第九届湖南省研究生数学建模竞赛承诺书

我们仔细阅读了湖南省高校研究生数学建模竞赛的竞赛规则。

我们完全明白，在竞赛开始后参赛队员不能以任何方式（包括电话、电子邮件、网上咨询等）与队外的任何人（包括指导教师）研究、讨论与赛题有关的问题。

我们知道，抄袭别人的成果是违反竞赛规则的，如果引用别人的成果或其他公开的资料（包括网上查到的资料），必须按照规定的参考文献的表述方式在正文引用处和参考文献中明确列出。

我们完全清楚，在竞赛中必须合法合规地使用文献资料和软件工具，不能有任何侵犯知识产权的行为。否则我们将失去评奖资格，并可能受到严肃处理。

我们郑重承诺，严格遵守竞赛规则，以保证竞赛的公正、公平性。如有违反竞赛规则的行为，我们将受到严肃处理。

我们授权湖南省研究生数学建模竞赛组委会，可将我们的论文以任何形式进行公开展示（包括进行网上公示，在书籍、期刊和其他媒体进行正式或非正式发表等）。

所属学校和学院（请填写完整的全名）：国防科技大学研究生院

参赛队员（打印后签名）：1.

2.

3.

指导教师或指导教师组负责人（打印后签名）：

日期：2024 年 7 月 3 日

（请勿改动此页内容和格式。以上内容请仔细核对，如填写错误，论文可能被取消评奖资格。）

第九届湖南省研究生数学建模竞赛

题　目：基于手机传感器数据的人体行为识别技术研究

摘要：基于智能手机传感器数据进行人体行为识别技术在健康监测、运动健身、智能家居等领域应用广泛，但面临数据复杂性、个体差异和隐私安全等挑战。

针对问题一:我们采用无监督数据分类方法 K-Means，并结合数据融合、频域分析、零偏误差补偿、异常数据处理及**小波阈值去噪**等预处理步骤，确保数据的准确性和一致性。通过引入**傅里叶变换**的频域信息，捕捉到时域中难以观察到的特征，提升了分类的有效性。在模型建立方面，优化的 **K-Means++**方法用于初始质心选择，并引入带约束的 K-Means 算法，以保证每类样本数量均衡。最终，该方法有效地将数据分成 **12** 类，提高了分类准确率，满足了问题需求。

针对问题二:问题二要求对比问题一中的分类模型与问题二中建立的判别模型的结果，并利用有标签数据训练的判别模型对 30 组无标签数据进行分类。我们设计了一种基于动态图神经网络（**DGNN**）的多元时间序列分类模型，以捕捉时间和空间上的依赖关系及不同维度之间的动态关联。模型包括时间序列图构建、动态图神经网络、动态图池化及输出模块。对有标签数据的训练结果显示，DGNN 模型在测试集上的判别准确率达到了 **93.83%**，显著高于 K-Means 方法的 **76.81%**。在对 30 组无标签数据进行分类时，DGNN 模型根据训练好的参数输出了各活动状态的预测标签。整体来看，DGNN 模型在处理多元时间序列分类任务上表现优异，能够有效捕捉复杂的时空特征，提高了分类的准确性和鲁棒性。

针对问题三:针对问题三中的具体任务，我们对于如何判断活动状态与人员特征数据间的**相关性**，由于这种相关性是非显式的，且活动状态的数据包含 6 个基础特征维度和异步数据，因此需要基于数据本身进行特征统一和数据压缩。随后，通过计算不同自变量（即活动状态特征数据）与因变量（即人员特征数据）间的 **Pearson 相关性系数**，从而对具体活动和人员特征进行相关性分析。最后，为实现基于活动状态数据对人员的画像，此时数据类型标签从活动状态类型转变为人员类型，进一步根据更新后的数据类型标签对模型进行针对性设计和训练，以满足人员类型判别的需要。

关键词：人体行为识别 K-Means++ 动态图神经网络 多元时间序列分类 相关性系数

目录

1 问题综述

1.1 问题背景

近年来，随着智能手机的普及和技术的不断进步，基于智能手机传感器数据进行人体行为识别的研究逐渐成为一个热门领域。智能手机内置了多种传感器，如加速度计、陀螺仪、磁力计、GPS 等，这些传感器能够实时捕捉用户的运动和位置数据。通过对这些数据的分析，可以识别出用户的各种行为，如步行、跑步、骑车、上下楼梯等。人体行为识别技术在众多领域中有着广泛的应用，例如健康监测、运动健身、智能家居和安防系统等。

人体行为识别技术在众多领域中有着广泛的应用。例如，在健康监测方面，利用智能手机传感器数据可以实时监测用户的日常活动，从而帮助医生更好地了解患者的健康状况，并为疾病的预防和康复提供数据支持。例如，通过长期监测用户的活动模式，可以早期发现帕金森病、抑郁症等疾病的症状。此外，对于老年人和慢性病患者，通过行为识别技术可以实现实时监控，及时发现异常行为，如跌倒等，从而快速采取相应措施，保障用户的安全。在运动和健身领域，行为识别技术可以为用户提供个性化的运动建议和反馈，帮助用户更科学地进行锻炼。通过分析用户的运动数据，可以评估其运动强度、卡路里消耗等指标，进而提供个性化的训练计划和建议。例如，通过识别用户的跑步姿态，可以纠正不正确的跑步姿势，减少运动损伤的风险。此外，在团队运动中，通过行为识别技术可以监测和分析队员的运动表现，从而优化训练策略，提高整体运动水平。在智能家居和安防系统中，行为识别技术也能够提高系统的智能化程度，为用户提供更便捷和安全的生活环境。例如，通过识别用户的日常活动模式，可以实现智能家居设备的自动化控制，如根据用户的作息时间自动调节室内温度和照明。对于安防系统，通过行为识别技术可以检测异常行为，如入侵、火灾等，从而及时发出警报，保障用户的安全。

尽管基于智能手机传感器的数据进行人体行为识别具有诸多优势，但仍面临一些挑战。首先，传感器数据受到多种因素的影响，如手机的放置位置、用户的个体差异等，导致数据的复杂性和多样性增加。不同的用户在相同的行为下可能产生不同的传感器数据，如何建立泛化能力强的模型是一个重要的研究课题。其次，如何高效地处理和分析大量的传感器数据，并从中提取出有用的特征信息，是另一个重要的研究方向。传统的特征提取方法可能无法充分利用高维的传感器数据，深度学习等先进的机器学习技术在这一领域展现出了巨大的潜力。此外，隐私和数据安全问题也是需要重点考虑的方面。智能手机传感器数据可能包含用户的敏感信息，如位置、行为习惯等，如何在保证用户隐私的前提下进行数据收集和分析，是行为识别技术应用的关键。当前，隐私保护技术如差分隐私、联邦学习等在这一领域的应用研究较为活跃，这些技术能够在保护用户隐私的同时，提升数据分析的准确性和可靠性。

目前，国内外学者已经开展了大量基于传统机器学习方法的人体活动识别研究。比如，Jennifer 等人利用手机加速度计数据进行活动识别，通过收集 29 名用户在日常活动中的加速度计数据，构建预测模型，实现对用户运动状态的自动识别，应用范围广泛，包括移动设备行为的自动定制和用户活动健康状况的分析。从实验结果来看，他们的模型对于上下楼的判断不是准确。Brezmes 等人实现了基于普通手机加速度计的实时人类运动分类系统，验证了无需服务器处理的去中心化监测方法的可行性，为低成本开发新应用提供了可能。伍俊杰基于智能手机传感器，采用隐马尔可夫模型算法实现了非特定人的行为识别系统，并通过数据滤波和算法融合解决了传感器噪音和 HMM 算法的收敛问题。周林等人提出了一种基于多传感器的人体行为识别系统，通过滑动时间窗内的数

据特征提取和基于决策树的两层分类算法，实现了对 8 种常见行为的高效识别，并在降低能耗的同时达到了 93.12%的平均识别率。范琳等人基于智能手机传感器数据，提取多种行为特征并构建三种决策树分类模型，通过优选特征实现了对不同手机位置的用户行为有效识别，其中行为模型在混合样本上的识别准确率达到了 80.29%。

随着深度学习技术的迅猛发展，传统机器学习方法在人体行为识别领域的局限性逐渐暴露。首先，传统方法依赖于复杂的特征工程，需要大量的专业知识和实验来提取有效特征，这不仅耗时且复杂。其次，手工提取的特征往往无法充分表达复杂的行为模式，限制了模型的表现。此外，传统方法对大规模标注数据的依赖性较强，数据标注成本高，且泛化能力较弱，难以应对多样化的数据和环境变化。实时性也是一个重要问题，许多传统方法在处理实时数据时表现不佳，无法满足实际应用中的需求。

为了解决这些问题，研究者们开始探索深度学习技术在人体行为识别中的应用。深度学习方法，如卷积神经网络（CNN）和长短时记忆网络（LSTM），能够自动提取特征，提高识别精度和效率。数据增强、多模态数据融合以及迁移学习等技术进一步提升了模型的泛化能力和实用性。通过引入这些先进技术，人体行为识别领域迎来了新的发展机遇，克服了传统方法的诸多不足，推动了研究的深入和应用的广泛。Vepakomma 等人提出了 A-Wristocracy，一个利用手腕佩戴设备传感技术识别细粒度和复杂家庭活动的新框架，通过深度学习和多模态传感，实现了对 22 种复杂日常活动的高精度识别，显著优于现有的可穿戴设备方法，并通过少量蓝牙信标提供粗略位置信息，保护用户隐私。Hammerla 等人深入探讨了深度学习、卷积和循环神经网络在三个代表性可穿戴传感器数据集上的应用，通过引入新颖的正则化方法并优化超参数，显著提升了人类活动识别的准确性，提供了适用于不同任务的深度学习模型选择指南。Yao 等人提出了 DeepSense，一个结合卷积和循环神经网络的深度学习框架，通过统一解决传感器噪声和特征定制问题，大幅提升了车辆跟踪、人类活动识别和用户身份认证等任务的准确性，并且在能耗和延迟方面适合智能手机应用。Ordóñez 等人提出了一种基于卷积和 LSTM 单元的通用深度学习框架，用于人类活动识别任务，通过自动特征提取和时间动态建模，显著提升了多模态可穿戴传感器数据的识别精度，并在多个数据集上取得了优于非递归网络的效果。

在人体行为识别的研究中，传统的神经网络模型如 DNN、CNN 和 RNN 各自都面临一些挑战。DNN 模型通常通过增加隐藏层来提高性能，但这也导致了建模复杂性和训练时间的增加。RNN 虽然能处理时间序列数据，但其训练难以并行化，导致学习速度慢且资源消耗大。CNN 需要将数据转换为类似图像的形式，这增加了算法开发的难度。此外，神经网络模型对数据质量要求较高，需要进行良好的数据预处理。针对这些问题，我提出了基于动态图神经网络的方法。动态图神经网络能够动态建模和实时特征提取，既解决了传统模型在数据处理和训练方面的困难，又提高了人体行为识别的准确性和效率。这种方法不仅简化了数据预处理过程，还能更有效地利用计算资源，展现出显著的优势。

1.2 问题提出

基于手机传感器数据的人体行动识别任务涉及多方面的问题，首先我们需要对获取到的传感器数据进行各种数据预处理，使其满足我们模型使用需求。而后对于没有记录运动状态且没有标签数据集进行训练的问题，我们只能进行聚类模型将其分类为需要的类别。接下来，对于包含标签的传感器数据集，我们可以通过训练一个深度学习模型用于分类。最后对于不同人员可能会对不同的活动状态数据有不同的影响，我们需要先对个体身高体重和年龄与各类行动数据进行相关性的实验和分析，之后通过也是用第二问

的深度学习模型，但是标签由 12 类活动类型换成了人员编号，并将每个人的身高体重和年龄信息进行处理，融合到输入信息中，从而训练出一个人员识别模型。

具体的来说，问题就是以下三个：

(1) 问题 1：根据 3 名实验人员的 60 组加速度计和陀螺仪的活动状态数据，将其分为 12 类，每类有 5 组数据。
(2) 问题 2：根据 10 名实验人员的 600 组加速度计和陀螺仪的活动状态数据，第二问的数据包含了活动标签，提取 12 类人员活动状态的典型特征，建立人员活动状态的判别模型。跟第一问提出的分类模型进行比较，分析两者分类的准确性差异，而后通过本问中建立的判别模型对没有标签的 30 次活动状态进行判断。
(3) 问题 3：根据前两问参与实验的 13 位实验人员的年龄、身高、体重等数据，分析不同人员的同一活动状态是否存在差异，并建立一个人员识别模型，判别 50 组没有人员标签的数据属于问题二的哪个实验人员。

1.3 资料条件

附件 1 中有 3 名实验人员的运动数据，包含每名实验人员每种活动状态的 5 组加速度计和陀螺仪数据，但实验时未记录数据所代表的活动状态。附件 2 中有 10 名实验人员的活动数据，包含每位实验人员每种活动状态的 5 组加速度计和陀螺仪数据，但实验时记录了每组数据所代表的实验人员的活动状态附件 3 中收集有某实验人员 30 次活动的状态数据附件 4 给出了问题 1 和问题 2 中参与实验的 13 位实验人员的年龄、身高、体重等数据附件 5 中给出了问题 2 的 10 位实验人员中的 5 位的某次活动数据，数据包含了每人的 12 类活动状态

2 模型假设与符号说明

2.1 模型基本假设

(1) 假设不同实验人员进行相同活动时，其数据模式也具有一定的相似性，尽管可能存在个体差异。
(2) 假设每种活动是独立进行的，即在一种活动进行过程中，没有其他活动的干扰或重叠。
(3) 假设每种活动状态的典型特征在不同的实验人员之间是稳定且可识别的。
(4) 假设所有实验人员的运动传感器安装位置和方式一致，数据采集过程无误差，且传感器性能一致。
(5) 假设实验环境对所有实验人员都是一致的，没有外部环境因素对数据采集造成显著影响。
(6) 假设所有实验人员严格按照实验要求完成各项活动，没有人为的误差和偏差。

2.2 符号说明

本文定义了如下使用次数较多的符号，并对问题主要相关概念进行说明，其余符号在使用时注明.

