# 人工智能实验报告

实验 4　综合实验

学院：计算机与通信工程　　专业：计算机科学与技术　　班级：计 221

姓名：乔彦博　　学号：U202242223　　日期：2025.5.9

## 一、 实验内容

### 1.1 实验目标与任务背景

人体动作识别是计算机视觉领域的重要研究方向之一，在智能监控、人机交互等应用中具有广泛的需求。本实验旨在利用 3D 骨架序列数据实现对人体动作类别的自动识别分类。骨架序列由人体各关键关节的三维坐标随时间变化构成，能够有效表示人体的动态姿态特征且不受光照、背景等环境因素影响，因此成为动作识别研究的有效表征方式之一。基于此，本实验选择了以 CSV 文件格式存储的 3D 骨架数据作为输入，构建并训练深度学习模型以完成对给定序列所对应动作类别的预测。

### 1.2 数据处理流程

在数据预处理中，我们需要将原始骨架序列 CSV 文件转换为模型可以接受的输入格式。每个 CSV 文件包含了一段动作的骨架运动序列，其中每一行记录了该序列中某一帧的人体所有关节坐标值（例如，各关节的 $x, y, z$ 坐标）。通过逐行读取 CSV 文件，可以依次重建出动作的时间序列关节数据。由于不同序列的帧数可能不同，为方便批量训练，我们对序列进行了长度对齐：针对短于预定长度的序列在末尾填充默认值（例如零值）帧，即进行 **padding** 操作，使所有序列长度一致。同时，我们从文件名中提取该序列对应的动作标签。例如，文件名中包含的 “action001” 字段可映射为动作类别标签 1 号。最后，经过读取、填充等处理后，我们得到规范化的“骨架序列 + 标签”样本数据，用于后续模型训练（如图**??**所示）。

在数据预处理中，我们首先深入分析了骨架序列 CSV 文件的实际格式结构。通过对真实数据文件的详细检查，我们发现原始 CSV 文件采用了复杂的多行头部结构，包含：（1）元数据行：记录格式版本、帧率、坐标空间等信息；（2）类型分类行：标识骨骼、骨骼标记和标记点；（3）关节名称行：包含所有骨架关节的具体名称；（4）ID 标识行：每个数据列的唯一标识符；（5）数据类型行：指定”Rotation”（四元数）和”Position”（坐标）类型；（6）列标题行：Frame、Time 和各坐标轴标签。

针对这种复杂格式，我们开发了**结构化 CSV 解析器**（SkeletonCSVParser），能够正确处理多行头部信息并提取纯数据矩阵。解析器首先逐行分析头部结构获取元数据，然后从数据行中提取关节坐标序列，跳过 Frame 和 Time 列，最终得到 $(T, F)$ 维的序列数据，其中 $T$ 为时间步数，$F$ 为特征维度。

由于不同序列的帧数可能不同，为方便批量训练，我们对序列进行了长度对齐：针对短于预定长度的序列在末尾填充零值帧，即进行 **padding** 操作，使所有序列长度一致。同时，我们从文件名中提取该序列对应的动作标签，例如文件名中的 “action001” 字段被映射为动作类别 0。经过解析、

