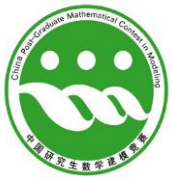


中国研究生创新实践系列大赛

“华为杯”第二十届中国研究生数学建模竞赛

学　　校	南华大学
参赛队号	23105550004
队员姓名	1.李洁
	2.周婷
	3.刘洋

中国研究生创新实践系列大赛
“华为杯”第二十届中国研究生
数学建模竞赛

题　目： 基于多模型融合的 UNet 网络预报强对流降水

摘　要：

我国地域广阔、自然条件复杂，导致灾害性天气种类多样，严重的对流性降水是许多灾害的主要原因。然而对流性降水的即时预测非常关键但又极具挑战性，因此利用双偏振雷达提供的微物理特征信息进行建模，对强对流预报有重要意义。具体做法如下：

针对问题一：（利用雷达观测量对强对流做临近预报）首先根据题目要求和数据量的大小决定采用数据驱动的神经网络模型，并选择了 3km 等高面的数据进行预报。对数据进行质量控制、过滤非雨天数据、双线性插值等操作，得到 10557 个样本并按照 9:1 划分训练测试数据；为了选择最优模型并评估模型性能，本文使用了相对误差、均方根误差、比率偏差和皮尔逊相关系数 4 个评估指标，对 ConvLSTM 时序模型和 FURNet 模型进行了比较和评估，发现 FURNet 模型较好，4 个评价指标分别为 0.4133，0.0553，0.8336，0.8512。并且通过可视化展示后 10 帧 Z_H 信息，验证了基于对称编码解码的 FURNet 模型在利用双偏振雷达数据进行临近预报中的有效性。

针对问题二：（优化模型解决“回归到平均”问题）在问题一建立的 FURNet 模型基础之上，结合多尺度特征融合的优点，引入 CBAM 注意力机制到上采样和下采样模块中，建立 FURNet+CBAM 模型来提高模型对细节的关注程度。并且融合 ConvLSTM 和 UNet 对时序数据预测的优点，引入了“高”这一维度在数据中的设计，实现多通道的 3D-UNet 模型。并将 FURNet+CBAM 模型、3D-UNet 模型和 FURNet 模型进行对比检验，发现 FURNet+CBAM 模型在 4 个评价指标的结果都有所提高，并在测试集的损失下降到 0.0021 也优于其他模型。揭示了通过引入注意力机制和时序特征处理，模型可以更加准确地预测强对流降水。

针对问题三：（利用 Z_H、Z_{DR} 预测降水量）建立了两个重要的数学模型：CSU-HIDRO 雷达优化模型和基于遗传算法的符号回归模型。研究选取了一个地区的数据进行处理，首先，利用单个 Z_H、组合 Z_H 和 Z_{DR} 使用 CSU-HIDRO 模型进行参数拟合。结果显示，$R(Z_H, Z_{DR})$ 预测优于 $R(Z_H)$。然而，由于 CSU-HIDRO 模型受多种因素影响，为提高预测结果的准确性，还引入了机器学习中的符号回归算法，同样得到类似的两个模型，但是拟合速度更快。此外，还对两种方法的预测结果进行比较，发现组合预测的误差较小，更接近实际情况；而符号回归预测的误差更接近于 0，表明符号回归模型的结果优于 CSU-HIDRO 模型。

针对问题四：（运用数据融合对强对流预报的突发性和局地性预测）使用了多源融合、加权平均或集成学习等多种数据融合方式，并充分考虑了强对流降水突发性和局地性的特

性。对 1km、3km 和 7km 的等高线数据构建图结构数据，使用 GNN（图神经网络）通过 3 层的消息传递，实现不同位置之间的信息的交互，更新图的信息，以此实现了多源数据融合。然后使用新的 3km 处的信息作为第一层上游任务，然后运用改进的（CBAM）UNET 系列网络进行临近预报。发现前两问建立模型在此融合数据后的性能有所提升，并且 GNN+Unet 处理融合后的数据更有优势，与真实值得接近度达到 0.881，说明此数据融合策略可以更好地应对突发性和局地性强的强对流天气。

关键词：UNET 神经网络；ConvLSTM；CBAM 注意力机制；CSU-HIDRO 模型；符号回归

目录

一、问题重述

1.1 问题背景

由于我国地域广阔和自然条件复杂多样，强对流天气现象如雷暴、大风、冰雹和龙卷风经常导致严重的经济和人员损失。这些天气现象往往具有突然性、局限性强和短暂的生命周期，这使得短时和临近预报成为一个巨大的挑战。

传统上，临近预报主要依靠雷达观测和经验性的 Z-R 关系来估算降水量。然而，这些方法通常忽略了许多影响强对流天气的复杂因素，并且可能不适用于所有地区或气象条件。随着大数据和计算能力的快速发展，深度学习已经在多个领域表现出强大的性能，特别是基于卷积神经网络（CNN）的模型如 U-Net，以及基于循环神经网络（RNN）的模型如 DGMR，为短期临近预报提供了新的可能性[1]。这种方法的一个重要优势是其对海量数据的处理能力，这在分析复杂和多变的强对流天气现象时尤为重要。

然而雨滴在下降的过程会受到空气阻力的影响，造成其对水平偏振（电场振动方向在水平面内）的电磁波和垂直偏振（电场振动方向在垂直平面内）的电磁波的反射特征是不一样的（如图 1.1 所示）。而传统的单偏振雷达只能在一个偏振方向（通常是水平或垂直）上发送和接收电磁波，因此其提供的信息相对有限[2]。不同的是，双偏振雷达（如图 1.2 所示）能在水平和垂直两个偏振方向上同时发送和接收电磁波，从而能够获取更多关于降水粒子（如雨滴或冰粒）的详细信息，包括其大小、形状、相态（液态或固态）以及含水量。故选择双偏振雷达接收信号不仅能提供更准确、更全面的天气信息，而且能极大地提升强对流天气事件的预测质量和准确性，从而更有效地预防和减少由此引发的经济和人员损失。

本题要求在应用双偏振雷达的基础上对强对流降水短临预报进行分析，并给出以.npy 为后缀的双偏振雷达数据和降水格点数据。数据集一共有 258 个降水过程，每个过程包含连续多帧数据，相邻两帧之间的时间间隔为 6 分钟，每一帧数据是 256×256，对应于 256km×256km 的平面区域。

