iloc[1:,0].reset_index(drop=True)) > 25, 'k+')
3 plt.xlim(1)
4 plt.show()
```

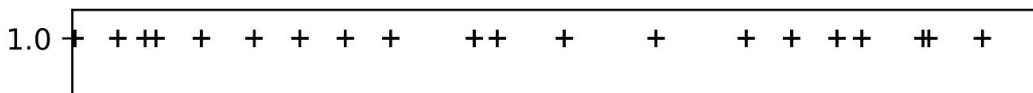


图 11 时间戳离群点可视化

使用了 **scipy** 中的 **interpolate.interp1d** 多项式插值对信号进行重采样。示例结果如下，途中橘红色点是重采样过后的点，时间分布严格按照 20ms 分开。

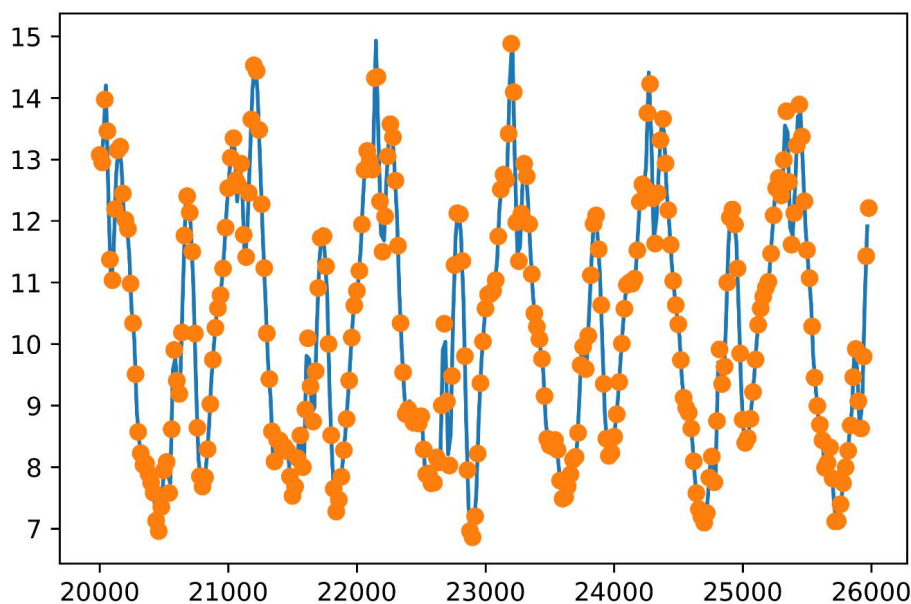


图 12 数据重采样示例

下一步我们进行时间序列的采样，这里并没有做出实质上划分并实例化数据集的子集，为了方便后续实验，我动态地划分了数据集的行号，然后根据行号取出对应数据集中对应的时间序列。同时针对性地实现了一个 Pytorch Dataset 类，方便后续大规模参数化实验，支持输入不同数据/分割不同长度序列/不同网络对数据格式不同需求等。具体细节可见代码实现，简图如下，

```
def index_split(data_length, window_len:int, stride=1, mode=1):
    """Split the data_index into train/test index set according to the given datatype"""
    if window_len <= 0:
        print("window len must be a positive integer")

    # series_length = (data_length-window_len)//stride + 1
    full_index = [i for i in range(0, data_length, stride) if i+window_len<=data_length]
    if mode==1:
        np.random.shuffle(full_index)
        train_index = full_index[:int(len(full_index)*0.8)]
        test_index = full_index[int(len(full_index)*0.8):]
    if mode == 2:
        np.random.shuffle(train_index)
        np.random.shuffle(test_index)
    return train_index, test_index

class SeriesDataset(Dataset):
    """3 dimension Acc or Gyr Dataset. Dynamically generate series data with index"""