表1 符号说明

符号	含义	单位
X	时间序列数据	由场景决定
N	时间序列数据的变量维度	无

符号	含义	单位
t	时间步	由场景决定
T	观测时长	由场景决定
R	实数集	无
G	图结构	无
V	点集合	无
E	边集合	无
A	邻接矩阵	无

多元时间序列：多元时间序列是指一个时间序列数据集包含多个观测变量，并且这些变量之间具有相关性，通常由同一个系统生成。在本文中，将多元时间序列定义为 $X \in R^{N \times T}$，其中 N 表示观测变量的数量，T 表示序列的长度；将第 i 个变量对应的时间序列记为 $X_i \in R^T$；令 $x_i^t \in R$ 表示第 i 个序列中第 t 个时间步长的观测值；将所有变量在第 t 个时间步长的观测值定义为 $x_g^t \in R^{N \times 1}$。

图结构数据：图结构数据也称为网络数据，通常定义为 $G = \{V, E, A\}$。其中 $V = \{v_1, v_2, ..., v_N\}$ 表示图中的节点集，$|V| = N$ 表示图结构数据 G 包含 N 个节点。E 表示图中的连边集合，$e_{ij} = (v_i, v_j) \in E$ 表示节点 v_i 和 v_j 之间存在连接关系。$A \in R^{N \times N}$ 表示图 G 对应的邻接矩阵，A_{ij} 表示邻接矩阵中第 i 行第 j 列的元素，它也表示节点 v_i 和节点 v_j 之间的关联关系。当 $A_{ij} \in \{0,1\}$ 则表示图 G 为无权图，此时 $A_{ij} = 1$ 表示节点 v_i 和节点 v_j 之间存在连边关系，$A_{ij} = 0$ 则表示节点 v_i 和节点 v_j 之间不存在连边。当 $A_{ij} \in R$ 则表示 G 为加权图，此时 A_{ij} 的数值大小表示节点 v_i 和节点 v_j 之间连边的强弱关系。当 A 为对称矩阵时，则表示图 G 是无向图，否则图 G 有向图。

3 数据预处理

在进行人体行为识别的研究中，数据预处理是确保模型性能的关键步骤。通过对传感器数据进行去噪、归一化和滑动窗口处理，可以提高数据的质量，增强模型的鲁棒性和准确性。

3.1 数据去噪

传感器数据在采集过程中容易受到环境噪声和设备误差的影响，导致数据质量下降。为了提高输入数据的准确性，首先需要对数据进行去噪处理。常用的方法包括低通滤波、高通滤波和中值滤波。

- 低通滤波：低通滤波器可以平滑数据，去除高频噪声。典型的低通滤波器包括移动平均滤波器和 Butterworth 滤波器。移动平均滤波器通过取一段时间窗口内数据的平均值来平滑数据，而 Butterworth 滤波器则可以在频域内设计滤波器的通带和阻带特性。
- 高通滤波：高通滤波器用于去除低频噪声，保留高频信号特征。高通滤波通常用于消除传感器信号中的漂移现象。
- 中值滤波：中值滤波是一种非线性滤波方法，通过取局部窗口内数据的中值来去除尖锐的噪声点，非常适合处理包含尖峰噪声的数据。

本研究采用了中值滤波方法，有效去除了数据中的离散噪声，保留了有用的信号特征。

3.2 数据归一化

由于不同维度的传感器数据可能具有不同的量纲和范围，为了消除量纲对模型训练的影响，需要对数据进行归一化处理。归一化可以将数据缩放到相同的范围内，使得不同特征具有相同的尺度，常用的方法有最小-最大归一化和标准化。

- 最小-最大归一化：将数据缩放到[0, 1]区间内，可以有效消除量纲差异

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

其中，x是原始数据，x'是归一化后的数据，$x_{\min}$和$x_{\max}$分别是数据的最小值和最大值。

- 标准化：通过减去均值并除以标准差，将数据转化为零均值和单位方差的分布。公式如下：

$$x' = \frac{x - \mu}{\sigma}$$

其中，μ是数据的均值，σ是数据的标准差。标准化可以有效消除不同特征之间的量纲差异，同时克服异常值的影响。

在本研究中，采用了最小-最大归一化方法，将数据缩放到[0, 1]区间内，以确保不同特征之间的可比性。

3.3 滑动窗口处理

传感器数据通常是时间序列数据，为了捕捉时间维度上的动态特征，本研究采用了滑动窗口处理方法。滑动窗口处理可以将时间序列数据划分为若干个固定长度的窗口，每个窗口作为一个独立的样本输入到模型中。滑动窗口的大小和步长是影响模型性能的重要参数。

窗口大小（Window Size, W）：窗口大小决定了每个样本包含的时间步数。较大的窗口可以捕捉更长时间范围内的动态变化，但也增加了计算复杂度。较小的窗口则可以更敏捷地反应短期变化。

步长（Stride, S）：步长决定了窗口移动的步数。较小的步长可以生成更多样本，有助于提高模型的训练数据量，但也增加了冗余信息。较大的步长可以减少冗余，但可能会忽略一些细节特征。

为了进一步提高数据处理效果，本研究使用了重叠滑动窗口方法，并对每个窗口内的数据进行均值处理。重叠滑动窗口已在实验中证明能够取得更好的识别精度，并增加活动分割次数，提高深度学习的泛化能力。

具体的滑动窗口处理算法如下：

1. 初始化窗口起始位置 *t=0*：从时间序列的起始位置开始。

2. 提取窗口数据：提取从 *t* 开始，长度为 *W* 的数据窗口。

3. 均值处理：对提取的数据窗口进行均值处理，将窗口内的数据进行平均，生成一个粗粒化的数据点。

4. 存储样本：将均值处理后的数据点作为一个独立样本存储。

5. 移动窗口：将窗口起始位置向前移动 *S* 个时间步，即 *t=t+S*。为了实现重叠滑动，通常 *S<W*。

6. 重复处理：重复步骤 2-5，直到处理完所有数据。

通过滑动窗口处理，可以将原始的时间序列数据转换为适合模型输入的样本集，从而更好地捕捉时间序列中的动态变化特征。这种处理方式在行为识别任务中尤其重要，因为人体行为通常具有时间上的连续性和依赖性。重叠滑动窗口方法和均值处理的结合，不仅增强了模型对短期和长期特征的捕捉能力，还能提高模型的鲁棒性和泛化能力，从而在行为识别等任务中取得更好的性能。

4 问题一分析与模型建立

4.1 问题一分析

题目以智能手机健康检测为背景,介绍了数据主要类别及检测的 12 种活动状态,问题一要求我们使用陀螺仪与加速度计数据，对附件 1 提供的活动数据进行分类，并将分类结果填写到表中。该问题可以看作双传感器时序数据多分类问题，主要难点在于数据维度高，数据量大且分类种类多。有文献表明，在分类算法上引入了 D-S 证据理论，再利用数据融合的思想识别行为，可以将识别准确率提高到 90%以上。此外采用主成分分析方法对数据统计特征进行降维处理，并将决策树与分类算法相结合可以将行为识别平均准确率提升至 97.5%。

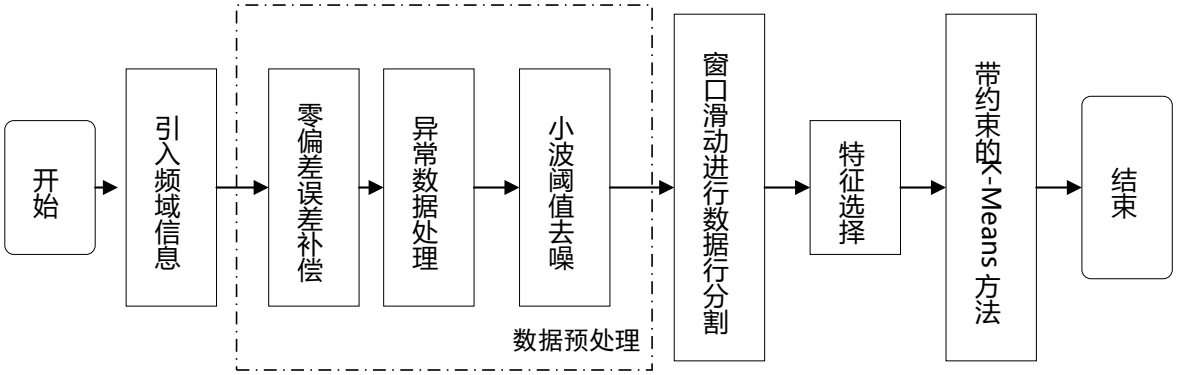

图 1 问题一方法流程图

基于问题描述及相关文献，我们考虑采用无监督数据分类方法即 K-Means 对数据进行有效分类，图 1 展示了我们的算法流程。具体操作如下：在时域数据基础上，通过 FFT 算法引入数据的频域信息，以引入更加多元的信息及更加全面的特征。为较好的对数据进行分析及处理，需要对数据进行融合处理。我们选择将加速度计与陀螺仪结合在，对采集到的数据进行预处理，具体包括使用零偏误差补偿、基于箱型图和均值修正法的异常数据处理、以及基于 SURE 阈值的软阈值去噪。采用固定长度及重叠率的滑动窗口对数据进行处理，将处理后的数据进行划分，将窗口数据提取特征用于描述行为，对提取的特征进行筛选以减少冗余特征影响。最后将特征选择后形成的新数据集用于分类器模型分类，为尽可能确保每个类样本数量尽可能均衡，我们选择使用带约束的 K-Means 方法进行分类，以得到尽可能接近原始样本数据的分类结果。

4.2 数据处理方法

数据质量决定了分类方法的上限，好的算法模型在不断地逼近这个上限。在实验过程中，可能因为传感器本身或其他因素引入数据噪声，清理数据噪声对分类准确率的提升有着较好的效果。此外，为对数据有着全面了解，可以对对传感器获得的数据进行统计性描述，以获得有价值的数据处理方案，为数据预处理提供有效的指导。我们结合文献及相关资料，主要进行了以下数据预处理：

4.2.1 时域—频域变换分析

傅里叶变换是时域—频域变换分析中最基本的方法之一，离散傅里叶变换是针对离散数据进行处理的一种傅里叶变换方法，其将时域信号的采样变换为在离散时间傅里叶变换频域的采样。具体来说离散傅里叶变换的运算为：

$$X(k) = DFT[x(n)] = \sum_{n=0}^{N-1} x(n) W_N^{kn}, k = 0,1,\cdots,N-1.$$

$$x(n) = IDFT[X(k)] = \sum_{k=0}^{N-1} X(k) W_N^{-kn}, n = 0,1,\cdots,N-1.$$

式中$W_N = e^{-j\frac{2\pi}{N}}$，改进 DFT 算法利用 DFT 中的$W_N = e^{-j\frac{2\pi}{N}}$周期性和对称性，使整个 DFT 的计算变成一系列迭代运算，大幅度提高了运算过程和运算量。经过傅里叶变换后，可以得到其频域数据，为将该数据与时域数据进行对齐，减少复数带来的影响，对频域数据求绝对值。

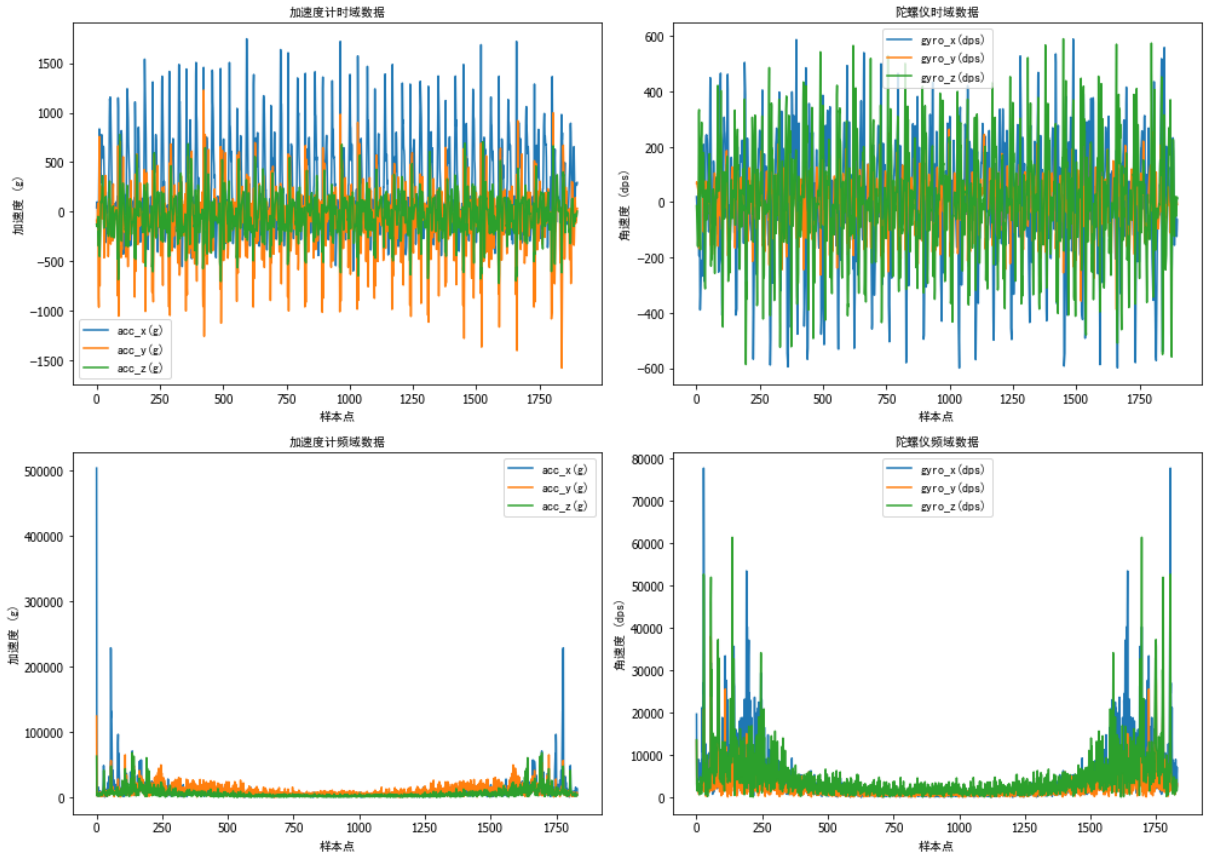

图 2 时域频域数据对比图

通过引入频域数据，可以清晰的看到时域中难以捕捉的信息，不同类别的数据，可能在时域上是相似的，如对于坐下与躺下的行为模式的区分，在加速度和陀螺仪上都是规律性的上下起伏数据，但在频域上存在着差距，其体现在高低频的峰值的大小及范围上，通过对频域进行处理，捕捉其区域分布范围与峰值大小，就可以很容易引入时间特征到分类中，为后续的任务提供了基础。