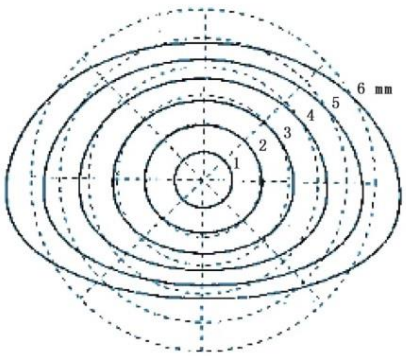


图 1.1 空气中雨滴的形态

图 1.2 双偏振雷达

1.2 问题分析

针对强对流预报中利用双偏振变量改进预报的问题，建立数学模型是一种常用的方法。下面是本文的问题分析和建模思路：

问题一：要求利用双偏振变量改进预报，主要用到的数据为 dBZ、ZDR、KDP 为主的雷达数据，需要从中提取 ZH、ZDR 和 KDP 中的微物理特征信息，然后建立数学模型。本题将对数据进行预处理，其选择卷积神经网络（**FURNet**）模型和循环神经网络（ConvLSTM）模型进行建模。

问题二：为了减少现有数据驱动算法中的平均值预测的模糊效应，要求在问题一的模型基础上进行改进。并引入模型融合的方法。本题将问题一中的两种模型进行融合处理，减少模糊效应，并提高强对流降水预测的准确性和可靠性。

问题三：旨在利用 ZH 和 ZDR 数据设计数学模型，以实现对降水量的定量估计。本文将采用非线性拟合方法和符号回归方法，通过学习输入和输出之间的映射关系，得到 ZH、ZDR 和降水量之间的关系，为降水量的定量估计提供有价值的指导。

问题四：旨在评估双偏振雷达资料在强对流降水临近预报中的贡献，并要求设计一个优化的融合策略。本文将使用三个微物理参数 ZH、ZDR 和 KDP 在不同等高面的数据融合，通过对每个参数比较分析贡献程度，以便更好地应对突发性和局地性强的强对流天气。

二、模型假设

在数学建模的过程中，在不影响模型意义与预测结果的前提下，为了使模型简单明确，本文建立了以下假设：

（1）假设输入数据中的相邻区域具有相关性，即相邻的数据点之间存在一定的空间相关性。

（2）假设在不同的位置上使用相同的权重能够有效地捕捉到数据的局部特征。

（3）假设输入数据中的特征在空间上具有平移不变性，即特征的位置变化不会影响其表示和识别。

（4）假设所有降水过程中为温度恒定，即为常温状态。

（5）假设冰晶和雹子这些非球形粒子对降水量的影响较小，可以忽略不计。

三、符号说明

符号	符号说明
Z_H /dBZ	水平反射率
Z_{DR}	差分反射率
K_{DP}	比差分相移
R	降水量
Z	雷达反射率
C_t	t 时刻的单元输出
H_t	t 时刻的隐藏状态

H_t	输入
RE	相对误差
RMSE	均方根误差
BIAS	比率偏差
CC	皮尔逊相关系数

注：其它符号在正文中详细标注。

四、问题一的分析与求解

4.1 问题分析

问题一要求利用双偏振变量改进预报，主要用到的数据为 dBZ、ZDR、KDP 为主的雷达数据，需要从中提取 Z_H、Z_{DR} 和 K_{DP} 中的微物理特征信息（这里以 Z_H 为例，画出它在一次降雨过程中的变化（如图 4.1 所示），然后建立数学模型。具体地，使用前一小时（10 帧）的雷达观测量（Z_H 、ZDR、KDP）为输入，得到后续一小时（10 帧）的 Z_H 预报。然而地面杂波的抑制会使水平信道和垂直信道的信号不相干/不平衡，回波受到地杂波污染，故分别定义为水平反射率因子，差分反射率以及比差分相移 Z_H，Z_{DR} 和 K_{DP} 的测量精度可能会降低。因此，需要先对原始数据（以 3km 登高面为例）进行预处理，接着建立深度学习模型分析前面一小时（10 帧）的雷达观测量与后续一小时（10 帧）的 Z_H 的因果关系。

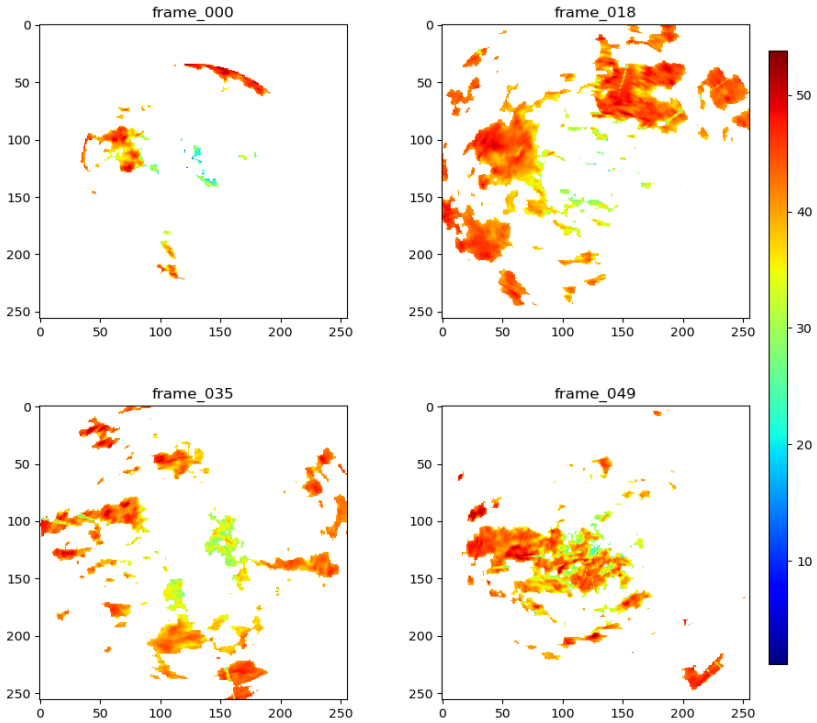


图 4.1 双偏振 dBZ 变量雷达数据

问题一的总体思路如图 4.2 所示，首先对数据进行质量控制、过滤非雨天样本、双线性插值和归一化等处理过程。接着利用微物理特征信息建立以 UNet 为基础的融合重分配

网络——FURNet 网络和以卷积神经网络（CNN）和长短期记忆网络（LSTM）为基础的 ConvLSTM[3][4]，然后利用训练数据对网络模型的参数进行梯度调优，并将两个试验数据的结果进行对比检验，最后以将模型应用于实际的强对流预报中。