    def __init__(self, datadict, datadict_idx, window_len, datatype:['acc','gyr','both'],
                 if_datatype_not_in:['acc','gyr','both']):
        if datatype not in ['acc', 'gyr', 'both']:
            print("datatype must be either 'acc' or 'gyr'")
            exit()
        if window_len <= 0:
            print("Window len must be a positive integer")
            exit()
        self.cls_cnt = len([i for i in datadict.keys() if i[:3] == 'acc'])
        if datatype == 'both':
            self.data = [(key[-1], pd.concat([datadict[key].iloc[:,1:], datadict['gyr'+str
            else:
            self.data = [(key[-1], datadict[key].iloc[:,1:].values) for key in sorted(dat
            self.index = datadict_idx
```

数据集划分代码

实验过程

数据集总结

经过以上处理，包含不对原始数据做预处理的组，在不考虑数据来源(acc, gyr, both)和序列长度的情况下，我一共得到了六种数据集合，分别是：

1. full_data
2. refull_data
3. butter_data
4. rebutter_data
5. wave_data
6. rewave_data

re 前缀代表进行了重采样，full 指代不进行任何降噪处理，wave 代表小波降噪，butter 为 Butterworth 低通滤波，此外，以上每个数据集都是一个字典，包含 acc, gyr 两种数据的原始序列，一会我们将在训练和测试过程中通过预划分好的时间窗索引动态地获取时间序列。

3.2 实验设计和模型选择

除了以上处理好的六种基本数据集之外，我们实际在进行训练的时候还需要进行时间按序列划分，需要选择到底用哪一类数据。就之前直观的数据观察结果来看，我在实验开始前经分析认为应该使用 cnn 架构并输入 acc/gyr 中的一种加降噪/resample 版本的数据可能会有比较好的效果，一则二者共用 both 模式时间戳不齐，二则此分类任务中预测时间序列可能会限定网络感受野，导致其无法捕获到序列中更全局的特征，三则小波变换是当前最好的非深度学习降噪方式。

基于以上思考优先选择了 cnn+acc+rebutter data/wave data 组合开始实验，并根据这个配置进一步探索序列长度对于模型性能的影响，得到合适的序列长度后，再进一步拓展实验对照范围，进行全量对照。序列长度我选择了五种长度 50, 100, 256, 512, 1024。

模型选择方面，除了前文已经提到的简单 1D cnn 结构，还设计了简单的双层 lstm 结构，考虑到数据集难度和数据量，这两个模型作为 baseline 都没有做很大的深度或者参数量，并且在后来的实验中这些 baseline 性能上都令人较为满意，从而侧面说明了在设计初期我对于参数规模和任务难度的估计比较准确。以下是 baseline 模型的结构，

1	summary(model.cuda(), (3, 1024))
---	----------------------------------

Layer (type)	Output Shape	Param #
Conv1d-1	[-1, 128, 1022]	1,280
BatchNorm1d-2	[-1, 128, 1022]	256
ReLU-3	[-1, 128, 1022]	0
MaxPool1d-4	[-1, 128, 511]	0
Conv1d-5	[-1, 256, 509]	98,560
BatchNorm1d-6	[-1, 256, 509]	512
ReLU-7	[-1, 256, 509]	0
AdaptiveAvgPool1d-8	[-1, 256, 10]	0
Linear-9	[-1, 512]	1,311,232
ReLU-10	[-1, 512]	0
Dropout-11	[-1, 512]	0
Linear-12	[-1, 10]	5,130

Total params:	1,416,970
Trainable params:	1,416,970
Non-trainable params:	0

Input size (MB):	0.01
Forward/backward pass size (MB):	6.51
Params size (MB):	5.41
Estimated Total Size (MB):	11.92