4.2.2 数据预处理

在进行手机传感器数据的行为识别分析之前，我们首先对加速度计和角加速度计收集到的数据执行了一系列关键的预处理步骤。这包括零偏误差补偿，通过在传感器启动

初期采集静态数据并计算平均值来校正常值零偏和零偏稳定性，此外，通过处理数据中的异常值，确保数据集的一致性和准确性。为去除噪声并保留有用信号，我们应用了小波阈值去噪技术进行处理，通过这些综合的数据预处理方法，我们为行为识别分析提供了准确和可靠的数据基础。具体方法介绍如下：

零偏误差补偿：在行为识别领域，传感器的精度需求通常不会特别高。因此，本文对传感器的零偏误差进行了一个较为简单的估算。零偏误差主要包括三个方面：恒定零偏、零偏的稳定性和零偏的不稳定性。对零偏误差的校正过程分为两个阶段：在传感器启动初期，通过收集数秒的静态数据并计算其平均值，以此来校正恒定零偏和零偏的稳定性；采用 N 秒平均法来评估传感器的零偏不稳定性。这种方法涉及到在较长时间内收集静态数据。通过这些平均值来计算它们的平均数和标准差，从而评估零偏的不稳定性。

表2 静态数据求零偏不稳定性测量结果

坐标轴	加速度计/（m/s^2）	陀螺仪/((°)/s)
X	0.0731	-0.0023
Y	-0.0059	0.0017
Z	0.0781	0.0029

表3 N 秒平均法求零偏不稳定性测量结果

坐标轴	加速度计/（m/s^2）	陀螺仪/((°)/s)
X	0.0017	9.3125
Y	0.2101	-3.1234
Z	-0.0015	10.1247

异常数据处理：本文对异常值的处理采用均值修正法，利用原异常值所在位置的左右各两个数据的平均值来替换该异常值。但在处理过程中,需要注意判断数据是否越界。当进行异常值处理时，先对去除零偏误差后的数据采用箱型图检测出异常值所在位置。若异常值所在位置的左右各两个值均在原数据可检索范围内,则正常进行均值修正。若在寻找异常值相邻数据时，左边或右边的相邻数据索引超出了原数据序列长度，此时只选择可检索到的数据求平均值来替换该异常值。经处理后异常值数量明显减少。

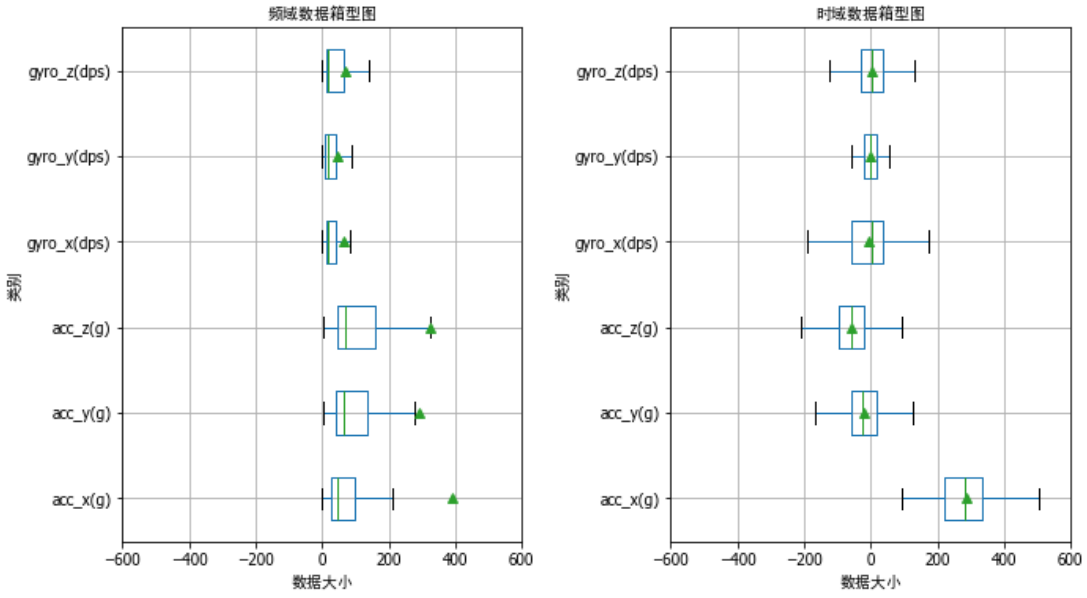

图 3 时域频域数据箱型图

小波阈值去噪：对于原始数据中存在的高频噪声，本文研究采用小波阈值去噪方法加以滤波，该方法不仅能够剔除信号中的噪声，而且可以尽可能地保留原始数据的特征。小波阈值去噪方法的主要流程如下：①选择一种小波基函数并确定分解层次；②对含有噪声的数据进行小波变换，以获得不同尺度下的小波系数；③选择合适的阈值，将噪声

产生的小波系数删除，只保留真实信号所产生的系数；④对处理后的小波系数利用小波逆变换实现信号重构，以获得去噪后的信号。在以上过程中，每个部分的设计均会直接影响对数据的去噪效果。

4.3 分类模型建立

问题一数据均为无标签数据，在分类模型选择上受到了巨大限制，诸如 SVM、随机森林等一系列机器学习方法都很难发挥作用，因此只能选择 K-Means 方法进行分类。传统 K-Means 方法能够实现较好的聚类效果。但对于问题一，最重要的一点在于，已经明确数据有 12 类，且每一类有 5 个样本，因此相当于明确了簇中样本数，在传统 K-Means 基础上，我们使用 K-Means++方法优化初始聚类中心，并结合动态数据池对其进行约束，最终建立适合本问题的约束 K-Means 方法，通过此算法，可以将问题一中数据均匀分成 12 类，且确保每一类中的样本数均为 5，满足了问题需求，提高了分类准确率。

4.3.1 K-means 聚类算法

（1）K-Means 算法基本介绍

K-Means 算法是无监督的聚类算法，它实现起来比较简单，聚类效果好，应用广泛。传统的 K-Means 算法可以实现数据的分类，K-Means 算法的思想很简单，对于给定的样本集，按照样本之间的距离大小，将样本集划分为 K 个簇。让簇内的点尽量紧密的连在一起，而让簇间的距离尽量的大。如果用数据表达式表示，假设簇划分为$(C_1, C_2, \ldots, C_k)$，则我们的目标是最小化平方误差E：

$$E = \sum_{i=1}^{k} \sum_{x \in C_i} ||x - \mu_i||_2^2$$

其中μ_i是簇C_i的均值向量，有时也称为质心，表达式为：

$$\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$$

直接求上式的最小值并不容易，这是一个 NP 难的问题，因此只能采用启发式的迭代方法。K-Means 采用的启发式方式很简单，用下图可形象化描述。

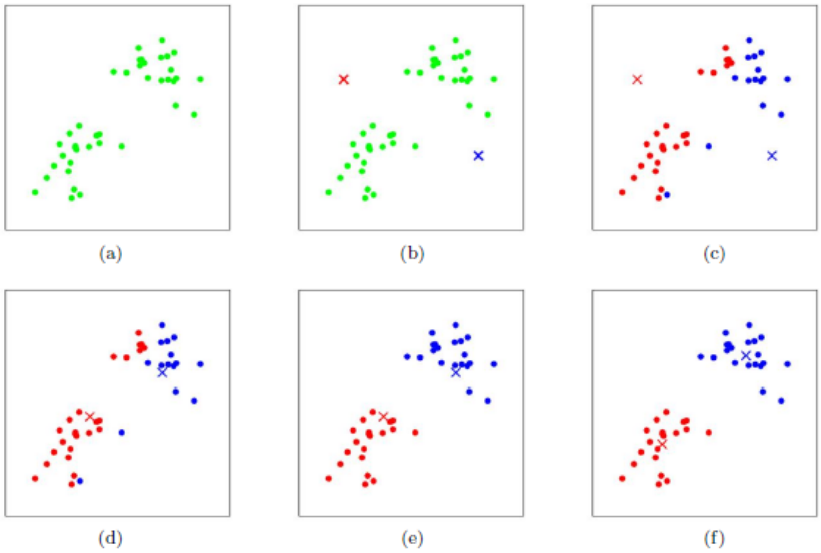

图 4 K-Means 数据分布状态示意图

上图 a 表达了初始的数据集，假设 k=2。在图 b 中，我们随机选择了两个 k 类所对应的类别质心，即图中的红色质心和蓝色质心，然后分别求样本中所有点到这两个质心的距离，并标记每个样本的类别为和该样本距离最小的质心的类别，如图 c 所示，经过计算样本和红色质心和蓝色质心的距离，我们得到了所有样本点的第一轮迭代后的类别。此时对当前标记为红色和蓝色的点分别求其新的质心，如图所示，新的红色质心和蓝色质心的位置已经发生了变动。图 e 和图 f 重复了我们在图 c 和图 d 的过程，即将所有点的类别标记为距离最近的质心的类别并求新的质心。最终得到的两个类别如图 f。

（2）K-Means 算法流程

对于 K-Means 算法，首先要注意的是 k 值的选择，一般来说，我们会根据对数据的先验经验选择一个合适的 k 值，如果没有什么先验知识，则可以通过交叉验证选择一个合适的 k 值。在确定了 k 的个数后，我们需要选择 k 个初始化的质心，就像上图 b 中的随机质心。由于我们是启发式方法，k 个初始化的质心的位置选择对最后的聚类结果和运行时间都有很大的影响，因此需要选择合适的 k 个质心，最好这些质心不能太近。我们总结 K-Means 算法流程：

输入是样本集$D = \{x_1, x_2, \dots x_m\}$,聚类的簇树$k$,最大迭代次数$N$，输出是簇划分$C = \{C_1, C_2, \dots C_k\}$，具体如下：

1) 从数据集D中随机选择k个样本作为初始的k个质心向量：$\{u_1, u_2, \dots u_k\}$
2) 对于$n = 1,2, \dots, N$
   a) 将簇划分C初始化为$Ct = \emptyset, t = 1,2 \dots k$
   b) 对于$i = 1,2 \dots m$,计算样本x_i和各个质心向量$\mu_j (j = 1,2, \dots k)$的距离：$d_{ij} = ||xi - \mu j||_2^2$，将$x_i$标记最小的为$d_{ij}$所对应的类别$\lambda_i$，更新$C_{\lambda_i} = C_{\lambda_i} \cup \{x_i\}$
   c) 对于$j = 1,2, \dots, k$,对Cj中所有的样本点重新计算新的质心$\mu_j = \frac{1}{|C_j|} \sum_{x \in C_j} x$
   d) 如果所有的k个质心向量都没有发生变化，则转到步骤 3）
3) 输出簇划分$C = \{C_1, C_2, \dots C_k\}$

（3）初始化优化 K-Means++算法

k 个初始化的质心的位置选择对最后的聚类结果和运行时间都有很大的影响，因此需要选择合适的 k 个质心。如果仅仅是完全随机的选择，有可能导致算法收敛很慢。K-Means++算法就是对 K-Means 随机初始化质心的方法的优化。

K-Means++的对于初始化质心的优化策略也很简单，如下：

a) 从输入的数据点集合中随机选择一个点作为第一个聚类中心μ_1
b) 对于数据集中的每一个点xi，计算它与已选择的聚类中心中最近聚类中心的距离$D(x_i) = arg\ min||x_i - \mu_r||_2^2\ r = 1,2, \dots k_{selected}$
c) 选择一个新的数据点作为新的聚类中心，选择的原则是：$D(x)$较大的点，被选取作为聚类中心的概率较大
d) 重复 b 和 c 直到选择出 k 个聚类质心
e) 利用这k个质心来作为初始化质心去运行标准的 K-Means 算法