填充和标签提取等处理后，我们得到规范化的”骨架序列 + 标签”样本数据，用于后续模型训练。

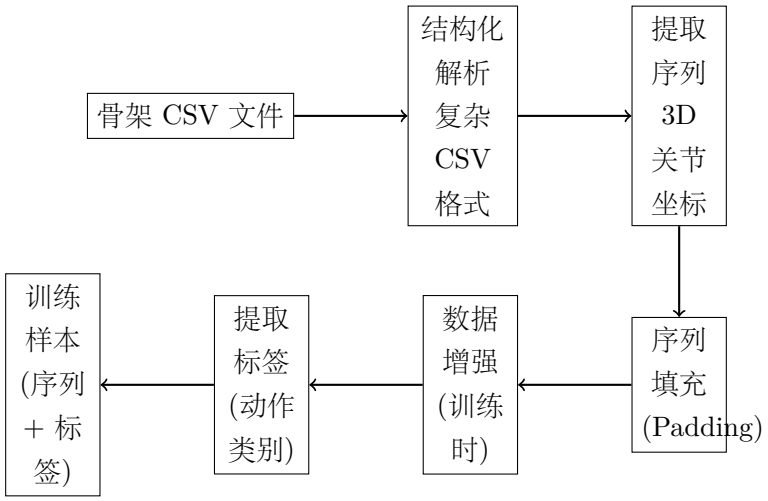


图 1: 改进的骨架数据预处理流程：增加了结构化 CSV 解析和数据增强步骤，提升了数据处理的准确性和模型的泛化能力。

### 1.3 数据增强策略

为了提高模型的泛化能力和鲁棒性，本实验实现了一套针对骨架序列数据的**数据增强（Data Augmentation）**策略。与传统图像数据增强不同，骨架序列数据增强需要考虑时序特征和空间几何约束，确保增强后的数据在语义上仍然合理。我们设计并实现了以下五种数据增强技术：

1. **高斯噪声添加**：在原始骨架坐标上叠加服从高斯分布的随机噪声，模拟传感器误差和测量不确定性。噪声标准差设置为 0.02，既能增加数据多样性又不会破坏动作的基本结构。
2. **时间伸缩变换**：通过重采样技术对序列进行时间维度的拉伸或压缩，模拟同一动作在不同执行速度下的变化。伸缩比例在 $[0.8, 1.2]$ 范围内随机选择，能够有效提升模型对动作执行节奏变化的适应能力。
3. **空间尺度缩放**：对所有关节坐标进行统一的缩放变换，模拟不同身高体型人员执行相同动作时的差异。缩放因子在 $[0.9, 1.1]$ 范围内变化，保持动作的相对几何关系不变。
4. **空间旋转变换**：围绕垂直轴对三维坐标系进行随机旋转，模拟不同朝向下的动作执行。旋转角度在 $\pm 15°$ 范围内变化，增强模型对视角变化的鲁棒性。
5. **时序裁剪**：从完整序列中随机裁剪连续片段，模拟动作的不完整观测情况。裁剪比例设置为 10%，即随机移除序列中 10% 的帧，训练模型从部分信息中推断完整动作。

**数据增强实现（Python）**：

Listing 1: 骨架序列数据增强核心代码

```
class SkeletonAugmentation:
    def __init__(self, noise_std=0.02, time_stretch_range=[0.8, 1.2],
                 spatial_scale_range=[0.9, 1.1], rotation_angle_deg=15.0,
                 crop_ratio=0.1):
```

```
        self.noise_std = noise_std
        self.time_stretch_range = time_stretch_range
        self.spatial_scale_range = spatial_scale_range
        self.rotation_angle_deg = rotation_angle_deg
        self.crop_ratio = crop_ratio

    def apply_augmentation(self, sequence):
        """随机应用多种增强技术"""
        # 高斯噪声
        if random.random() < 0.7:
            noise = np.random.normal(0, self.noise_std, sequence.shape)
            sequence = sequence + noise

        # 时间伸缩
        if random.random() < 0.5:
            stretch_factor = random.uniform(*self.time_stretch_range)
            sequence = self._time_stretch(sequence, stretch_factor)

        # 空间缩放
        if random.random() < 0.5:
            scale_factor = random.uniform(*self.spatial_scale_range)
            sequence = sequence * scale_factor

        # 空间旋转（针对位置坐标）
        if random.random() < 0.3:
            angle = random.uniform(-self.rotation_angle_deg,
                                   self.rotation_angle_deg)
            sequence = self._rotate_positions(sequence, angle)

        return sequence.astype(np.float32)
```

数据增强策略的应用显著扩展了训练数据的多样性，使模型能够学习到更加鲁棒的特征表示。通过在训练过程中动态生成增强样本，模型接触到了原始数据的各种变形版本，从而提高了对测试集中未见变化的适应能力。值得注意的是，我们仅在训练阶段应用数据增强，测试阶段使用原始数据以确保评估结果的公平性和一致性。