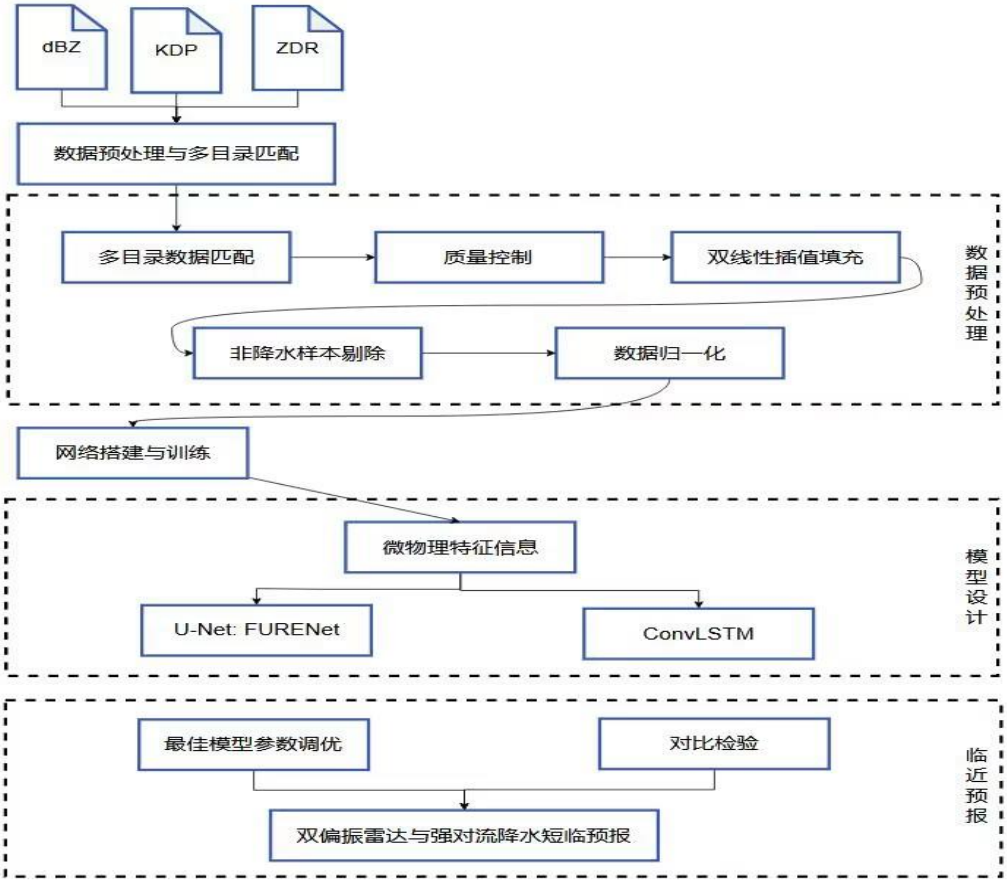


图 4.2 问题一的总体分析流程图

4.2 数据预处理

本研究使用南京大学运行的双偏振天气雷达的数据。数据集涵盖了 258 个降水事件数据。我们使用 3 公里高度的 dBZ、KDP、ZDR 雷达数据，包括了雷达中心周围 256×256 公里的区域。数据集的时间分辨率为 6 分钟。预处理部分主要包括质量控制、过滤非雨天样本、双线性插值和归一化等步骤（代码见附录 1）。

4.2.1 质量控制

质量控制是为了去除雷达数据中的噪声和非气象信号。常见的质量控制方法包括去除雷达反射率小于一定阈值的像素，去除强度不连续的区域（例如经过强降水的降雨区域），以及根据双偏振参数的一致性进行筛选。具体筛选公式如下：

$$\begin{aligned} & A < a_1, \\ & |\Delta A| > a_2, \\ & \left|A - \overline{A}\right| < 2\sqrt{Var(A)} \end{aligned} \quad ,$$

其中 A 是雷达数据中的参数，ΔA 是连续时间的差值，$\overline{\mathrm{A}}$ 和 $Var(A)$ 是均值和方差，a_1 和 a_2 是阈值。

4.2.2 过滤非雨天样本

具体来说，设置了一个强度阈值 $I_{threshold}$ 和一个面积阈值 $A_{threshold}$。在每个时间步长中，对雷达反射率超过阈值的区域进行计数并记录为 a 评价，如果评价序列(样本的最后 10 帧)中 a 评价的最大值小于 a 阈值，则将该样本识别为“非雨样本”，并在训练和测试数据集中进行过滤。在实际应用中，$I_{threshold}$ 和 $A_{threshold}$ 分别设置为 35 dBZ 和 256 km^2，合理地反映了 256km×256km 域的对流降水事件。

4.2.3 双线性插值

前面的质量控制和过滤会产生空值，需要相应坐标系下将数据进行插值。常见的雷达数据插值方法包括双线性插值、径向基函数插值等。这里可以选择双线性插值的插值方法，并将数据插值到所需的坐标格点上。具体做法如下：

$$f(x,y)=\frac{1}{(x_2-x_1)(y_2-y_1)}[f(x_1,y_1)\cdot(x_2-x)(y_2-y)+f(x_2,y_1)\cdot(x-x_1)(y_2-y)+f(x_1,y_2)\cdot(x_2-x)(y-y_1)+f(x_2,y_2)\cdot(x-x_1)(y-y_1)],$$

其中 $f(x_1, y_1), f(x_2, y_1), f(x_1, y_2), f(x_2, y_2)$ 是二维网格上有四个已知数据点。

4.2.4 特征归一化

数据归一化是一种常用于数据预处理的技术，目的是将不同范围的数据缩放到相同的范围内，以便机器学习算法能够更好地处理。常见的数据归一化方法包括：最小-最大归一化，Z-score 标准化，小数定标标准化等。根据数据分布和机器学习模型的需求选择，选择最小-最大归一化，将数据线性地缩放到一个指定的范围，通常是[0, 1]。公式为：

$$\tilde{x}=\frac{x-\min}{\max-\min},$$

其中 x 是原始数据，$\tilde{x}$ 是归一化后的数据，min 和 max 分别是特征的最小值和最大值。

4.3 模型建立

4.3.1 模型通道构建

现在预测问题被定义为使用当前时间之前的 1 小时雷达反射率（Z_H）和变量（K_{DP} 和 Z_{DR}），来预测接下来 1 小时的反射率（Z_H）。不同时间步长的观测结果叠加在一起形成通道维度：

$$\hat{Z}_{H;t+k},\hat{Z}_{H;t+k-1},\cdots,\hat{Z}_{H;t+1}=F\begin{pmatrix}Z_{H;t},Z_{H;t-1},\cdots,Z_{H;t-j+1};\\ Z_{DR;t},Z_{DR;t-1},\cdots,Z_{DR;t-j+1};\\ K_{DP;t},K_{DP;t-1},\cdots,K_{DP;t-j+1}\end{pmatrix},$$

其中 F 为临近预报模型，j 和 k 分别为输入和输出的时间步长，均为 10，雷达观测的时间间隔为 6。因此，单个样本共包含 20 个时间步长的雷达数据。在数据预处理后，数据集中剩下 10557 个样本，训练集与测试集的划分比例为 9：1，训练集数据大小为 9,501，测试集数据大小为 1056。