4.3.2 带约束的 K-means 聚类算法

该题对每一簇的聚类样本点数量有限制，每一簇存在满足约束的最大能力，这时候应该怎么添加约束，并且不破坏聚类的效果。这里设置一个目标——在满足需求的情况下最小聚类样本数量，具体来说，其算法为：

表4 带约束的 K-means 聚类算法图

1	算法输入：K-簇的数目，D-数据集；
2	算法流程： 1）初始化：从 D 中随机选择 K 个对象作为初始簇中心； 2）重复以下操作操作； 3）将每个对象分配到最相似的簇中； 4）重新计算每个簇中对象的均值，作为新的簇中心； 5）直到 K 个簇的中心不再发生变化
3	算法输出：K 个簇的集合。
4	算法说明：由于存在需求约束，那么就存在最小的 K，可以从最小值 K 开始遍历，约束加在 3）中，将对象分配到最相似的簇中，

具体修改为：在满足约束的情况下，将对象分配到最相似的簇中，如果最相似的簇无法容纳，就找第二相似的簇，依次类推。在遍历中，对遍历对象进行修改，通常 K-means 是随机/按列表顺序遍历对象，将对象分配到最相似的簇，这里改进为：取所有对象中距离某个簇最近的点，优先分配（这算是随机/按列表顺序遍历对象的一个特殊情况）。为了减小 K-means 对初始解敏感的问题，这里采用多次求解的方式保留最优的方式降低对初始解的依赖。聚类效果使用轮廓系数进行评价。

4.4 结果分析

表5 问题一结果

分类	Person1	Person2	Person3
第 1 类	SY7 SY33 SY38 SY42 SY58	SY2 SY10 SY12 SY26 SY47	SY13 SY27 SY29 SY51 SY56
第 2 类	SY14 SY17 SY28 SY41 SY47	SY16 SY21 SY29 SY40 SY43	SY9 SY14 SY30 SY37 SY43
第 3 类	SY5 SY8 SY46 SY56 SY59	SY3 SY20 SY38 SY51 SY60	SY8 SY23 SY39 SY42 SY52

第 4 类	SY21 SY25 SY29 SY48 SY53	SY19 SY23 SY30 SY37 SY46	SY11 SY25 SY31 SY44 SY60
第 5 类	SY27 SY31 SY37 SY49 SY52	SY7 SY24 SY33 SY44 SY57	SY19 SY24 SY35 SY36 SY49
第 6 类	SY19 SY26 SY43 SY44 SY50	SY1 SY4 SY48 SY49 SY50	SY18 SY22 SY57 SY58 SY59
第 7 类	SY1 SY2 SY15 SY23 SY36	SY13 SY27 SY28 SY34 SY42	SY4 SY5 SY10 SY41 SY53
第 8 类	SY4 SY13 SY18 SY24 SY39	SY5 SY11 SY22 SY54 SY56	SY6 SY17 SY21 SY33 SY38
第 9 类	SY10 SY12 SY20 SY57 SY9	SY6 SY15 SY25 SY35 SY58	SY2 SY12 SY34 SY47 SY48
第 10 类	SY22 SY32 SY34 SY35 SY40	SY31 SY32 SY39 SY52 SY59	SY3 SY32 SY50 SY54 SY55
第 11 类	SY3 SY6 SY11 SY16	SY8 SY9 SY36 SY41	SY1 SY7 SY26 SY40

	SY55	SY55	SY45
第 12 类	SY30 SY45 SY51 SY54 SY60	SY14 SY17 SY18 SY45 SY53	SY15 SY16 SY20 SY28 SY46

5 问题二分析与模型建立

5.1 问题分析

问题二分别为两个小问，第一部分是要比较第一问建立的分类模型和第二问建立的判别模型之间的结果，根据第二问的标签来分析第一问的分类模型的准确度。第二部分是要通过用有标签数据训练的判别模型，对 30 组无标签数据的活动状态进行判别。

具体来说，首先就是要通过深度学习模型训练一个判别模型，而后在进行后续的其他的比较和分类。给出的数据的命名是标签，例如 a1t1，a 后面的 1 代表的是第一类活动，t 后面的 1 代表的是第几次采样。其中的数据一共有六列，每一列就是传感器采样的线加速度和角加速度 x，y，z 三个维度。行代表的是每个时间刻度，采样频率是 100Hz，所以时间间隔为 0.01s。总的来看，这就是一个多元时间序列分类问题，现有的很多深度学习模型都可以解决这种问题。但是这些方法大多数忽略了不同的特征维度之间的关系，在训练模型时，单独的考虑每个特征维度。还有的深度学习模型没有很好的提取到时序数据的这种动态特性，所以我们要建立一个更加完备的模型去解决问题。

5.2 基于动态图神经网络的多元时间序列分类模型建立

5.2.1 多元时间序列分类任务

多元时间序列分类是机器学习与数据挖掘中的重要任务，旨在对由多个时间序列组成的数据进行分类。每个时间序列包含多个变量，这些变量在时间上是同步的。该任务在金融市场分析、医疗诊断、工业监控和运动识别等诸多领域有着广泛应用。例如，通过分析股票价格、交易量及其他经济指标的时间序列数据，可以预测市场走势或分类不同的市场状态；在医疗领域，利用患者的心率、血压和体温等生理指标的时间序列数据，可以进行疾病分类。

然而，多元时间序列分类也面临诸多挑战。首先是数据的高维性，处理和分析难度较大；其次是时间依赖性，需要考虑时间上的相关性和模式。此外，数据不均衡问题和数据中的噪声与缺失值也增加了分类的难度。为了应对这些挑战，研究人员开发了多种方法，包括传统的特征工程、动态时间规整（DTW）和隐马尔可夫模型（HMM）等方法，以及机器学习中的支持向量机（SVM）和随机森林等方法。近年来，深度学习方法如卷积神经网络（CNN）、循环神经网络（RNN）及其变种长短期记忆网络（LSTM）和注意力机制（Attention Mechanism）在处理时间序列数据方面表现出色。

评估多元时间序列分类模型的常用指标包括准确率、精确率、召回率、F1 评分和混淆矩阵。这些指标帮助研究人员和工程师评估模型在不同类别上的表现。为了支持这一领域的研究，许多数据集如 UCR 时间序列分类数据集和 PhysioNet 数据库提供了丰富的时间序列数据。此外，tslearn、sktime 等 Python 库以及深度学习框架 TensorFlow 和

PyTorch 为研究提供了强大的工具与支持。通过这些方法和工具，研究人员在多元时间序列分类领域取得了显著成果。

虽然基于深度学习的模型当前很热门，但仍有很明显的缺点和局限性。大多数的模型都是假设每个变量之间都是相同的，这样就不能有效的建模出变量之间的关系。为了解决这个问题，我们考虑到使用图对多元时间序列进行建模。

5.2.2 动态图神经网络

动态图神经网络（Dynamic Graph Neural Networks, DGNN）是一种深度学习模型，专门用于处理节点和边随时间变化的图结构。与静态图神经网络不同，DGNN 能够捕捉图结构在时间上的动态变化，这使得它在社交网络分析、交通网络预测、金融市场分析和推荐系统等领域具有广泛的应用。

DGNN 的核心特点在于其时间维度的建模能力，能够处理节点和边属性随时间的变化，甚至节点和边的出现或消失。为了准确预测未来的图结构，DGNN 需要有效地利用历史信息，这通常通过时间序列模型、记忆网络或递归神经网络（RNN）来实现。

常见的动态图神经网络方法包括基于时间快照的方法、时间递归神经网络和连续时间图神经网络。基于时间快照的方法将动态图按时间切片，每个时间点上的图称为一个快照，然后对每个快照应用静态图神经网络。时间递归神经网络则结合图神经网络和递归神经网络，通过引入时间维度来处理图的时间变化。连续时间图神经网络则直接对图的连续时间变化进行建模，常用的方法包括时间编码和时间注意力机制。

在实际应用中，DGNN 能够用于分析和预测社交网络中的用户行为和关系变化，交通网络中的交通流量和道路状态，金融市场中的交易网络和价格波动，以及推荐系统中的用户兴趣和物品流行度的动态变化。通过捕捉和建模节点和边在时间上的动态变化，DGNN 显著提升了对动态图数据的处理能力，尽管仍面临许多挑战和研究问题，但其应用前景十分广阔。

当前运用于多元时间序列分类任务的动态图模型几乎没有，这个领域还有很多改进的空间。所以我们建立了一个动态图模型用于多元时间序列分类任务上，来丰富这个领域。

5.2.3 模型总体结构

本文专门使用了一种为处理多元时间序列数据而设计的深度学习模型，其主要目标是捕捉时间和空间上的依赖关系以及不同维度之间的动态关联。这个模型由几个核心模块构成，包括时间序列图构建模块、动态图神经网络模块、动态图池化模块以及输出模块。在时间序列图构建模块中，初始关系通过生成一组图邻接矩阵来表示，这些矩阵分别对应不同时间槽的变量关系，并通过浅层嵌入和稀疏化技术在迭代过程中不断学习和优化，以更准确地反映变量之间的依赖关系。动态图神经网络模块则负责利用这些优化后的邻接矩阵，捕捉时间图之间的动态关联。该模块由多个时间卷积层组成，每个卷积层处理时间序列数据，从而提取出时空依赖特征。为了避免平坦池化问题，动态图池化模块引入了一种可微分且分层的图池化方法，该方法通过可学习的时态卷积参数来提高模型捕捉复杂时态特征的能力。最后，输出模块通过平均池化和全连接层的结合，生成每个类别的数值，完成分类任务。整体而言，通过各个模块的协同工作，有效地捕捉并处理多元时间序列数据中的复杂依赖关系，从而显著提高模型的性能和准确性。

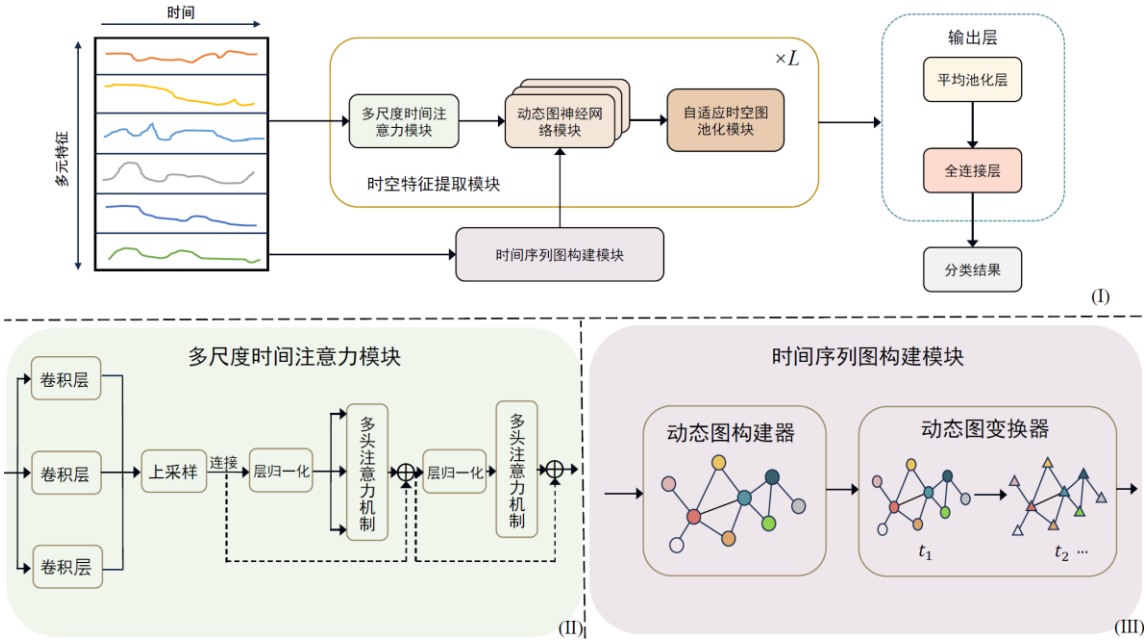

图 5 模型概述图

5.2.4 时间序列图构建模块

在动态图神经网络的研究中，时间序列数据的每个时间槽可以被表示为一个图，其邻接矩阵通过两个可学习的嵌入向量生成。这些嵌入向量分别表示源节点和目标节点。通过嵌入向量的乘积生成初始邻接矩阵，并通过稀疏化保留前 m 个最大权重的元素，从而实现计算的高效性和图结构的灵活表示。具体步骤如下：

首先，对于每个时间槽 t，初始化两个长度为 e 的向量$\boldsymbol{\Phi}_t$和$\boldsymbol{\Omega}_t$。这些向量的元素是可学习参数，随机初始化。通过对这两个向量的乘积，可以生成一个初始邻接矩阵$\mathbf{A}_t$。具体来说，嵌入向量$\boldsymbol{\Phi}_t$和$\boldsymbol{\Omega}_t$的乘积$\mathbf{A}_t = \boldsymbol{\Phi}_t^{\mathrm{T}} \cdot \boldsymbol{\Omega}_t$表示时间槽 t 的初始邻接矩阵。

然而，直接使用这个初始邻接矩阵在计算上可能会非常昂贵。为了降低计算成本，我们对邻接矩阵$\mathbf{A}_t$进行稀疏化处理。具体方法是保留邻接矩阵中前$\mathbf{A}_t$个最大权重的元素，并将其他值设置为零。稀疏化过程包括以下几个步骤：首先，对邻接矩阵$\mathbf{A}_t$的元素进行排序，找到前 m 最大值的索引。然后，创建一个稀疏邻接矩阵$\mathbf{A}_t^{\text{sparse}}$，只保留前 m 最大值的元素，其余元素设置为零。

$$\mathbf{A}_t^{\text{sparse}}[i,j] = \begin{cases} \mathbf{A}_t[i,j] & \text{如果}(i,j) \in \text{argtop}(\mathbf{A}_t, m) \\ 0 & \text{否则} \end{cases}$$