```
BaselineModel(  
  (features): Sequential(  
    (0): Conv1d(3, 128, kernel_size=(3,), stride=(1,))  
    (1): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (2): ReLU(inplace)  
    (3): MaxPool1d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (4): Conv1d(128, 256, kernel_size=(3,), stride=(1,))  
    (5): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (6): ReLU(inplace)  
    (7): AdaptiveAvgPool1d(output_size=10)  
  )  
  (classifier): Sequential(  
    (0): Linear(in_features=2560, out_features=512, bias=True)  
    (1): ReLU(inplace)  
    (2): Dropout(p=0.5)  
    (3): Linear(in_features=512, out_features=10, bias=True)  
  )  
)
```

Baseline CNN 模型的参数配置和输出形状

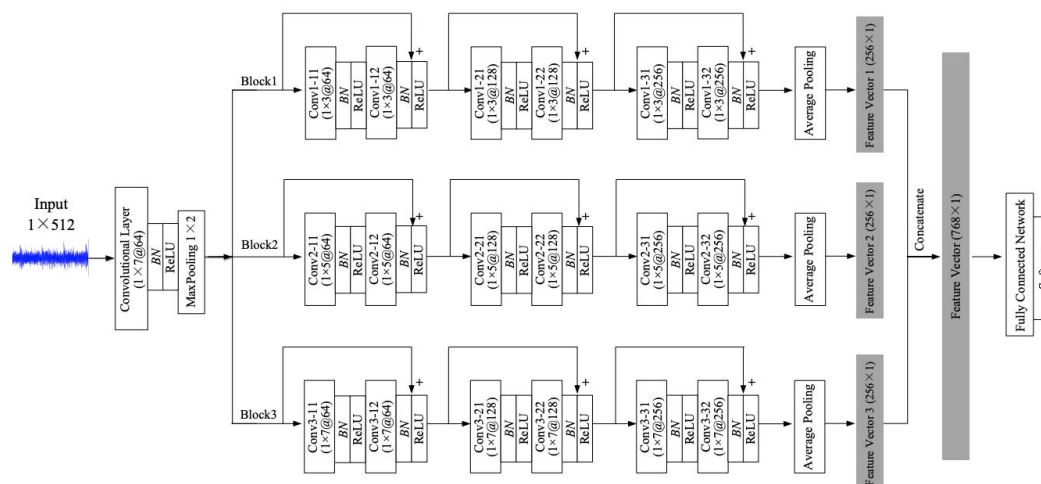
1	model
<pre>BaselineModel((features): Sequential((0): LSTM(3, 32, num_layers=2, batch_first=True)) (classifier): Sequential((0): Linear(in_features=640, out_features=10, bias=True)))</pre>	

Baseline LSTM 模型的参数配置

此处使用的 1D 卷积主要设计思路是通过卷积降噪，然后加入 maxpool 提取特征，第二层卷积通过压缩单通道信息提升信息维度尽可能丰富特征种类，最后做平均池化进一步降噪过滤时域相关的特征信号。中间还引入了 BN，最后 fc 层之前还做了 dropout 这些都是为了正则化减少过拟合，同时 BN 的加入也有很好地抵抗时间戳偏移的效果。而 Baseline 中 lstm 部分的设计就比较直接，采用了双层 lstm 核，最后接一个全连接层，中间层传递给下一层的是网络输出，而非隐状态，因为我们要预测的是序列种类而非生成序列故直接使用网络输出比较好。通过初步实验后，发现 cnn 的模型表现符合我的假设，同时观察到一般情况下序列长度越长网络性能和收敛性都越好，猜测出现这种现象的原因是人们确实在运动过程中充满了随机因素的干扰，从而导致短时间内人们的移动序列呈现出多种模式叠加的状态，也即一个人的步态从短时间内来看并没有很明显的区分度和个人特点，故而我们需要加长输入序列长度，

在 baseline 基础上，通过搜索相关资料，进一步引入了一个多尺度序列一维卷积残差网络(Multi-Scale ResNet 1D)。主要设计思路是当我们的输入序列逐渐变长的时候，网络对与序列中部分细节可能会变得不够敏感，但是单纯地使用短序列输入又提取不到全局的时域不变特征。此外不在 cnn 中加入自注意力机制（那样会需要更多的数据量）并且前文 lstm 效果并不好，因此为了在保留长时间依赖（也即时域不变的一些特征）的同时，引入更多不同尺度上的细节特征，考虑直接使用多路网络处理不同长度的序列，并按照 resnet 网络架构搭建分支网络，根据原始论文中的经验性设计结论，网络的最大深度决定了网络的性能上限，网络的最短路径决定了其收敛性，我们也在每一个分支中都引入了跨层连接，最短路径实际上反而比我们的 cnn baseline 还短一点，同时输出的最后一层 feature 维度也差不多数量接近。值得注意的是，虽然我们的深度更大，绝对参数体积更大，但是前传中数据复用率高，平均总的模型大小竟然比 baseline cnn 还少一点。（12MB 和 11MB）。

该网络结构图如下，



Multi-Scale ResNet1D 结构图

BatchNorm1d-74	[-1, 128, 23]	256
Conv1d-75	[-1, 128, 29]	8,192
BatchNorm1d-76	[-1, 128, 29]	256
ReLU-77	[-1, 128, 23]	0
BasicBlock7x7-78	[-1, 128, 23]	0
Conv1d-79	[-1, 256, 10]	229,376
BatchNorm1d-80	[-1, 256, 10]	512
ReLU-81	[-1, 256, 10]	0
Conv1d-82	[-1, 256, 6]	458,752
BatchNorm1d-83	[-1, 256, 6]	512
Conv1d-84	[-1, 256, 12]	32,768
BatchNorm1d-85	[-1, 256, 12]	512
ReLU-86	[-1, 256, 6]	0
BasicBlock7x7-87	[-1, 256, 6]	0
AvgPool1d-88	[-1, 256, 1]	0
Linear-89	[-1, 10]	7,690

Total params: 2,118,474

Trainable params: 2,118,474

Non-trainable params: 0

Input size (MB): 0.01

Forward/backward pass size (MB): 2.68

Params size (MB): 8.08

Estimated Total Size (MB): 10.77

Multi-Scale ResNet1D 参数量和输出形状

由于层数过多，此处就不再放出详细的参数列表，每层大小和参数列表请见文末附录及代码。

以下是最终形成的实验方案总结，

- Data Input
 - Test with different time window length
 - 20 with 95% overlap(stride=1) a total of $10 * ((30500-20)/1+1) = 304810$ groups of data
 - length = 50
 - length = 100
 - length = 256
 - length = 512
 - length = 1024
 - Test with different dataset type
 - acc
 - gyr
 - hybrid(both)
 - Test with different dataset preprocessing
 - no preprocessing
 - Butterworth Lowpass filter
 - Wavelet denoising
 - Resampled version of above three datasets
- CNN and LSTM baseline finetune
- Modified ResNet-50

实验关键模块说明

Index_Split