4.3.2 基于时空特性的 ConvLSTM 网络模型

通过 ConvLSTM 设立了对照实验（代码见附录 3）。ConvLSTM 是一种用于时空预测的递归神经网络，在输入到状态和状态到状态的转换中都具有卷积结构。一个显著特征是将所有输入 $X_1,\cdots,X_t$，单元输出 $C_1,\cdots,C_t$，隐藏状态 $H_1,\cdots,H_t$ 关起来。ConvLSTM 通过网格中某个单元的输入和其本地邻居的过去状态来确定该单元的未来状态[5]。这可以通过在状态到状态和输入到状态转换中使用卷积算子轻松实现（见图 4.3）。ConvLSTM 的关键方程如下所示：

$$
\begin{aligned}
i_t &= \sigma(W_{xi} * X_t + W_{hi} * H_{t-1} + W_{ci} \circ C_{t-1} + b_i) \\
f_t &= \sigma(W_{xf} * X_t + W_{hf} * H_{t-1} + W_{cf} \circ C_{t-1} + b_i) \\
C_t &= f_t \circ C_{t-1} + i_t \circ \tanh(W_{xc} * X_t + W_{hc} * H_{t-1} + b_c)\,, \\
o_t &= \sigma(W_{xo} * X_t + W_{ho} * H_{t-1} + W_{co} \circ C_t + b_o) \\
H_t &= o_t \circ \tanh(C_t)
\end{aligned}
$$

其中 $*$ 表示卷积算子，$\circ$ 为 Hadamard 积，输入门 i_t，遗忘门 f_t，输出门 o_t。

如果将状态视为运动物体的隐藏表示，那么具有较大过渡核的 ConvLSTM 应该能够捕获更快的运动，而具有较小核的 ConvLSTM 可以捕获较慢的运动为了确保状态具有与输入相同的行数和列数，在应用卷积操作之前需要填充。在这里，边界点上隐藏状态的填充可以看作是使用外部世界的状态进行计算。通常，在第一个输入到来之前，将 LSTM 的所有状态初始化为实际上是将外部世界的状态设置为零，并且假设没有关于外部世界的先验知识。通过填充状态，可以对边界点进行不同的处理，这在很多情况下都很有用。对于时空序列预测问题，使用如图 4.3 所示的结构，它由两个网络组成，一个编码网络和一个预测网络。预测网络的初始状态和单元输出是从编码网络的最后状态复制的。这两种网络都是通过堆叠多个 ConvLSTM 层形成的。由于预测目标与输入具有相同的维数，将预测网络中的所有状态连接起来，并将它们馈送到 1×1 的卷积层中以生成最终的预测。编码 LSTM 将整个输入序列压缩成一个隐藏状态张量，预测 LSTM 将这个隐藏张量展开给出最终预测的状态:

$$
\begin{aligned}
\hat{X}_{t+1},\cdots,\hat{X}_{t+K} &= \underset{X_{t+1},\cdots X_{t+K}}{\arg\max}\ p(X_{t+1},\cdots X_{t+K} \mid \hat{X}_{t-J+1},\hat{X}_{t-J+2}\cdots,\hat{X}_t) \\
&= \underset{X_{t+1},\cdots X_{t+K}}{\arg\max}\ p(X_{t+1},\cdots X_{t+K} \mid f_{encoding}(\hat{X}_{t-J+1},\hat{X}_{t-J+2}\cdots,\hat{X}_t)) \\
&= g_{forecasting}(f_{encoding}(\hat{X}_{t-J+1},\hat{X}_{t-J+2}\cdots,\hat{X}_t))
\end{aligned}
$$

。

该网络有多个堆叠的 ConvLSTM 层，它具有很强的表征能力，这使得它适合在复杂的动力系统中给出预测。

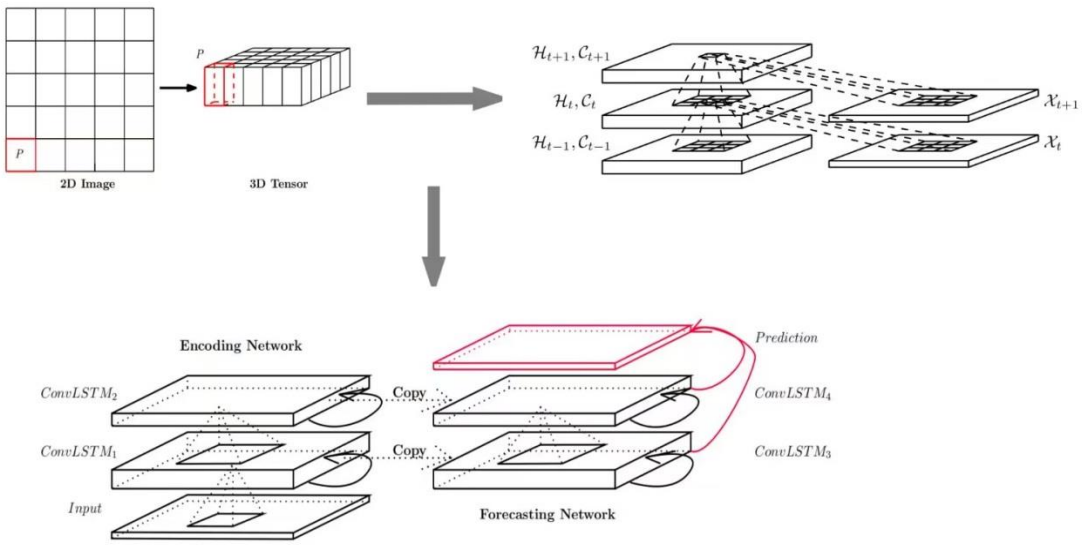


图 4.3 ConvLSTM 网络简图

4.3.3 基于对称编码解码的 FURNet 网络模型

本次研究使用 Xiang Pan 等人提出的 FURNet 网络结构（代码见附录 2），是基于 UNet 实现的融合与重分配网络。UNet 是一种常用于图像分割任务的深度学习网络模型，由 Olaf Ronneberger 等人于 2015 年提出。它的结构灵感来自 U 形结构，因此得名 UNet。UNet 的设计目标是在保持上下文信息的同时，能够精确地定位目标。

其整体结构由对称的编码器（下采样路径）和解码器（上采样路径）组成[2]（如图 4.4 右图）。编码器用于从输入图像中提取特征，通过连续的卷积和池化操作逐渐减小特征图的尺寸和通道数。解码器则通过反卷积和跳跃连接的方式逐渐恢复特征图的尺寸和通道数（如图 4.4 黑色框中所示）。跳跃连接是指将编码器中的某一层的特征图与解码器中对应的层的特征图进行连接，以保留更多的上下文信息。