其中，$\text{argtop}(\mathbf{A}_t, m)$返回邻接矩阵$\mathbf{A}_t$中前 m 最大值的索引。通过这种方式，我们可以有效地降低动态图神经网络的计算成本，同时保留了关键的图结构信息。稀疏化后的邻接矩阵$\mathbf{A}_t^{\text{sparse}}$能够更好地捕捉变量之间的隐藏依赖关系，并在训练过程中进行优化。

这种方法通过浅层嵌入和稀疏化技术，显著提高了动态图神经网络的计算效率，并为时间序列数据提供了灵活且高效的图结构表示。邻接矩阵的值在训练过程中进行优化，从而更好地反映时间序列数据中不同变量之间的依赖关系。这种灵活性和高效性使得该方法在处理复杂的时间序列数据时具有显著的优势。

5.2.5 动态图神经网络模块

现有方法在处理每个时间槽的图时，往往通过添加相同数量的顶点来增加图的复杂度，尤其是顶点数量较多时，这种方式会带来额外的计算开销。为了解决这一问题，可以引入一种时间混合机制，通过时间卷积和时间注意力来更高效地处理动态图数据。

时间卷积使用一维卷积处理每个节点在不同时间点的特征，从而捕捉时间序列中的局部特征。

$$\mathbf{h}_v^{(t)} = \text{Conv1D}(\mathbf{h}_v^{(t-\Delta t)}, \mathbf{h}_v^{(t)}, \mathbf{h}_v^{(t+\Delta t)}$$

其中，$\mathbf{h}_v^{(t)}$表示节点 v 在时间 t 的特征，Δt是时间间隔。

时间注意力机制根据不同时间点的特征重要性进行加权聚合。公式如下：

$$\mathbf{h}_v^{(t)} = \sum_{\tau \in \mathcal{T}} \alpha_\tau \mathbf{h}_v^{(t-\tau)}$$

其中，α_τ是时间注意力权重，通过学习得到。

通过这种方式，不需要显式地增加顶点数量，而是通过时间卷积和注意力机制在时间维度上有效地聚合信息，从而提高动态图变换的效率。

现有方法中，尽管一些静态的图神经网络在区分图结构上表现优异，但在处理动态图数据时可能无法充分利用时间信息。此外，邻接矩阵归一化可能会导致信息丢失，特别是在处理稀疏图时。为了解决这些问题，可以引入时空图神经网络（Spatio-Temporal Graph Neural Network, STGNN），将时间信息和空间信息统一到一个框架中进行处理。

时空图卷积在每一层中同时处理时间和空间信息。公式如下：

$$\mathbf{h}_v^{(l,t)} = \sigma\left(\sum_{u \in \mathcal{N}(v)} \frac{1}{\sqrt{d_v d_u}} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1,t)} + \mathbf{W}_{\text{time}} \mathbf{h}_v^{(l-1,t-1)} \right)$$

其中，$\mathbf{W}^{(l)}$是空间卷积权重，$\mathbf{W}_{\text{time}}$是时间卷积权重，σ是激活函数。

时空注意力机制在每一层中引入时空注意力机制，根据节点在不同时间点和邻居节点之间的相关性进行加权聚合。公式如下：

$$\mathbf{h}_v^{(l,t)} = \sum_{u \in \mathcal{N}(v)} \alpha_{uv}^{(l,t)} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1,t)} + \sum_{\tau \in \mathcal{T}} \beta_\tau^{(l,t)} \mathbf{W}_{\text{time}} \mathbf{h}_v^{(l-1,t-\tau)}$$

其中，$\alpha_{uv}^{(l,t)}$和$\beta_\tau^{(l,t)}$是空间和时间注意力权重，通过学习得到。

通过这种方法，可以在时空维度上更全面地聚合信息，提高模型对动态图数据的处理能力。本文的模型能够高效地处理动态图数据，充分利用时间信息和空间信息，提高模型的性能和准确性，为动态图神经网络模块提供了更好的信息传递和特征聚合策略。

5.2.6 自适应时空图池化模块

传统的池化方法，如最大池化、平均池化和总和池化，由于在聚合特征时会丢失大量信息，因此在处理具有复杂结构和时间依赖性的动态图数据时并不理想。为了更好地捕捉动态图中的时空特征，我们提出了一种改进的时序图池化方法，称为自适应时空图池化（Adaptive Spatio-Temporal Graph Pooling, ASTGP），该方法结合了图池化和时间过程，同时通过自适应学习机制有效减少信息丢失。

ASTGP 的核心思想是通过自适应学习机制，动态地将节点聚合到簇中，同时保留重要的时空信息。我们引入一个自适应权重矩阵$\mathbf{W}_{\text{adaptive}}$，用于学习每个节点的聚合权重，公式如下：

$$\mathbf{M} = \text{softmax}(\mathbf{W}_{\text{adaptive}} \cdot \mathbf{X}^{(l)})$$

其中，$\mathbf{X}^{(l)}$是第 l 层的输入节点嵌入矩阵，$\mathbf{M}$是自适应权重矩阵，通过 *softmax* 激活函数确保权重矩阵的行和为 1，从而保证节点特征的归一化聚合。

然后，通过自适应权重矩阵$\mathbf{M}$和节点嵌入矩阵$\mathbf{X}^{(l)}$计算输出嵌入矩阵$\mathbf{X}^{(l+1)}$：

$$\mathbf{X}^{(l+1)} = \mathbf{M} \cdot \mathbf{X}^{(l)}$$

这样，我们可以自适应地将节点聚合到簇中，同时保留节点的特征信息。

为了生成新的邻接矩阵$\mathbf{A}^{(l+1)}$，我们采用自适应学习机制，通过自适应权重矩阵$\mathbf{M}$和原始邻接矩阵$\mathbf{A}^{(l)}$进行计算：

$$\mathbf{A}^{(l+1)} = \mathbf{M} \cdot \mathbf{A}^{(l)} \cdot \mathbf{M}^{\mathrm{T}}$$

其中，$\mathbf{A}^{(l+1)}$表示新的聚类节点之间的连接关系和相应的权重，这保证了在聚合过程中节点间的关系得以保留和优化。

自适应时空图池化方法通过自适应学习机制和动态权重矩阵，在池化过程中最大程度地保留节点特征和时间依赖性。与传统方法相比，ASTGP 能够更有效地处理具有复杂时空依赖性的动态图数据，提高模型的性能和准确性。这种改进不仅解决了传统池化方法信息丢失的问题，还增强了特征聚合的灵活性和适应性。

5.2.7 多尺度时间注意力模块

我们提出了一种新的时间特征提取方法，称为多尺度时间注意力网络（Multi-Scale Temporal Attention Network, MSTAN）。该方法结合了多尺度卷积和注意力机制，以更有效地捕捉时间序列中的复杂依赖关系。

首先，我们引入多尺度卷积来处理时间序列数据。通过使用不同大小的卷积核（例如 3, 5, 7）来捕捉不同时间跨度上的特征，这些卷积核能够在不同的时间窗口内提取特征，从而更全面地捕捉时间依赖性。假设输入时间序列为$X \in \mathbb{R}^{T \times D}$，其中$T$为时间步数，$D$为特征维度。多尺度卷积操作可以表示为：

$$C_k = \text{Conv}(X, W_k)$$

其中，W_k为第k个卷积核的权重，C_k为对应的卷积输出。

为了同时提取短期和长期时间依赖关系，我们将多尺度卷积层并行应用于输入的时间序列数据，得到多个特征图：

$$C = [C_1, C_2, \dots, C_K]$$

其中，K为不同尺度卷积核的数量。

为了进一步增强时间特征的提取，我们引入注意力机制对多尺度卷积的输出进行加权融合。通过自注意力机制计算时间序列中每个时间步之间的相关性，从而动态调整各时间步的权重。自注意力机制可以表示为：

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

其中，Q,K,V分别为查询、键和值矩阵，d_k为键向量的维度。多头注意力机制增强了模型的表示能力：

$$\text{MultiHead}(Q, K, V) = [\text{head}_1, \text{head}_2, \dots, \text{head}_h]W^O$$

其中，$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$，$W_i^Q, W_i^K, W_i^V, W^O$为可训练权重矩阵，$h$为注意力头数。

5.3 结果分析展示

5.3.1 问题二第一问结果展示

问题二给出了具体的人体行为，即为有标签数据，在使用问题一的 K-Mean 方法时，我们在不使用标签的情况下，分别用到附件 2 的 10 个人的数据中，并将分类后的结果与实际标签进行对比，进而得到最终的判别结果，此外，为和深度学习算法进行公平比较，我们也使用数据混合后直接分类的方式进行了测试，即未直接判别结果。

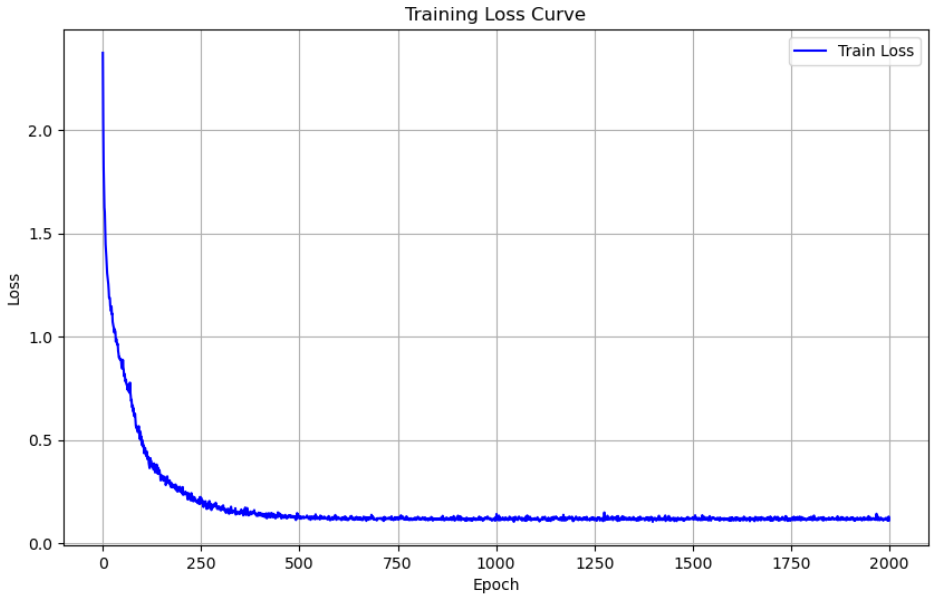

图 6 活动状态识别训练损失曲线

对于基于动态图神经网络的多元时间序列分类模型，模型本身为监督模型，其将附件 2 中的 70%数据划分为训练集，20%为测试集，剩余 10%为测试集，共计训练 2000 轮次，图 6 展示了在这个过程中的损失函数，可以看到，在训练过程中训练损失值下降明显，说明整个训练过程模型得到了有效的训练。在对模型进行测试时，为防止数据泄露引起的不公平，我们使用测试集进行测试，即使用未参与训练的 60 个数据进行最终的测试，测试结果见下表。

表6 带约束的 K-means 聚类算法图

人员类别	判别结果（%）	人员类别	判别结果（%）
Person4	83.33	Person9	73.33
Person5	71.67	Person10	71.67
Person6	83.33	Person11	71.67
Person7	71.67	Person12	80.00
Person8	83.33	Person13	71.67
K-Means 平均判别结果		76.81	
K-Means 直接判别结果		63.83	
多元时间序列分类模型判别结果		93.83	

表 6 展示了最终的结果。可以看到 K-Means 在监督数据上表现较好，平均能够达到 76.81%的判别准确率，但这与文献以及本文中使用的深度学习算法相比还有较大差距。特别是当使用 600 条数据进行分类时，判别结果明显下降，这是因为在特征空间中，存在了大量的数据，总体分类数远小于数量集，使得簇中心难于确定，结果具有很强的随机性。仅依赖数据之间的相似性进行分类很难的到正确的结果。与深度学习模型相比，K-Means 方法本身为无监督训练方法，不依赖于数据的时间序列特性，缺少标签指导，且模型对应的解空间很小，泛化能力弱，而深度学习方法通过，能够捕捉时间序列数据中的时序依赖性和动态特性，可以自动学习数据中的复杂特征。

5.3.2 问题二第二小问结果展示

我们保存了第一小问训练的模型参数，第二小问使用同样的参数，直接将无标签的活动数据输入到模型中，输出预测状态标签，如表 7 所示。

表7 问题 2 第 2 问结果

活动类型	判别状态
SY1	5
SY2	1
SY3	7
SY4	11
SY5	7
SY6	9
SY7	2
SY8	6
SY9	7
SY10	8
SY11	9
SY12	7
SY13	4
SY14	3
SY15	4
SY16	1
SY17	4
SY18	5
SY19	9
SY20	10
SY21	6
SY22	2
SY23	10

SY24	5
SY25	2
SY26	9
SY27	9
SY28	5
SY29	6
SY30	5

6 问题三分析与模型建立

6.1 问题分析

对于问题三，在问题一二的基础上，进一步得到参与实验的 13 名人员的年龄、身高、体重数据，要求分析不同人员的同一活动状态是否存在差异，并判断活动状态数据与实验人员的年龄、身高、体重间是否存在潜在的相关性，最后使用判别模型对附件 5 中数据进行活动人员判别。

