第 3 章　动作识别

3.1 任务介绍

动作识别是计算机视觉和人工智能领域的重要研究方向之一，旨在通过分析视频或传感器数据，自动识别和理解人体的动态行为。随着深度学习的发展以及硬件计算能力的提升，动作识别已从传统的手工特征提取方法逐步过渡到以神经网络为核心的端到端学习方法。根据输入数据类型的不同，当前主流的动作识别方法分为多种，本文主要介绍基于 RGB 视频的视觉方法和基于点云数据的三维几何方法，如图 3.1 所示，a) 为 RGB 动作识别，b) 为点云动作识别[1]。两者在特征建模策略与应用场景上各具优势。

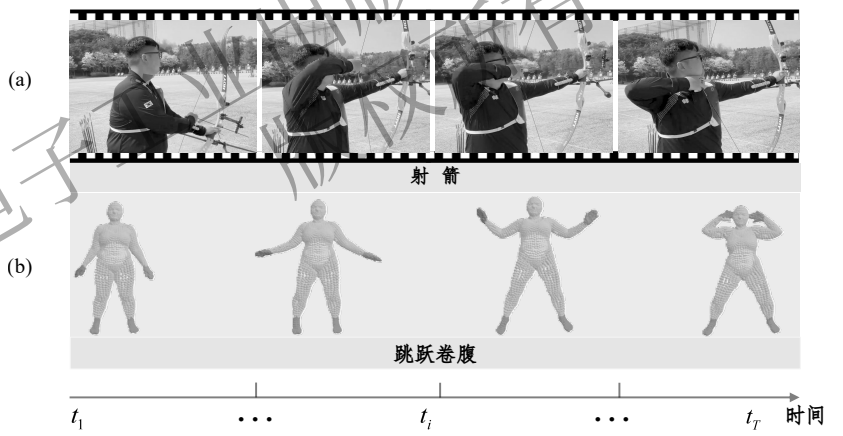

图 3.1　动作识别任务示意图

RGB 动作识别是指利用普通彩色视频数据，基于连续帧中的颜色和外观信息，来判断和理解人体所执行的动作。该任务通常依赖于视频序列中可见的形体变化、姿态转移，以及背景环境的上下文线索，对动作进行分类或分段。点云动作识别是指基于三维点云序列，对人体在空间中的动态行为进行理解与识别。点云数据由传感器获取的空间坐标构成，直接反映人体在三维空间中的形状与动作轨迹。与 RGB 数据相比，点云数据更注重动作的几何结构、空间关系，以及深度信息，对光照和背景变化不敏感，因此更适合在复杂或结构化环境中进行高精度动作分析。

近年来，对于动作识别任务的研究持续深入，呈现出多模态融合、时空建模优化和高

效端到端学习等发展趋势。借助深度学习技术的不断演进，动作识别在特征表达能力、模型泛化能力和识别精度等方面取得进展。作为理解人体动态行为的核心技术之一，动作识别正被广泛应用于智能安防、行为分析、人机交互、医疗健康等多个实际场景，已经成为计算机视觉与人工智能领域的重要研究方向。

3.2 方法总览

3.2.1 概况

RGB 动作识别作为最早发展的一类视觉动作识别方法，起初依赖手工设计的特征和传统分类器进行识别。随着深度学习的兴起，研究重心逐步转向卷积神经网络和三维卷积网络，能够自动从视频中提取时空特征。随后，双流网络引入光流信息增强对运动特征的建模，而时序建模方法进一步提升对长时间动作的识别能力。近年来，自注意力机制与 Transformer 架构的引入改善了对复杂时序依赖的建模能力，使得 RGB 动作识别在多个公开数据集上取得突破性进展。目前，RGB 动作识别方法可分为四类 [如图 3.2 (a) 所示]：基于双流卷积神经网络的动作识别、基于三维卷积神经网络的动作识别、基于循环神经网络的动作识别和基于 Transformer 的动作识别。

点云动作识别的核心目标是从随时间变化的点云帧中里提取出描述动作本质的信息，从而判断动作的类型，例如行走、奔跑、挥手、跳跃等。早期的点云动作识别方法多依赖手工设计的特征（如轨迹、点密度、几何关系等），结合传统分类器进行识别，但这些方法在面对复杂动作、视角变化或部分遮挡时往往表现不佳。随着深度学习的发展，尤其是 PointNet[2] 及其后续模型的提出，使得神经网络可以直接处理无序的点集，从而推动点云动作识别进入数据驱动建模的新时代。从监督范式的角度来讲，当前点云动作识别任务主要分为全监督点云动作识别、半监督点云动作识别和自监督点云动作识别，如图 3.2 (b) 所示。

3.2.2 基于双流卷积神经网络的动作识别

基于双流卷积神经网络的动作识别方法利用两个独立的 CNN 流来处理视频数据，包含空间流和时序流两个维度的数据。空间流处理 RGB 图像，用于捕捉空间布局和外观信息；时序流处理连续帧的光流图（或差分图像），用于捕捉动作的动态信息。两个数据流分别提取空间和时序特征，然后将这些特征融合，以实现更准确的动作识别。这种方法有效利用视频中的空间和时序信息，可以较好地适配动作识别任务。

为了捕捉来自静止帧和帧间运动的互补外观信息，Simonyan 等人提出一种由空间流和时序流组成的双流 CNN 模型[3]，其中空间流负责捕捉独立帧的表面信息（关于物体、场景等），即每个 RGB 帧的外观信息；而时序流指帧间的光流，携带着帧之间的运动信息。相应的，所提出的网络结构由两个深度网络组成，分别处理时间与空间的维度，最终将输入的双流信息通过求平均方法或以 L2 正则化的 Softmax 输出作为特征，训练多分类线性 SVM 的方法进行分类得分的融合。在此基础上，许多后续研究对双流模型进行改进。为了将 ConvNet 融入进时间与空间，Feichtenhofer 等人研究多种融合策略[4]，并提出在最后一

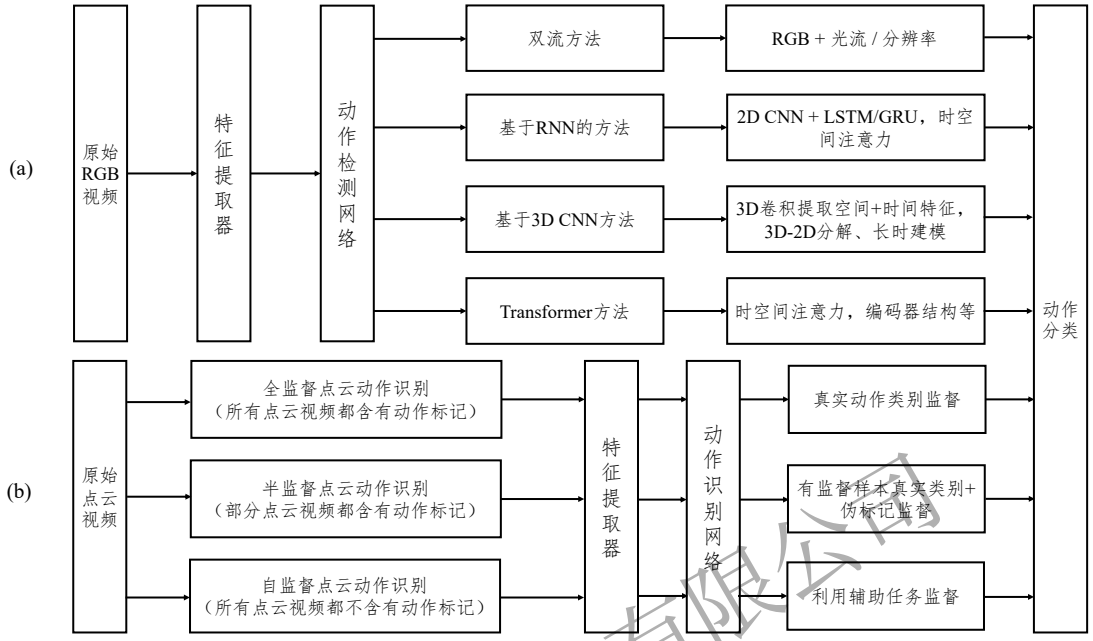

图 3.2　动作识别方法分类示意图

个卷积层中融合空间流和时序流，从而在保证准确性的同时减少参数量；双流 CNN 方法存在一个缺陷，即无法对长时间的视频进行有效建模，仅能从连续的几帧中提取有限的时序上下文信息。为了解决这一问题，Wang 等人提出时序段网络（Temporal Segment Networks，TSN）[5]。该方法将整个视频划分为 K 个部分，从每个部分中随机选取一个短片段，并对这些片段分别应用双流 CNN 模型，最终融合各片段提取到的特征以获取整体视频的表达。为了提升模型学习效率，TSN 还引入 RGB 差异图和变形光流场作为额外的输入模态。在此基础上，为了更好地在不同抽象层次上建模空间与时间结构之间的关联关系，后续工作中进一步提出时空金字塔网络（Spatiotemporal Pyramid Network，SPN）[6]，通过金字塔结构整合多尺度的时空特征，增强空间与时间之间的相互作用。尽管这些方法多采用光流图来捕获运动信息，但光流图的计算代价较高。因此，在更进一步的研究中，研究者推广 RGB 差异的思路，设计时间差异模块（Temporal Difference Module，TDM）[7]，用于对视频中的运动信息进行高效建模，其核心做法是计算相邻帧之间的差异，以替代传统的光流提取方式。

虽然基于双流网络的架构能在时空特征上应用 CNN 来捕捉各自数据流的语义信息，但是存在很多弊端。双流网络需要分别处理空间和时间信息，导致计算开销增大，并且预计算的光流图在计算和存储方面成本较高；双流网络的空间流通常基于静态帧进行处理，时间流则依赖光流来获取运动信息。对于动态变化比较快或运动复杂的场景，光流的质量可能不足，导致时间流的学习能力受到限制，从而影响整个模型的表现；由于双流网络中空间流和时间流是独立处理的，缺乏跨流的信息共享，无法充分挖掘空间信息和时间信息之间的交互，从而限制模型的表达能力。

3.2.3 基于三维卷积神经网络的方法

预计算光流计算量大，存储要求高，不利于大规模训练或实时部署。从概念上理解视频的一个简单方法是把它视为一个具有两个空间维度和一个时间维度的三维张量。因此，产生了基于三维卷积神经网络（3D Convolutional Neural Network，3DCNN）来进行动作识别的方法。与传统的二维卷积神经网络不同，3DCNN 通过直接处理视频数据的时空信息来实现对动作的有效识别，在卷积操作中同时考虑空间维度和时间维度，能够捕捉连续帧之间的动态变化。这种方法通过在视频序列上滑动三维卷积核，提取出包含运动特征的深层次时空特征，从而提高动作识别的准确性和鲁棒性。

为了解决视频动作识别中时空信息建模不足的问题，研究者开始探索能够同时捕捉空间和时间特征的深度模型。为了更好地学习时空特征，Tran 等人提出三维卷积神经网络 C3D[8]，将视频帧序列作为一个整体输入网络，通过 3D 卷积操作在时间和空间两个维度上同时提取特征，实现端到端的动作识别。随后，为了更好地利用已有的 2D 图像预训练模型，同时提升对时空特征的建模能力，Carreira 等人提出膨胀 3D 卷积网络 I3D[9]，将 2D 卷积核膨胀为 3D 卷积核，并引入双流结构，分别处理 RGB 帧和光流信息，从而增强模型对空间结构和运动信息的联合建模能力。

在提升网络深度和性能方面，由于残差结构在缓解梯度消失问题上的优势，Hara 等人将 ResNet 架构扩展至三维卷积，提出 3D 残差网络 R3D[10]，该方法在不依赖光流预处理的前提下，实现对视频数据的高效建模，具有更轻量的计算成本。由于传统的 3D 卷积网络在计算效率和模型规模上存在局限性，Feichtenhofer 等人提出快慢网络（SlowFast）[11]，设计两个互补的分支来捕捉视频中的多尺度时序特征，慢速（Slow）分支负责捕捉长时特征，使用较大的时间步长，快速（Fast）分支负责捕捉短时特征，使用较小的时间步长进行密集采样，两个分支的特征最终进行融合，从而在保持高精度识别的同时降低计算成本。

基于 3D 卷积的动作识别方法能够同时捕获空间和时间特征，从而有效地学习视频帧之间的动态变化和物体运动信息。与传统的 2D 卷积神经网络不同，3DCNN 在空间维度的基础上增加时间维度，使得网络能够处理时序信息，从而提高动作识别的准确性。此外，3D 卷积网络的端到端训练方式使得模型可以从原始视频数据中自动学习最有效的特征，避免人工提取特征的复杂性，可以处理较为复杂的时空模式。

3.2.4 基于循环神经网络的动作识别

视频除空间维度外，最大的痛点是时间序列问题。循环神经网络（Recurrent Neural Network，RNN）在 NLP 领域中取得了巨大成功，非常适合处理序列数据。基于 RNN 的动作识别方法通过捕捉视频帧之间的时序依赖关系来识别动作，RNN 中的长短期记忆网络（Long ShortTerm Memory，LSTM）和双向长短期记忆网络（Bi-Directional Long ShortTerm Memory，Bi-LSTM）能够有效地建模长期依赖和上下文信息，从而在动作识别任务中达到更高的准确性。这类方法尤其适用于需要理解动作演变过程的应用场景。

为了增强模型对视频中时间动态的建模能力，研究者开始尝试将循环神经网络 RNN 引入动作识别任务中。在此背景下，Ng 等人提出一种将预训练的 2D 卷积神经网络用于特征

提取，并将提取的帧级特征送入堆叠的 LSTM 结构进行时序建模的方法[12]，从而实现对动作的识别。为进一步提升模型对长期依赖关系的理解能力，Donahue 等人设计了一种能够在空间和时间两个维度上学习联合表示的网络结构[13]，通过引入非线性映射增强对复杂时序关系的建模效果。

考虑到视频信息不仅具有前向的时间演化过程，还包含后向的关联依赖，He 等人提出采用双向 LSTM 结构[14]，充分利用双向时序信息以提升识别性能。在减小模型复杂度的同时保持建模能力方面，部分研究[15-16] 采用参数更少但性能相近的门控循环单元（Gated Recurrent Unit，GRU）以实现更加高效的动作识别。此外，为了提高模型对关键帧和重要特征的关注能力，Meng 等人将注意力机制集成到基于 LSTM 的网络结构中[17]，有效增强模型的判别能力和识别准确率。

3.2.5 基于 Transformer 的方法

基于 Transformer 的动作识别方法利用注意力机制来捕捉视频中的长期依赖关系和时空特征，通过将视频帧序列输入到 Transformer 编码器中，有效地处理视频中复杂的时空关系。相较于传统的 3D 卷积网络，基于 Transformer 的方法在计算效率和灵活性上更具优势，特别是在处理长序列数据时，能够提升动作识别任务的性能。

在视频领域，Bertasius 等人将 ViT（Vision Transformer）扩展至视频任务[18]，先将视频切分成一系列帧级补丁，并引入分割注意力机制，通过时空注意力机制捕捉视频中的时空特征。另外，研究人员通过应用多种注意力 (包括空间注意力、时序注意力，以及时空注意力)，使得模型能够灵活应对不同类型的视频数据，同时解决只使用时空注意力所导致的计算复杂度高的问题。当前许多工作都致力于减少计算复杂度和内存成本，如 RegionViT[19]。在视频理解和动作识别领域，传统的监督学习方法依赖于大量的标记数据，但对视频进行数据标记的成本过高，时间开销较大；现有的自监督学习方法虽然能够在一定程度上缓解对标记数据的依赖，但是在视频数据上的表现仍有待提高。为此，Tong 等人和 Wang 等人分别提出基于 Transformer 的视频特征提取器（Video Masked Auto Encoders，VideoMAE）[20] 和 VideoMAEv2[21]，利用掩码掩蔽部分视频帧并让模型预测被掩蔽的部分，使得 VideoMAE 能够在无需大量标记数据的情况下学习到有用的时空特征。

基于 Transformer 的动作识别方法能有效地学习到视频中复杂的时空关系，相较于传统的 3D 卷积神经网络，在计算效率和灵活性上更具优势，特别是在处理长序列数据时，能够提升动作识别性能。

3.2.6 全监督点云动作识别

全监督点云动作识别是指在大量带有动作类别标记的点云序列数据的指导下，对三维动作进行建模和分类的任务。这一研究方向近年来得到快速发展，特别是在 3D 传感器逐渐普及、点云数据采集更为便捷的背景下，研究者们开始关注如何在全监督学习框架下更高效、更准确地理解点云序列中的时空信息，从而完成动作识别。现阶段，全监督点云动作识别任务主要采用基于卷积 + Transformer、基于追踪 + PointNet 两种方法。

3.2.6.1 基于卷积 + Transformer 的点云动作识别

近几年，随着 Transformer 在自然语言处理领域的广泛应用，不少研究者开始尝试将其应用到视觉领域。由于 Transformer 无须按照时间顺序逐步处理序列，而是通过并行计算和全局注意力机制捕捉序列中任意位置之间的依赖关系，从而在处理长距离依赖和大规模数据方面具有更高的效率和表现力，因此十分契合点云数据无序性的特点，Fan 等人提出点 4D Transformer（Point 4D Transformer，P4-Transformer）[22]，成功地将 Transformer 应用到点云序列中。该模型首先选取关键帧构建时空邻域，并使用点 4D 卷积提取邻域特征，然后将提取的局部区域特征输入到 Transformer 中提取全局特征，最后将融合后的局部-全局特征输入到多层感知机中得到分类结果。然而，点云序列在空间上是连续的，在时间上是离散的，直接将两个维度融合处理会导致时空关系紊乱，Fan 等人提出点时空卷积网络（Point Spatio-Temporal Convolution Network，PSTCN）[23]，该模型将时间维度和空间维度解耦并引进 PST 卷积提取特征。为了融合点时空卷积和 Transformer 的优点，Fan 等人用 PST 卷积代替 P4-Transformer 中的点 4D 卷积，提出点时空 Transformer（Point SpatioTemporal-Transformer，PST-Transformer）[24]。

3.2.6.2 基于追踪 + PointNet 的点云动作识别

PointNet 是首个直接处理原始点云数据的深度学习框架，通过对每个点独立处理并使用对称函数聚合，实现对点云整体形状的学习与分类。而 PointNet ++[25] 在 PointNet 的基础上引入分层结构，结合局部邻域信息对点云进行逐层特征提取，能够更好地捕捉细粒度的空间结构，已经作为点云特征提取器被广泛应用到各种点云下游任务中。为了提取点云序列的时空特征，Liu 等人提出 MeteorNet[26]，该模型使用直接聚合（Direct grouping）和流聚合（Chained-flow grouping）两种聚合方式构建点云序列的时空局部区域，然后针对每一个局部区域使用 PointNet++ 提取特征并分类。然而，直接聚合随着时间的变化查询半径变大，追踪精度随之下降，而流聚合需要额外训练模型。Wang 等人提出三维动态体素网络（3D Dynamic Voxel，3DV）[27] 用于点云动作识别，首先将点云体素化后使用 Temporal Rank Pooling 技术提取三维动态体素，为了消除背景无关信息的影响，使用 YOLOv3-Tiny[28] 检测人的位置以便提取特征，然后使用一个基于 PointNet++ 的双流深度学习模型提取运动特征和几何特征用于最后的分类。然而，体素化的过程极其消耗计算资源，为了在减少计算量的同时减少追踪误差，Ben 等人提出一种基于 patch 追踪的人体动作识别方法[1]，首先在第一帧点云中选取若干关键点，然后在之后的每一帧构建 patch，最后基于 patch 的轨迹提取动作特征。相较于单纯的点追踪，patch 追踪不仅能更好地描述人体运动，还能拥有比点追踪更高的容错率。

3.2.7 半监督点云动作识别

半监督点云动作识别是一种结合少量带标记数据与大量无标记数据，学习点云动作表示并完成分类的任务。该方法旨在解决点云数据标记成本高、全监督学习对大规模人工标记依赖强的问题，尤其适用于实际应用中标记稀缺但无标记数据丰富的场景。鉴于此，Chen

等人提出一种掩码伪标记自编码器（Masked Pseudo-Labeling AutoEncoder，MAPLE）[29] 用于半监督点云动作识别，该模型首先使用带标记样本训练出一个初始模型，然后利用未标记样本通过掩码重构的方式微调模型。