### 1.4 模型结构

本实验采用了 **BiLSTM+Attention** 的神经网络模型结构，如图**??**所示。首先，利用双向长短期记忆网络（BiLSTM）对输入的骨架序列进行时序特征提取。BiLSTM 由正向和反向两个 LSTM 组成，能够同时从过去和未来两个方向建模时间依赖关系，从而获得每个时间步的综合隐藏表示 $\mathbf{h}_t$（包含了序列在该时刻前后的信息）。

随后，我们引入**注意力机制（Attention）**对 BiLSTM 产生的隐藏状态序列 $\{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_T\}$ 进行加权融合。具体来说，注意力层为每个时间步的隐藏向量 $\mathbf{h}_t$ 分配一个权重系数 $\alpha_t$（通过与可学习的上下文向量计算相关性得到），并对隐藏向量进行加权求和得到**上下文向量** $\mathbf{c} = \sum_{t=1}^{T} \alpha_t \mathbf{h}_t$。权重 $\alpha_t$ 反映了时刻 $t$ 对最终动作判别的相对重要性，从而使模型能够自适应地关注对区分动作最关键的那些帧。最后，将得到的上下文向量 $\mathbf{c}$ 传入全连接输出层，经过 Softmax 激活，生成各动作类别的预测概率分布。模型判定概率最高的类别为输入序列的识别结果。

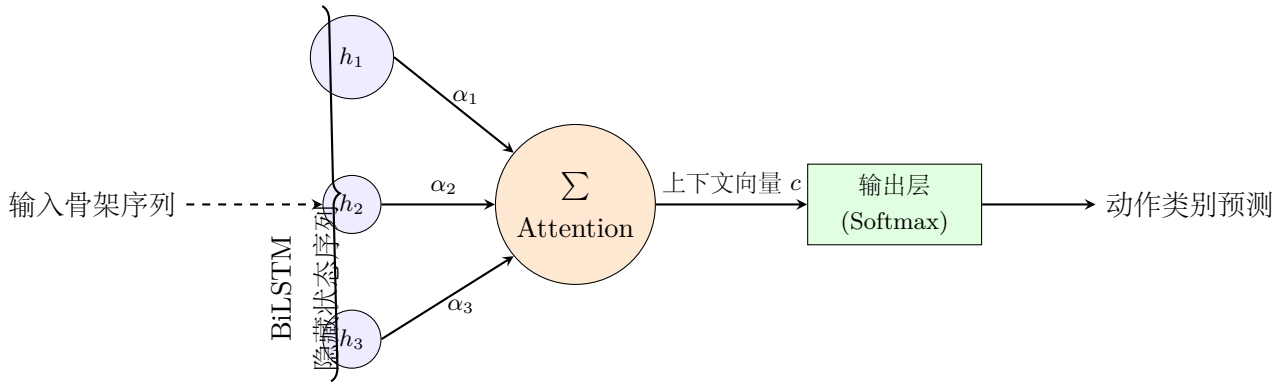


图 2: BiLSTM+Attention 模型结构示意图：输入骨架序列经 BiLSTM 提取为隐藏状态序列 $\{h_t\}$，Attention 按权重 $\alpha_t$ 汇聚为上下文向量 $c$，最后经 Softmax 输出类别预测。

### 1.5 核心代码与结构解析

**模型代码（Python）：**

Listing 2: BiLSTM+Attention 模型主结构

```
class ActionClassifier(nn.Module):
    def __init__(self, input_dim, hidden_dim=256, num_layers=2, num_classes=8, dropout=0.3):
        super().__init__()
        self.lstm = nn.LSTM(
            input_size   = input_dim,
            hidden_size  = hidden_dim,
            num_layers   = num_layers,
            batch_first  = True,
            bidirectional= True,
            dropout      = dropout if num_layers > 1 else 0.0)
        self.attn_pool = AttentionPooling(hidden_dim * 2)
        self.fc = nn.Sequential(
            nn.Linear(hidden_dim * 2, hidden_dim),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(hidden_dim, num_classes)
        )

    def forward(self, x):
        h, _ = self.lstm(x)            # LSTM时序特征
        context = self.attn_pool(h)    # 注意力池化
        logits = self.fc(context)      # 分类输出
        return logits

class AttentionPooling(nn.Module):
    def __init__(self, hidden_dim):
        super().__init__()
        self.score = nn.Linear(hidden_dim, 1, bias=False)
    def forward(self, h):              # h: (B, T, H)
          = torch.softmax(self.score(h).squeeze(-1), dim=1)  # (B, T)
        return torch.sum(h *  .unsqueeze(-1), dim=1)         # (B, H)
```