```
def index_split(data_length, window_len:int, stride=1, mode=2):
    """Split the data_index into train/test index set according to the given datatype"""
    if window_len <= 0:
        print("window len must be a positive integer")

    # series_length = (data_length-window_len)//stride + 1
    full_index = [i for i in range(0, data_length, stride) if i+window_len<=data_length]
    if mode==1:
        np.random.shuffle(full_index)
        train_index = full_index[:int(len(full_index)*0.8)]
        test_index = full_index[int(len(full_index)*0.8):]
    if mode == 2:
        np.random.shuffle(train_index)
        np.random.shuffle(test_index)
    return train_index, test_index
```

数据划分代码

此处有两种数据划分方式，一种是先 **shuffle** 然后在切割，但我很快发现这种划分方式如同数据泄露，因为序列之间的重合度太高了，划分之后测试集基本可以看作是一个训练集的子集。故而改进了采样方法，先对 **index** 进行划分这样保证了训练和测试数据从时间上完全隔绝，然后再分别 **shuffle** 保证了训练和测试集内部的随机性。

Series Dataset

```

class SeriesDataset(Dataset):
    """3 dimension Acc or Gyr Dataset. Dynamically generate series data with index"""

    def __init__(self, datadict, datadict_idx, window_len, datatype:['acc', 'gyr', 'both'], model_mode:['cnn', 'lstm'], transform=None):
        if datatype not in ['acc', 'gyr', 'both']:
            print("datatype must be either 'acc' or 'gyr'")
            exit()
        if window_len <= 0:
            print("Window len must be a positive integer")
            exit()
        self.cls_cnt = len([i for i in datadict.keys() if i[:3] == 'acc'])
        if datatype == 'both':
            self.data = [(key[-1], pd.concat([datadict[key].iloc[:,1:], datadict['gyr'+str(key[-1])].iloc[:,1:]], 1).values) for key in
            else:
                self.data = [(key[-1], datadict[key].iloc[:,1:].values) for key in sorted(datadict.keys()) if key[:3] == datatype]
        self.index = datadict_idx
        self.window = window_len

    def __len__(self):
        return self.cls_cnt*len(self.index)

    def __getitem__(self, idx):
        """Return:
        (data, target)
        """
        data_idx = self.index[idx//10]
        cls = idx%10
        if model_mode == 'cnn':
            sample = (self.data[cls][1][data_idx:data_idx + self.window].T.astype('float32'), cls)
            # (3, 20)
            sample = (sample[0][0].reshape([1, -1]), cls) # (1, 20)
        elif model_mode == 'lstm':
            sample = (self.data[cls][1][data_idx:data_idx + self.window].astype('float32'), cls)
            # (20, 3)
        else:
            sample = ([], cls)
        return sample

```

自定义动态序列数据集

这里为了方便大规模实验，实现了一个自定义的 **pytorch** 数据类。核心在于动态地根据前文 **index** 和输入的模型类别，要求的数据类别，时间窗长度返回对应的时间序列。值得注意的是 **both** 模式下我会将 3 通道 **acc** 和 3 通道 **gyr** 数据合并在一起得到 6 通道的数据，同时 **lstm** 和 **cnn** 的输入中 **channel** 的位置也不一样。

Config