3.2.8 自监督点云动作识别

自监督点云动作识别是一种无须人工标记，通过设计辅助任务引导模型自主学习时空特征，从而实现动作识别的学习范式。与全监督或半监督方法相比，自监督方法完全摆脱对人工标记数据的依赖，降低数据准备成本，尤其适用于点云数据这种难以标记、结构复杂的数据形式。Wang 等人提出一种自监督点云视频分析技术[30]，通过使用片段顺序恢复的辅助任务学习点云视频的特征：首先将点云序列分为若干片段（每个片段包含若干帧），然后以片段为单位随机打乱顺序，并训练模型使其能够正确地恢复片段的原始顺序，从而学习到点云序列的时空特征，应用到各种下游任务。然而，上述方法没有很好地学习到点云视频的空间特征，且容易受到相似帧的干扰，Shen 等人提出掩码时空结构预测（Masked Spatio-Temporal Structure Prediction，MaST-Pre）[31]，该模型通过预测掩码的辅助任务学习点云视频的空间特征；另一方面，引进时序基数差（Temporal Cardinality Difference，CD）的概念，并通过预测时序基数差学习点云视频的时序特征。MaST-Pre 结合两个辅助任务，成功地学习到点云视频的时空特征，但是时序基数差的计算开销较大且在复杂场景中容易出错。Zhang 等人提出完整到部分的 4D 蒸馏（Complete-to-partial 4D Distillation，Ctp-4D）[32]，首先利用相机轨迹生成对应的点云序列，将原始点云输入到教师网络，生成点云输入到学生网络，然后使用非对称的教师-学生网络实现从整体到部分的蒸馏。虽然 Ctp-4D 模型优化比较简单，但是通过相机轨迹的方法从原始点云中生成部分点云不太稳定。Xu 等人提出一个以人为中心的点云视频理解的统一框架（A Unified Framework for Human-centric Point Cloud Video Understanding，UniPVU-Human）[33]，框架采用双流网络的形式，在外观流网络中，首先利用蒙皮多人线性模型（Skinned Multi-Person Linear Model，SMPL）生成人体数据，并使用生成数据训练分割网络，将人体按部位分割，然后时序上以帧为单位，空间上以部位为单位进行掩码，并通过生成掩码重构的方式训练特征提取器；在运动流网络中，使用运动流估计器学习运动流特征，通过融合全局特征、部位特征、点特征对编码器进行微调。

3.3 典型方法

3.3.1 基于双流卷积神经网络的 RGB 动作识别

3.3.1.1 基于卷积双流网络融合的动作识别

本节主要介绍基于卷积双流网络融合的动作识别方法[4]。

问题描述：视频中的动作识别是计算机视觉领域的一个活跃研究方向，但现有系统与人类的识别能力相比仍有较大差距。近年来，研究人员尝试将卷积神经网络（ConvNets）应

用于该任务，通过结合外观（空间信息）和运动（时间信息）来提升性能。然而，与图像分类、人脸识别等其他视觉任务相比，动作识别并未从卷积神经网络中获得性能提升。部分原因在于当前用于训练的数据集（如 UCF-101 和 HMDB51）规模较小或噪声较多。例如，ImageNet 数据集每个类别有 1000 个样本，而 UCF-101 每个类别仅有 100 个样本，难以应对视频中动作因运动和视角变化带来的复杂性。此外，现有的一些卷积神经网络架构无法充分利用时间信息，导致性能主要依赖于空间（外观）识别。例如，在某些动作（如射箭）中，单帧图像的外观足以识别动作，但对于其他动作（如区分走路和跑步），仅靠外观信息往往不够，时间线索至关重要。传统的双流架构 [3] 虽通过分别训练空间和时间网络来融合信息，但融合仅发生在分类层，无法实现空间和时间的像素级对应，也无法有效捕捉长时间的动作演变。

方法动机：针对上述问题，设计一种新的卷积神经网络架构，能够在特征抽象的多个粒度级别上融合空间和时间线索，并实现空间与时间的有效整合。三个关键问题：①如何在考虑空间配准的情况下融合空间和时间网络；② 在网络的哪个位置进行融合最为有效；③如何在时间维度上融合网络以捕捉动作的动态变化。

方法特点：该方法提出一种新的时空融合架构，相较于传统的双流网络，该架构在融合方式和时间建模上取得突破。首先，在卷积层而不是 Softmax 层进行空间和时间网络的融合，不仅不会降低性能，还能大幅减少参数量。其次，在最后一个卷积层进行空间融合优于在早期层，同时在类别预测层进行额外融合可以进一步提高准确性。最后，通过在时空邻域上对抽象卷积特征进行池化（如 3D 池化）。

具体方法：

此方法为一种改进的双流网络架构，用于处理视频数据。该方法基于经典的双流网络（空间流处理 RGB 帧，时间流处理光流帧），并通过空间融合、时间融合和整体架构设计三个部分进行优化。

1. 空间融合（Spatial Fusion）

空间融合的目标是将空间流和时间流的特征图在卷积层中进行有效整合，同时保持空间位置的对应性。假设空间流的特征图表示为 $\boldsymbol{x}^a \in \mathbb{R}^{H\times W\times D}$，时间流的特征图表示为 $\boldsymbol{x}^b \in \mathbb{R}^{H\times W\times D}$，融合的目标是通过函数 $f:\boldsymbol{x}^a,\boldsymbol{x}^b \rightarrow \boldsymbol{y}$ 生成输出特征图 $\boldsymbol{y} \in \mathbb{R}^{H\times W\times D'}$。具体方法包括：

加和融合即对相同位置和通道的元素求和，公式为：

$$\boldsymbol{y}^{\text{sum}}_{i,j,d} = \boldsymbol{x}^a_{i,j,d} + \boldsymbol{x}^b_{i,j,d}, \tag{3.1}$$

这种方法虽简单但通道对应性较弱，依赖后续层优化。

最大值融合，即取最大值，公式为：

$$\boldsymbol{y}^{\max}_{i,j,d} = \max\{\boldsymbol{x}^a_{i,j,d}, \boldsymbol{x}^b_{i,j,d}\}, \tag{3.2}$$

与加和融合类似，通道对应性较弱。

拼接融合，即沿通道维度堆叠，公式为：

$$\boldsymbol{y}^{\text{cat}}_{i,j,2d} = \boldsymbol{x}^a_{i,j,d}, \quad \boldsymbol{y}^{\text{cat}}_{i,j,2d-1} = \boldsymbol{x}^b_{i,j,d}, \tag{3.3}$$

经沿通道维度堆叠后输出维度翻倍，留给后续层学习对应关系。

卷积融合，即先堆叠特征图，再用卷积滤波器 $\boldsymbol{f} \in \mathbb{R}^{1\times1\times2D\times D}$ 处理，b 为偏置，公式为：

$$\boldsymbol{y}^{\text{conv}} = \boldsymbol{y}^{\text{cat}}\boldsymbol{f} + b, \tag{3.4}$$

这种方法通过可训练滤波器学习通道间的加权组合。

双线性融合，即计算每个像素位置的外积并求和，公式为：

$$\boldsymbol{y}^{\text{bil}} = \sum_{i=1}^{H}\sum_{j=1}^{W} \boldsymbol{x}_{i,j}^{a} \otimes \boldsymbol{x}_{i,j}^{b}. \tag{3.5}$$

上述公式虽捕捉通道间的乘性交互关系，但维度较高。

2. 时间融合（Temporal Fusion）

为捕捉动作的时间演变，该方法提出在时间维度上融合特征图 $\boldsymbol{x} \in \mathbb{R}^{H\times W\times T\times D}$，包括：2D Pooling：仅在空间维度池化，忽略时间信息；3D Pooling：在时空邻域上池化，如使用 $3\times3\times3$ 的池化核；3D Conv + Pooling：先用 3D 卷积滤波器 $\boldsymbol{f} \in \mathbb{R}^{W''\times H''\times T''\times D}$ 处理输入再进行 3D 池化。实验显示，3D Conv + Pooling 性能最佳，能有效建模时空特征。

此方法的时空融合架构如图 3.3 所示。在 ReLU 层通过 3D Conv Fusion 将时间流融合到空间流中，形成时空流，并进行 3D 池化。同时保留时间流，单独进行 3D 池化。训练时使用两个流的损失，测试时平均两个流的预测。此外，在时间上以间隔 τ 采样 T 个时间块（如 $T=5$），使网络能在短时间尺度（如光流堆栈的 10 帧）和长时间尺度（如 $T\times\tau$）上捕捉特征。实现中采用预训练的 VGG-M-2048 和 VGG-16 模型，并通过数据增强（如随机裁剪和抖动）防止过拟合。

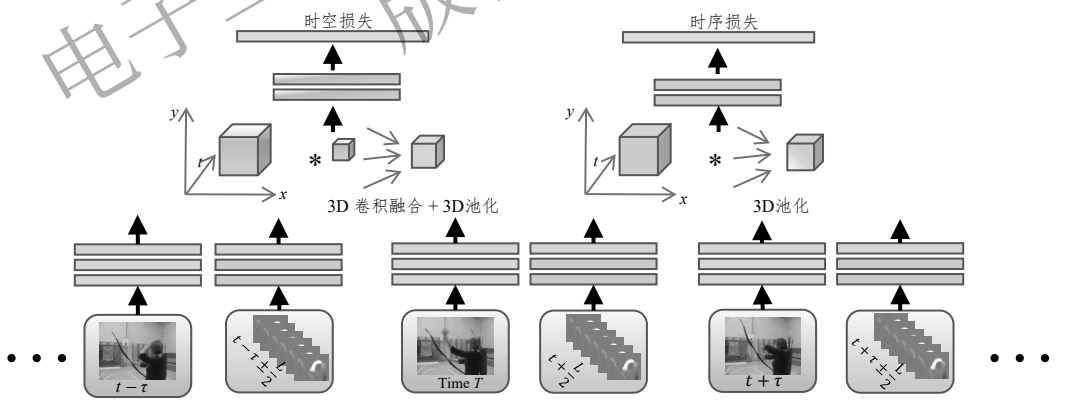

图 3.3　时空融合架构示意图[4]

3.3.1.2 基于双流协同学习与时空注意力的动作识别

本节主要介绍基于双流协同学习与时空注意力的动作识别方法[34]。

问题描述：视频分类是一个在视频搜索、智能监控等领域中具有重要意义的任务，其目标是对视频内容进行准确的类别识别。然而，视频分类面临三大主要挑战。首先，视频内

容的复杂性使得分类任务变得困难。例如，“生日派对”这一类别可能包含花朵、蛋糕、刀具等多种物体和场景，导致视频内容多样且难以辨识。其次，类内差异大增加分类的复杂度。同一类别的视频可能具有完全不同的背景，例如“生日派对”可能发生在餐厅或后院，背景差异显著。第三，类间相似性高进一步加大了分类难度。例如，“足球杂耍”和“足球点球”都包含足球、运动员和草地等相似元素，难以通过单一特征区分。这些挑战在大规模视频数据处理中尤为突出，传统方法难以应对。

方法动机：传统视频分类方法依赖手工特征，例如用于静态信息的方向梯度直方图和用于运动信息的光流直方图，但这些特征的区分能力有限。然而，现有深度学习方法存在两个主要局限。首先，忽略空间和时间注意力的共存关系。空间注意力关注帧中的显著区域，时间注意力强调视频中的关键帧，但现有方法通常分别处理二者，未能联合建模它们的相互关系。其次，忽略静态信息和运动信息的强互补性。静态信息（如物体外观）和运动信息（如动作序列）是视频的两个互补方面，但现有方法往往单独处理，未能充分利用它们的协同作用。这些局限促使新的方法出现，以更有效地解决视频分类问题。

方法特点：双流协同学习与时空注意力（Two-stream Collaborative Learning with Spatial-Temporal Attention，TCLSTA）的主要创新体现在两个方面。首先是空间-时间注意力模型，该模型联合建模空间和时间注意力，通过强调帧中的显著区域和视频中的关键帧，以相互增强的方式学习判别性特征。其次是静态-动态协同模型，通过协同学习静态和运动信息，不仅增强特征提取能力，还自适应地学习两者的融合权重，从而提升分类性能。

具体方法：

此方法是一种用于视频分类的空间-时间注意力模型与静态-动态协同模型的结合方法，通过充分利用视频中的静态信息（帧）和动态信息（光流），提升分类性能。以下详细介绍方法的各个组成部分。

空间-时间注意力模型采用如图3.4所示的双流架构，分别处理视频中的静态信息（基于帧）和动态信息（基于光流）。每个流包含三个核心组件：连接网络、空间级注意力网络和时间级注意力网络。这些组件协同工作，通过突出帧内的显著区域和视频序列中的关键帧，提取判别性特征。

连接网络是特征提取的骨干网络，基于 ResNet-50 架构进行修改以整合空间和时间注意力机制。其最后一层池化层被替换为加权池化层，该层与空间级注意力网络相连，使空间注意力能够通过强调重要区域来引导特征学习。此外，加权池化层的输出还作为时间级注意力网络的输入，从而实现空间和时间注意力的联合优化。连接网络从输入帧或光流图像中提取高级特征，随后通过注意力机制进一步精练。

空间级注意力网络旨在识别并强调每帧中对分类至关重要的显著区域。它利用类别激活映射（Class Activation Mapping，CAM）计算不同空间位置的重要性。其架构基于连接网络构建，在卷积层共享权重。随后添加一个具有 1024 个滤波器的卷积层（CAM_conv）、全局平均池化层（GAP）和 Softmax 层。对于给定帧，卷积层中单元 k 在空间位置 (x, y) 的激活表示为 $a_k(x, y)$。位置 (x, y) 对类别 c 的重要性计算为：

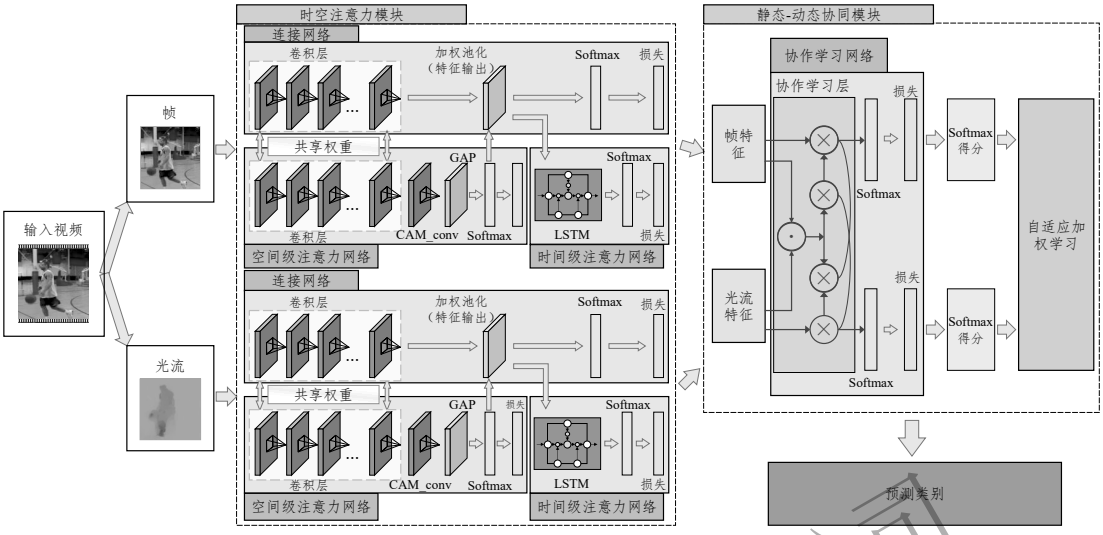

图 3.4　基于时空注意力与双流协同学习的动作识别方法框图[34]

$$m_c(x,y)=\sum_k w_k^c a_k(x,y), \tag{3.6}$$

其中，w_k^c 是单元 k 对类别 c 的权重。然后，该值被归一化为：

$$\tilde{m}_c(x,y)=\frac{g\cdot\exp(m_c(x,y))}{\sum_{x,y}\exp(m_c(x,y))}, \tag{3.7}$$

其中，g 是帧的空间网格数量。归一化的空间注意力 $\tilde{m}_c(x,y)$ 被用于连接网络的加权池化层，与相应的卷积特征相乘并执行池化，从而使网络聚焦于显著区域。

时间级注意力网络通过识别对分类任务最重要的帧，捕捉视频的时间动态。它使用 LSTM 建模连接网络提取的特征序列。其架构包括一个具有 512 个单元的 LSTM 层，后接一个具有 N 个单元（N 为类别数）的 Softmax 层。对于视频序列 $\boldsymbol{x}=[\boldsymbol{x}_1,\boldsymbol{x}_2,\cdots,\boldsymbol{x}_T]\in\mathbb{R}^{D_f\times T}$，LSTM 生成隐藏状态 $\boldsymbol{H}=[\boldsymbol{h}_1,\boldsymbol{h}_2,\cdots,\boldsymbol{h}_T]\in\mathbb{R}^{D_h\times T}$，$D_f$ 为每帧特征向量的维度，D_h 为 LSTM 隐藏状态的维度。亲和矩阵 $\boldsymbol{C}$ 计算为 $\boldsymbol{C}=\tanh(\boldsymbol{H}^{\mathrm{T}}\boldsymbol{H})$，该矩阵捕获帧之间的成对相关性。第 t 帧的时间注意力为 $\gamma_t=\sum_{i=1}^{T}c_{i,t}$，其中 $c_{i,t}$ 是 $\boldsymbol{C}$ 中表示帧 i 和 t 相关性的元素。结合空间和时间注意力，连接网络输出的帧 t 特征（含空间注意力）为 $\alpha_t\in\mathbb{R}^{c\times 1}$，最终特征 β_t 计算为：

$$\beta_t=\frac{\alpha_t\cdot\exp(\gamma_t)}{\sum_{j=1}^{T}\exp(\gamma_j)}, \tag{3.8}$$

此公式确保特征同时强调空间显著区域和时间重要帧。

静态-动态协同模型基于空间-时间注意力模型提取的特征，进一步提升分类性能。它包括协作学习网络和自适应加权学习两个主要部分。

协作学习网络利用静态信息和动态信息的互补性，通过两流之间的相互引导进行优化。其输入为空间-时间注意力模型提取的静态特征（来自帧）和动态特征（来自光流）。网络

采用交替优化方案，在每个步骤中，一类特征引导另一类特征的优化。例如，在时间 t，静态特征引导动态特征的优化：

$$\boldsymbol{H} = \tanh(\boldsymbol{W}^m \boldsymbol{V}^m + (\boldsymbol{W}_o^s \boldsymbol{O}^s)\mathbf{1}^{\mathrm{T}}), \tag{3.9}$$

$$\boldsymbol{z}^m = \mathrm{Softmax}(\boldsymbol{W}_p^m \boldsymbol{H}), \tag{3.10}$$

$$\boldsymbol{O}^m = \sum z_i^m v_i^m, \tag{3.11}$$

其中，$\boldsymbol{H} \in \mathbb{R}^{D_z \times N}$ 为中间隐藏状态矩阵，D_z 是隐藏层维度，由网络设计决定，$\boldsymbol{W}^m \in \mathbb{R}^{D_z \times D_v}$ 为动态特征的权重矩阵，$\boldsymbol{V}^m \in \mathbb{R}^{D_v \times N}$，为动态特征矩阵，包含 N 个光流特征向量，D_v 是光流特征维度，$\boldsymbol{W}_o^s \in \mathbb{R}^{D_z \times D_s}$ 为静态特征的权重矩阵，$\boldsymbol{O}^s \in \mathbb{R}^{D_s \times 1}$ 为前一步合并的静态特征向量，D_s 是静态特征维度，$\boldsymbol{z}^m \in \mathbb{R}^{N \times 1}$ 为动态特征的优化系数，$\boldsymbol{O}^m \in \mathbb{R}^{D_v \times 1}$ 为优化后的动态特征。在时间 $t+1$，角色反转，动态特征引导静态特征的优化，采用类似的公式。网络输出经过相互引导优化的静态和动态特征，充分利用两流之间的互补信息。

自适应加权学习针对不同视频类别中静态和动态信息的重要性差异，学习类别特定的融合权重。对于每个类别 j，融合权重 $\boldsymbol{W}_j = [w_{j,1}, w_{j,2}] \in \mathbb{R}^{1\times 2}$ 被学习，其中 $w_{j,1}$ 和 $w_{j,2}$ 分别对应静态流和动态流的权重，且满足 $w_{j,1} + w_{j,2} = 1$。权重通过最大化以下目标函数学习：

$$\arg\max_{\boldsymbol{W}_j} \mathcal{P}_j - \lambda \mathcal{N}_j, \tag{3.12}$$

其中，$\mathcal{P}_j = \sum_{i=1}^{n_j} \boldsymbol{W}_j \boldsymbol{S}_i^j \boldsymbol{J}_j$ 为最大化类别 j 正样本的预测分数，而 $\mathcal{N}_j = \sum_{k=1,k\neq j}^{c} \sum_{i=1}^{n_k} \boldsymbol{W}_j \boldsymbol{S}_i^k \boldsymbol{J}_j$ 为最小化负样本的预测分数。这里，$\boldsymbol{S}_i^j = [s_{i,1}^{j\,\mathrm{T}}, s_{i,2}^{j\,\mathrm{T}}]^{\mathrm{T}} \in \mathbb{R}^{2\times c}$ 是类别 j 中第 i 个样本的预测分数向量，$\boldsymbol{J}_j = [0, \cdots, 1, \cdots, 0]^{\mathrm{T}} \in \mathbb{R}^{c\times 1}$ 是类别 j 的独热向量，λ 平衡正负样本的影响。通过线性规划求解，得到自适应融合权重，根据类别的相关性组合静态和动态流。通过整合空间-时间注意力模型和静态-动态协同模型，本方法结合空间和时间注意力，并充分利用静态和动态信息的互补优势。