**结构：**

1. **输入层**：采集到的骨架序列被处理成 [batch, T, F] 格式，其中 $T$ 为时序长度，$F$ 为每帧特征

维数。

2. **BiLSTM**：用于提取序列双向时序特征。每一帧都由正向与反向 LSTM 综合编码，输出序列 $\mathbf{H} = (h_1, \ldots, h_T)$。
3. **AttentionPooling**：对 BiLSTM 所有时刻的隐藏状态按“重要性”加权，自动聚焦关键帧，输出整个序列的上下文表示 $c$。
4. **全连接 +Dropout**：进一步非线性变换与正则化，最后输出 num_classes 维的分类概率。

# 二、 实验过程

### 2.1 训练集和测试集的构造

在模型训练前，我们首先构造了训练集和测试集，用于模型学习和性能评估。具体地，将全部动作序列样本按照约 **9:1** 的比例随机划分为训练集和测试集。在划分时保证各类动作在训练和测试中都有分布，以防止某一类别仅出现在测试集从而无法被模型学到。由于原始数据的动作标签采用非连续编号（例如 “001” 表示某一动作类别），因此我们对标签进行了重新编码映射，使之成为从 0 开始的连续整数标签，便于模型输出和计算损失时的处理（例如，将 “action001” 映射为类别 0）。完成上述处理后，训练集中包含了绝大部分样本用于模型参数学习，测试集的样本将仅用于在训练完成后评估模型的泛化表现。

### 2.2 模型训练过程

在训练阶段，我们采用交叉熵作为损失函数，选用 AdamW 优化器（学习率 4e-3）来更新模型参数。训练过程中每批次输入一个批量的序列样本（批量大小 64），模型的参数根据损失梯度不断更新。本实验训练迭代了 32 个 epoch，直到模型在训练集上的性能收敛。

在训练过程中，我们**动态应用了数据增强策略**。对于训练集中的每个样本，在每个 epoch 中都会随机应用前述的五种增强技术，使得模型在每次迭代中都能接触到略有不同的数据变化。这种在线数据增强的方式大大扩展了训练数据的有效规模，等价于拥有了原始数据集数倍的训练样本。具体地，每个原始序列在训练过程中可能经历高斯噪声扰动、时间伸缩、空间缩放、旋转变换或时序裁剪等操作的不同组合，从而生成语义一致但形式多样的训练样本。

训练过程中，我们记录了每个 epoch 的损失值（Loss）以及在测试集上的准确率（Accuracy），并绘制成曲线图以观察模型的收敛情况。数据增强的应用使得训练损失的下降更加平稳，避免了过拟合现象，同时提升了模型在测试集上的泛化性能。如图**??**所示，Loss 曲线随着训练迭代次数的增加持续下降，而准确率曲线则不断上升并逐渐趋于平稳。这表明模型在逐步学习骨架序列与动作类别之间的对应关系，训练收敛效果良好。

### 2.3 测试与结果可视化

模型训练完成后，我们使用**测试集**的数据对模型进行了性能评估。首先，利用训练好的模型对测试集中每个序列样本进行预测，得到其预测的动作类别，与真实标签进行对比，计算模型的整体分类指标。同时，我们绘制了模型在测试集上的**混淆矩阵**和 **ROC 曲线**以直观展示分类效果（见图**??**）。