```

window = 1024
dataset = full_data
data_mode = 'acc' # 'acc', 'gyr', 'both'
model_mode = 'lstm' # 'cnn', 'lstm', 'xl'
save_name = 'cnn_full2_acc1024_best.pth'
model = BaselineModel(model_mode, in_channel=3, num_classes=10)
# model2 = MSResNet(input_channel=3, layers=[1, 1, 1, 1], num_classes=10)
# model = BaselineModel(model_mode, num_classes=10)

```

模型配置

代码块中可以在此处直接配置你希望的实验，然后设定一个权重存储名称，最佳权重会被自动保存在 **./pth** 目录下。

结果和结论

对照实验结果

由于实验维度过多，我们此处将按照网络架构来分类性能对比表格。

Baseline CNN	full	refull	butter	rebutter	wave	rewave
Acc50	0.9761	0.8050	0.9762	1.0	0.9746	1.0
Acc100	0.9782	0.8078	0.9793	1.0	0.9771	1.0
Acc256	0.9906	0.8099	0.9929	1.0	0.9893	1.0
Acc512	0.9994	0.8160	0.9963	1.0	0.9964	1.0
Acc1024	0.9985	0.8099	0.9985	1.0	0.9970	1.0
Gyr50	0.9698	0.7956	0.9671	1.0	0.9655	1.0
Gyr100	0.9791	0.8035	0.9741	1.0	0.9747	1.0

Gyr256	0.9829	0.7996	0.9845	1.0	0.9876	1.0
Gyr512	0.9913	0.8044	0.9881	1.0	0.9948	1.0
Gyr1024	1.0	0.8033	1.0	1.0	1.0	1.0
Both50	0.9755	0.8044	0.9761	1.0	0.9753	1.0
Both100	0.9802	0.7985	0.9818	1.0	0.9812	1.0
Both256	0.9876	0.8008	0.9887	1.0	0.9893	1.0
Both512	0.9927	0.8026	0.9927	1.0	0.9917	1.0
Both1024	0.9973	0.8110	0.9962	1.0	0.9972	1.0

表 1 Baseline CNN 实验结果

Baseline LSTM	full	refull	butter	rebutter	wave	rewave
Acc50	0.9332	0.7927	0.9326	0.1	0.9235	0.1
Acc100	0.9540	0.7873	0.9520	0.1	0.9394	0.1
Acc256	0.9321	0.7884	0.9371	0.1	0.9429	0.1
Acc512	0.9298	0.8152	0.9241	0.1	0.9153	0.1
Acc1024	0.8812	0.7762	0.9324	0.1	0.9445	0.1
Gyr50	0.9281	0.7558	0.9246	0.1	0.9182	0.1
Gyr100	0.9412	0.7651	0.9273	0.1	0.9371	0.1
Gyr256	0.9328	0.7563	0.9329	0.1	0.9410	0.1
Gyr512	0.9570	0.8067	0.9519	0.1	0.9629	0.1
Gyr1024	0.9437	0.7972	0.9921	0.1	0.9585	0.1
Both50	0.9492	0.7879	0.9561	0.1	0.9513	0.1
Both100	0.9530	0.8172	0.9557	0.1	0.9524	0.1
Both256	0.9711	0.7965	0.9735	0.1	0.9509	0.1
Both512	0.9493	0.7908	0.9512	0.1	0.9715	0.1
Both1024	0.9534	0.8315	0.9421	0.1	0.9626	0.1

Baseline LSTM 实验结果

Multi-Scale ResNet1D	Full
Acc512	0.9927
Gyr512	0.9992
Both512	0.9920

Multi-Scale ResNet1D 实验结果

经验性总结和讨论

从上面的实验结果表格中我们得到了一些经验性的推论和发现。这些启发和规律不仅适用于这个实验，也适用于更加广泛的各种深度学习任务中。

参考资料

[1] 手机上到底有哪些传感器，它们都有什么作用呢？你知道吗？

[2] 利用 Python `scipy.signal.filtfilt()` 实现信号滤波

[3] Muhammad Muaaz, Rene Mayrhofer: Smartphone-Based Gait Recognition: From Authentication to Imitation. IEEE Trans. Mob. Comput. 16(11): 3209-3221 (2017)