编码器部分用于逐步提取昏像的高级特征。其计算公式如下:

$$
\begin{aligned}
c_1 &= Conv(x, w_1) \\
p_1 &= MaxPooling(c_1) \\
c_2 &= Conv(p_1, w_2) \\
p_2 &= MaxPooling(c_2), \\
&\vdots \\
c_2 &= Conv(p_{n-1}, w_2) \\
p_2 &= MaxPooling(c_2)
\end{aligned}
$$

其中，x 表示输入阂陈，w_i 表示第 i 个卷积层的权重，Conv 表示卷积操作，MaxPooling 表示最大池化操作。

解码器部分用于将编码器提取的特征进行逐步上采样和融合，生成分割结果。其计算公式如下:

$$
\begin{aligned}
u_{n-1} &= UpConv(c_n, w_{n-1}) \\
m_{n-1} &= Concat(u_{n-1}, c_{n-1}) \\
u_{n-2} &= UpConv(m_{n-1}, w_{n-2}) \\
m_{n-2} &= Concat(u_{n-2}, c_{n-2}) \\
&\vdots \\
u_1 &= UpConv(m_2, w_1) \\
m_1 &= Concat(u_1, c_1) \\
output &= Conv(m_1, w_{output})
\end{aligned}
,
$$

其中，u_i 表示第 i 个上采样层的输出，m_i 表示上采样层输出与对应编码器层的特征融合结果，UpConv 表示上采样卷积操作,Concat 表示特征融合操作,output 表示分割结果。

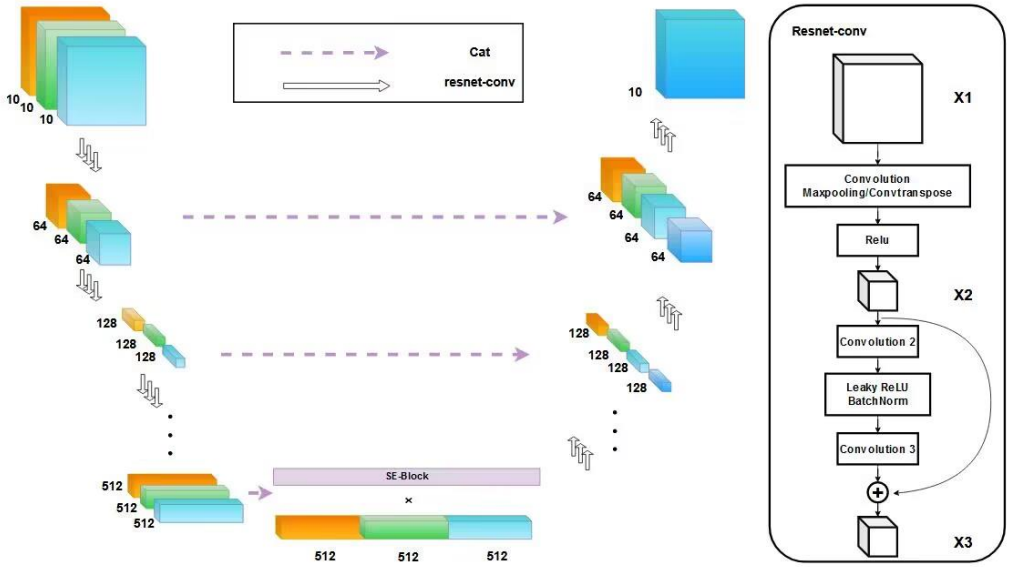


图 4.4　FURNet 网络结构简图

4.4 模型求解与检验

4.4.1 减少内存占用

在实际应用的过程中，发现需要处理大内存的数据，采取一些特殊措施来减少内存使用。以下缓解内存问题的技术：

（1）分批次处理：因为数据集很大，本文将数据分成多个较小的批次进行处理。通过使用 PyTorch 的 DataLoader 类和指定适当的 batch_size 参数来实现。

（2）数据类型转换：使用较低的精度（如 float16）而不是默认的 float32 可以减少内存使用。

（3）内存复用：在训练期间，将所有权重张量和梯度张量共享相同的内存块，从而减少内存使用。这可以通过在优化器上调用 optimizer.zero_grad( )和 model.to( ) 来实现。

（4）模型压缩：对深度学习模型进行压缩可以减少内存使用。例如，可以使用网络剪枝或量化技术等方法来减少模型中的参数数量。

4.4.2 设计损失函数与评估标准

在设计损失函数时，考虑到模型的泛化能力，与数据集的数值分布不平衡问题, MSE 和 MAE 分别衡量了预测值与真实值之间的平方差和绝对差，它们对异常值的敏感程度不同。将 MSE 和 MAE 结合起来，可以综合考虑两者的优点，更全面地评估模型的性能。在本次研究使用的微物理特征信息样本中也可能存在以上问题，所以通过设计的融合 MSE 与 MAE 的损失函数可以提高模型的性能。计算表达式如下：

$$L = w_1\left(\frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2\right)+w_1\left(\frac{1}{n}\sum_{i=1}^{n}|\,y_i-\hat{y}_i\,|\right)。$$

本文采用相对误差（RE）、均方根误差（RMSE）、比率偏差（BIAS）和皮尔逊相关系数（CC）（汪舵等，2017）作为评估指标，对不同估测算法进行评估。计算表达式如下：

（1）相对误差（Relative Error，RE）：用来衡量估测值与实际观测值之间的差异程度。计算公式为：

$$R_E=\left(\frac{\frac{1}{n}\sum_{i=1}^{n}\left|X_{g,i}-X_{r,i}\right|}{\overline{X_g}}\right)\times 100\%。$$

相对误差越接近于 0，表示估测值与实际观测值越接近。其中 X_g 代表估计值，X_r 代表真实值，n 代表样本数据个数。

（2）均方根误差（Root Mean Square Error，RMSE）：用来衡量估测值与实际观测值之间的整体差异程度。计算公式为：

$$R_{MSE}=\sqrt{\frac{\sum_{i=1}^{n}\left(X_{g,i}-X_{r,i}\right)^2}{n}}。$$

均方根误差越小，表示估测值与实际观测值之间的差异越小。

（3）比率偏差（Bias）：用来衡量估测值与实际观测值之间的平均差异程度。计算公式为：

$$B_{IAS}=\frac{\frac{1}{n}\sum_{i=1}^{n}X_{r,i}}{\frac{1}{n}\sum_{i=1}^{n}X_{g,i}}。$$

比率偏差越接近于 1，表示估测值的平均值与实际观测值的平均值越接近。

（4）皮尔逊相关系数（Pearson Correlation Coefficient，CC）：用来衡量估测值与实际观测值之间的线性相关程度。计算公式为：

$$C_C=\frac{\sum_{i=1}^{n}\left(X_{g,i}-\overline{X_g}\right)\left(X_{r,i}-\overline{X_r}\right)}{\sqrt{\sum_{i=1}^{n}\left(X_{g,i}-\overline{X_g}\right)\sum_{i=1}^{n}\left(X_{r,i}-\overline{X_r}\right)}}。$$