3.3.2 基于三维卷积神经网络的 RGB 动作识别

本节主要介绍基于快慢网络的动作识别方法[11]。

问题描述：视频识别是一个复杂任务，核心问题在于如何从视频数据中提取并理解空间信息（每帧中的内容）和时间信息（内容随时间的变化），以实现动作分类或事件检测。传统的视频识别方法，如基于三维卷积神经网络（3D CNN）的模型，通常对空间和时间维度采取对称处理，即在网络设计中将两者视为同等重要。然而，这种对称性假设可能并不完全适用于自然视频的特性。在自然视频中，空间语义（如物体身份或场景类别）往往变化缓慢，而动作或运动状态（如从走路变为跑步）可能在短时间内发生显著变化。传统方法未能充分利用这一特性，可能导致对时间动态的建模不足或计算资源的浪费。因此，存在一个关键问题：如何设计一种架构，能够更有效地分别处理视频中的空间语义和时间动态，而不是简单地沿用对称处理的方式。

研究动机：此方法提出的动机源于对自然视频统计特性和生物视觉机制的双重观察。首先，从视频的统计特性来看，空间语义（如“手”或“人”）通常在动作发生时保持稳定，

而运动本身（如挥手或跳跃）的变化速度远快于语义变化。这意味着，识别空间语义可能并不需要高时间分辨率的密集采样，而捕捉快速运动则需要更高的帧率支持。其次，受到灵长类生物视觉系统的启发。在灵长类的视网膜神经节细胞中，约 80% 是 Parvocellular 细胞（P-cells），负责处理高空间分辨率的细节和颜色，但对时间变化反应较慢；而约 15% ~ 20% 是 Magnocellular 细胞（M-cells），对快速时间变化敏感，却对空间细节和颜色不甚关注。这一生物学现象提示，空间和时间信息的处理可以分担给不同的专门路径，而不是统一处理。基于此，Feichtenhofer 提出一种非对称的视频识别模型，旨在通过两个不同速率的路径分别优化空间和时间信息的建模，从而提升识别性能并降低计算冗余。

方法特点：SlowFast 的核心创新在于其如图 3.5 所示的双路径架构设计，通过 Slow 路径和 Fast 路径分别针对空间语义和时间动态进行专门处理。Slow 路径以低帧率运行，专注于捕捉视频中变化缓慢的空间语义信息。与之相对，Fast 路径以高帧率运行，例如每 2 帧只采样 1 帧（由帧率比 $\alpha = 8$ 控制），旨在捕捉快速变化的动作细节。此外，两个路径通过横向连接（Lateral Connections）融合，允许 Fast 路径的动态信息在网络的不同阶段传递给 Slow 路径，从而形成一个综合的时空表征。

具体方法：

SlowFast 的具体实现依托于三维卷积神经网络架构（如 ResNet），并通过以下方式设计和优化。

首先，Slow 路径处理低帧率的视频片段，例如从 64 帧的原始视频中采样 4 帧 ($T = 4, \tau = 16$)。它基于 3D ResNet backbone，但在时间维度上使用大步长（Large Temporal Stride），避免过多的时间下采样，以保留空间分辨率。早期层中的卷积核在时间维度上退化为 2D，仅在较深层（如 Res3 和 Res5）使用非退化（Temporal Kernel > 1）的 3D 卷积。这一设计基于以下观察：当时间步长较大时，早期层的时空相关性较弱，而深层卷积能更好地捕捉空间语义。

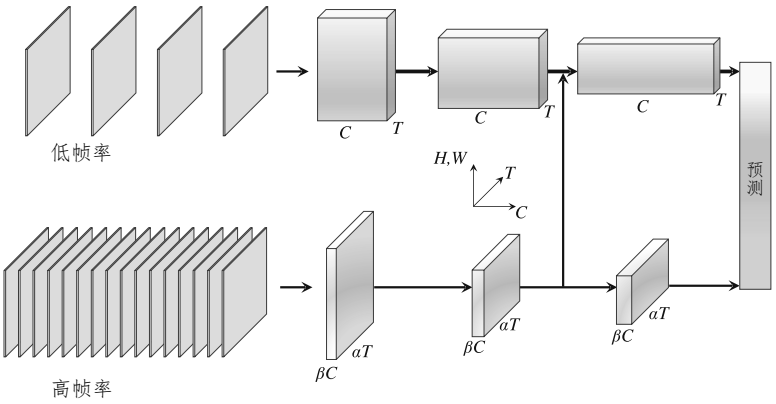

图 3.5　SlowFast Networks 示意图[11]

其次，Fast 路径处理高帧率的视频片段，例如从同一 64 帧中采样 32 帧（$\alpha T = 32$）。它同样基于 3D ResNet，但全程不进行时间下采样，以保持高时间分辨率。Fast 路径的通道数显著减少（例如 $\beta = 1/8$），使其计算量远低于 Slow 路径。为了增强对动作的建模能力，每

个残差块中都使用非退化的 3D 卷积核，充分利用其高时间分辨率。

接着，横向连接用于融合两条路径的信息。由于 Fast 路径的时间维度（αT）比 Slow 路径（T）长，融合前需通过变换匹配特征尺寸。此方法采用了多种融合方式，包括：时间到通道（Time-to-Channel，将时间帧打包到通道中）、时间步长采样（Time-strided Sampling，每 α 帧采样 1 帧）和时间步长卷积（Time-strided Convolution，使用 5×1^2 卷积核，步长为 α）。实验表明，时间步长卷积效果最佳，成为默认选择。融合后，Fast 路径的特征通过加法或拼接的方式注入 Slow 路径，增强其对动作的感知能力。

3.3.3 基于循环神经网络的 RGB 动作识别

本节主要介绍基于长期循环卷积网络（LRCN）的动作识别方法[13]。

问题描述：在计算机视觉领域，传统的卷积神经网络（CNN）在静态图像识别任务中表现优异，但当任务涉及序列数据时，例如视频活动识别、图像描述生成或视频描述生成，其局限性逐渐暴露。这些任务需要模型能够处理可变长度的输入和输出序列，同时捕捉时间维度上的长期依赖关系。然而，传统的 CNN 架构主要针对空间特征的提取，缺乏对时间动态的建模能力。

方法动机：该方法的提出源于对现有方法局限性的深刻认识。在视频处理领域，传统方法通常采用两种极端策略：一种是学习完全通用的时空滤波器，例如基于 3D 卷积的模型；另一种是对固定时间窗口或视频片段进行简单的时序池化（如平均池化或最大池化）。这些方法在处理长时间依赖关系或可变长度序列时效果有限，无法充分捕捉复杂的动态变化。受深度学习在空间和时间维度上的成功启发，一种在时间上也具备“深度”的模型被提出，即通过引入循环机制的卷积网络，使其能够在空间和时间上学习组合表示。

方法特点：该方法提出长期循环卷积网络架构，此架构的关键思想是将视觉输入（例如图像或视频帧）通过 CNN 处理，生成固定长度的特征向量，然后将这些特征向量输入到循环序列学习模块中，以捕捉时间上的长期依赖关系。这种架构具有以下几个创新点：首先，它在空间和时间维度上均具备“深度”，能够学习复杂的时空组合表示；其次，整个模型支持端到端训练，使得视觉特征提取与序列学习过程能够协同优化；此外，该架构具有高度的灵活性，可以应用于多种视觉任务，包括活动识别、图像描述和视频描述；最后，通过引入 LSTM 等循环单元，模型能够有效学习长期依赖关系，克服传统 RNN 中的梯度消失问题。

具体方法：

LRCN 的架构由两大部分组成，如图3.6所示：视觉特征提取模块和序列学习模块。视觉特征提取模块通常采用一个预训练的卷积神经网络（CNN），例如基于 CaffeNet 和 Zeiler & Fergus 网络的混合模型，在 ImageNet 数据集上预训练，用于将每个视觉输入 $\boldsymbol{x}_t \in \mathbb{R}^{H \times W \times C}$ 转换为固定长度的特征向量 $\phi_V(\boldsymbol{x}_t) \in \mathbb{R}^d$。随后，这些特征向量被输入到序列学习模块，该模块由长短期记忆网络（LSTM）实现，用于处理序列数据并捕捉长期依赖关系。LSTM 通过其记忆单元和门控机制（如输入门、遗忘门和输出门）更新隐藏状态。

$$
\begin{aligned}
\boldsymbol{i}_t &= \sigma(\boldsymbol{W}_{xi}\boldsymbol{x}_t + \boldsymbol{W}_{hi}\boldsymbol{h}_{t-1} + b_i),\\
\boldsymbol{f}_t &= \sigma(\boldsymbol{W}_{xf}\boldsymbol{x}_t + \boldsymbol{W}_{hf}\boldsymbol{h}_{t-1} + b_f),\\
\boldsymbol{o}_t &= \sigma(\boldsymbol{W}_{xo}\boldsymbol{x}_t + \boldsymbol{W}_{ho}\boldsymbol{h}_{t-1} + b_o),\\
\boldsymbol{g}_t &= \tanh(\boldsymbol{W}_{xc}\boldsymbol{x}_t + \boldsymbol{W}_{hc}\boldsymbol{h}_{t-1} + b_c),\\
\boldsymbol{c}_t &= \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \boldsymbol{g}_t,\\
\boldsymbol{h}_t &= \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t),
\end{aligned}
\tag{3.13}
$$

其中，$\boldsymbol{i}_t$、$\boldsymbol{f}_t$、$\boldsymbol{o}_t$ 分别为输入门、遗忘门和输出门，均为 N 维向量。$\boldsymbol{c}_t$ 为记忆单元，$\boldsymbol{h}_t$ 为隐藏状态，$\sigma(\cdot)$ 和 $\tanh(\cdot)$ 分别为 Sigmoid 和双曲正切激活函数，$\odot$ 表示元素级乘法。$\boldsymbol{W}_{xi}, \boldsymbol{W}_{xf}, \boldsymbol{W}_{xo}, \boldsymbol{W}_{xc} \in \mathbb{R}^{N\times d}$ 为输入到门的权重矩阵，将 d 维输入映射到 N 维。$\boldsymbol{W}_{hi}, \boldsymbol{W}_{hf}, \boldsymbol{W}_{ho}, \boldsymbol{W}_{hc} \in \mathbb{R}^{N\times N}$ 为隐藏状态到门的权重矩阵，映射 N 维隐藏状态。

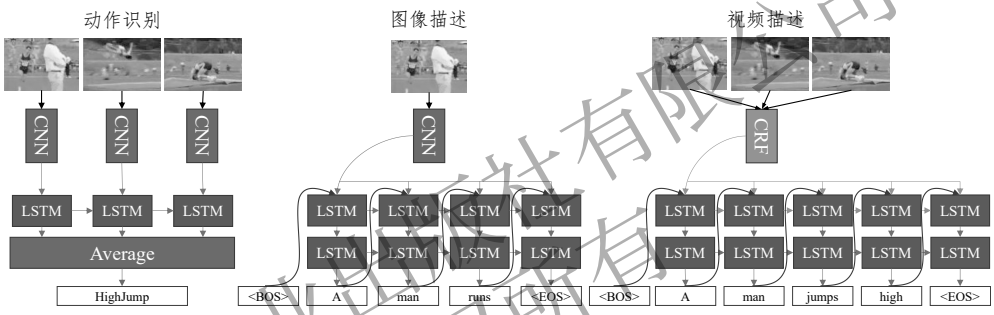

图 3.6　长期循环卷积网络示意图[13]

根据具体任务的不同，LRCN 的架构将进行相应的调整。对于活动识别，输入是视频帧序列 $[\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_T]$，输出是一个活动标记 y，模型在每个时间步预测标记分布并通过平均所有时间步的预测结果得到最终标记 $P(y=c) = \frac{1}{T}\sum_{t=1}^{T} P(y_t = c)$；对于图像描述，输入是单个图像 $\boldsymbol{x}$，输出是描述性句子 $[y_1, y_2, \cdots, y_T]$，模型将图像特征重复输入到 LSTM 中以生成序列化文本；对于视频描述，输入是视频帧序列 $[\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_T]$，输出是描述性句子 $[y_1, y_2, \cdots, y_T]$，模型采用编码器-解码器结构，先将视频帧编码为固定向量，再解码为句子。在训练过程中，LRCN 通过最大化真实标记（Ground Truth）输出的似然来优化参数，使用随机梯度下降和反向传播算法计算梯度。

3.3.4 基于 Transformer 的 RGB 动作识别

3.3.4.1 基于掩码自编码器的动作识别

本节主要介绍基于掩码自编码器的动作识别方法[20]。

问题描述：自监督视频预训练（Self-Supervised Video Pre-Training，SSVP）中存在数据效率问题。在视频识别任务中，训练高效的视频变换器（Video Transformers）通常需要依赖大规模的监督数据集。然而，与图像数据集相比，视频数据集的规模较小，例如 Kinetics-400 仅有约 24 万个训练视频，而图像数据集如 ImageNet 通常包含数百万个样本。这种数据规

模的限制使得从头训练视频变换器变得极具挑战性，因为模型难以在有限的数据上学习到泛化能力强的表示。传统的解决方法依赖于图像预训练模型（如 ImageNet 预训练）或额外的大型视频数据集，但这引入了图像模型的偏见，且对纯视频数据的利用效率不高。自监督学习通过利用无标记数据学习通用表示，为这一问题提供一种潜在的解决方案。VideoMAE 旨在探索如何在小规模视频数据上高效地进行自监督预训练，从而减少对大规模监督数据或外部数据的依赖。

方法动机：VideoMAE 的研究动机源于图像领域中掩码自编码器（Masked Autoencoder，MAE）的成功，尤其是 ImageMAE 的出色表现。ImageMAE 通过随机掩码图像的 75% 区域并重建这些区域，在自监督图像预训练中取得成果。将这一方法扩展到视频领域存在着些许困难，视频数据具有独特的特性：时序冗余（Temporal Redundancy）和时序相关性（Temporal Correlation）。时序冗余指的是视频帧之间内容变化缓慢，信息密度低于图像；而时序相关性则意味着相邻帧之间存在强相关性，可能导致模型通过邻近帧信息“作弊”重建掩码区域。这些特性使得直接应用 ImageMAE 的方法效果有限。因此，需要提出针对视频特性设计定制化的掩码策略和模型结构，以提高自监督任务的挑战性，从而迫使模型学习更有效的时空表示。

方法特点：VideoMAE 的主要创新体现在以下几个方面。首先，该方法使用一种极高比例的掩码策略，将掩码比例提升至 90% ~ 95%，远高于图像中的 75%。这种设计利用视频的时序冗余特性，使得重建任务更具挑战性，从而推动模型学习更高级的表示。其次，引入时序管状掩码（Tube Masking）策略，即在整个时序轴上应用相同的掩码图。这种方法避免模型利用时序邻居信息简单重建掩码区域，鼓励模型捕捉更高层次的时空结构。第三，VideoMAE 在数据效率上表现出色，仅使用约 3000~ 4000 个视频的小规模数据集即可取得优异结果，无须额外数据，这得益于其挑战性任务设计。

具体方法：

VideoMAE 的方法设计围绕视频特性展开，包含多个关键组件，如图 3.7 所示。首先，生成输入时，采用时序下采样（Temporal Downsampling）策略，通过步幅采样压缩视频帧数，以减少时序冗余并提升预训练效率。例如，在 Kinetics 数据集上步幅设为 4，在 Something-Something V2 数据集上设为 2。接着，通过立方体嵌入（Cube Embedding）将视频划分为 $2 \times 16 \times 16$ 的时空立方体，每个立方体映射为一个 Token embedding，从而降低时空维度并缓解冗余。核心的掩码策略是高比例管道掩码，掩码比例高达 90% ~ 95%，且掩码图在所有帧上保持一致，数学上可表示为：

$$\mathbb{I}[p_{x,y} \in \boldsymbol{\Omega}] \sim \text{Bernoulli}(p_{\text{mask}}), \tag{3.14}$$

其中，$p_{x,y}$ 是空间位置的掩码概率，$\boldsymbol{\Omega}$ 是掩码 Token 集合，Bernoulli 为伯努利分布，p_{mask} 是掩码比例，且时序上共享相同掩码。这种设计增加了重建难度，迫使模型学习深层时空特征。

模型采用非对称编码器-解码器架构，编码器仅处理未掩码的 Token，解码器则重建掩码区域。编码器中引入联合时空注意力（Joint Space-Time Attention）机制，允许所有 Token 交互以捕捉高级时空信息。损失函数使用均方误差（MSE）衡量重建与原始 Token 的差异，

定义为：

$$\mathcal{L} = \frac{1}{|\boldsymbol{\Omega}|} \sum_{p \in \boldsymbol{\Omega}} \|\boldsymbol{I}(p) - \hat{\boldsymbol{I}}(p)\|_2^2, \tag{3.15}$$

其中，$\boldsymbol{I}(p) \in \mathbb{R}^{C \times T' \times H \times W}$ 是输入视频的 Token，$\hat{\boldsymbol{I}}(p)$ 是重建 Token，$\boldsymbol{\Omega}$ 是掩码 Token 集合。这种设计结合高效的预处理、挑战性的任务和强大的注意力机制，使得 VideoMAE 能在小规模数据上学习到鲁棒的视频表示。

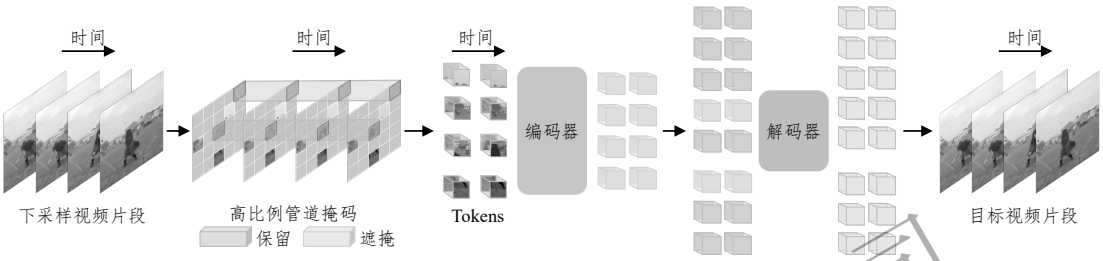

图 3.7　掩码自编码器方法示意图[20]

3.3.4.2 基于双重掩码自编码器的动作识别

本节主要介绍基于双重掩码自编码器的动作识别方法[21]。

问题描述：此方法所针对的核心问题是视频领域自监督预训练模型在大规模数据和模型上的表现。视频数据因其高维度（空间和时间维度）和时序特性，相较于图像和语言数据，对计算资源的需求更高，训练大规模模型存在困难。一些方法如 VideoMAE 虽已通过非对称编码器-解码器架构提升效率，但在扩展到亿级参数模型时，计算成本和内存消耗仍成为瓶颈。此外，视频数据集的规模和多样性不足限制了模型的泛化能力，而直接在小规模下游任务数据集上微调大规模预训练模型可能导致过拟合，影响性能。VideoMAE V2 通过改进 VideoMAE，使其在更大规模的模型和数据上高效训练，并在动作识别、时空动作检测和时序动作检测等下游任务中表现出色。

方法动机：动机源于以下几个关键点：首先，在语言和图像领域，大规模模型和数据已被证明能提升性能，但在视频领域，由于数据复杂性和资源限制，这一潜力尚未完全释放；其次，尽管 VideoMAE 通过高比例掩码策略降低计算需求，但在亿级参数模型训练中仍需进一步优化效率；再次，视频数据集规模较小（如 Kinetics-400 仅有 24 万个视频，远低于 ImageNet 的 1420 万个图像），且多样性不足，限制模型的学习能力；最后，直接微调大规模模型到小数据集易导致过拟合，亟需更有效的迁移学习策略。这些挑战促使研究者探索如何扩展 VideoMAE 以构建强大的视频基础模型。

方法特点：VideoMAE V2 提出三大创新点以应对上述问题：第一是双重掩码策略，在编码器和解码器中均应用掩码操作，进一步降低计算成本，同时保持自监督任务的有效性；第二是大规模预训练，通过整合多源数据构建包含 135 万个视频片段的无标记混合数据集，为大规模模型提供充足的训练数据；第三是渐进式训练范式，包括在无标记数据上的自监督预训练、在有标记混合数据集上的后预训练，以及在目标数据集上的微调，从而缓解过拟合并提升下游任务性能。这些创新共同推动视频掩码自编码器向亿级参数模型的扩展。