针对问题三中上述具体任务，首先考虑对不同人员的同一活动状态是否存在差异进行判定，即可使用统计方法（如方差分析 ANOVA）来检测不同人员在同一活动状态下是否存在显著差异，将同一活动状态的数据提取出来，使用 ANOVA 或 Kruskal-Wallis H 检验（如果数据不满足正态分布）等方法进行差异分析；其次，对于判断活动状态与人员特征数据间的相关性，由于这种相关性是非显式的，且活动状态的数据包含 6 个基础特征维度和异步数据，因此需要基于数据本身进行特征统一和数据压缩，而后通过计算不同自变量，即活动状态特征数据，对因变量，即人员特征数据间的 Person 相关性系数，从而对具体活动和人员特征进行相关性分析；最后，为实现基于活动状态数据对人员的画像，此时数据类型标签从活动状态类型转变为人员类型，因此，需要进一步根据更新后的数据类型标签对模型进行针对性设计并训练，以满足人员类型的判别需要。

6.2 模型建立

6.2.1 相关性分析

为分析活动状态数据与实验人员的年龄、身高、体重有无关系，我们使用 Pearson 相关性系数两两计算数据间相关性。需要注意的是，活动状态数据为多类型的多维时序数据，无法与实验人员的多维特征信息开展相关性分析，因此，需要对数据进行有效的压缩规整，从而提取关键特征，并为探究相关性关系保留主要信息。具体相关性分析流程图如下。

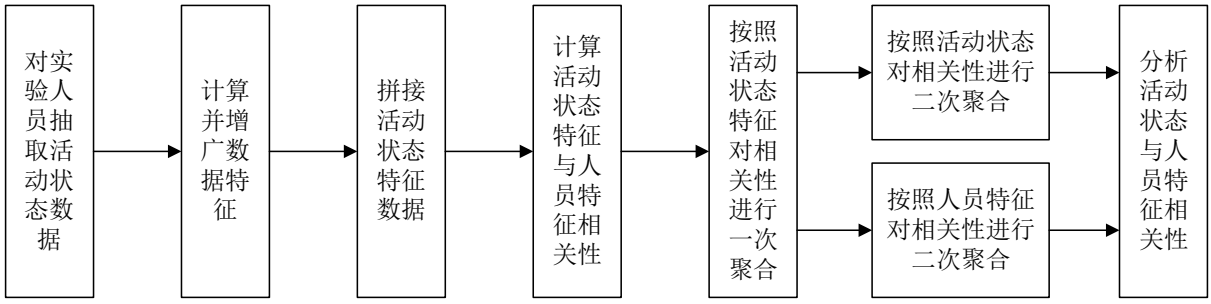

图 7 相关性分析流程

首先，针对不同实验人员的不同类型活动状态数据，采用分层抽样的方法，对每名实验人员的每类活动状态数据抽取其一作为相关性分析样本；其次，对活动状态中的特征维度进行统一，针对每个活动数据中的加速度计特征 acc_x，acc_y，acc_z 和陀螺仪特征 gyro_x，gyro_y，gyro_z，计算其上下四分位数和平均数作为时序特征，从而压缩活动数据的时间维度并对每个活动状态数据得到共计 3×6=18 维增广特征；随后，拼接不同人员不同活动状态类型的特征数据，并分别计算与人员特征信息（年龄、身高、体重）的相关性；最后，为探究各活动状态类型与人员特征的相关性，进一步将每类活动状态的不同特征相关性进行绝对值加和，从而得到不同活动状态类型与不同人员特征信息的相关性得分。

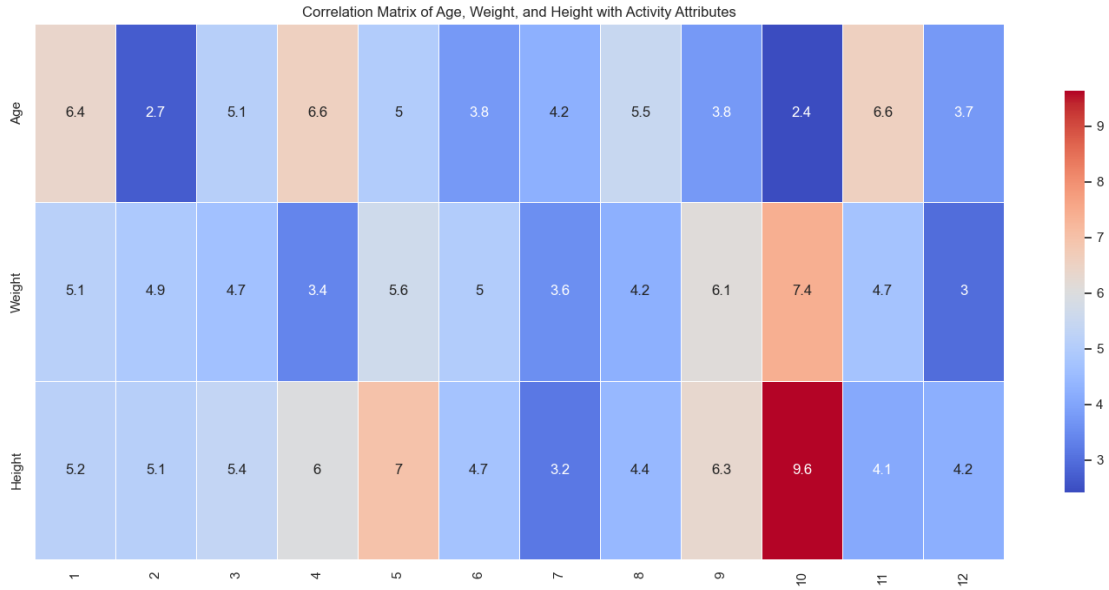

图 8 活动状态与人员特征相关性热力图

观察活动状态与人员特征的相关性热力图可知，实验人员身高与活动状态 10，即躺下之间的相关性最为明显，同时人员体重对躺下的影响也较为明显，与此相似的还有站立状态等。因此，可以看出不同活动状态与不同人员特征间的相关性分布并不相对平均，也即说明活动状态数据与实验人员的年龄、身高、体重间存在着潜在的相关性。分别按照活动状态和人员特征对相关性得分进行进一步聚合，从而便于观察不同人员特征对人员活动的相关性以及不同活动状态对人员信息的相关性。

表8 不同人员特征与活动信息相关性得分

人员特征	与人员活动相关性得分
年龄	55.75
体重	57.76
身高	**65.18**

表9 不同活动状态与人员信息相关性得分

活动状态	与人员信息相关性得分
1	**16.72**
2	12.69

3	15.14
4	15.95
5	**17.60**
6	13.50
7	10.95
8	14.12
9	16.21
10	**19.48**
11	15.40
12	10.93

根据上述相关性得分聚合情况可以看出，在不同类型的活动状态中，躺下、步行下楼、向前走三种活动状态与人员信息相关性最为显著，而在不同人员特征中，身高特征与人员活动的相关性最为显著。综上，可以得知人员活动状态数据与实验人员的年龄、身高、体重虽然无法直观看出其隐式关联，但仍具有一定的相关性，且身高特征对活动状态数据的影响最大，因此结合人员特征信息对人员活动状态数据进行人员画像具有一定的信息辅助作用。

6.2.2 人员判别模型任务设计

针对问题 3 中要求对 12 类活动数据所属的未知实验人员进行识别判定，我们利用 5.2 节中提出的特征提取模型和附件 2 中提供的完整数据进行人员判别模型任务设计。

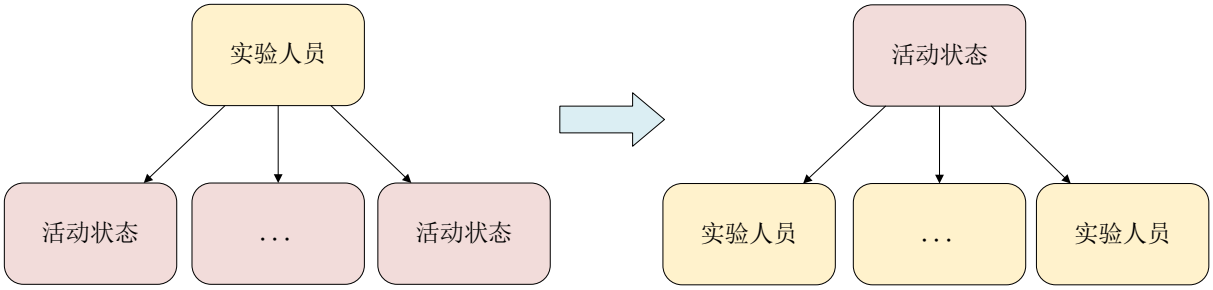

图 9 数据层次结构转换

此时分析任务为对不同人员的活动状态数据进行人员识别，因此，在进行模型训练任务的数据标签设定时，应从活动状态类型标签转换为实验人员标签，相应的，对于数据结构进行训练集处理时，应与问题 2 中的数据层次处理相反。

为进行人员判别训练集设置，对原始数据标签进行结构化处理，此时对同一实验人员的 12 类不同活动状态的共计 60 组活动数据统一标签为实验人员序号，即 4-13；设置梯度下降循环为 2000 代，训练数据批次为 16，预热学习率从 0.001 开始；按照 7：3 的比例划分数据集，训练损失曲线结果如下。

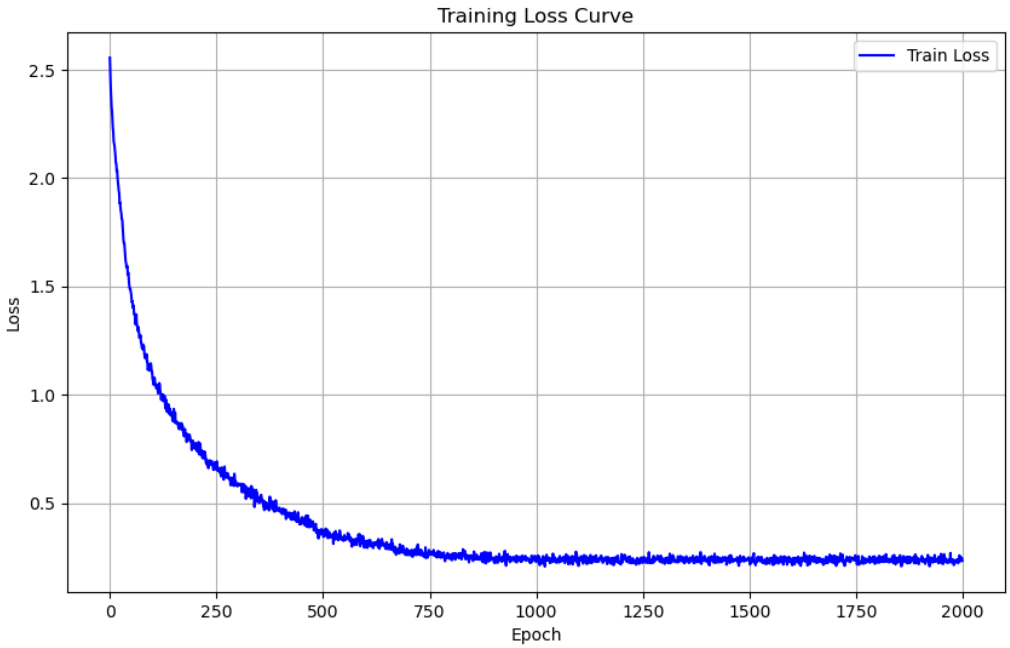

图 10 人员判别训练损失曲线

6.3 问题求解

利用 5.2 节中建立的判别模型，我们对附件 5 中所给的 5 名未知人员的实验数据进行测试，分别预测其 12 种活动数据对应的人员标签编号，得到的结果为[9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 7, 6, 6, 6, 9, 6, 6, 6, 6, 6, 6, 5, 6, 5, 5, 5, 5, 5, 5, 5, 5, 6, 8, 5, 6, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 12, 12, 12, 12, 12, 12, 12, 12, 12, 11, 12, 12]，其中，上述输出结果顺序表示 unknow1 到 unknow5 的各自 12 组活动状态数据判别得到的人员类型编号。最后，通过对每名未知人员的 12 组判别类型编号进行众数提取，得到问题 3 结果如下。

表10 问题 3 结果

活动类型	判别结果
Unknow1	9
Unknow2	6
Unknow3	5
Unknow4	8
Unknow5	12

7 模型评价与推广

7.1 模型的优点

(1) 实际结合度高：模型充分结合实际情况，简化了复杂的前提条件和约束，例如传感器的放置位置和数据采集的环境一致性，考虑了个体差异、传感器数据的多样性等重要因素，构建了合理的模型。这使得模型在实际应用中具备较高的可操作性和推广价值。

(2) 参数设定合理：模型运用了特征工程和深度学习的思想，抓住了影响人体行为识别的关键因素，将复杂的多元时间序列分类问题转化为动态图建模问题。合理设置了时间窗口、滑动步长和神经网络的层数等参数，使得模型的输出结果符合预期，能够有效解决实际问题。

(3) 算法优势明显：本文使用的动态图神经网络和自适应时空图池化算法具有高效处理时间序列数据、动态建模和自适应特征提取等优点。这些算法在处理高维度、复杂依赖关系的时间序列数据时表现优异，显著提升了模型的准确性和鲁棒性。

(4) 结果稳定可靠：模型得到的识别结果具有效率高、输出稳定、分类准确等特点，基本不存在数据噪声、分类偏差等问题。在现有条件下，能够有效提高行为识别的性能，为健康监测、运动健身和智能家居等应用提供了可靠的技术支持。

7.2 模型的不足

(1) 因素考虑不全面：实际应用中，可能还存在其他重要因素，例如传感器的佩戴方式变化、用户的情绪和健康状况等，这些因素未能在模型中充分考虑，可能会在一定程度上影响模型的准确性和泛化能力。

(2) 适用范围有限：本文提出的模型在现有数据集和实验条件下表现较好，但由于时间和资源限制，未能对其他情形（例如，不同传感器的组合、不同数据采集频率等）进行全面验证。在其他应用场景中，模型的性能可能无法达到同样的效果。

(3) 非线性因素简化：实际上，人体行为与传感器数据之间的关系不一定是线性的，本文将其作为线性因子处理，忽略了潜在的非线性和边际效应。这可能导致模型在处理复杂行为模式时的表现有所欠缺。

7.3 模型的推广

(1) 在健康监测领域，可以将模型中使用的传感器数据参数替换为更高精度的生理传感器数据，例如心率、血氧饱和度等，从而解决更广泛的健康监测问题。例如，通过融合多模态传感数据，可以实现更全面的健康状态监测和早期疾病预警。

(2) 结合参考文献中的联邦学习和差分隐私技术，进一步考虑数据隐私保护和分布式计算的影响，从而在保护用户隐私的同时，提升模型的准确性和可靠性。具体而言，可以在模型训练过程中引入差分隐私机制，对梯度信息进行扰动，或者采用联邦学习框架，在数据不离开本地设备的情况下进行模型训练，提高数据安全性和用户信任度。

参考文献

[1] Kwapisz J R, Weiss G M, Moore S A. Activity recognition using cell phone accelerometers[J]. ACM SigKDD Explorations Newsletter, 2011, 12(2):74-82.