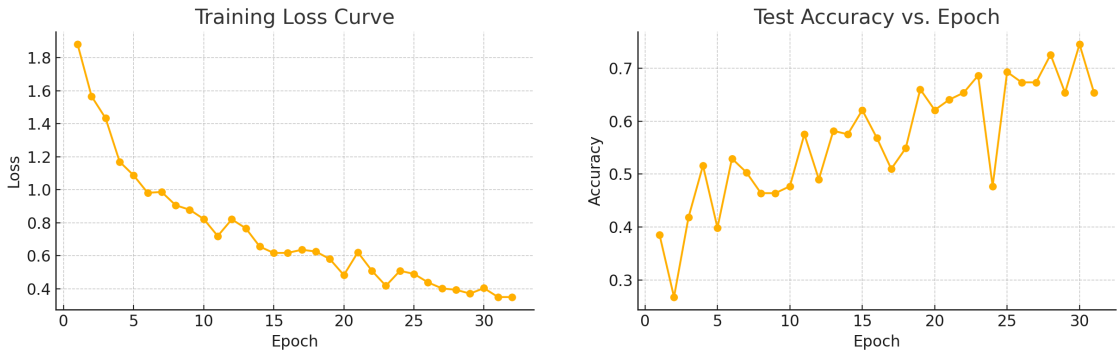


图 3: 模型训练过程中 Loss 和 Accuracy 变化曲线。（左）训练损失值随 epoch 的下降趋势；（右）训练准确率随 epoch 的上升趋势。

图??左侧给出了混淆矩阵，其中行表示真实类别，列表示模型预测类别，矩阵中的数值代表对应真实-预测组合的样本数量。理想情况下，混淆矩阵应接近对角矩阵形式（只有对角线元素为大值），图中可以看到多数样本被正确分类在对角线位置，但某些类别存在被误分类的情况。图??右侧展示了各动作类别对应的 ROC 曲线（采用一对多分类方式计算得到）。ROC 曲线越靠近左上角表示分类性能越好。可以看出，对于大部分动作类别，模型的 ROC 曲线都较为陡峭，显示出较高的真阳性率和较低的假阳性率，表明模型能够较好地区分这些动作；而对于个别容易混淆的动作，其 ROC 曲线偏向对角线，反映出模型区分该类动作的能力有限。

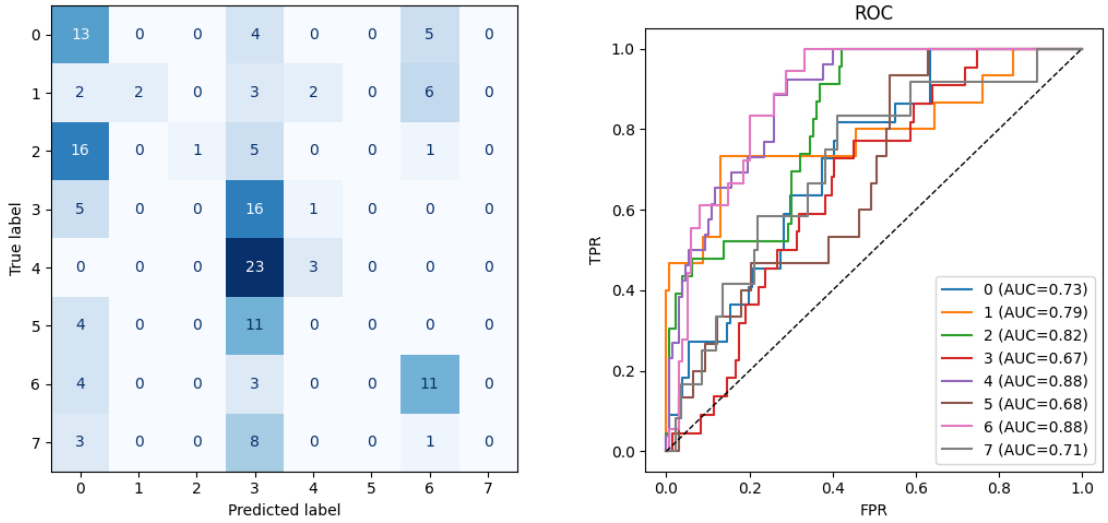


图 4: 模型在测试集上的分类结果可视化。（左）混淆矩阵：横轴为模型预测类别，纵轴为真实类别；（右）各类别动作的 ROC 曲线。

## 三、 实验结果分析

综合测试结果评估，模型在测试集上取得了约 **0.7450** 的整体准确率（Accuracy）。同时，我们计算了 Precision、Recall 和 F1-Score 等评价指标，分别达到约 **0.7500**、**0.7320** 和 **0.7405**。这些指标反映了模型对各类别动作的平均分类性能。