皮尔逊相关系数的取值范围为-1 到 1，越接近于 1 表示估测值与实际观测值之间的线性相关性越强。

4.4.3 模型参数调优

参数调优是机器学习和深度学习模型开发中非常重要的一环。下面是本文基本的参数调优过程：

（1）确定参数空间：首先，需要确定待调优的参数及其可能的取值范围。这些参数包括学习率、批次大小、正则化系数、网络结构的层数和宽度等。学习率的范围为（0.001,0.0001），批次大小的范围为（1,4），网络层数的范围为（6,7,8）。

（2）设定评估指标：根据任务需求，选择合适的评估指标来衡量模型性能。对于回归问题可以使用均方误差（RMSE）等指标。

（3）初始化参数：根据参数空间的定义，在合理的范围内对模型的参数进行初始化。

（4）参数搜索与评估：根据定义的参数空间，尝试不同的参数组合，并在验证集上评估模型性能，并网格搜索（GridSearchCV）来搜索参数空间。

（5）反复迭代：参数调优是一个迭代的过程，可能需要多次尝试不同的参数组合，反复进行模型的训练、验证和评估，直至获得满意的结果为止。

在参数调优过程中，采用学习率衰减和早停法，在每个训练步骤或每个训练轮数结束时，将学习率乘以一个衰减因子，本文设置衰减因子为 0.1。当损失结果连续 5 步内没有下降时，启用早停法，以防止过拟合，来进一步提升模型性能和减少训练时间。

4.4.4 求解结果与评价

图 4.5 显示的是 ConvLSTM 网络的损失结果图（代码见附录 4），模型在训练集的损失结果为红色曲线，整体上损失函数值随着训练步骤的进行而逐渐减小，一开始损失值较大，但是从第 3 个 epoch 循环开始大幅度下降，在 110 epoch 时达到最低（红点标记，最低为 0.00765）。由于在第 150 个 epoch 后损失值开始大幅度震荡，因此，本文选取的训练总步数为 110。测试集的结果损失结果为蓝色曲线，局部出现一些震荡或波动，但整体上呈下降趋势，最终损失为 0.00756（蓝点标记）。

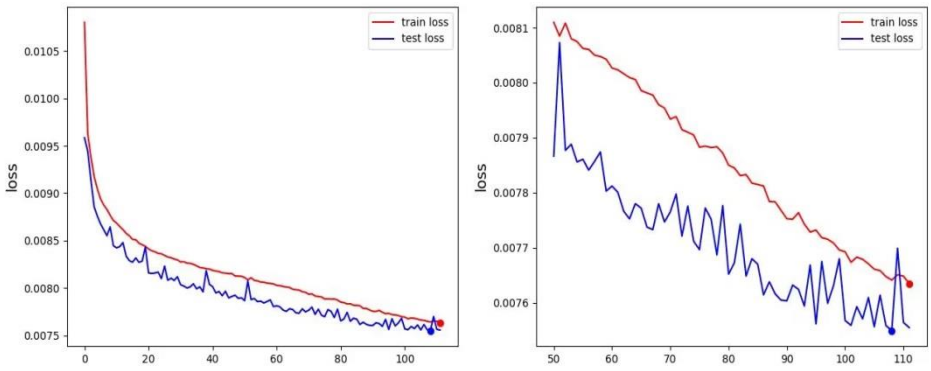


图 4.5 ConvLSTM 网络的损失结果图（右为局部放大图）

图 4.6 显示的是 FURNet 网络的损失结果图，模型在训练集的损失结果为红色曲线，整体上损失函数值随着训练步骤的进行而逐渐减小，一开始损失值较大，但是从第 1 个 epoch 循环开始大幅度下降，在 80 epoch 时大致收敛，在 102 epoch 时达到最低（红点标记，最低为 0.0024）。测试集的结果损失结果为蓝色曲线，局部出现一些震荡或波动，但整体

上呈下降趋势，损失值最后下降到 0.0028。

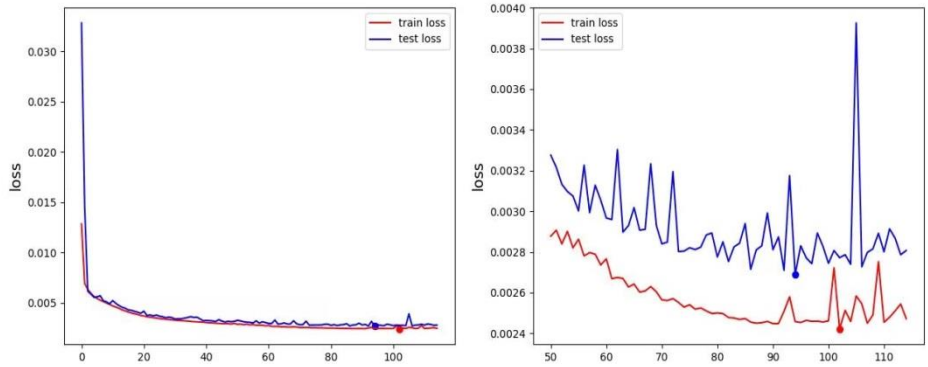


图 4.6 FURNet 网络的损失结果图（右为局部放大图）

将 ConvLSTM 网络和 FURNet 网络的损失结果进行对比（如图 4.7），右侧局部放大图显示了更详细的结果。在训练集上，FURNet 模型的损失值用红色表示，ConvLSTM 网络的损失值用浅蓝色曲线表示。两者整体上随着训练步骤的进行而逐渐减小。初始时 ConvLSTM 的损失结果几乎是 FURNet 的两倍，从第 3 个 epoch 之后，FURNet 损失结果几乎比 ConvLSTM 小一个数量级。在测试集上，损失结果 FURNet 模型的损失值用红色表示，ConvLSTM 网络的损失值用浅蓝色曲线表示，总体上呈下降趋势，但从图中可以看出，FURNet 模型损失值约为 ConvLSTM 的损失的 0.3。