具体方法：

VideoMAE V2 建立在原始 VideoMAE 方法之上，后者是一种基于掩码自编码器（MAE）的自监督预训练框架。如图 3.8 所示，其核心步骤包括：首先通过立方体嵌入（Cube Embedding）将视频帧序列 $\boldsymbol{I} \in \mathbb{R}^{C\times T\times H\times W}$（通道数 C、时间帧数 T、高度 H、宽度 W）转换为一系列 Token $\boldsymbol{T} = \Phi_{\text{emb}}(\boldsymbol{I}) \in \mathbb{R}^{N\times D}$。接着，采用管状掩码（Tube Masking）策略以高比例 ρ（如 90%）丢弃 Token，仅保留未掩码的 Token $\boldsymbol{T}^u \in \mathbb{R}^{N^u\times D}$。然后，视频 Transformer 编码器 $\Phi_{\text{enc}}(\cdot)$ 处理 $\boldsymbol{T}^u$，生成潜在表示 $\boldsymbol{Z} = \Phi_{\text{enc}}(\boldsymbol{T}^u) \in \mathbb{R}^{N^u\times D}$。最后，解码器 $\Phi_{\text{dec}}(\cdot)$ 将 $\boldsymbol{Z}$ 与可学习的掩码 Token [MASK] 组合为 $\boldsymbol{Z}^c \in \mathbb{R}^{(N^u+N^d)\times D}$，重建视频帧 $\hat{\boldsymbol{I}} = \Phi_{\text{dec}}(\boldsymbol{Z}^c) \in \mathbb{R}^{C\times T\times H\times W}$。$N^u$ 为未掩码 Token 的数量，N^d 为解码器掩码下选择的 Token 数量。损失函数为均方误差（MSE），仅在掩码区域计算：

$$\mathcal{L} = \frac{1}{|\mathcal{M}(\rho)|}\sum_{i\in\mathcal{M}(\rho)} \|\boldsymbol{I}_i - \hat{\boldsymbol{I}}_i\|_2^2. \tag{3.16}$$

此方法利用非对称架构（编码器输入远少于解码器）提升效率，但在大规模场景下仍需改进。

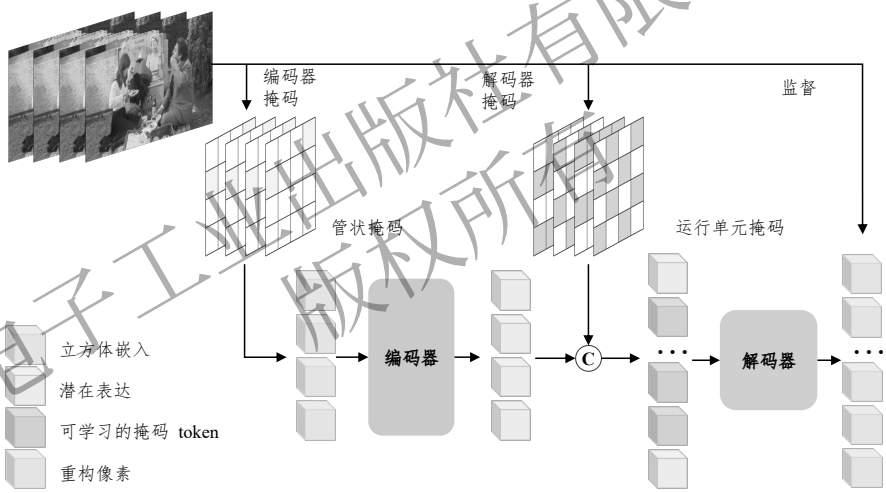

图 3.8　双重掩码自编码器网络示意图[21]

双重掩码策略

VideoMAE V2 引入双重掩码策略，在编码器和解码器中均应用掩码，进一步优化效率。编码器掩码 $\mathcal{M}_e$ 沿用原始 VideoMAE 的高比例管道掩码（如 90%），仅处理少量可见 Token。解码器掩码 $\mathcal{M}_d$ 则采用运行单元掩码（Running Cell Masking），选择一组代表性立方体进行重建，而非全部 Token。解码器输入为编码器输出 $\boldsymbol{Z}$ 与部分剩余可见 Token 的组合：

$$\boldsymbol{Z}^c = \boldsymbol{Z} \cup \{\mathcal{M}_i\}_{i\in\mathcal{M}_d\cap\mathcal{M}_e^c}, \tag{3.17}$$

其中，$\mathcal{M}_i$ 为可学习掩码 Token，$\mathcal{M}_e^c$ 表示编码器未掩码区域。解码器仅重建 $\mathcal{M}_d$ 下的可见 Token，损失函数调整为仅在解码器可见且编码器不可见的 Token 上计算：

$$\mathcal{L} = \frac{1}{|\mathcal{M}_d\cap\mathcal{M}_e^c|}\sum_{i\in\mathcal{M}_d\cap\mathcal{M}_e^c} \|\boldsymbol{I}_i - \hat{\boldsymbol{I}}_i\|_2^2. \tag{3.18}$$

这一策略利用视频数据冗余，减少解码器输入 Token 数量，降低计算成本和内存消耗。例如，相较于仅编码器掩码，双重掩码将训练时间缩短约三分之一，且性能几乎无损。这一过程迫使模型学习视频的空间-时间结构和语义信息，形成对动作、物体和场景的通用表示。虽然预训练目标是像素重建，但其本质是让模型捕捉视频中丰富的时空模式，这些模式对动作识别至关重要。

3.3.5 全监督点云动作识别

3.3.5.1 基于点 4D Transformer 的点云动作识别

本节主要介绍基于点 4D Transformer 的点云动作识别方法[22]。

问题描述：点云视频在空间维度存在不规则性与无序性，不同帧的点云构成不一致，点可能跨帧流入或流出，导致基于点跟踪的动态建模方法面临轨迹计算困难、跟踪误差累积及依赖点颜色（无法处理无色点云）等问题；而基于体素化的方法需将点云转换为规则网格，存在计算效率低、难以满足实时性需求等缺陷，传统方法在时空结构建模中比较困难。

方法动机：现有依赖点跟踪和体素化的点云视频建模方法存在局限性：点跟踪受限于长视频跟踪误差、颜色依赖及点的动态变化，体素化方法则因量化误差和额外计算成本影响效率。因此，须探索一种无需点跟踪、直接处理原始点云视频的端到端方法，以高效捕捉时空关联，提升模型对无色点云的适应性和计算效率。

方法特点：P4Transformer 设计点 4D 卷积模块直接对原始点云操作，通过时空半径和步长构建局部区域以编码时空结构，并利用降采样减少后续计算量；引入 Transformer 模块，将 4D 坐标（空间坐标与时间）与局部特征嵌入融合以增强位置感知，通过视频级自注意力机制在整个视频范围内动态捕捉全局外观和运动信息，替代显式点跟踪，解决点跨帧流动问题，同时采用多头注意力机制提升多尺度特征表示能力，实现端到端的高效时空建模，且该框架可同时适用于 3D 动作识别（视频级分类）和 4D 语义分割（点级预测）任务。

具体方法：

P4Transformer 主要有点 4D 卷积（Point 4D Convolution，P4Conv）和 m 个 Transformer 块组成，其结构如图 3.9 所示。

点 4D 卷积：令 $\boldsymbol{P}_t \in \mathbb{R}^{N\times 3}$ 和 $\boldsymbol{F}_t \in \mathbb{R}^{N\times C}$ 分别表示点云视频第 t 帧的点坐标和特征，N 代表点的数量，C 代表特征通道数。给定点云视频 $([\boldsymbol{P}_1;\boldsymbol{F}_1],[\boldsymbol{P}_2;\boldsymbol{F}_2],\cdots,[\boldsymbol{P}_L;\boldsymbol{F}_L])$，使用点 4D 卷积提取局部结构和下采样点，生成 $([\boldsymbol{P}'_1;\boldsymbol{F}'_1],[\boldsymbol{P}'_2;\boldsymbol{F}'_2],\cdots,[\boldsymbol{P}'_{L'};\boldsymbol{F}'_{L'}])$，供后续 Transformer 处理。模仿传统的卷积操作，点 4D 卷积可以表示为：

$$\begin{aligned}\boldsymbol{F}_t'^{(x,y,z)} &= \sum_{(\delta_x,\delta_y,\delta_z,\delta_t)\in G} \zeta(\delta_x,\delta_y,\delta_z,\delta_t)\cdot \boldsymbol{F}_{t+\delta_t}^{(x+\delta_x,y+\delta_y,z+\delta_z)} \\ &= \sum_{\delta_t=-r_t}^{r_t} \sum_{||(\delta_x,\delta_y,\delta_z)||\leqslant r_s} \zeta(\delta_x,\delta_y,\delta_z,\delta_t)\cdot \boldsymbol{F}_{t+\delta_t}^{(x+\delta_x,y+\delta_y,z+\delta_z)},\end{aligned} \tag{3.19}$$

其中，$\boldsymbol{F}'_t \in \mathbb{R}^{C'\times N'}$ 为输出特征，$(x,y,z)\in \boldsymbol{P}_t$，$(\delta_x,\delta_y,\delta_z,\delta_t)$ 表示时空位移，$\boldsymbol{F}_t^{(x,y,z)}$ 表示点云视频中位于 (x,y,z,t) 的点的特征，$\sum$ 可以用不同的池化方法实现，G 为某个点周围的时

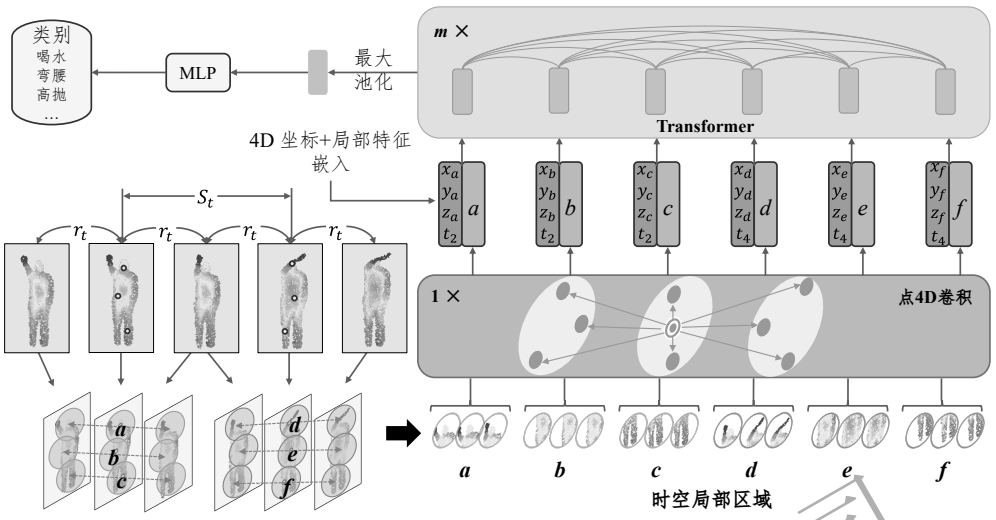

图 3.9　点 4D Transformer（P4Transformer）结构图[22]

空局部区域。因为空间和时间相互正交且独立，可以将区域 G 划分为一系列空间区域，由空间半径和时间半径定义。ζ 是 $(\delta_x, \delta_y, \delta_z, \delta_t)$ 的一个参数化函数：

$$\zeta(\delta_x, \delta_y, \delta_z, \delta_t) \cdot \boldsymbol{f} = (\boldsymbol{W}_d \cdot (\delta_x, \delta_y, \delta_z, \delta_t)^{\mathrm{T}}) \odot (\boldsymbol{W}_f \cdot \boldsymbol{f}), \tag{3.20}$$

其中，$\boldsymbol{f} = \boldsymbol{F}_{t+\delta_t}^{(x+\delta_z, y+\delta_y, z+\delta_z)}$ 为点特征，$\boldsymbol{W}_d \in \mathbb{R}^{C' \times 4}$，$\boldsymbol{W}_f \in \mathbb{R}^{C' \times C}$ 用于增加点特征维度从而改善特征表示能力。

由于点云空间上不规则且无序，不能像传统的视频那样通过滑动网格可以很容易的执行基于网格的卷积。而且，和基于网格的卷积不同，点 4D 卷积需要避免空的区域，所以在卷积之前选择和 PSTNet 相似的方法构建时空区域。具体地，首先基于时序步长 s_t 选取一些帧，其次使用最远点采样在选取帧中下采样 $N' = N/s_s$ 个点，s_s 为空间下采样率，然后将这些下采样点迁移到最近 r_t 帧中，最后使用半径为 r_s 的球查询构建空间邻域。

Transformer：在经过点 4D 卷积后，第 t 帧的时空邻域被编码为 $\boldsymbol{F}'_t$。因为相似的局部区域共享相似的表示，且点的位置也一定程度上反映局部区域的关系。因此可以将坐标和局部区域特征结合起来作为 Transformer 的输入：

$$\boldsymbol{I}^{(x,y,z,t)} = \boldsymbol{W}_i \cdot (x, y, z, t)^{\mathrm{T}} + \boldsymbol{F}_t'^{(x,y,z)}, \tag{3.21}$$

其中，$\boldsymbol{W}_i \in \mathbb{R}^{C' \times 4}$ 是转换 4D 坐标的权重。$\boldsymbol{I} \in \mathbb{R}^{C' \times L' N'}$ 是自注意力的输入。得到输入后，使用自注意力机制来捕捉不同邻域之间的关系：

$$\boldsymbol{Q} = \boldsymbol{W}_q \cdot \boldsymbol{I},\ \boldsymbol{K} = \boldsymbol{W}_k \cdot \boldsymbol{I},\ \boldsymbol{V} = \boldsymbol{W}_v \cdot \boldsymbol{I}, \tag{3.22}$$

$$\mathrm{Attention}(\boldsymbol{Q}, \boldsymbol{K}) = \mathrm{Softmax}(\frac{\boldsymbol{Q}^{\mathrm{T}} \cdot \boldsymbol{K}}{\sqrt{C^k}}), \tag{3.23}$$

$$\boldsymbol{O} = \boldsymbol{V} \cdot \mathrm{Attention}(\boldsymbol{Q}, \boldsymbol{K}), \tag{3.24}$$

其中，$\boldsymbol{W}_q \in \mathbb{R}^{C^k \times C'}$，$\boldsymbol{W}_k \in \mathbb{R}^{C^k \times C'}$，$\boldsymbol{W}_v \in \mathbb{R}^{C^v \times C'}$，$C^k$ 和 C^v 分别是键、值的维度，$\boldsymbol{Q} \in \mathbb{R}^{C^k \times L'N'}$，$\boldsymbol{K} \in \mathbb{R}^{C^k \times L'N'}$，$\boldsymbol{V} \in \mathbb{R}^{C^v \times L'N'}$，$\boldsymbol{O} \in \mathbb{R}^{C^v \times L'N'}$。得到 Transformer 的输出后，经过一个最大池化和多层感知机后得到模型的预测结果。

3.3.5.2 基于 t-patch 追踪的点云动作识别

本节主要介绍基于 t-patch 追踪的点云动作识别方法[1]。

问题描述：现有 3D 点云动作识别领域面临多重挑战：点云数据具有非结构化、排列不变性和点数可变的特性，导致难以学习时空表示；时序点云间缺乏显式的点对应关系，传统方法难以捕捉动态特征；公开标记的 3D 动作数据集稀缺，尤其是针对时序点云的细粒度动作标记不足。此外，现有方法多基于 RGB 视频或静态点云，直接应用于 3D 点云视频时性能有限，无法充分利用 3D 几何信息来理解人类动作的时空动态。

方法动机：随着 3D 传感器的普及，利用点云视频进行动作识别在自动驾驶、机器人协作等安全关键场景中具有重要价值，但现有方法存在局限性。纯 RGB 方法依赖视觉外观，在低光照、遮挡等场景下泛化能力不足，而 3D 点云能提供几何结构信息，但缺乏适用于时序点云的动态特征建模方法。因此，需要设计一种不依赖骨架先验、能直接从 3D 点云视频中提取时空特征的方法，以解决数据稀疏性、对应关系缺失和模型泛化性差等问题，推动 3D 点云在动作识别领域的实际应用。

方法特点：3DInAction 通过 t-patch 捕捉点云在时间维度的动态演化，将每个查询点及其相邻帧的对应点组合成 t-patch，建模局部运动模式；设计层次化神经网络架构，通过多层 t-patch 模块逐步提取从局部到全局的时空特征，每层通过多层感知机和卷积层处理 t-patch 特征，实现对动作细节和全局结构的分层建模；引入双向 t-patch 和噪声扰动策略，通过双向 t-patch 提取和高斯噪声扰动缓解 t-patch 坍缩问题，增强模型对遮挡和点密度变化的鲁棒性。该方法无需骨架标记，能直接从原始点云视频中学习判别性特征。

具体方法：

令 $\boldsymbol{S} = \{\boldsymbol{x}_j \in \mathbb{R}^3 | j = 1, 2, \cdots, N\}$ 表示包含 N 个点的 3D 点云。$\boldsymbol{S} = \{\boldsymbol{S}^0, \boldsymbol{S}^1, \cdots, \boldsymbol{S}^T\}$ 表示一个点云视频，由点云帧 $\boldsymbol{S}^t = \{\boldsymbol{x}_j^t | j = 1, 2, \cdots, N^t\}$ 组成，t 表示点云在视频中的索引。一个 t-patch $\boldsymbol{P}_q$ 是一个点集视频，$\boldsymbol{P}_q = \langle \boldsymbol{\Psi}_q^t \rangle_{t=0}^T$，$\boldsymbol{\Psi}_q^t = \Phi(\boldsymbol{\Psi}_q^{t-1})$，$\boldsymbol{\Psi}_q^0$ 为初始（静态）patch，Φ 是逐点映射函数。在实践中，Φ 难以获取，因此基于 k 近邻查询提出一种简单的方法：给定一个查询点 $\boldsymbol{x}_q^0$，提取一个 patch 记为 $\boldsymbol{\Psi}_q^0$，然后使用下一帧的最近邻点作为新的查询点并迭代地提取对应的 patch，即 $\overrightarrow{\boldsymbol{\Psi}}_q^t = \text{knn}(\boldsymbol{x}_q^{t-1}, \boldsymbol{S}^t)$，$\boldsymbol{x}_q^t = \text{nn}(\boldsymbol{x}_q^{t-1}, \boldsymbol{S}^t)$，$t = 1, 2, \cdots, T$。简化后的 t-patch 公式为：

$$\overrightarrow{\boldsymbol{P}}_q = \langle \overrightarrow{\boldsymbol{\Psi}}_q^t | t = 0, \cdots, T \rangle. \tag{3.25}$$

t-patch 时序坍缩：提取 t-patch 固有地存在两个或多个查询点在某一帧后坍缩为相同点的问题，这种问题称为时序坍缩，即 $\boldsymbol{x}_q^0 \neq \boldsymbol{x}_p^0$ 且 $\boldsymbol{x}_q^t = \boldsymbol{x}_p^t$。这将导致随着时间推移查询点的数量逐渐减少，丢失的信息越来越多。为了缓解这个问题，提出两个解决方案：① 添加小的噪声到每次迭代的查询点，即 $\overrightarrow{\boldsymbol{\Psi}}_q^t = \text{knn}(\boldsymbol{x}_q^t + \epsilon, \boldsymbol{S}^{t+1})$，$\epsilon \sim \mathcal{N}(\mu, \sigma^2)$ 是一个小的高斯噪声；② 从第一帧到最后一帧构建 t-patch，但也可以反向构建，分别初始化为 $\boldsymbol{\Psi}_q^0$ 和 $\boldsymbol{\Psi}_q^T$，利

用下述公式得到双向 t-patch：

$$\overleftrightarrow{\boldsymbol{P}} = \left(\cup_q \overrightarrow{\boldsymbol{P}}_q\right) \cup \left(\cup_p \overleftarrow{\boldsymbol{P}}_p\right), \tag{3.26}$$

其中，$\cup_q$ 和 $\cup_p$ 表示一个点云序列的所有下采样点，$\overleftarrow{\boldsymbol{P}}_p = \langle \overleftarrow{\boldsymbol{\Psi}}_p^t \rangle$ 表示反向的 t-patch，$\overleftarrow{\boldsymbol{\Psi}}_p^t = \mathrm{knn}(\boldsymbol{x}_p^{t+1}, \boldsymbol{S}^t)$，$t = T-1, \cdots, 0$。

层次化结构：所提出的结构由 l 个连续的 t-patch 模块组成，具体的网络结构如图 3.10 所示。t-patch 提取器首先对点云序列的第一帧进行下采样，使用最远点采样形成一个由 M 个查询点组成的集合，用于形成 t-patch；随后将 t-patch 输入 t-patch 网络，先通过多层感知机对每个 t-patch 的非时间维度（空间坐标）进行特征提取，共享权重的多层感知机逐步将点云特征映射至高维空间，捕捉局部几何结构；接着通过二维时序卷积层对时间维度和特征维度进行联合建模，捕捉跨帧运动依赖关系，生成每个 t-patch 的时空特征向量；不同层级的 t-patch 模块通过逐步下采样和跨层特征传递形成从局部补丁到全局结构的分层表示；最终通过最大池化聚合所有 t-patch 的特征向量，得到全局时空表示，再经全连接层和时间平滑卷积完成分类。