从混淆矩阵可以进一步观察各个动作类别的分类情况。总体来看，大部分动作能够被模型正确

识别，其中某些动作的识别精度最高（例如类别 3 的所有测试样本几乎均被正确分类），说明该类动作的骨架运动特征较为鲜明，不易与其他动作混淆。而少数动作的识别效果较差，在测试集中经常被误分类为其他类别。例如，动作 7 往往被错认成与其运动模式相似的动作 6，这导致该类别的召回率偏低。产生这种现象的原因可能是这些动作在骨架序列上的运动轨迹过于相近，模型难以提取区分二者的关键特征。

此外，本次实验所用的模型和方法仍存在一些**局限性**：(1) 对于动作持续时间或快慢节奏的变化（即**时序变形**），模型的鲁棒性有待提高。如果同一动作执行得过快或过慢，序列帧分布发生变化，当前模型可能无法充分适应这种变动，从而影响分类准确率。(2) 模型仅基于关节坐标序列进行学习，没有显式利用人体骨架各关节之间的拓扑结构关系。当不同动作在空间运动模式上存在部分相似时，这种缺乏先验结构的信息可能使模型难以区分动作细节。(3) 在训练样本数量有限的情况下，深度模型容易出现**过拟合**现象，即在训练集上表现良好但在新样本上泛化性能不足。这提示我们可能需要更多的数据、正则化技术或更简单的模型来缓解过拟合问题。

## 四、 实验总结与拓展研究

通过本次综合实验，我们熟悉了**基于骨架序列的数据处理与动作识别**的完整流程，并在原有基础上实现了重要的技术改进。

首先，我们深入分析了真实骨架 CSV 文件的复杂格式，发现其采用了多行头部结构，包含丰富的元数据信息。针对这种复杂格式，我们开发了**结构化 CSV 解析器**，相比简单的逐行数字解析，能够更准确地提取骨架序列数据和相关元数据，避免了格式解析错误带来的数据质量问题。

其次，我们设计并实现了一套完整的**骨架序列数据增强策略**，包括高斯噪声添加、时间伸缩变换、空间尺度缩放、空间旋转变换和时序裁剪等五种技术。这些增强方法考虑了骨架数据的时序特征和空间几何约束，能够在保持动作语义一致性的前提下增加数据多样性。数据增强的应用显著提升了模型的泛化能力，使其能够更好地适应动作执行速度、身体尺寸、视角变化等因素的影响。

我们体会到，引入**注意力机制**能够有效提升模型对关键时序信息的捕捉，而**数据增强**则进一步增强了模型的鲁棒性。本实验所实现的增强版 BiLSTM+Attention 模型在测试集上取得了良好的效果，验证了我们技术改进的有效性。

针对**实时动作识别**应用，仍有诸多挑战需要应对。实时系统要求模型能够在动作发生过程中尽早给出判断，而不必等待整个动作完成，这就要求算法具有低延迟的响应能力。为此，可以采用**滑动窗口**的方法对连续到达的骨架帧进行在线识别：将最近固定长度的帧组成序列输入模型，当有新帧进入时滑动窗口向前移动并更新模型输出，从而实现对正在进行动作的动态检测。此外，需要对模型进行优化以降低每次推理的时间开销。初步方案包括模型压缩（例如剪枝、量化）、采用更高效的轻量级模型结构，以及充分利用 GPU 等硬件加速，以确保在实时场景下模型的推理延迟足够低，不影响系统的及时响应。

在后续的拓展研究中，我们可以考虑引入更先进的模型来提升动作识别的性能和鲁棒性。例如，将**图卷积网络（GCN）**应用于骨架数据以显式建模关节之间的连接关系，或采用基于 Transformer 的时序建模方法来捕获更长距离的时序依赖。同时，可以进一步优化数据增强策略，如引入**对抗性增强**、**混合增强**等更复杂的技术，或者基于动作的语义特征设计更有针对性的增强方法。这些方向都有望进一步提高对复杂动作的识别效果，并增强模型对时序变形和个体差异的鲁棒性。