[2] Brezmes T, Gorricho J L, Cotrina J. Activity recognition from accelerometer data on a mobile phone, 2009[C]. Berlin: International Work-Conference on Artificial Neural Neworks,2009:796-799.

[3] 伍俊杰. 基于智能手机多传感器的非特定人行为识别方法研究[D].哈尔滨:哈尔滨工业大 学, 2016.

[4] 周林,雷丽平,杨龙频.基于多传感器的人体行为识别系统[J].传感器与微系统, 2016,35(03):89-91.

[5] 范琳,王忠民.穿戴位置无关的手机用户行为识别模型[J].计算机应用研究, 2015, 32(1):63-66.

[6] Vepakomma P, De D, Das S K, et al. A-Wristocracy: Deep learning on wrist-worn sensing for recognition of user complex activities, 2015[C]. IEEE: 2015 IEEE 12th International conference on wearable and implantable body sensor networks (BSN),2015:1-6.

[7] Hammerla N Y, Halloran S, Plötz T. Deep, convolutional, and recurrent models for human activity recognition using wearables[J]. arXiv preprint arXiv, 2016, [2016-04-29].

[8] Yao S, Hu S, Zhao Y, et al. Deepsense: A unified deep learning framework for time-series mobile sensing data processing, 2017[C]. Proceedings of the 26th International Conference on World Wide Web,2017:351-360.

[9] Ordóñez F J, Roggen D. Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition[J]. Sensors, 2016, 16(1):115.

[10] Kim S H, Lee D H, Kim K J. EWMA-PRIM: Process optimization based on time-series process operational data using the exponentially weighted moving average and patient rule induction method[J]. Expert Systems with Applications, 2022, 195: 116606.

[11] Seong B. Smoothing and forecasting mixed-frequency time series with vector exponential smoothing models[J]. Economic Modelling, 2020, 91: 463-468.

[12] Syavasya C, Muddana A L. Machine learning based Time series prediction using Holt-Winters Exponential Smoothing with Multiplicative Seasonality[C]//2021 5th International Conference on Electrical, Electronics, Communication, Computer Technologies and Optimization Techniques (ICEECCOT). IEEE, 2021: 174-182.

[13] Lei T, Yang K, Li J, et al. Multichannel spatial–temporal graph convolution network based on spectrum decomposition for traffic prediction[J]. Expert Systems with Applications, 2024, 238: 122281.

[14] Song D, Baek A M C, Kim N. Forecasting stock market indices using padding-based fourier transform denoising and time series deep learning models[J]. IEEE Access, 2021, 9: 83786-83796.

[15] Mohammadi H A, Ghofrani S, Nikseresht A. Using empirical wavelet transform and high-order fuzzy cognitive maps for time series forecasting[J]. Applied Soft Computing, 2023, 135: 109990.
[16] Zhongda T, Shujiang L, Yanhong W, et al. A prediction method based on wavelet transform and multiple models fusion for chaotic time series[J]. Chaos, Solitons & Fractals, 2017, 98: 158-172.
[17] Hong-fa W. Clustering of hydrological time series based on discrete wavelet transform[J]. Physics procedia, 2012, 25: 1966-1972.
[18] Kumar A, Tomar H, Mehla V K, et al. Stationary wavelet transform based ECG signal denoising method[J]. ISA transactions, 2021, 114: 251-262.
[19] Jimenez J R, Wu H R. Empirical Mode Decomposition as a tool for data analysis[C]//2011 6th IEEE Conference on Industrial Electronics and Applications. IEEE, 2011: 2538-2543.
[20] Cleveland R B, Cleveland W S, McRae J E, et al. STL: A seasonal-trend decomposition[J]. J. Off. Stat, 1990, 6(1): 3-73.
[21] Bishop G, Welch G. An introduction to the kalman filter[J]. Proc of SIGGRAPH, Course, 2001, 8(27599-23175): 41.
[22] Ye L, Keogh E. Time series shapelets: a new primitive for data mining[C]//Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining. 2009: 947-956.
[23] Müller M. Dynamic time warping[J]. Information retrieval for music and motion, 2007: 69-84.
[24] Geler Z, Kurbalija V, Ivanović M, et al. Weighted kNN and constrained elastic distances for time-series classification[J]. Expert Systems with Applications, 2020, 162: 113829.
[25] Chen T T, Lee S J. A weighted LS-SVM based learning system for time series forecasting[J]. Information Sciences, 2015, 299: 99-116.
[26] Davis R A, Nielsen M S. Modeling of time series using random forests: Theoretical developments[J]. 2020.
[27] Ji C, Zou X, Hu Y, et al. XG-SF: An XGBoost classifier based on shapelet features for time series classification[J]. Procedia computer science, 2019, 147: 24-28.
[28] Ke G, Meng Q, Finley T, et al. Lightgbm: A highly efficient gradient boosting decision tree[J]. Advances in neural information processing systems, 2017, 30.
[29] Zaremba W, Sutskever I, Vinyals O. Recurrent neural network regularization[J]. arXiv preprint arXiv:1409.2329, 2014.
[30] Karim F, Majumdar S, Darabi H, et al. Multivariate LSTM-FCNs for time series classification[J]. Neural networks, 2019, 116: 237-245.

附录

附录 A:问题一核心代码

```
from sklearn.metrics import pairwise_distances_argmin_min
from matplotlib.font_manager import FontProperties
from sklearn.metrics import silhouette_score
from sklearn.cluster import KMeans
from matplotlib.pylab import mpl
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import random
import math
import os
# 设置中文字体
import matplotlib.pyplot as plt

font_path = 'C:/Windows/Fonts/simhei.ttf'  # 使用 Windows 系统自带的 SimHei 字体
font = FontProperties(fname=font_path)
plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['font.sans-serif'] = ['Tahoma']

directory_path = 'C:/Users/zhout/OneDrive/桌面/2024 年湖南省研究生数学建模竞赛赛
题及附件/2024 年湖南省研究生数学建模竞赛赛题/A 题/附件 2/Person5'
# 列出目录下的所有文件
files = os.listdir(directory_path)

X = []
f_X = []
for file in files:
    # scaler = StandardScaler()
    excel_path = directory_path + "/" + file
    df = pd.read_excel(excel_path)
    # df = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
    fft_df = pd.DataFrame()
    for col in df.columns:
        df[col] = pd.to_numeric(df[col], errors='coerce')
        df = df[(df[col] >= -600) & (df[col] <= 600)]
        if col not in ['gyro_x(dps)', 'gyro_y(dps)', 'gyro_z(dps)']:
            df[col] = df[col] * 300

    for col in df.columns:
        fft_df[col] = np.abs(np.fft.fft(df[col]))

    var = df.var()
    avg = df[len(df) // 2:len(df)].mean()
    Q1 = df.quantile(0.25)
    median = df.quantile(0.5)
    Q3 = df.quantile(0.75)
    x_var = fft_df.var()
    x_avg = fft_df.mean()
    x_Q1 = fft_df.quantile(0.25)
    x_median = fft_df.quantile(0.5)
    x_Q3 = fft_df.quantile(0.75)
    x = np.concatenate((Q1.to_numpy(), median.to_numpy(), Q3.to_numpy()), 0)
    f_x = np.concatenate((x_Q1.to_numpy(), x_median.to_numpy(),
x_Q3.to_numpy()), 0)
    # x = np.concatenate((var.to_numpy(),me-
dian.to_numpy(),avg.to_numpy()),0)
```

```
    # f_x = np.concatenate((x_var.to_numpy(),x_me-
dian.to_numpy(),x_avg.to_numpy()),0)
    x = np.concatenate((x, f_X), 0)
    X.append(x)

    plt.figure(figsize=(10, 6))  # 可以调整图形大小
    plt.subplot(1, 2, 1)
    ax = df.boxplot(showmeans=True, showfliers=False, vert=False)
    ax.set_xlim([-600, 600])  # 假设 x 轴的范围是从 0 到 10
    # 设置标题和标签
    plt.title('时域数据箱型图', fontproperties=font)
    plt.xlabel('数据大小', fontproperties=font)
    plt.ylabel('类别', fontproperties=font)

    plt.figure(figsize=(10, 6))  # 可以调整图形大小
    plt.subplot(1, 2, 2)
    ax = fft_df.boxplot(showmeans=True, showfliers=False, vert=False)
    ax.set_xlim([-600, 600])  # 假设 x 轴的范围是从 0 到 10
    # 设置标题和标签
    plt.title('频域数据箱型图', fontproperties=font)
    plt.xlabel('数据大小', fontproperties=font)
    plt.ylabel('类别', fontproperties=font)

# K-Means++算法
# 假设 X 是特征矩阵，K 是给定的类别数
K = 12  # 给定的类别数
# 使用 K-Means++算法初始化质心，这有助于平衡簇的大小
kmeans = KMeans(n_clusters=K, init='k-means++', n_init=15, max_iter=5000)
# 训练模型
kmeans.fit(X)
# 打印簇标签
print("簇标签：", kmeans.labels_)
# 打印质心位置
print("质心位置：", kmeans.cluster_centers_)
# 打印每个簇中的数据点个数
unique, counts = np.unique(kmeans.labels_, return_counts=True)
print("每个簇中的数据点个数：", dict(zip(unique, counts)))
print(files)

##带约束的 K-Means 算法
from sklearn.cluster import KMeans


def custom_kmeans(X, num_clusters, desired_size, max_iterations=1000):
    kmeans = KMeans(n_clusters=num_clusters, init='k-means++', n_init=12,
max_iter=1)
    labels = kmeans.fit_predict(X)
    while not all(len(np.where(labels == i)[0]) == desired_size for i in
range(num_clusters)):
        # 计算每个簇的大小
        cluster_sizes = np.bincount(labels)
        # 找出大小不符合期望的簇
        overfull_clusters = np.where(cluster_sizes > desired_size)[0]
        lessfull_clusters = np.where(cluster_sizes < desired_size)[0]
        print(np.where(cluster_sizes > desired_size), np.where(cluster_sizes
< desired_size))
        if not overfull_clusters.size:
```

```
            # 如果没有簇的大小超过 desired_size，则退出循环
            break

        # 从每个超额的簇中移除一个点
        for cluster_id in overfull_clusters:
            # 找出当前簇中的点
            points_in_cluster = np.where(labels == cluster_id)[0]

            # 计算这些点到簇中心的距离
            distances = np.linalg.norm(X[points_in_cluster] - kmeans.clus-
ter_centers_[cluster_id])

            # 选择最远的点进行重新分配
            farthest_index = np.argmax(distances)

            farthest_point = points_in_cluster[farthest_index]

            # 找到除了当前簇之外最近的簇中心
            other_centers = kmeans.cluster_centers_[lessfull_clusters]
            closest_other_center_index = np.argmin(np.linalg.norm(X[far-
thest_point] - other_centers, axis=1))
            # 更新该点的标签
            closest_label = lessfull_clusters[closest_other_center_index]
            labels[farthest_point] = closest_label

        # 更新簇中心
        # kmeans.cluster_centers_ = np.array([X[labels == i].mean(axis=0)
for i in range(num_clusters)])

        # 打印当前迭代的簇大小
        print("Cluster sizes = {}".format(cluster_sizes))

    return labels, kmeans.cluster_centers_


# 示例使用
# 假设 X 是一个 NumPy 数组，包含你需要聚类的数据点
X = np.array(X)
# num_clusters 为簇的数量，desired_size 为每个簇期望的节点数
num_clusters = 12
desired_size = 5

labels, cluster_centers = custom_kmeans(X, num_clusters, desired_size)

print("簇分配:", labels)
print("簇中心:\n", cluster_centers)
# 打印每个簇中的数据点个数
unique, counts = np.unique(labels, return_counts=True)
right = 0
for i in range(12):
    tmp = labels[5 * i:5 * i + 5]
    unique_values, counts = np.unique(tmp, return_counts=True)

    max_count = np.max(counts)
    right += max_count
acc = right / 60
print(files)
print(str(acc * 100) + "%")
```