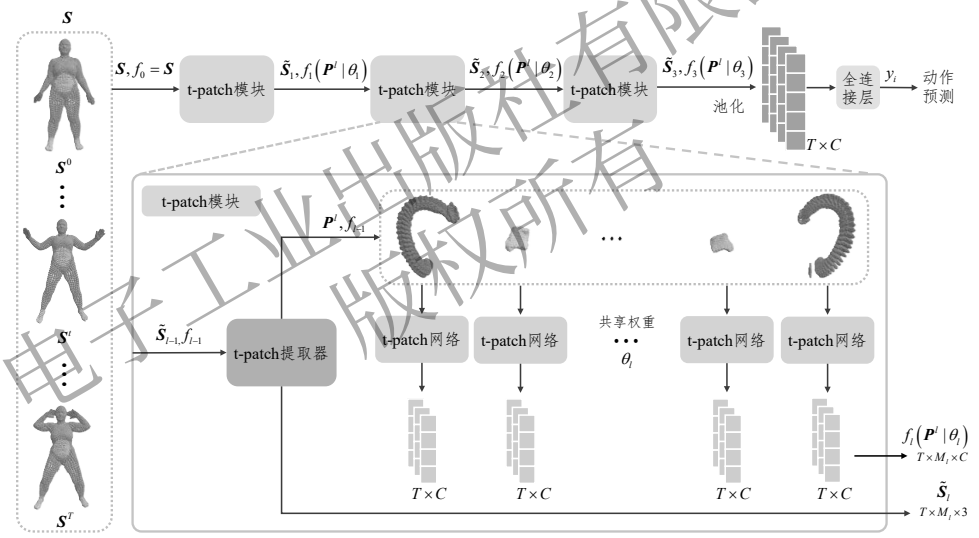

图 3.10　3DInAction 网络结构图[1]

3.3.6 半监督点云动作识别

本节主要介绍基于掩码伪标记自编码的半监督点云动作识别方法[29]。

问题描述：点云视频动作识别面临高标记成本与计算效率低下的双重挑战：现有方法依赖大量人工标记数据，且主干网络（如 Transformer）计算成本高，难以适配自动驾驶、机器人等实时场景；同时，点云的不规则性、无序性及跨帧点流动特性，导致传统体素化方法（引入量化误差与额外计算）和点跟踪方法（依赖颜色特征、跟踪误差累积）难以建模时空结构。此外，图像领域的方法因点云的不变性和无序性无法直接套用，须构建适配点云特性的半监督学习框架。

方法动机：为突破点云动作识别的实际应用瓶颈，须解决三大核心需求：首先，由于点云标记成本高于 2D RGB 视频，须利用半监督学习框架结合少量标记数据与海量无标记数据以降低标记依赖；其次，传统时空建模方法如体素化卷积、点跟踪效率低下，且现有 Transformer 方法计算复杂度高，须设计轻量化架构解耦时空维度以优化时空建模效率；同时，借鉴图像领域自监督（如 MAE）与半监督结合的思路，针对点云特性设计新的掩码重构与伪标记监督机制，以挖掘无标记数据的潜在价值从而拓展半监督范式。

方法特点：MAPLE 使用解耦的时空 Transformer（Decoupled Spatial-temporal Transformer，DestFormer）作为主干网络，将点云视频的空间与时间维度解耦，通过空间提取器利用点 4D 卷积和空间 Transformer 提取短期局部特征，时间聚合器通过时间编码器聚合长期全局信息，以高效自注意力机制建模时空特征并降低计算复杂度；构建掩码伪标记自动编码器结构，在训练中以高掩码率随机丢弃短期全局特征，利用分类头生成的伪标记作为监督信号，引导模型从可见帧重构掩码帧的特征，避免直接重构原始特征导致的训练不稳定，利用无标记数据学习判别性表示。该方法采用两阶段训练策略，先通过有监督预训练初始化模型，再结合无标记数据利用伪标记监督的特征重构优化，且可与传统半监督方法结合，是半监督点云动作识别的首次尝试。

具体方法：

MAPLE 主要由解耦的时空编码器（DestFormer）和掩码伪标记自编码器组成。

DestFormer：如图 3.11 所示，首先通过数据准备阶段利用最远点采样（Fastest Point Sampling，FPS）构建点云视频的局部时空区域，并使用点 4D 卷积提取紧凑的短期局部特征；然后通过空间提取器中的空间 Transformer 模块融合空间信息，生成短期全局特征；接着利用时序聚合中的时序编码器对短期全局特征进行长程时间依赖建模，最后通过预测头生成类别标记。

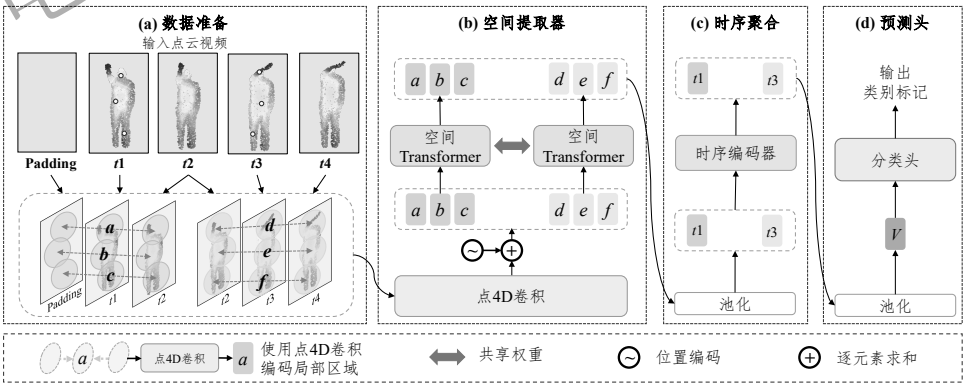

图 3.11　解耦的时空 Transformer（DestFormer）结构图[29]

掩码伪标记自编码器：用大量未标记点云视频作为输入，首先通过预训练好的空间提取器提取空间特征；然后对输出进行随机丢弃并将剩下的空间特征输入到预训练的时序编码器中；接着添加可学习的掩码 Token，通过时序解码器重构特征；最后通过预训练好的分类头生成伪标记，并使用 Kullback-Leibler（KL）散度损失来计算原始空间特征和重构空间

特征生成伪标记之间的分布差异：

$$L_u = L_{\text{maple}} = \frac{1}{|D_u|} \sum_{x_i \in D_u} \text{KL}(f(\boldsymbol{P}_i|x_i)||f(\hat{\boldsymbol{P}}_i|x_i)), \tag{3.27}$$

其中，$|D_u|$ 表示未标记数据集的大小，$\boldsymbol{P}_i$ 是根据原始特征生成的伪标记，$\hat{\boldsymbol{P}}_i$ 是根据重构特征生成的伪标记。

此外，框架采用两阶段训练策略：首先在标记数据集上通过有监督学习预训练 DestFormer，然后在无标记数据集上利用掩码伪标记机制进行半监督微调，同时可与虚拟对抗训练（Virtual Adversarial Training，VAT）、条件熵最小化（Conditional Entropy Minimization，EntMin）等传统半监督方法结合，进一步提升模型对低标记数据的学习能力。具体的结构图和算法流程分别如图 3.12 和算法 3.1 所示。

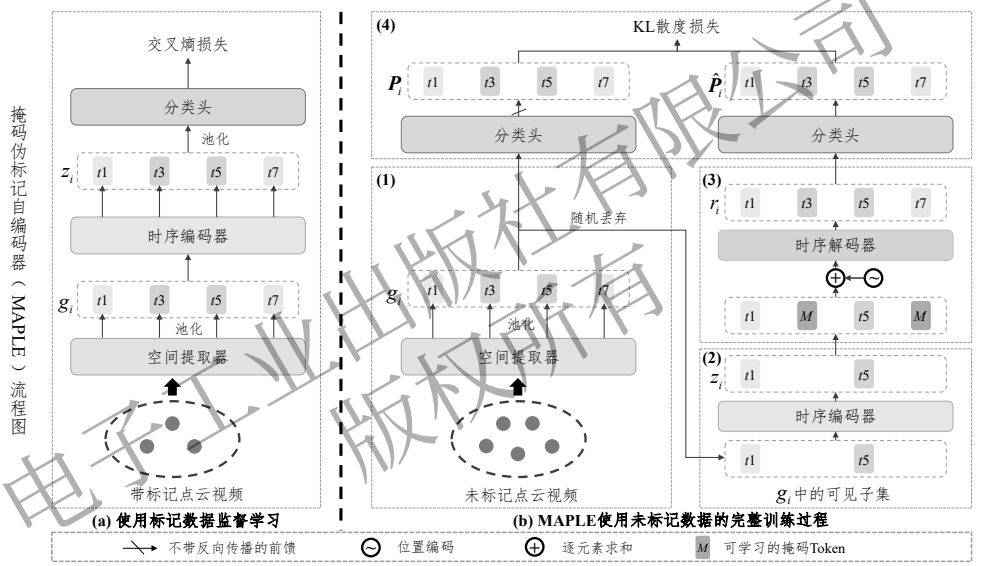

图 3.12　掩码伪标记自编码器（MAPLE）流程图[29]

3.3.7 自监督点云动作识别

本节主要介绍基于掩码时空结构预测的自监督点云动作识别方法[31]。

问题描述：当下，点云视频理解领域的多数方法高度依赖大量带标记的数据，然而对点云视频进行标记往往成本极高。并且，仅借助分类、分割等少数传统任务来训练模型，难以让模型学习到点云视频中时空结构的细微细节。尽管自监督学习在图像、视频和静态点云领域已得到应用，但在点云视频这类 4D 数据的处理中，尚未有成熟的自监督学习方法来充分利用其外观和运动结构信息。

方法动机：为解决点云视频标记成本高昂以及传统监督学习对时空结构细节捕捉不足的问题，研究人员期望通过自监督学习，利用数据自身的监督信号，使模型能够从海量数据中自动学习，无须人工标记，进而识别数据中更细微的模式。由于点云视频兼具空间不

算法 3.1 MAPLE 的训练过程.

输入： 带标记数据 D_l, 未标记数据 D_u, DestFormer 网络的初始化参数 θ, 基础学习率 η, 带标记样本的批量大小 b_l, 未标记样本的批量大小 b_u, 初始化的监督损失 L_l, 初始化的无监督损失 L_u 和其正标量权重 α.

输出： 动作类别.

Repeat：
  $t = 1, \cdots$, 最大迭代次数:
    从 D_l 获得最小批次的数据 d_l;
    在 d_l 上计算损失 L_l;
    更新 $\theta^t = \theta^{t-1} - \eta \bigtriangledown L_l$;
Until：验证集上的准确率和损失稳定;
Repeat：
  $t = 1, \cdots$, 最大迭代次数:
    分别从 D_l 和 D_u 获得最小批次的数据 d_l 和 d_u;
    在 d_l 和 d_u 上计算损失 $L = L_l + \alpha L_u$;
    更新 $\theta^t = \theta^{t-1} - \eta \bigtriangledown L$;
Until：验证集上的准确率和损失稳定.

规则性和时间规律性，研究人员需要设计专门的自监督学习方法捕捉其外观信息（物体的空间特征）和运动信息（物体的动态变化），填补 4D 点云视频自监督学习领域的研究空白。

方法特点：掩码时空结构预测的自监督学习方法设计基于时空点管道（Point Tube）掩码的 4D 掩蔽预测框架，通过重建掩码点管道捕捉点云视频的外观结构，并提出时间基数差异（Temporal Cardinality Difference，CD）预测任务，将点管的球形支撑域划分为 8 个卦限，通过计算相邻帧卦限内的点数概率分布差异来建模动态特征，迫使模型学习点云的运动信息。该方法无需人工标记，通过双流任务联合学习外观与运动结构，且时间基数差异可直接从原始输入在线计算，无需额外参数。

具体方法：

如图 3.13 所示，首先将其划分为多个点管。接下来执行掩码操作，将这些点管分为可见部分和不可见部分。可见的点管被输入到编码器中，然后其更新后的嵌入表示与掩码的点管一起输入到解码器中。随后，执行双流预测任务，分别用于恢复掩码点管内的点坐标并推断其时间基数差异。各部分的详细介绍如下：

掩码：包括三个步骤：Input Division、Embedding 和 Masking。（1）Input Division：给定一个点云视频 $\boldsymbol{P}$，用最远点采样选择 N 个关键点 $\hat{\boldsymbol{p}}$。为每个关键点构建一个点管道，以第 i 个关键点为中心的点管道表示为 $\mathbf{Tube}_{\hat{p}_i} = \{\boldsymbol{p} | \boldsymbol{p} \in \boldsymbol{P},\ \mathcal{D}_s(\boldsymbol{p}, \hat{\boldsymbol{p}}_i) < r,\ \mathcal{D}_t(\boldsymbol{p}, \hat{\boldsymbol{p}}_i) < \frac{l}{2}\}$，然后对每个空间邻居随机采样选取 n 个点。这样一来，一个点云视频被分为 N 个点管道，每个点管道包含 $l \times n$ 个点。为了确保所有的点都被点管道覆盖，设置 r 和 l 以保持相邻点管之间的轻微重叠。（2）Embedding：遵循 P4Transformer 的操作，给定点管道，得到位置编码，

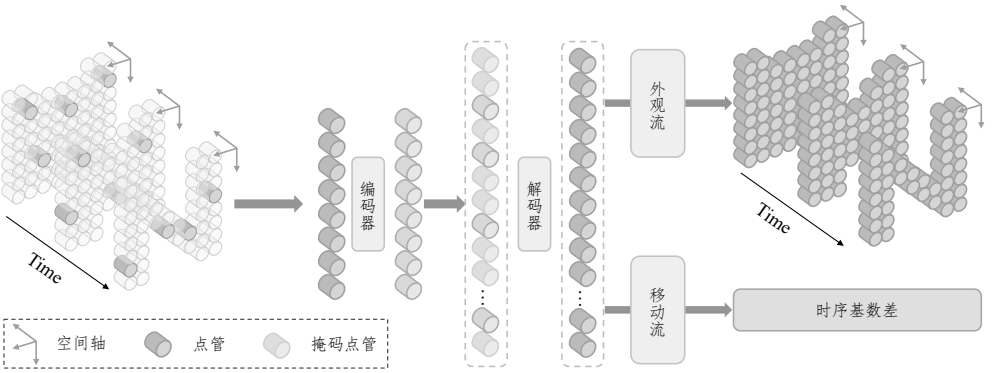

图 3.13　掩码时空结构预测（MaST-Pre）结构图[31]

具体公式如下：

$$\boldsymbol{E}_{\hat{\boldsymbol{p}}_i} = \sum_{t=1}^{l} \sum_{\boldsymbol{p} \in \mathbf{Tube}_{\hat{\boldsymbol{p}}_i}^t} f(\boldsymbol{p} - \hat{\boldsymbol{p}}_i), \tag{3.28}$$

其中，$f(\cdot)$ 是基于多层感知机的特征提取器。（3）Masking：由于时空连贯性，点云视频的冗余度高于低维数据，所以采用随机掩码策略和较高的掩码比例，高掩码比例有助于缓解信息泄露，并使重建成为有意义的自监督任务。

自编码：基于 P4Transformer，使用不对称的编码器和解码器。编码器联合时空注意力，只对可见的点管道进行时空位置编码；解码器相比于编码器采用轻量级的 Transformer。将已编码的点管道和掩码的点管道作为输入，通过向所有标记添加完整的时空位置嵌入，为自监督学习提供位置线索。解码后，只将掩码点管道的嵌入信息馈送到后续的预测头。

双流预测：外观流通过重建掩码点管道的坐标，利用 Chamfer 距离损失函数（式（3.29））让模型学习点云视频的外观结构：

$$L_{\text{app}} = \frac{1}{l} \sum_{i=1}^{l} \left\{ \frac{1}{|\boldsymbol{P}_{\text{pre}}^i|} \sum_{a \in \boldsymbol{P}_{\text{pre}}^i} \min_{b \in \boldsymbol{P}_{\text{gt}}^i} \|a - b\|_2^2 + \frac{1}{|\boldsymbol{P}_{\text{gt}}^i|} \sum_{b \in \boldsymbol{P}_{\text{gt}}^i} \min_{a \in \boldsymbol{P}_{\text{pre}}^i} \|b - a\|_2^2 \right\}, \tag{3.29}$$

其中，$|\boldsymbol{P}_{\text{pre}}|$ 和 $|\boldsymbol{P}_{\text{gt}}|$ 分别表示重构点云和真实点云中点的数量，l 是点云视频的总长度。运动流提出时间基数差预测任务，将点管中关键帧的球形支撑域划分为 8 个卦限，计算各卦限的点数概率分布，通过相邻帧的基数直方图差异来表征点的流动情况，计算预测和真实坐标差异损失：

$$L_m^i = \begin{cases} 0.5 \times (\boldsymbol{M}_{\text{pre}}^i - \boldsymbol{M}_{\text{gt}}^i)^2, & \text{if } |\boldsymbol{M}_{\text{pre}}^i - \boldsymbol{M}_{\text{gt}}^i| < 1, \\ |\boldsymbol{M}_{\text{pre}}^i - \boldsymbol{M}_{\text{gt}}^i| - 0.5, & \text{otherwise.} \end{cases}, \tag{3.30}$$

$$L_{\text{motion}} = \frac{1}{l-1} \sum_{i=1}^{l-1} L_m^i, \tag{3.31}$$

其中，$\boldsymbol{M}_{\text{pre}} \in \mathbb{R}^{(l-1)\times 8}$，$\boldsymbol{M}_{\text{gt}} \in \mathbb{R}^{(l-1)\times 8}$ 分别表示重构点云和真实点云的时序基数差预测。然后通过式（3.31）得到移动流损失，总的损失如下所示：

$$L_{\text{total}} = L_{\text{app}} + L_{\text{motion}}. \tag{3.32}$$

3.4 代码实例

3.4.1 RGB 动作识别方法

本实例主要介绍基于时序差分网络的高效动作识别方法[7]。

时序差分网络（Temporal Difference Networks，TDN）是一种高效的视频动作识别框架，其核心创新在于通过 Temporal Difference Module (TDM) 捕捉视频中的运动信息，替代传统的光流或 3D 卷积方法。该文献提出，TDN 采用双层时序差分建模范式，分为短期时序差分模块 (S-TDM) 和长期时序差分模块 (L-TDM)，分别用于局部和全局运动建模。整个网络架构基于 Temporal Segment Network (TSN) 的稀疏采样策略，将视频划分为多个片段，每段采样一帧作为输入。

```
class TDN_Net(nn.Module):
"""Temporal Difference Networks网络类"""
    def __init__(self, resnet_model, resnet_model1, apha, belta):
        super(TDN_Net, self).__init__()

        # 提取主干网络的第一个卷积层和批归一化层
        self.conv1 = list(resnet_model.children())[0]  # 主干网络的第一个卷积层
        self.bn1 = list(resnet_model.children())[1]  # 主干网络的第一个批归一化层
        self.relu = nn.ReLU(inplace=True)  # ReLU 激活函数

        # 实现 conv1_5 并扩展其权重维度
        self.conv1_temp = list(resnet_model1.children())[0]  # 辅助网络的第一个卷积层
        params = [x.clone() for x in self.conv1_temp.parameters()]  # 复制参数
        kernel_size = params[0].size()  # 获取卷积核尺寸
        new_kernel_size = kernel_size[:1] + (3 * 4,) + kernel_size[2:]  # 扩展输入通道
            维度
        new_kernels = params[0].data.mean(dim=1, keepdim=True).expand(new_kernel_size)
            .contiguous()
        # 创建新的卷积层并初始化权重
        self.conv1_5 = nn.Sequential(
            nn.Conv2d(12, 64, kernel_size=7, stride=2, padding=3, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True)
        )
        self.conv1_5[0].weight.data = new_kernels  # 设置新卷积层的权重

        # 定义最大池化层用于处理时间差特征
        self.maxpool_diff = nn.MaxPool2d(kernel_size=3, stride=2, padding=1,
            dilation=1, ceil_mode=False)

        # 提取辅助网络的 layer1 和主干网络的 maxpool 层
        self.resnext_layer1 = nn.Sequential(*list(resnet_model1.children())[4])
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1,
            dilation=1, ceil_mode=False)

        # 提取主干网络的各个残差块
        self.layer1_bak = nn.Sequential(*list(resnet_model.children())[4])  # 主干网络
            的 layer1
```

```
        self.layer2_bak = nn.Sequential(*list(resnet_model.children())[5])  # 主干网络的 layer2
        self.layer3_bak = nn.Sequential(*list(resnet_model.children())[6])  # 主干网络的 layer3
        self.layer4_bak = nn.Sequential(*list(resnet_model.children())[7])  # 主干网络的 layer4

        # 全局平均池化层和全连接层
        self.avgpool = nn.AvgPool2d(7, stride=1)  # 全局平均池化
        self.avg_diff = nn.AvgPool2d(kernel_size=2, stride=2)  # 下采样池化
        self.fc = list(resnet_model.children())[8]  # 全连接层 (分类器)

        # 存储融合系数
        self.apha = apha  # 主分支权重
        self.belta = belta  # 辅助分支 (时间差) 权重

    def forward(self, x):
        # 将输入分割成多个帧
        x1 = x[:, 0:3, :, :]
        x2 = x[:, 3:6, :, :]
        x3 = x[:, 6:9, :, :]
        x4 = x[:, 9:12, :, :]
        x5 = x[:, 12:15, :, :]

        # 计算相邻帧之间的差异，并通过 conv1_5 卷积层提取时间差特征
        x_c5 = self.conv1_5(
            self.avg_diff(
                torch.cat([x2 - x1, x3 - x2, x4 - x3, x5 - x4], 1)
                .view(-1, 12, x2.size()[2], x2.size()[3])
            )
        )
        x_diff = self.maxpool_diff(1.0 / 1.0 * x_c5)

        # 存储中间结果用于后续融合
        temp_out_diff1 = x_diff
        x_diff = self.resnext_layer1(x_diff)  # 辅助网络的 layer1

        # 主干网络前向传播
        x = self.conv1(x3)
        x = self.bn1(x)
        x = self.relu(x)

        # 融合 layer1 的输出与时间差特征
        x = self.maxpool(x)
        temp_out_diff1 = F.interpolate(temp_out_diff1, x.size()[2:])  # 上采样时间差特征
        x = self.apha * x + self.belta * temp_out_diff1  # 加权融合

        # 融合 layer2 的输出与时间差特征
        x = self.layer1_bak(x)
        x_diff = F.interpolate(x_diff, x.size()[2:])  # 上采样时间差特征
        x = self.apha * x + self.belta * x_diff  # 加权融合