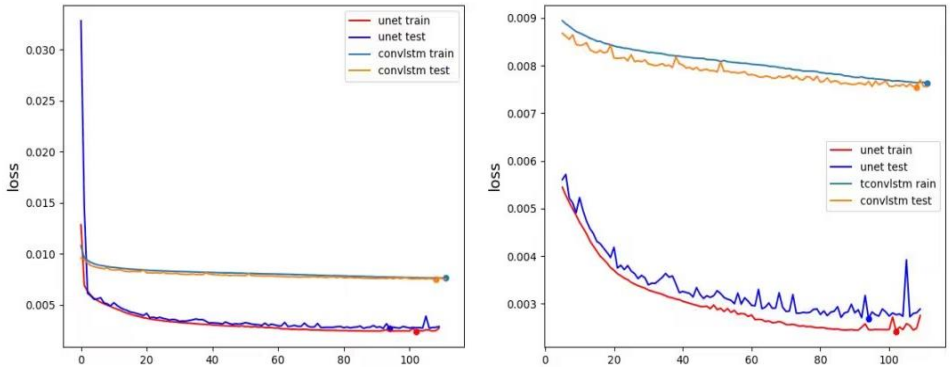


图 4.7 FURNet 网络与 ConvLSTM 网络损失函数对比图

在实际效果预测中，使用两个网络模型在测试集上的输出结果绘制了微物理特征信息的临近预报预测值的热力图，将预测出的结果进行可视化分析，如图 4.8 所示，给出了测试数据集中的一个代表性案例来说明建立的两个模型如何进行临近预报。图 4.8 的对流数据 ZH_True、ZH_ConvLSTM、分别表示一小时内雷达观测的 ZH_True，ConvLSTM 模型预测。为简洁起见，每两帧画出雷达图。从真实序列 ZH_True 可以看出，对流系统早期演化阶段降水较大。从预测序列 ZH_ConvLSTM 可以看出，该模型不能较好地预测地区的对流形成。而 ZH_UNet 预测序列能准确预报风暴的形成，在空间范围和强度分布上对地区风暴的预报也更为准确。定量评价（表 4.1）也证明了基于对称编码解码的 FURNet 预测的优越性。

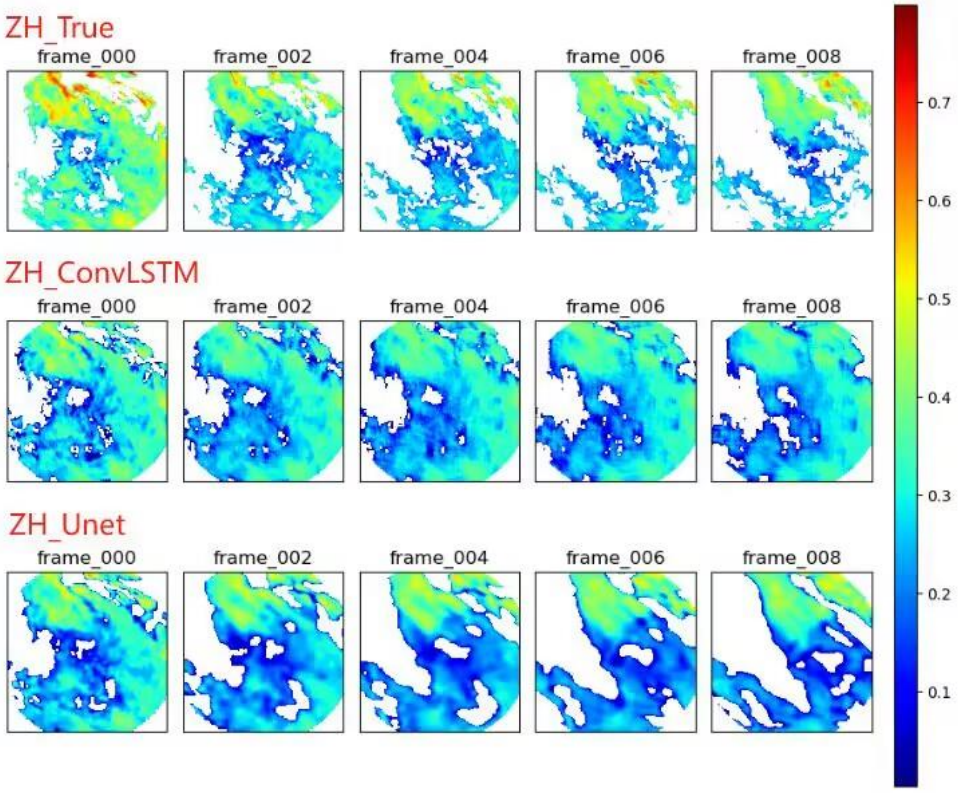


图 4.8 真实 ZH 与模型预测的对比图

对于模型的效果采用了相对误差（RE）、均方根误差（RMSE）、比率偏差（BIAS）和皮尔逊相关系数（CC）进行验证。如表 4.1 所示在 FURNet 模型中，相对误差为 0.4133，比 ConvLSTM 预测出来的误差较大，同时 FURNet 模型的均方根误差比 ConvLSTM 预测的结果更趋向 0，且比率偏差和皮尔逊相关系数也更趋向 1，这表明 FURNet 结果比 ConvLSTM 结果更接近实际。

表 4.1 ConvLSTM 与 FURNet 的评价指标

误差 / 方法	R_E	R_{MSE}	B_{IAS}	C_C
ConvLSTM	0.8301	0.1006	0.5685	0.5015
FURNet	0.4133	0.0553	0.8336	0.8512

主要原因为 FURNet 在 UNet 的基础上引入了 ResNet、Late Fusion、SE-Block 模块，增强了特征提取和上下文信息的能力，进行多尺度特征融合保留了关键信息，并且能够进行多特征融合。在强对流临近预报中，使用 UNet-FURNet 模型可以更好地捕捉双偏振雷达资料中微物理特征的细节和上下文信息，从而提高预报的准确性。相比之下，ConvLSTM 虽然可以处理时空序列数据，并有效地捕捉时序特征和空间关系，且在预测时考虑时序信息。然而，在处理双偏振雷达资料中的微物理特征信息时，ConvLSTM 却可能无法充分利用图像的上下文信息并且经过不断的卷积和池化操作，可能会丢失许多关键信息，影响网络结果的判断。

五、问题二的分析与求解

5.1 问题分析

由于以往一些数据驱动的算法在进行强对流预报时，倾向于生成接近于平均值的预报，即存在“回归到平均（Regression to the mean）”问题，导致预测结果缺乏清晰的细节和差异，无法准确地捕捉到强对流的空间分布和强度变化。预测结果可能会呈现出过于平滑的趋势，无法准确地反映出强对流的局部细节和特征。因此，需要建立数学模型以缓解预报的模糊效应，使预报出的雷达回波细节更充分、更真实。在问题一的基础上（因此本题不需要对数据再进行预处理），已经建立利用微物理特征信息数据对强对流临近预报的网络模型，然而在强对流临近预报中，预测结果缺乏明显的细节和差异，预报结果模糊，因此考虑在 UNet 的降采样层和上采样层中引入 CBAM 注意力机制。

问题二的总体思路流程图如图 5.1 所示，大致流程和问题一相同，不同的是对第一题的两个网络模型，ConvLSTM 与 UNet-FURNet 进行融合。将 ConvLSTM 中时序维度的观念，融合到 UNet 中来，把时序维度设计成二维数据的高，构建了 3D 数据集，在 UNet 中使用了 3D 卷积来行处理。