附录 B：问题二、三核心代码

数据预处理主要代码：

```
import os
import pandas as pd
import numpy as np


def calculate_statistics(file_path):
    df = pd.read_excel(file_path)

    statistics = []
    for col in df.columns:
        q1 = df[col].quantile(0.25)
        q3 = df[col].quantile(0.75)
        mean = df[col].mean()
        statistics.extend([q1, q3, mean])

    return statistics


def process_activity_files(folder_path):
    activity_statistics = []
    for i in range(1, 13):  # 12 种活动状态
        filename = f'a{i}t1.xlsx'  # 只处理第一个实验数据
        file_path = os.path.join(folder_path, filename)
        if os.path.exists(file_path):
            stats = calculate_statistics(file_path)
            activity_statistics.extend(stats)  # 使用 extend 而不是 append

    return activity_statistics


def process_person_folders(root_folder):
    all_person_statistics = []
    for person_folder in sorted(os.listdir(root_folder)):
        person_path = os.path.join(root_folder, person_folder)
        if os.path.isdir(person_path):
            activity_stats = process_activity_files(person_path)
            all_person_statistics.append(activity_stats)

    return all_person_statistics


def create_header(num_activities, num_dimensions):
    header = []
    for i in range(1, num_activities + 1):
        for j in range(1, num_dimensions + 1):
            header.extend([f'a{i}_dim{j}_Q1', f'a{i}_dim{j}_Q3',
f'a{i}_dim{j}_Mean'])
    return header


# 设置根文件夹路径
root_folder = './附件 2/'

# 处理所有 Person 文件夹中的数据
all_statistics = process_person_folders(root_folder)
```

```
# 生成表头
num_activities = 12  # 活动状态数
num_dimensions = 6  # 每个文件中的维度数
header = create_header(num_activities, num_dimensions)

# 创建 DataFrame 并保存到 Excel
df = pd.DataFrame(all_statistics, columns=header)
output_path = 'Q3dataprocess.xlsx'
df.to_excel(output_path, index=False)

print(f"二维向量已保存至 {output_path}")
```

网络层代码：

```
import torch
import torch.nn as nn
import torch.nn.functional as F

from torch import Tensor
from torch.nn import init
from torch.nn.parameter import Parameter


class multi_shallow_embedding(nn.Module):

    def __init__(self, num_nodes, k_neighs, num_graphs):
        super().__init__()

        self.num_nodes = num_nodes
        self.k = k_neighs
        self.num_graphs = num_graphs

        self.emb_s = Parameter(Tensor(num_graphs, num_nodes, 1))
        self.emb_t = Parameter(Tensor(num_graphs, 1, num_nodes))

    def reset_parameters(self):
        init.xavier_uniform_(self.emb_s)
        init.xavier_uniform_(self.emb_t)

    def forward(self, device):
        # adj: [G, N, N]
        adj = torch.matmul(self.emb_s, self.emb_t).to(device)

        # remove self-loop
        adj = adj.clone()
        idx = torch.arange(self.num_nodes, dtype=torch.long, device=device)
        adj[:, idx, idx] = float('-inf')

        # top-k-edge adj
        adj_flat = adj.reshape(self.num_graphs, -1)
        indices = adj_flat.topk(k=self.k)[1].reshape(-1)

        idx = torch.tensor([i // self.k for i in range(indices.size(0))],
device=device)

        adj_flat = torch.zeros_like(adj_flat).clone()
        adj_flat[idx, indices] = 1.
        adj = adj_flat.reshape_as(adj)

        return adj
```

```
class Group_Linear(nn.Module):

    def __init__(self, in_channels, out_channels, groups=1, bias=False):
        super().__init__()

        self.out_channels = out_channels
        self.groups = groups

        self.group_mlp = nn.Conv2d(in_channels * groups, out_channels *
groups, kernel_size=(1, 1), groups=groups,
                                   bias=bias)

        self.reset_parameters()

    def reset_parameters(self):
        self.group_mlp.reset_parameters()

    def forward(self, x: Tensor, is_reshape: False):
        """
        Args:
            x (Tensor): [B, C, N, F] (if not is_reshape), [B, C, G, N, F//G]
(if is_reshape)
        """
        B = x.size(0)
        C = x.size(1)
        N = x.size(-2)
        G = self.groups

        if not is_reshape:
            # x: [B, C_in, G, N, F//G]
            x = x.reshape(B, C, N, G, -1).transpose(2, 3)
        # x: [B, G*C_in, N, F//G]
        x = x.transpose(1, 2).reshape(B, G * C, N, -1)

        out = self.group_mlp(x)
        out = out.reshape(B, G, self.out_channels, N, -1).transpose(1, 2)

        # out: [B, C_out, G, N, F//G]
        return out


class DenseGCNConv2d(nn.Module):

    def __init__(self, in_channels, out_channels, groups=1, bias=True):
        super().__init__()

        self.in_channels = in_channels
        self.out_channels = out_channels

        self.lin = Group_Linear(in_channels, out_channels, groups,
bias=False)

        if bias:
            self.bias = Parameter(torch.Tensor(out_channels))
        else:
            self.register_parameter('bias', None)

        self.reset_parameters()

    def reset_parameters(self):
```

```
        self.lin.reset_parameters()
        init.zeros_(self.bias)

    def norm(self, adj: Tensor, add_loop):
        if add_loop:
            adj = adj.clone()
            idx = torch.arange(adj.size(-1), dtype=torch.long, device=adj.de-
vice)
            adj[:, idx, idx] += 1

        deg_inv_sqrt = adj.sum(-1).clamp(min=1).pow(-0.5)

        adj = deg_inv_sqrt.unsqueeze(-1) * adj * deg_inv_sqrt.unsqueeze(-2)

        return adj

    def forward(self, x: Tensor, adj: Tensor, add_loop=True):
        """
        Args:
            x (Tensor): [B, C, N, F]
            adj (Tensor): [B, G, N, N]
        """
        adj = self.norm(adj, add_loop).unsqueeze(1)

        # x: [B, C, G, N, F//G]
        x = self.lin(x, False)

        out = torch.matmul(adj, x)

        # out: [B, C, N, F]
        B, C, _, N, _ = out.size()
        out = out.transpose(2, 3).reshape(B, C, N, -1)

        if self.bias is not None:
            out = out.transpose(1, -1) + self.bias
            out = out.transpose(1, -1)

        return out


class DenseGINConv2d(nn.Module):

    def __init__(self, in_channels, out_channels, groups=1, eps=0,
train_eps=True):
        super().__init__()

        # TODO: Multi-layer model
        self.mlp = Group_Linear(in_channels, out_channels, groups,
bias=False)

        self.init_eps = eps
        if train_eps:
            self.eps = Parameter(Tensor([eps]))
        else:
            self.register_buffer('eps', Tensor([eps]))

        self.reset_parameters()

    def reset_parameters(self):
        self.mlp.reset_parameters()
        self.eps.data.fill_(self.init_eps)
```

```
    def norm(self, adj: Tensor, add_loop):
        if add_loop:
            adj = adj.clone()
            idx = torch.arange(adj.size(-1), dtype=torch.long, device=adj.de-
vice)
            adj[..., idx, idx] += 1

        deg_inv_sqrt = adj.sum(-1).clamp(min=1).pow(-0.5)

        adj = deg_inv_sqrt.unsqueeze(-1) * adj * deg_inv_sqrt.unsqueeze(-2)

        return adj

    def forward(self, x: Tensor, adj: Tensor, add_loop=True):
        """
        Args:
            x (Tensor): [B, C, N, F]
            adj (Tensor): [G, N, N]
        """
        B, C, N, _ = x.size()
        G = adj.size(0)

        # adj-norm
        adj = self.norm(adj, add_loop=False)

        # x: [B, C, G, N, F//G]
        x = x.reshape(B, C, N, G, -1).transpose(2, 3)

        out = torch.matmul(adj, x)

        # DYNAMIC
        x_pre = x[:, :, :-1, ...]

        # out = x[:, :, 1:, ...] + x_pre
        out[:, :, 1:, ...] = out[:, :, 1:, ...] + x_pre
        # out = torch.cat( [x[:, :, 0, ...].unsqueeze(2), out], dim=2 )

        if add_loop:
            out = (1 + self.eps) * x + out

        # out: [B, C, G, N, F//G]
        out = self.mlp(out, True)

        # out: [B, C, N, F]
        C = out.size(1)
        out = out.transpose(2, 3).reshape(B, C, N, -1)

        return out


class Dense_TimeDiffPool2d(nn.Module):

    def __init__(self, pre_nodes, pooled_nodes, kern_size, padding):
        super().__init__()

        # TODO: add Normalization
        self.time_conv = nn.Conv2d(pre_nodes, pooled_nodes, (1, kern_size),
padding=(0, padding))

        self.re_param = Parameter(Tensor(kern_size, 1))
```

```
    def reset_parameters(self):
        self.time_conv.reset_parameters()
        init.kaiming_uniform_(self.re_param, nonlinearity='relu')

    def forward(self, x: Tensor, adj: Tensor):
        """
        Args:
            x (Tensor): [B, C, N, F]
            adj (Tensor): [G, N, N]
        """
        x = x.transpose(1, 2)
        out = self.time_conv(x)
        out = out.transpose(1, 2)

        # s: [ N^(l+1), N^l, 1, K ]
        s = torch.matmul(self.time_conv.weight,
self.re_param).view(out.size(-2), -1)

        # TODO: fully-connect, how to decrease time complexity
        out_adj = torch.matmul(torch.matmul(s, adj), s.transpose(0, 1))

        return out, out_adj
```

网络整体架构代码：

```
from math import ceil
from layer import *
class GNNStack(nn.Module):
    def __init__(self, gnn_model_type, num_layers, groups, pool_ratio,
kern_size,
                 in_dim, hidden_dim, out_dim,
                 seq_len, num_nodes, num_classes, dropout=0.5, activa-
tion=nn.ReLU()):

        super().__init__()

        # TODO: Sparsity Analysis
        k_neighs = self.num_nodes = num_nodes

        self.num_graphs = groups

        self.num_feats = seq_len
        if seq_len % groups:
            self.num_feats += (groups - seq_len % groups)
        self.g_constr = multi_shallow_embedding(num_nodes, k_neighs,
self.num_graphs)

        gnn_model, heads = self.build_gnn_model(gnn_model_type)

        assert num_layers >= 1, 'Error: Number of layers is invalid.'
        assert num_layers == len(kern_size), 'Error: Number of kernel_size
should equal to number of layers.'
        paddings = [(k - 1) // 2 for k in kern_size]

        self.tconvs = nn.ModuleList(
            [nn.Conv2d(1, in_dim, (1, kern_size[0]), padding=(0, pad-
dings[0]))] +
            [nn.Conv2d(heads * in_dim, hidden_dim, (1, kern_size[layer + 1]),
padding=(0, paddings[layer + 1])) for
             layer in range(num_layers - 2)] +
```

```
            [nn.Conv2d(heads * hidden_dim, out_dim, (1, kern_size[-1]), pad-
ding=(0, paddings[-1]))]
        )

        self.gconvs = nn.ModuleList(
            [gnn_model(in_dim, heads * in_dim, groups)] +
            [gnn_model(hidden_dim, heads * hidden_dim, groups) for _ in
range(num_layers - 2)] +
            [gnn_model(out_dim, heads * out_dim, groups)]
        )

        self.bns = nn.ModuleList(
            [nn.BatchNorm2d(heads * in_dim)] +
            [nn.BatchNorm2d(heads * hidden_dim) for _ in range(num_layers -
2)] +
            [nn.BatchNorm2d(heads * out_dim)]
        )

        self.left_num_nodes = []
        for layer in range(num_layers + 1):
            left_node = round(num_nodes * (1 - (pool_ratio * layer)))
            if left_node > 0:
                self.left_num_nodes.append(left_node)
            else:
                self.left_num_nodes.append(1)
        self.diffpool = nn.ModuleList(
            [Dense_TimeDiffPool2d(self.left_num_nodes[layer],
self.left_num_nodes[layer + 1], kern_size[layer],
                                  paddings[layer]) for layer in range(num_layers
- 1)] +
            [Dense_TimeDiffPool2d(self.left_num_nodes[-2],
self.left_num_nodes[-1], kern_size[-1], paddings[-1])]
        )

        self.num_layers = num_layers
        self.dropout = dropout
        self.activation = activation

        self.softmax = nn.Softmax(dim=-1)
        self.global_pool = nn.AdaptiveAvgPool2d(1)

        self.linear = nn.Linear(heads * out_dim, num_classes)

        self.reset_parameters()

    def reset_parameters(self):
        for tconv, gconv, bn, pool in zip(self.tconvs, self.gconvs,
self.bns, self.diffpool):
            tconv.reset_parameters()
            gconv.reset_parameters()
            bn.reset_parameters()
            pool.reset_parameters()

        self.linear.reset_parameters()

    def build_gnn_model(self, model_type):
        if model_type == 'dyGCN2d':
            return DenseGCNConv2d, 1
        if model_type == 'dyGIN2d':
            return DenseGINConv2d, 1
```

```
    def forward(self, inputs: Tensor):

        if inputs.size(-1) % self.num_graphs:
            pad_size = (self.num_graphs - inputs.size(-1) % self.num_graphs)
/ 2
            x = F.pad(inputs, (int(pad_size), ceil(pad_size)), mode='con-
stant', value=0.0)
        else:
            x = inputs

        adj = self.g_constr(x.device)

        for tconv, gconv, bn, pool in zip(self.tconvs, self.gconvs,
self.bns, self.diffpool):
            x, adj = pool(gconv(tconv(x), adj), adj)

            x = self.activation(bn(x))

            x = F.dropout(x, p=self.dropout, training=self.training)

        out = self.global_pool(x)
        out = out.view(out.size(0), -1)
        out = self.linear(out)

        return out
```