```

```
        # 继续主干网络的前向传播
        x = self.layer2_bak(x)
        x = self.layer3_bak(x)
        x = self.layer4_bak(x)

        # 全局平均池化和全连接层
        x = self.avgpool(x)
        x = x.view(x.size(0), -1)  # 展平
        x = self.fc(x)  # 全连接层输出

        return x
```

在 TDN 的实现中，TDN_Net 类是网络的核心入口，负责处理输入张量并整合短期和长期运动特征。假设输入张量为 x，为了便于计算相邻帧的差分，代码首先将 x 切分为多个部分。输入被分为 x1、x2、x3、x4 和 x5，分别对应于视频中的 5 个采样帧。具体切分方式如下：

x1 = x[:, 0:3, :, :]：第一帧的 RGB 通道，形状为 [batch_size, 3, height, width]。

x2 = x[:, 3:6, :, :]：第二帧的 RGB 通道。

x3 = x[:, 6:9, :, :]：第三帧的 RGB 通道。

x4 = x[:, 9:12, :, :]：第四帧的 RGB 通道。

x5 = x[:, 12:15, :, :]：第五帧的 RGB 通道。

短期时序差分模块 (S-TDM) 的核心任务是通过相邻帧的 RGB 差分捕捉局部运动信息。具体计算过程是，代码使用 torch.cat([x2-x1, x3-x2, x4-x3, x5-x4], dim=1) 将四组差分特征拼接起来。每组差分 (如 x2-x1) 的形状为 [batch_size, 3, height, width]，四组拼接后得到形状为 [batch_size, 12, height, width] 的差分特征张量。这 12 个通道来源于 4 组差分，每组 3 个 RGB 通道。

接下来，这组差分特征被送入一个 2D 卷积层 conv1_5 进行处理。conv1_5 的输入通道数为 12，输出通道数为 64，卷积核大小通常为 3×3。通过这一层，网络将原始的 RGB 差分转化为更高维的特征表示，模拟了 3D 卷积的时序建模效果。处理后的特征 (记为 x_c5) 再经过一个最大池化层 maxpool_diff 下采样，生成短期运动特征 x_diff，其形状变为 [batch_size, 64, height/2, width/2] (假设池化步幅为 2) 。这一过程不仅提取了短期运动信息，还通过池化减少了特征的空间维度，从而降低后续计算的复杂度。

主干网络与短期特征融合：

短期 TDM 生成的特征 x_diff 在网络的早期阶段与主干特征 x 融合。融合之前，x_diff 会通过一个 ResNeXt 层 (resnext_layer1) 进一步处理，生成 temp_out_diff1。融合公式为：

$$x = \text{self.apha} \times x + \text{self.belta} \times \text{temp\_out\_diff1},$$

其中，self.apha 和 self.belta 为超参数。这种残差连接方式将短期运动信息注入主干特征，增强了网络对局部动态的感知能力。在进入 layer2 之前，x_diff (经过插值调整尺寸以匹配主干特征) 再次与 x 融合，融合方式相同。这种在网络早期多次融合短期特征的设计，体现了文献中“短期运动建模”的思想，即通过低层特征捕捉细粒度的运动细节。

```
class BottleneckShift(nn.Module):
    expansion = 4  # 设置expansion为4，这是ResNet中bottleneck块的扩展系数
    def __init__(self, num_segments, inplanes, planes, stride=1, downsample=None):
        super(BottleneckShift, self).__init__()  # 调用父类的初始化方法
        # 定义第一个1x1卷积层，输入通道为inplanes，输出通道为planes
        self.conv1 = nn.Conv2d(inplanes, planes, kernel_size=1, bias=True)
        self.bn1 = nn.BatchNorm2d(planes)  # 定义批归一化层，通道数为planes
        self.num_segments = num_segments  # 保存num_segments参数，表示时间片段数量
        # 实例化mSEModule，输入通道为planes，时间片段数为num_segments，索引为1
        self.mse = mSEModule(planes, n_segment=self.num_segments, index=1)
        # 实例化ShiftModule，输入通道为planes，时间片段数为num_segments，通道分组数为8
        self.shift = ShiftModule(planes, n_segment=self.num_segments, n_div=8, mode='
            shift')
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=stride, padding
            =1, bias=True)
        self.bn2 = nn.BatchNorm2d(planes) # 定义批归一化层，通道数为planes
        self.conv3 = nn.Conv2d(planes, planes * 4, kernel_size=1, bias=True)
        self.bn3 = nn.BatchNorm2d(planes * 4)  # 定义批归一化层，通道数为planes*4
        self.relu = nn.ReLU(inplace=True)  # 定义ReLU激活函数，inplace=True表示原地操
            作
        self.downsample = downsample  # 保存downsample参数，用于残差连接的下采样
        self.stride = stride  # 保存stride参数，表示卷积步长

    def forward(self, x):
        residual = x  # 保存残差连接的输入
        out = self.conv1(x)  # 通过第一个1×1卷积层
        out = self.bn1(out)  # 通过批归一化层
        out = self.relu(out)  # 通过ReLU激活函数
        out = self.mse(out)  # 通过mSEModule模块，增强时序注意力
        out = self.shift(out)  # 通过ShiftModule模块，进行特征移位
        out = self.conv2(out)  # 通过第二个3×3 卷积层
        out = self.bn2(out)  # 通过批归一化层
        out = self.relu(out)  # 通过ReLU激活函数
        out = self.conv3(out)  # 通过第三个1×1卷积层
        out = self.bn3(out)  # 通过批归一化层
        if self.downsample is not None:  # 如果downsample不为空
            residual = self.downsample(x)  # 对残差连接的输入进行下采样
        out += residual  # 加上残差连接
        out = self.relu(out)  # 通过ReLU激活函数
        return out  # 返回输出
```

BottleneckShift 块在标准 ResNet 的瓶颈结构 (1×1 卷积降维、3×3 卷积特征提取、1×1 卷积升维) 基础上，加入了 mSEModule 和 ShiftModule 两个组件，用于增强时序建模能力。

```
class mSEModule(nn.Module):
    def __init__(self, channel, n_segment=8, index=1):
        # (省略)

    def forward(self, x):
        bottleneck = self.conv1(x)  # 通过第一个1x1卷积层，降维特征
        bottleneck = self.bn1(bottleneck)  # 通过批归一化层
        # 重塑特征为[n, t, c, h, w]，分离时间维度
        reshape_bottleneck = bottleneck.view((-1, self.n_segment) + bottleneck.size()
```

```
            [1:])
        # 分割得到前向特征，去掉最后一个时间步
        t_fea_forward, _ = reshape_bottleneck.split([self.n_segment -1, 1], dim=1)
        # 分割得到后向特征，去掉第一个时间步
        _, t_fea_backward = reshape_bottleneck.split([1, self.n_segment -1], dim=1)
        conv_bottleneck = self.conv2(bottleneck)  # 通过第二个3x3卷积层
        # 重塑特征为[n, t, c, h, w]
        reshape_conv_bottleneck = conv_bottleneck.view((-1, self.n_segment) +
            conv_bottleneck.size()[1:])
        # 分割得到t+1前向特征，去掉第一个时间步
        _, tPlusone_fea_forward = reshape_conv_bottleneck.split([1, self.n_segment-1],
             dim=1)
        # 分割得到t+1后向特征，去掉最后一个时间步
        tPlusone_fea_backward, _ = reshape_conv_bottleneck.split([self.n_segment-1,
            1], dim=1)
        # 计算前向差分，表示时间变化
        diff_fea_forward = tPlusone_fea_forward - t_fea_forward
        # 计算后向差分，表示时间变化
        diff_fea_backward = tPlusone_fea_backward - t_fea_backward
        # 对前向差分进行填充，补齐时间维度
        diff_fea_pluszero_forward = F.pad(diff_fea_forward, self.pad1_forward, mode="
            constant", value=0)
        # 重塑为[nt, c, h, w]
        diff_fea_pluszero_forward = diff_fea_pluszero_forward.view((-1,) +
            diff_fea_pluszero_forward.size()[2:])
        # 对后向差分进行填充，补齐时间维度
        diff_fea_pluszero_backward = F.pad(diff_fea_backward, self.pad1_backward, mode
            ="constant", value=0)
        diff_fea_pluszero_backward = diff_fea_pluszero_backward.view((-1,) +
            diff_fea_pluszero_backward.size()[2:])  # 重塑为[nt, c, h, w]
        # 对前向差分应用2×2平均池化，提取小尺度特征
        y_forward_smallscale2 = self.avg_pool_forward2(diff_fea_pluszero_forward)
        # 对后向差分应用2×2平均池化，提取小尺度特征
        y_backward_smallscale2 = self.avg_pool_backward2(diff_fea_pluszero_backward)
        # 使用原始尺度的前向差分，作为更大尺度特征
        y_forward_smallscale4 = diff_fea_pluszero_forward
        # 使用原始尺度的后向差分，作为更大尺度特征
        y_backward_smallscale4 = diff_fea_pluszero_backward
        # 对小尺度2前向特征通过卷积和批归一化
        y_forward_smallscale2 = self.bn3_smallscale2(self.conv3_smallscale2(
            y_forward_smallscale2))
        # 对小尺度2后向特征通过卷积和批归一化
        y_backward_smallscale2 = self.bn3_smallscale2(self.conv3_smallscale2(
            y_backward_smallscale2))
        # 对小尺度4前向特征通过卷积和批归一化
        y_forward_smallscale4 = self.bn3_smallscale4(self.conv3_smallscale4(
            y_forward_smallscale4))
        # 对小尺度4后向特征通过卷积和批归一化
        y_backward_smallscale4 = self.bn3_smallscale4(self.conv3_smallscale4(
            y_backward_smallscale4))
        # 对小尺度2前向特征插值到原始大小
        y_forward_smallscale2 = F.interpolate(y_forward_smallscale2,
            diff_fea_pluszero_forward.size()[2:])
```

```
        # 对小尺度2后向特征插值到原始大小
        y_backward_smallscale2 = F.interpolate(y_backward_smallscale2,
            diff_fea_pluszero_backward.size()[2:])
        # 融合多尺度前向特征并通过卷积和批归一化
        y_forward = self.bn3(self.conv3(1.0/3.0*diff_fea_pluszero_forward + 1.0/3.0*
            y_forward_smallscale2 + 1.0/3.0*y_forward_smallscale4))
        # 融合多尺度后向特征并通过卷积和批归一化
        y_backward = self.bn3(self.conv3(1.0/3.0*diff_fea_pluszero_backward + 1.0/3.0*
            y_backward_smallscale2 + 1.0/3.0*y_backward_smallscale4))
        # 对前向特征应用Sigmoid激活并减0.5，生成注意力权重
        y_forward = self.sigmoid_forward(y_forward) - 0.5
        # 对后向特征应用Sigmoid激活并减0.5，生成注意力权重
        y_backward = self.sigmoid_backward(y_backward) - 0.5
        y = 0.5*y_forward + 0.5*y_backward  # 平均前向和后向注意力权重
        output = x + x*y  # 将注意力权重应用于输入特征
        return output  # 返回输出
```

多尺度时序注意力模块 (mSEModule) 是 TDN 中长期时序差分模块 (L-TDM) 的核心组成部分，旨在通过跨视频片段的时序差分和多尺度特征融合捕捉长期运动信息，从而增强动作识别的性能。其设计灵感来源于对长范围时序结构的建模需求，通过计算相邻片段间的特征差分并生成注意力图来突出关键时序信息。输入特征首先通过 1×1 卷积降维至通道数的 1/16，随后重塑为 [n, t, c, h, w] 的形状，其中 t 表示片段数，以分离时间维度。接着，模块计算前向差分 (t+1 帧减 t 帧) 和后向差分 (t 帧减 t+1 帧)，并通过填充补齐时间维度。这些差分特征在三个尺度 (原尺度、2×2 池化和 4×4 池化) 下分别处理：原尺度直接使用差分，2×2 池化和 4×4 池化通过平均池化降低空间分辨率，每种尺度再通过 3×3 卷积和批归一化进一步提取特征。多尺度特征随后以 1:1:1 的比例融合，通过 1×1 卷积和 Sigmoid 激活函数生成前向和后向注意力图，公式为 y_forward = Sigmoid(Conv(1/3diff_fea + 1/3y_smallscale2 + 1/3y_smallscale4)) – 0.5 和类似的后向公式。最终注意力图通过平均前向和后向权重 (y = 0.5y_forward + 0.5y_backward) 得到，并以残差形式应用于原始特征，即 output = x + x*y。这种多尺度双向注意力机制有效捕捉了视频中长期的时序依赖，增强了特征的表达能力，同时保持了计算效率，符合 TDN 高效动作识别的设计目标。由于 mSEModule 类代码过长，故此处只给出其 forward 函数部分。

3.4.2 基于膨胀三维卷积神经网络的动作识别

本实例主要介绍基于膨胀三维卷积神经网络的动作识别方法[9]。

I3D（Inflated 3D ConvNet）模型是为视频动作识别设计的深度学习架构，其核心由两个主要模块组成：Unit3D 和 InceptionI3d。Unit3D 是基础构建单元，通过执行 3D 卷积、批量归一化（batch normalization）和非线性激活函数，处理视频数据的空间和时间特征。它是模型能够捕捉视频中时空信息的基本组件。InceptionI3d 则是完整的模型架构，通过集成多个 Unit3D 层、池化操作，以及 Inception 模块，构建了一个深度网络，用于视频中的动作分类。Inception 模块采用多尺度卷积分支设计，能够提取不同尺度的特征，这一架构是在 2D Inception-v1 模型基础上扩展为 3D 输入，并利用 ImageNet 预训练权重提升性能。

```
class Unit3D(snt.AbstractModule):  # 定义Unit3D类，继承自Sonnet的AbstractModule基类
```

```
    def __init__(self, output_channels,  # 初始化函数，接收输出通道数参数
                 kernel_shape=(1, 1, 1),  # 卷积核形状参数，默认值为(1, 1, 1)，表示时间、高、宽维度
                 stride=(1, 1, 1),  # 卷积步长参数，默认值为(1, 1, 1)，表示时间、高、宽维度
                 activation_fn=tf.nn.relu,  # 激活函数参数，默认为ReLU函数
                 use_batch_norm=True,  # 是否使用批量归一化的布尔参数，默认为True
                 use_bias=False,  # 是否在卷积中使用偏置的布尔参数，默认为False
                 name='unit_3d'):  # 模块名称参数，默认为'unit_3d'

      super(Unit3D, self).__init__(name=name)  # 调用父类的初始化方法，设置模块名称
      self._output_channels = output_channels  # 将输出通道数保存为实例变量
      self._kernel_shape = kernel_shape  # 将卷积核形状保存为实例变量
      self._stride = stride  # 将卷积步长保存为实例变量
      self._use_batch_norm = use_batch_norm  # 将是否使用批量归一化保存为实例变量
      self._activation_fn = activation_fn  # 将激活函数保存为实例变量
      self._use_bias = use_bias  # 将是否使用偏置保存为实例变量

    def _build(self, inputs, is_training):  # 定义构建函数，连接输入到模块的计算图
      """
      Args:  # 参数说明
        inputs: Unit3D组件的输入.
        is_training: 是否使用 snt.BatchNorm 的训练模式，布尔类型.
      Returns:  # 返回值说明
        返回模块的输出张量
      """
      # 使用Sonnet的Conv3D执行3D卷积，指定输出通道数
      net = snt.Conv3D(output_channels=self._output_channels,
                       kernel_shape=self._kernel_shape,  # 指定卷积核形状
                       stride=self._stride,  # 指定卷积步长
                       padding=snt.SAME,  # 使用SAME填充，确保输出尺寸与输入匹配（考虑步长）
                       use_bias=self._use_bias)(inputs)  # 指定是否使用偏置，并将卷积应用于输入
      if self._use_batch_norm:  # 如果启用了批量归一化
        bn = snt.BatchNorm()  # 创建Sonnet的BatchNorm对象
        # 对卷积输出应用批量归一化，指定训练模式
        net = bn(net, is_training=is_training, test_local_stats=False)
      if self._activation_fn is not None:  # 如果指定了激活函数
        net = self._activation_fn(net)  # 将激活函数应用于当前张量
      return net  # 返回处理后的输出张量
```

InceptionI3d 是 I3D 模型的主类，实现了基于 Inception-v1 的 3D 架构。它通过一系列卷积层、池化层和 Inception 模块构建网络，能够处理视频输入并生成动作分类的预测。文献中提到，该模型通过将 2D Inception 架构膨胀为 3D，并利用 ImageNet 预训练权重，显著提升了动作识别性能。以下代码展示了其核心结构。

```
class InceptionI3d(snt.AbstractModule):  # 定义InceptionI3d类，继承自Sonnet的AbstractModule基类
    VALID_ENDPOINTS = (  # 定义有效端点元组，列出模型中所有可能的输出层名称
        'Conv3d_1a_7x7',  # 第一个7×7×7卷积层
        'MaxPool3d_2a_3x3',  # 第一个3×3最大池化层
        'Conv3d_2b_1x1',  # 第二个1×1×1卷积层
```

```
        'Conv3d_2c_3x3',  # 第三个3×3×3卷积层
        'MaxPool3d_3a_3x3',  # 第二个3×3最大池化层
        'Mixed_3b',  # 第一个Inception模块
        'Mixed_3c',  # 第二个Inception模块
        'MaxPool3d_4a_3x3',  # 第三个3×3最大池化层
        'Mixed_4b',  # 第三个Inception模块
        'Mixed_4c',  # 第四个Inception模块
        'Mixed_4d',  # 第五个Inception模块
        'Mixed_4e',  # 第六个Inception模块
        'Mixed_4f',  # 第七个Inception模块
        'MaxPool3d_5a_2x2',  # 2×2最大池化层
        'Mixed_5b',  # 第八个Inception模块
        'Mixed_5c',  # 第九个Inception模块
        'Logits',  # 分类logits层
        'Predictions',  # 最终预测层
    )

    # 初始化函数，接收类别数和空间压缩参数
    # 指定最终端点和模块名称，默认为'Logits'和'inception_i3d'
    def __init__(self, num_classes=400, spatial_squeeze=True,
                 final_endpoint='Logits', name='inception_i3d'):

      super(InceptionI3d, self).__init__(name=name)  # 调用父类的初始化方法，设置模块
          名称
      self._num_classes = num_classes  # 将类别数保存为实例变量
      self._spatial_squeeze = spatial_squeeze  # 将是否压缩空间维度保存为实例变量
      self._final_endpoint = final_endpoint  # 将最终端点保存为实例变量
```

以上为 InceptionI3d 的有效端点元组设置以及构建函数，构建函数的参数说明如下：num_classes: 类别数参数，默认为 400，与 Kinetics 数据集匹配。spatial_squeeze: 是否压缩空间维度的布尔参数，默认为 True。final_endpoint: 最终端点参数，指定构建模型的最后一个端点，除最终端点外，所有中间端点输出也将以字典形式返回，必须是 VALID_ENDPOINTS 中的一个，默认为“Logits”。name: 模块名称参数，可选，类型为字符串。下面是构建函数的示例代码。

```
    def _build(self, inputs, is_training, dropout_keep_prob=1.0):
      net = inputs  # 将输入张量赋值给net，作为网络的起点
      end_points = {}  # 初始化端点字典，用于存储所有中间输出
      end_point = 'Conv3d_1a_7x7'  # 定义第一个端点名称为'Conv3d_1a_7x7'

      # 使用Unit3D创建第一个卷积层，输出64通道，卷积核为7×7×7
      # 设置步长为2×2×2，并将结果应用于net
      net = Unit3D(output_channels=64, kernel_shape=[7, 7, 7],
                   stride=[2, 2, 2], name=end_point)(net, is_training=is_training)
      end_points[end_point] = net  # 将当前输出存入端点字典
      # 如果达到最终端点，返回net和端点字典
      if self._final_endpoint == end_point: return net, end_points
      end_point = 'MaxPool3d_2a_3x3'
      # 使用3D最大池化，核大小为1×3×3，步长为1×2×2
      net = tf.nn.max_pool3d(net, ksize=[1, 1, 3, 3, 1], strides=[1, 1, 2, 2, 1],
                             padding=snt.SAME, name=end_point)
      end_points[end_point] = net
```

```
18
19      if self._final_endpoint == end_point: return net, end_points
20      end_point = 'Conv3d_2b_1x1'
21      net = Unit3D(output_channels=64, kernel_shape=[1, 1, 1],
22                   name=end_point)(net, is_training=is_training)  # 将结果应用于net
23      end_points[end_point] = net
24
25      if self._final_endpoint == end_point: return net, end_points
26      end_point = 'Conv3d_2c_3x3'
27      net = Unit3D(output_channels=192, kernel_shape=[3, 3, 3],
28                   name=end_point)(net, is_training=is_training)  # 将结果应用于net
29      end_points[end_point] = net
30
31      if self._final_endpoint == end_point: return net, end_points
32      end_point = 'MaxPool3d_3a_3x3'  # 定义第五个端点名称为'MaxPool3d_3a_3x3'
33      # 使用3D最大池化，核大小为1×3×3，步长为1×2×2
34      net = tf.nn.max_pool3d(net, ksize=[1, 1, 3, 3, 1], strides=[1, 1, 2, 2, 1],
35                             padding=snt.SAME, name=end_point)
36      end_points[end_point] = net
37
38      if self._final_endpoint == end_point: return net, end_points
39
40      # 定义端点名称为'Mixed_3b'，第一个Inception模块，此处仅给出一个示例，其他
            Inception模块类似
41      end_point = 'Mixed_3b'
42      with tf.variable_scope(end_point):  # 使用TensorFlow变量作用域命名该端点
43        with tf.variable_scope('Branch_0'):  # 定义分支0的作用域
44          # 分支0：1×1×1卷积，输出64通道
45          branch_0 = Unit3D(output_channels=64, kernel_shape=[1, 1, 1],
46                            name='Conv3d_0a_1x1')(net, is_training=is_training)
47        with tf.variable_scope('Branch_1'):  # 定义分支1的作用域
48          # 分支1第一步：1×1×1卷积，输出96通道
49          branch_1 = Unit3D(output_channels=96, kernel_shape=[1, 1, 1],
50                            name='Conv3d_0a_1x1')(net, is_training=is_training)
51          branch_1 = Unit3D(output_channels=128, kernel_shape=[3, 3, 3],
52                            name='Conv3d_0b_3x3')(branch_1, is_training=is_training)
53        with tf.variable_scope('Branch_2'):  # 定义分支2的作用域
54          branch_2 = Unit3D(output_channels=16, kernel_shape=[1, 1, 1],
55                            name='Conv3d_0a_1x1')(net, is_training=is_training)
56          branch_2 = Unit3D(output_channels=32, kernel_shape=[3, 3, 3],
57                            name='Conv3d_0b_3x3')(branch_2, is_training=is_training)
58        with tf.variable_scope('Branch_3'):  # 定义分支3的作用域
59          branch_3 = tf.nn.max_pool3d(net, ksize=[1, 3, 3, 3, 1],
60                                      strides=[1, 1, 1, 1, 1], padding=snt.SAME,
61                                      name='MaxPool3d_0a_3x3')
62          branch_3 = Unit3D(output_channels=32, kernel_shape=[1, 1, 1],
63                            name='Conv3d_0b_1x1')(branch_3, is_training=is_training)
64        # 将四个分支的输出在通道维度（第4维）上拼接
65        net = tf.concat([branch_0, branch_1, branch_2, branch_3], 4)
66      end_points[end_point] = net  # 将当前输出存入端点字典
67      # 如果达到最终端点，返回net和端点字典
68      if self._final_endpoint == end_point: return net, end_points
69
```

```
    # 以下为Inception后续模块和层的示例，由于完整代码较长，此处仅展示部分关键结构
    end_point = 'Logits'  # 定义'Logits'端点，表示分类层
    with tf.variable_scope(end_point):  # 使用TensorFlow变量作用域命名该端点
      net = tf.nn.avg_pool3d(net, ksize=[1, 2, 7, 7, 1],  # 3D平均池化，核大小为2×7×7，步长为1
                             strides=[1, 1, 1, 1, 1], padding=snt.VALID)
      net = tf.nn.dropout(net, dropout_keep_prob)  # 应用dropout，保留概率由参数指定
      # 使用Unit3D创建分类卷积层，输出通道数为类别数
      logits = Unit3D(output_channels=self._num_classes,
                      kernel_shape=[1, 1, 1],  # 卷积核为1×1×1
                      activation_fn=None,  # 无激活函数
                      use_batch_norm=False,  # 不使用批量归一化
                      use_bias=True,  # 使用偏置
                      name='Conv3d_0c_1x1')(net, is_training=is_training) # 应用于net
      if self._spatial_squeeze:  # 如果启用了空间压缩
        # 压缩空间维度（高和宽）
        logits = tf.squeeze(logits, [2, 3], name='SpatialSqueeze')
    # 在时间维度（第1维）上取平均值，得到平均logits
    averaged_logits = tf.reduce_mean(logits, axis=1)
    end_points[end_point] = averaged_logits  # 将当前输出存入端点字典
    if self._final_endpoint == end_point: return averaged_logits, end_points

    end_point = 'Predictions'  # 定义'Predictions'端点，表示最终预测
    predictions = tf.nn.softmax(averaged_logits)  # 对平均logits应用softmax函数，生成概率预测
    end_points[end_point] = predictions  # 将预测存入端点字典
    return predictions, end_points  # 返回最终预测和所有端点字典
```

参数说明：inputs: 输入参数，模型的输入张量，预期形状为 batch_size × num_frames × 224 × 224 × num_channels。is_training: 是否为训练模式的布尔参数，用于批量归一化。dropout_keep_prob: dropout 保留概率参数，范围为 [0, 1)。返回一个元组，包含指定 final_endpoint 处的网络输出，以及所有中间端点的字典，以端点名称为键。

3.4.3 基于点 4D Transformer 的点云动作识别

本实例主要介绍基于点 4D Transformer 的点云动作识别方法[22]。

由于点云视频在空间维度存在不规则性和无序性、不同帧中点的出现不一致，导致难以建模时空结构，且传统方法依赖点跟踪存在轨迹计算困难、对无色点云处理能力不足等问题。因此，该方法通过点 4D 卷积对原始点云视频的时空局部结构进行编码，在避免体素化的同时减少后续处理的点数，再利用 Transformer 对嵌入的局部特征进行自注意力操作，捕捉整个视频的外观和运动信息，通过注意力权重自适应融合相关局部区域，建模原始点云视频的时空结构。

1）局部区域特征提取：下述代码实现 P4Transformer 的局部区域特征提取模块，通过下采样减少点的数量并利用卷积来编码点云邻域特征。

```
for t in range(self.temporal_kernel_size//2, len(xyzs)-self.temporal_kernel_size//2, self.temporal_stride):  # 遍历时间关键帧

```

```
    #----------------------构建局部邻域----------------------
    # 使用FPS采样得到空间关键点索引
    anchor_idx = pointnet2_utils.furthest_point_sample(xyzs[t], npoints//self.
        spatial_stride)
    anchor_xyz_flipped = pointnet2_utils.gather_operation(xyzs[t].transpose(1, 2).
        contiguous(), anchor_idx)  # 根据索引取关键点坐标
    anchor_xyz_expanded = torch.unsqueeze(anchor_xyz_flipped, 3)  # 扩展维度用于后续计
        算
    anchor_xyz = anchor_xyz_flipped.transpose(1, 2).contiguous()  # 转置矩阵
    new_feature = []  # 存储当前关键帧帧的特征

    for i in range(t-self.temporal_kernel_size//2, t+self.temporal_kernel_size//2+1):

        #计算邻域中点到关键点的时间和空间位移
        neighbor_xyz = xyzs[i]  # 当前邻域帧的点坐标
        # 球查询找邻居点索引
        idx = pointnet2_utils.ball_query(self.r, self.k, neighbor_xyz, anchor_xyz)
        neighbor_xyz_flipped = neighbor_xyz.transpose(1, 2).contiguous()  # 转置
        # 取出邻居坐标
        neighbor_xyz_grouped = pointnet2_utils.grouping_operation(neighbor_xyz_flipped
            , idx)
        xyz_displacement = neighbor_xyz_grouped - anchor_xyz_expanded  # 计算空间位移
        # 创建时间位移
        t_displacement = torch.ones((xyz_displacement.size()[0], 1, xyz_displacement.
            size()[2], xyz_displacement.size()[3]),dtype=torch.float32, device=device)
             * (i-t)
        # 拼接空间+时间位移
        displacement = torch.cat(tensors=(xyz_displacement, t_displacement), dim=1,
            out=None)
        displacement = self.conv_d(displacement)  # 通过卷积调整维度

        #----------------------计算邻域特征----------------------
        if self.in_planes != 0:  # 如果有输入特征
            # 计算邻居特征
            neighbor_feature_grouped = pointnet2_utils.grouping_operation(features[i],
                 idx)
            feature = self.conv_f(neighbor_feature_grouped)  # 特征卷积
            if self.operator == '+':  # 使用加法
                feature = feature + displacement
            else:  # 使用乘法
                feature = feature * displacement
        else:  # 如果没有输入特征
            feature = displacement
        feature = self.mlp(feature)  # 通过 MLP 提取特征

        #----------------------空间池化----------------------
        if self.spatial_pooling == 'max':  # 空间池化：最大值
            # 对空间维度的最后一个维度进行最大池化
            feature = torch.max(input=feature, dim=-1, keepdim=False)[0]
        elif self.spatial_pooling == 'sum':  # 空间池化：求和
            # 对空间维度的最后一个维度进行求和池化
            feature = torch.sum(input=feature, dim=-1, keepdim=False)
        else:  # 空间池化：平均
```

```
48              # 对空间维度的最后一个维度进行平均池化
49              feature = torch.mean(input=feature, dim=-1, keepdim=False)
50          new_feature.append(feature)  # 存储该时间帧特征
51      new_feature = torch.stack(tensors=new_feature, dim=1)  # 堆叠邻域时间帧特征
52
53      #----------------------时序池化----------------------
54      if self.temporal_pooling == 'max':  # 时间池化：最大值
55          # 对时间维度（dim=1）进行最大池化
56          new_feature = torch.max(input=new_feature, dim=1, keepdim=False)[0]
57      elif self.temporal_pooling == 'sum':  # 时间池化：求和
58          # 对时间维度（dim=1）进行求和池化
59          new_feature = torch.sum(input=new_feature, dim=1, keepdim=False)
60      else:  # 时间池化：平均
61          # 对时间维度（dim=1）进行平均池化
62          new_feature = torch.mean(input=new_feature, dim=1, keepdim=False)
63      new_xyzs.append(anchor_xyz)  # 存储关键点坐标
64      new_features.append(new_feature)  # 存储关键点特征
65
66  new_xyzs = torch.stack(tensors=new_xyzs, dim=1)  # 堆叠所有关键点坐标
67  new_features = torch.stack(tensors=new_features, dim=1)  # 堆叠所有关键点特征
```

在上述代码中，首先通过遍历时间关键帧（行 1），在每个时间点使用最远点采样选取空间关键点索引（行 5~9），计算临域中点到关键点的时间和空间位移（行 14~25）。接着，对选定的关键点及其邻居进行特征聚合（行 28~37），根据是否存在输入特征选择加法或乘法操作来结合位移信息与邻居特征，利用多层感知机进一步提炼特征（行 38），最后采用指定的空间和时间池化策略（如最大值、求和或平均）合并特征（行 41~64）。整个过程自适应地搜索相关或相似的点，避免直接追踪点或者限制时间建模范围的问题，通过捕捉点云视频中的时空动态，增强对复杂场景的理解能力。

2）注意力模块：以下代码定义了 P4Transformer 框架的注意力模块，旨在处理点云视频数据并通过注意力机制捕捉其中复杂的时空结构。

```
1   class Attention(nn.Module):
2       #----------------------初始化----------------------
3       def __init__(self, dim, heads=8, dim_head=64, dropout=0.):
4           super().__init__()
5           inner_dim = dim_head * heads  # 多头维度总和
6           project_out = not (heads == 1 and dim_head == dim)  # 判断是否需要输出映射
7           self.heads = heads  # 头数
8           self.scale = dim_head ** -0.5  # 缩放因子
9           self.norm = nn.LayerNorm(dim)  # 层归一化
10          self.to_qkv = nn.Linear(dim, inner_dim * 3, bias=False)  # 线性层，生成qkv
11          self.spatial_op = nn.Linear(3, dim_head, bias=False)  # 空间位移特征映射
12          # 输出映射（如果 project_out 为 True）
13          self.to_out = nn.Sequential(
14              nn.Linear(inner_dim, dim),
15              nn.GELU(),
16              nn.Dropout(dropout)
17          ) if project_out else nn.Identity()  # 否则直接输出
18
19      #----------------------前向传播----------------------
20      def forward(self, xyzs, features):
```

```
21          b, l, n, _, h = *features.shape, self.heads  # 获取批量、帧、点数、特征维度、
                头数
22          norm_features = self.norm(features)  # 特征归一化
23          qkv = self.to_qkv(norm_features).chunk(3, dim=-1)  # 生成q,k,v，并在维度-1上切
                分为三部分
24          # 重排维度到多头格式
25          q, k, v = map(lambda t: rearrange(t, 'b l n (h d) -> b h (l n) d', h=h), qkv)
26          xyzs_flatten = rearrange(xyzs, 'b l n d -> b (l n) d')  # 将xyzs展开
27          # 计算点对之间的空间位移
28          delta_xyzs = torch.unsqueeze(xyzs_flatten, dim=1) - torch.unsqueeze(
                xyzs_flatten, dim=2)
29          dots = einsum('b h i d, b h j d -> b h i j', q, k) * self.scale  # 计算注意力
                分数
30          attn = dots.softmax(dim=-1)  # softmax归一化，得到注意力权重
31          v = einsum('b h i j, b h j d -> b h i d', attn, v)  # 计算加权的value
32          attn = torch.unsqueeze(attn, dim=4)  # 扩充注意力矩阵维度
33          delta_xyzs = torch.unsqueeze(delta_xyzs, dim=1)  # 扩充空间位移矩阵维度
34          # 加权求和，得到空间加权位移
35          delta_xyzs = torch.sum(attn * delta_xyzs, dim=3, keepdim=False)
36          displacement_features = self.spatial_op(delta_xyzs)  # 将位移映射到特征空间
37          out = v + displacement_features  # 空间增强特征和原特征相加
38          out = rearrange(out, 'b h m d -> b m (h d)')  # 拼接多头特征
39          out = self.to_out(out)  # 输出映射
40          out = rearrange(out, 'b (l n) d -> b l n d', l=l, n=n)  # 还原维度
41          return out + features  # 残差连接，输出增强后的特征
```

在 Attention 类中，在初始化时接收维度 (dim)、头数 (heads)、每头维度 (dim_head) 及 dropout 率作为参数（行 3），计算内部维度 (inner_dim) 并判断是否需要输出映射 (project_out)（行 6）。通过线性层生成查询 (q)、键 (k) 和值 (v) 向量（行 23），并对输入特征进行层归一化处理，同时定义 spatial_op 线性层来处理空间位移特征映射。在前向传播过程中，利用视频级自注意机制 (Video-level Self-attention)，基于输入的空间坐标 xyzs 和特征 features，通过一系列操作如切分生成 qkv 三元组、重排适应多头格式、计算点对之间的空间位移等步骤，自适应地搜索整个视频中相关或相似的点（行 23~28）。通过计算注意力分数并使用 Softmax 函数进行归一化得到注意力权重（行 29~30），利用这些权重来加权求和值向量 v，并通过考虑空间位移信息更新参考点特征（行 31~32）。最终，经过变换和重组操作后，输出增强后的特征，保留原始点云视频的时空结构（行 33~41）。

3.4.4 基于掩码伪标记自编码的半监督点云动作识别

本实例主要介绍基于掩码伪标记自编码的半监督点云动作识别方法[29]。

现阶段，点云视频动作识别面临高标记成本和计算效率低下的挑战，现有方法依赖大量标记数据且主干网络计算开销大，难以满足实时场景需求。点云的不规则性、无序性和跨帧点流动特性，使得体素化和点跟踪方法难以建模时空结构，图像领域方法也无法直接应用。为此，该方法提出适用于半监督点云动作识别的 MAPLE 框架并使用基于 Transformer 的时空融合模块整合长期全局信息。MAPLE 通过高掩码率随机丢弃短期全局特征，利用分类头生成的伪标记，引导模型从可见帧重构掩码帧特征，避免训练不稳定并高效利用无标

记数据。

1）基本函数：以下代码定义两个关键函数 ShuffleIndex() 和 MaskEmbeeding()，它们共同作用于 MAPLE。该方法为了处理点云视频数据并减少对人工标记的依赖，首先通过 ShuffleIndex() 根据设定的比例从输入索引列表中随机采样，并生成相应的掩码索引列表（行 5~9）。此过程确保即使对于长度较短的索引列表也能灵活操作，同时保证采样后的样本数量符合预期比例。接着，MaskEmbeeding() 接收经过 patch 嵌入和位置嵌入后的 Token 嵌入作为输入，利用 ShuffleIndex() 产生的掩码索引（行 15），从中选择未被掩码的部分 Token 进行后续处理。

```
1  def ShuffleIndex(index: list, sample_ratio: float):
2      if len(index) < 4:
3          raise ValueError("ipnuts must be more than 4")  # 如果输入索引长度小于4，抛出
               异常
4      else:
5          sample_length = int(sample_ratio * len(index))  # 根据采样比例计算采样的长度
6          sample_list = random.sample(index, sample_length)   # 从索引中随机采样
               sample_length个索引
7          mask_list = [x for x in index if x not in sample_list]  # 获取剩余索引（即被
               mask掉的索引）
8          assert len(sample_list) == int(len(index) * sample_ratio), "sample length must
               be same as the ratio!!!"   # 确保采样长度符合采样比例
9      return sorted(sample_list), sorted(mask_list)   # 返回排序后的采样索引和mask索引
10
11 def MaskEmbeeding(token_emb, mask_ratio):   #获取经过patch_emb+pos_emb后的掩码嵌入
12     token_length = token_emb.shape[1]  # token的数量
13     token_index = list(range(0, token_length))  # 创建索引列表 [0,1,2,...,token_length
           -1]
14     # 根据 mask_ratio 得到保留和掩码的索引
15     sample_index, mask_index = ShuffleIndex(token_index, sample_ratio=1 - mask_ratio)
16     x = token_emb[:, sample_index, :]          # 选出被保留（未mask）的位置上的token_emb
17     return x, sample_index, mask_index       # 返回未mask的token_emb、未mask索引、被
           mask索引
```

2）P4Transformer：用于带标记数据的训练，该模块代码可参考3.4.3。

3）整体框架：通过点 4D 卷积和空间 Transformer 提取空间特征的空间特征提取器、利用时序编码器捕捉长时依赖的时间特征聚合器以及用于分类的分类头。同时引入掩码机制，对短期全局特征进行高比例随机掩码，通过时间解码器结合可见特征潜在表示和掩码标记重建掩码帧特征，并重建目标为分类头生成的伪标记。

```
1  class P4Transformer(nn.Module):
2      def __init__(self, radius, nsamples, spatial_stride,
3                   temporal_kernel_size, temporal_stride,
4                   emb_relu,
5                   dim, depth_xyz, depth_t, heads, dim_head,
6                   mlp_dim, num_classes, only_cls=False, depth_t_decoder=8, dim_decoder
                        =512,
7                   mask_ratio=0.5):
8          super().__init__()
9          self.only_cls = only_cls  # 是否仅输出类别结果
10         # tube_embedding：使用P4DConv提取Tube的特征
```

```
11          self.tube_embedding = P4DConv(in_planes=0, mlp_planes=[dim], mlp_batch_norm=[
                False],
12                                        mlp_activation=[False],
13                                        spatial_kernel_size=[radius, nsamples],
14                                        spatial_stride=spatial_stride,
15                                        temporal_kernel_size=temporal_kernel_size,
16                                        temporal_stride=temporal_stride,
17                                        temporal_padding=[1, 0],
18                                        operator='+',
19                                        spatial_pooling='max',
20                                        temporal_pooling='max')
21          # 使用一维卷积进行位置编码
22          self.pos_embedding = nn.Conv1d(in_channels=4, out_channels=dim, kernel_size=1,
                 stride=1, padding=0, bias=True)
23          self.emb_relu = nn.ReLU() if emb_relu else False    # embedding后是否使用ReLU
24          # 空间（xyz）Transformer
25          self.transformer_1 = Transformer(dim, depth_xyz, heads, dim_head, mlp_dim)
26          # 时间（t）Transformer
27          self.transformer_2 = Transformer(dim, depth_t, heads, dim_head, mlp_dim)
28          if not only_cls:    # 如果不仅仅是分类
29              self.transformer_decoder_3 = Transformer(dim_decoder, depth_t_decoder,
                    heads, dim_head, mlp_dim)  # Transformer解码器
30              self.proj = nn.Linear(dim, dim_decoder)  # encoder维度到decoder维度投影
31              self.restruction = nn.Linear(dim_decoder, dim)  # 重建回dim维度
32              self.mask_ratio = mask_ratio  # mask比例
33              self.unmask_embed = UnMaskEmbeeding(dim_decoder)  # unmask过程
34              self.norm_encoder = nn.LayerNorm(dim)  # 对encoder输出做LayerNorm
35              self.decoder_pos_embedding = PositionEmbed(12, dim_decoder, 0)()  # 生成
                    decoder位置编码
36          # MLP分类头
37          self.mlp_head = nn.Sequential(
38              nn.LayerNorm(dim),  #层归一化
39              nn.Linear(dim, mlp_dim),    #线性层
40              nn.GELU(),  # GELU激活函数
41              nn.Linear(mlp_dim, num_classes),    #线性层
42          )
```

在 P4Transformer 类的初始化中。调用 P4DConv 类来实现 tube_embedding()（行 11），使用一维卷积实现 pos_embedding()（行 22），调用 Transformer 类定义空间 Transformer 和时序 Transformer（行 25~27），还为其他任务定义 decoder()、proj()、restruction()、unmask_embed()、norm_encoder()、decoder_pos_embedding()（行 28~35），最后定义 MLP 由层归一化、线性层、GELU 激活函数、线性层构成（行 37~42）。

```
1       def low_feature_forward(self, input_clips):  # 特征提取部分
2           device = input_clips.get_device()  # 获取设备
3           # 提取tube_embedding：输出点xyz位置
4           xyzs, features = self.tube_embedding(input_clips)
5           xyzts = []  #用于存储位置信息
6           # 拆成L帧，按时间维度分离，[B, 1, n, 3]
7           xyzs = torch.split(tensor=xyzs, split_size_or_sections=1, dim=1)
8           # 消除第一维度，[B, n, 3]
9           xyzs = [torch.squeeze(input=xyz, dim=1).contiguous() for xyz in xyzs]
```

```
10          for t, xyz in enumerate(xyzs):  # 遍历时间维度
11              t = torch.ones((xyz.size()[0], xyz.size()[1], 1), dtype=torch.float32,
                    device=device) * (t+1)  # 生成时间维度信息
12              xyzt = torch.cat(tensors=(xyz, t), dim=2)  # 拼接xyz和t
13              xyzts.append(xyzt)  # 加入列表
14          xyzts = torch.stack(tensors=xyzts, dim=1)  # [B, L, n, 4]
15          # 重塑位置矩阵形状
16          xyzts = torch.reshape(input=xyzts, shape=(xyzts.shape[0]*xyzts.shape[1], xyzts
                .shape[2], xyzts.shape[3]))
17          features = features.permute(0, 1, 3, 2)     # 调换特征矩阵的第2个和第3个维度
18          B, L, n, C = features.shape   # 提取特征矩阵的维度
19          features = torch.reshape(input=features, shape=(B*L, n, C))    # 重塑特征矩阵
                形状
20          xyzts = self.pos_embedding(xyzts.permute(0, 2, 1)).permute(0, 2, 1)  # 做位置
                编码
21          return xyzts, features, [B, L, n, C]  # 返回位置编码、特征和形状
```

上述代码定义了 P4Transformer 类的 low_feature_forward() 函数，主要用于特征提取。首先通过 tube_embedding() 得到点的坐标和特征（行 4），使用 torch.split() 将点云视频拆成单帧点云（行 7），然后为每帧点云添加时序信息并重新拼接为点云视频（行 10~19），然后调整特征矩阵维度，最后使用 pos_embedding() 生成位置编码（行 20~21），并同特征一起返回。

```
1       def forward(self, input_clips, forward_type='cls'):  # 前向过程，默认只做分类
2           assert forward_type in ['cls', 'mask', 'both']  # forward_type合法性检查
3           xyzts, features, shapes = self.low_feature_forward(input_clips)  # 低层特征提
                取
4           B, L, n, C = shapes
5           embedding = xyzts + features  # 将位置编码和特征相加
6           if self.emb_relu:  # 如果需要ReLU
7               embedding = self.emb_relu(embedding)
8           embedding_xyz = self.transformer_1(embedding)  # 空间维度Transformer
9           # 修改embedding_xyz形状
10          embedding_xyz = torch.reshape(input=embedding_xyz, shape=(B, L, n, C))
11          # 空间维度最大池化
12          embedding_xyz = torch.max(input=embedding_xyz, dim=-2, keepdim=False, out=None
                )[0]
13          device_gpu = embedding_xyz.device
14
15          #----------------------分类----------------------
16          if forward_type == 'cls':
17              embedding_t = self.transformer_2(embedding_xyz)  # 时间维度Transformer
18              # 对时间维度做最大池化
19              output_cls = torch.max(input=embedding_t, dim=1, keepdim=False, out=None)
                    [0]
20              output_cls = self.mlp_head(output_cls)  # 通过MLP得到类别输出
21              return output_cls
22
23          #----------------------掩码重建----------------------
24          elif not self.only_cls and forward_type == 'mask':
25              embedding_xyz_clone = embedding_xyz.clone()  # 保存一个副本
26              # 对序列做mask，返回：mask后的embedding、被采样的索引、被mask的索引
```

```
            mask_embedding_xyz, sample_index, mask_index = MaskEmbeeding(embedding_xyz
                , mask_ratio=self.mask_ratio)
            mask_embedding_t = self.transformer_2(mask_embedding_xyz)  # 经过时序
                Transformer
            norm_embedding_t = self.norm_encoder(mask_embedding_t)  # 将Transformer输
                出做归一化
            mask_proj_embedding_t = self.proj(norm_embedding_t)  # 投影到decoder维度
            assert self.decoder_pos_embedding.requires_grad == False  # decoder位置编
                码不更新
            # 将masked特征+unmask嵌入后恢复完整维度（unmask）
            proj_embedding_t = self.unmask_embed(mask_proj_embedding_t, sample_index,
                mask_index)
            proj_embedding_t = proj_embedding_t + self.decoder_pos_embedding.to(
                proj_embedding_t.device)  # 添加位置编码
            # 输入到decoder进行掩码重建
            proj_embedding_t = self.transformer_decoder_3(proj_embedding_t)
            restruct_embedding_xyz = self.restruction(proj_embedding_t)  # 还原到原维
                度
            # 输出重建特征、原始特征、mask索引
            return restruct_embedding_xyz, embedding_xyz_clone, mask_index

        #----------------------掩码重构+分类----------------------
        elif not self.only_cls and forward_type == 'both':
            embedding_xyz_clone = embedding_xyz.clone()  # 克隆一份embedding_xyz用于后
                续输出
            # 对序列做mask，返回：mask后的embedding、被采样的索引、被mask的索引
            mask_embedding_xyz, sample_index, mask_index = MaskEmbeeding(embedding_xyz
                , mask_ratio=self.mask_ratio)
            mask_embedding_t = self.transformer_2(mask_embedding_xyz)  # 经过时序
                Transformer
            # 对时间维度做最大池化
            output_cls = torch.max(input=mask_embedding_t, dim=1, keepdim=False, out=
                None)[0]
            output_cls = self.mlp_head(output_cls)  # 通过MLP得到类别输出
            norm_embedding_t = self.norm_encoder(mask_embedding_t)     # 将Transformer
                输出做归一化
            mask_proj_embedding_t = self.proj(norm_embedding_t)      # 投影到decoder所
                需的维度
            assert self.decoder_pos_embedding.requires_grad == False     # decoder位置
                编码不更新
            # 将masked特征+unmask嵌入后恢复完整维度（unmask）
            proj_embedding_t = self.unmask_embed(mask_proj_embedding_t, sample_index,
                mask_index)
            proj_embedding_t = proj_embedding_t + self.decoder_pos_embedding.to(
                proj_embedding_t.device)  # 添加位置编码
            # 输入到decoder进行掩码重建
            proj_embedding_t = self.transformer_decoder_3(proj_embedding_t)
            restruct_embedding_xyz = self.restruction(proj_embedding_t)  # 还原到原维
                度
            # 输出重建特征、原始特征、mask索引、类别
            return restruct_embedding_xyz, embedding_xyz_clone, mask_index, output_cls
        else:
            assert False, "forward_type must in ['cls', 'mask', 'both']"  # 错误提示
```

上述代码定义 P4Transformer 类的 forward() 函数，根据“forward_type”的值来决定前向传播的流程。如果“forward_type”为“cls”，进入到分类分支，使用时序 Transformer 处理 embedding_xyz 后经过最大池化和多层感知机输出预测类别（行 17~21）；如果“forward_type”为“mask”，进入到掩码重建分支。首先先保留原始特征（行 25），然后用 MaskEmbedding() 对特征进行掩码（行 27），将可见特征输入到时序 Transformer 得到潜在表示（mask_embedding_t）（行 28），通过投影层映射到解码器维度（行 30）。利用 unmask_embed() 在掩码位置插入可学习的掩码标记（行 33~34），恢复完整时间序列特征（proj_embedding_t），并添加固定位置编码。特征经时间解码器重建后，与原始未掩码特征（embedding_xyz_clone）对比，计算重建损失；如果“forward_type”为“both”，同时执行分类和掩码重建，可见特征用于分类，输出动作类别（output_cls），掩码特征用于重建，输出重建结果（restruct_embedding_xyz）和原始特征（embedding_xyz_clone）。

3.5 本章习题

3.5.1 简答题

1）动作识别中的“时空特征”指什么？为何视频分析需要同时建模空间和时间信息？
2）Two-stream 网络的基本结构是什么？其优势和局限性分别是什么？
3）三维卷积神经网络（3D CNN）如何解决传统二维卷积的不足？其代表模型有哪些？
4）循环神经网络（RNN）在动作识别中发挥什么作用？有哪些典型结构改进？
5）基于 Transformer 的方法相比 CNN/RNN 有何优势？其代表性模型有哪些创新点？
6）点云数据常用的获取方式有哪些？
7）点云数据与视频数据在动作识别中的本质区别是什么？
8）点云动作识别的主要技术挑战有哪些？
9）不同监督范式的点云动作识别任务有什么区别和联系？
10）阐述一下全监督点云动作识别任务中不同网络的优缺点。

3.5.2 动手实践题

1）动作识别技术在智能安防、体育训练评估、动作指令交互等众多场景中都具有广泛应用价值：在监控视频中自动识别人群行为，可及时发现异常事件并触发告警；在竞技体育或健身场景中，系统能够对运动员或用户的标准动作进行打分、反馈与纠正；在智能家居和机器人领域，基于动作技术的人机交互可以实现更加自然、直观的控制。本实践部分将要求读者使用 SlowFast 模型，对公开数据集中预定义的多种动作类别（例如行走、挥手、跳跃等）进行自动分类，以构建一个端到端的视频动作识别系统。

为保证实践的可复现性与效果对比，选用包含数十至数百类动作、总计数万条视频样本的主流公开数据集（如 Kinetics-400 或 HMDB51）。这些数据集涵盖日常生活、体育运动和舞蹈等多种动作场景，每条样本都已标记动作类别标记。模型训练与测试时，将按常见的训练/验证/测试划分进行，并以 top-1 和 top-5 分类准确率衡量整体性能；在关注某些关

键类别（如跌倒、打架等）时，还将通过精确率、召回率与 F1 分数评估模型在特定动作上的识别质量；此外，为了更全面地反映多类别性能，也可计算平均精度均值（mAP）作为补充指标。

请基于 SlowFast 模型实现上述要求。SlowFast 由 Facebook AI 提出，采用双分支时空卷积网络来同时捕捉视频中的静态语义与快速运动信息。其中“Slow”分支以较低帧率抽取场景和物体的语义特征，“Fast”分支以高帧率聚焦捕获细微的动作变化；两者通过侧连（lateral connection）在多个层级进行特征融合，从而实现对长时依赖与短时动态的协同建模。在大规模视频理解数据集（如 Kinetics-400、AVA）上，SlowFast 已取得领先的识别效果。实践基于开源的 PyTorch 实现，加载预训练权重并对指定数据集进行微调，最终得到一个面向多类别动作识别任务的高效模型。

2）在一个大型物流仓库中，工人们需要不断地进行搬运、打包、上架等作业。为了提高工作效率并确保安全，仓库安装了多个 3D 激光雷达（LiDAR）传感器，以捕捉实时的三维点云数据，反映工人的位置、姿势及动作轨迹。现在已经使用 P4Transformer 技术设计了一个识别系统，可以自动识别搬运动作（弯腰、抬举）以监测工人的姿势是否符合安全规范，及时提醒错误姿势，减少腰背受伤风险；还可以识别走动、转身等动作，追踪工人的路线，优化仓库作业流程；同时实时监测危险动作（滑倒、跌倒）并立即报警，防止事故扩大。当工人需要弯腰搬运货物时，系统通过点云序列中的姿态变化自动检测腰部弯曲角度和抬举姿势是否安全，如检测到危险动作，立刻发出警报并通知现场管理人员。但是最近需要延长检测时间，旧的系统出现了一些问题。经过相关专家分析，这有可能是因为点云视频在时间维度上是有序的，在空间上是无序的，所以当检测时间较长时，对应的点云视频长度增大，P4Transformer 在特征提取时容易出现时空紊乱。借鉴相关领域方法，专家认为将时空解耦可以缓解这一现象，于是设计了一个新的网络结构。

数据集：MSRAction3D。

评价指标：Top-1 视频级分类准确率。

任务：参考 PSTNet 中的时空解耦方式修改 3.4.3 中的局部区域特征提取模块的代码，使其实现时间和空间上的解耦，并在 MSRAction3D 数据集上验证方法是否有效。

参考文献

[1] BEN-SHABAT Y, SHROUT O, GOULD S. 3DInAction: understanding human actions in 3d point clouds[C]// Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2024: 19978-19987.

[2] QI C R, SU H, MO K, et al. PointNet: deep learning on point sets for 3d classification and segmentation[C]// Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2017: 652-660.

[3] SIMONYAN K, ZISSERMAN A. Two-stream convolutional networks for action recognition in videos[C]// Proceedings of the 28th International Conference on Neural Information Processing Systems (NeurIPS). 2014: 568-576.

[4] FEICHTENHOFER C, PINZ A, ZISSERMAN A. Convolutional two-stream network fusion for video action recognition[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016: 1933-1941.

[5] WANG L, XIONG Y, WANG Z, et al. Temporal segment networks: towards good practices for deep action recognition[C]//Proceedings of the European Conference on Computer Vision (ECCV). 2016: 20-36.

[6] WANG Y, LONG M, WANG J, et al. Spatio-temporal pyramid network for video action recognition[C]// Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2017: 1529-1538.

[7] WANG L, TONG Z, JI B, et al. TDN: temporal difference networks for efficient action recognition[C]// Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2021: 1895-1904.

[8] TRAN D, BOURDEV L, FERGUS R, et al. Learning spatiotemporal features with 3D convolutional networks [C]//Proceedings of the IEEE International Conference on Computer Vision (ICCV). 2015: 4489-4497.

[9] CARREIRA J, ZISSERMAN A. Quo vadis, action recognition? a new model and the kinetics dataset[C]// Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2017: 6299-6308.

[10] HARA K, KATAOKA H, SATOH Y. Learning spatio-temporal features with 3D residual networks for action recognition[C]//Proceedings of the IEEE International Conference on Computer Vision (ICCV). 2017: 3154-3160.

[11] FEICHTENHOFER C, FAN H, MALIK J, et al. SlowFast networks for video recognition[C]//Proceedings of the IEEE International Conference on Computer Vision (ICCV). 2019: 6202-6211.

[12] NG J Y H, HAUSKNECHT M, VIJAYANARASIMHAN S, et al. Beyond short snippets: deep networks for video classification[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2015: 4694-4702.

[13] DONAHUE J, HENDRICKS L A, ROHRBACH M, et al. Long-term recurrent convolutional networks for visual recognition and description[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 2017, 39: 677-691.

[14] HE J Y, WU X, CHENG Z Q, et al. DB-LSTM: densely-connected bi-directional LSTM for human action recognition[J]. Neurocomputing (IJON), 2021, 444: 319-331.

[15] SIAM M, VALIPOUR S, JAGERSAND M, et al. Temporal reasoning in videos using convolutional gated recurrent units[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2018: 1224-1229.

[16] SHI Y, TIAN Y, WANG Y, et al. Learning long-term dependencies for action recognition with a biologically-inspired deep network[C]//Proceedings of the IEEE International Conference on Computer Vision (ICCV). 2017: 716-725.

[17] MENG L, ZHAO B, CHANG B, et al. Interpretable spatio-temporal attention for video action recognition [C]//Proceedings of the IEEE International Conference on Computer Vision (ICCV). 2019: 1513-1522.

[18] BERTASIUS G, WANG H, TORRESANI L. Is space-time attention all you need for video understanding? [C]//Proceedings of the International Conference on Machine Learning (ICML). 2021: 813-824.

[19] CHEN C F, PANDA R, FAN Q. RegionViT: regional-to-local attention for vision transformers[C]// Proceedings of the International Conference on Learning Representations (ICLR). 2022.

[20] TONG Z, SONG Y, WANG J, et al. Videomae: masked autoencoders are data-efficient learners for self-supervised video pre-training[J]. Advances in Neural Information Processing Systems (NeurIPS), 2022, 35: 10078-10093.

[21] WANG L, HUANG B, ZHAO Z, et al. VideoMAE V2: scaling video masked autoencoders with dual masking[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2023: 14549-14560.

[22] FAN H, YANG Y, KANKANHALLI M. Point 4d transformer networks for spatio-temporal modeling in point cloud videos[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2021: 14204-14213.

[23] FAN H, YU X, DING Y, et al. PSTNet: point spatio-temporal convolution on point cloud sequences[C]// International Conference on Learning Representations (ICLR). 2021.

[24] FAN H, YANG Y, KANKANHALLI M. Point spatio-temporal transformer networks for point cloud video modeling[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 2023, 45(2): 2181-2192.

[25] QI C R, YI L, SU H, et al. PointNet++: deep hierarchical feature learning on point sets in a metric space[C]// Advances in Neural Information Processing Systems (NeurIPS). 2017: 5099-5108.

[26] LIU X, YAN M, BOHG J. MeteorNet: deep learning on dynamic 3d point cloud sequences[C]//Proceedings of the IEEE International Conference on Computer Vision (ICCV). 2019: 9246-9255.

[27] WANG Y, XIAO Y, XIONG F, et al. 3DV: 3D dynamic voxel for action recognition in depth video[C]// Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2020: 511-520.

[28] REDMON J, FARHADI A. YOLOv3: an incremental improvement[A]. 2018. arXiv: 1804.02767.

[29] CHEN X, LIU W, LIU X, et al. MAPLE: masked pseudo-labeling autoencoder for semi-supervised point cloud action recognition[C]//Proceedings of the ACM International Conference on Multimedia (ACMMM). 2022: 708-718.

[30] WANG H, YANG L, RONG X, et al. Self-supervised 4d spatio-temporal feature learning via order prediction of sequential point cloud clips[C]//Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV). 2021: 3762-3771.

[31] SHEN Z, SHENG X, FAN H, et al. Masked spatio-temporal structure prediction for self-supervised learning on point cloud videos[C]//Proceedings of the IEEE International Conference on Computer Vision (ICCV). 2023: 16580-16589.

[32] ZHANG Z, DONG Y, LIU Y, et al. Complete-to-partial 4D distillation for self-supervised point cloud sequence representation learning[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2023: 17661-17670.

[33] XU Y, YE K, HAN X, et al. A unified framework for human-centric point cloud video understanding[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2024: 1155-1164.

[34] PENG Y, ZHAO Y, ZHANG J. Two-stream collaborative learning with spatial-temporal attention for video classification[C]//IEEE Transactions on Circuits and Systems for Video Technology (TCSVT). 2019: 773-786.