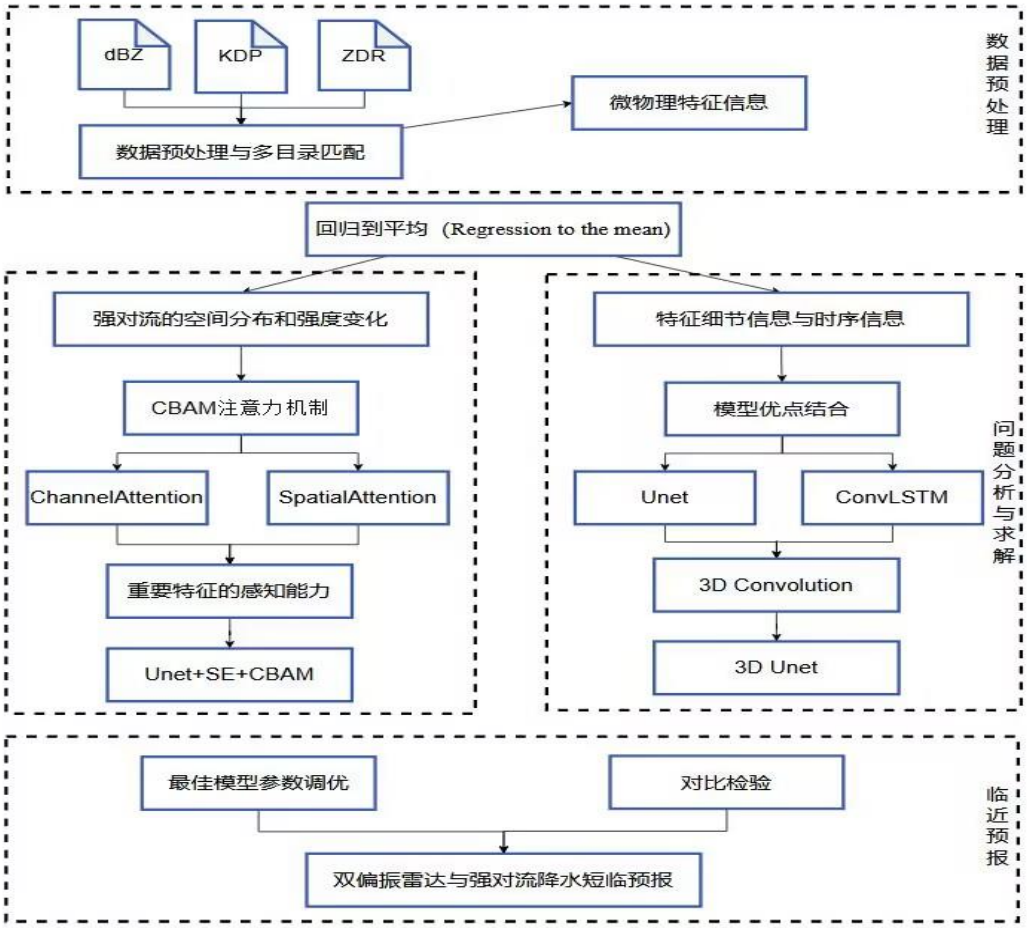


图 5.1 问题二的总体思路流程图

5.2 模型建立

5.2.1 基于注意力机制的 FURNet+CBAM 模型

“回归到平均”问题的原因可以归结为模型在处理双偏振雷达资料中的微物理特征信息时，没有明确地关注和强调重要的特征信息。FURNet 虽然在降采样的最底层的高级语义特征中使用了 SE 注意力机制进行多通道的加权融合（代码见附录 5），但是其在降采样层和上采样层中与传统的 UNet 模型类似，只使用了使用卷积和池化操作来提取特征，但这些操作没有明确地区分和强调不同特征的重要性。这导致了所有特征都被平等处理，无论其重要性如何，最终导致预测结果过于平均化。

为了解决这个问题，可以考虑在 UNet 的降采样层和上采样层中引入注意力机制（如图 5.2 所示），例如 CBAM（Convolutional Block Attention Module）注意力机制。CBAM 是一种用于卷积神经网络的注意力机制，旨在改善模型对图像中重要特征的提取能力。CBAM 模块可以应用于 CNN 的不同层级，以增强模型的感知能力和表征能力。CBAM 模块由两个子模块组成：通道注意力模块和空间注意力模块。

通道注意力模块（Channel Attention Module）：该模块通过学习图像不同通道的权重来提高模型对通道特征的关注度。它通过计算全局平均池化（Global Average Pooling）得到通道级别的特征表示，然后通过一个堆叠的全连接层来获取每个通道的权重。这些权重将应用于原始特征图上的每个通道，以增强对重要特征的感知能力。具体公式如下：

$$Channel\ Attention(X)=\sigma(FC(\mathrm{Re}LU(GloableAvgPooling(X)))),$$

其中，X 表示输入的特征图，MaxPooling 表示最大池化操作，AvgPooling 表示平均池化操作，Concat 表示拼接操作,Conv 表示卷积层,ReLU 表示激活函数(通常是 ReLU), σ 表示 Sigmoid 函数。

空间注意力模块（Spatial Attention Module）：该模块通过学习图像不同位置的权重来提高模型对空间特征的关注度。它采用两个并行的卷积操作，一个用于捕捉特征图在水平方向上的相关性，另一个用于捕捉特征图在垂直方向上的相关性。然后，通过对这两个方向上的特征图进行求和或者连接操作，得到最终的注意力图。这个注意力图将与原始特征图相乘，以突出重要的空间位置。具体公式如下：

$$Spatial\ Attention(X)=\sigma(Conv(\mathrm{Re}LU(Concat(MaxPooling(X),AvgPooling(X))))),$$

其中，X 表示输入的特征图，MaxPooling 表示最大池化操作，AvgPooling 表示平均池化操作，Concat 表示拼接操作,Conv 表示卷积层,ReLU 表示激活函数(通常是 ReLU), σ 表示 Sigmoid 函数。

通过使用 CBAM 模块，卷积神经网络能够自适应地学习图像中不同通道和空间位置的重要性，并且能够动态地调整特征图的权重，从而提高模型的表征能力和准确性。CBAM 注意力机制能够自动学习和调整特征图中不同空间位置的重要性，从而更好地捕捉到关键特征。它通过两个关键模块，即通道注意力模块（Channel Attention Module）和空间注意力模块（Spatial Attention Module），来提取和调整特征图的通道和空间信息。具体公式如下：

$$CBAM(X)=Channel\ Attention(X)\times Spatial\ Attention(X),$$

其中，X 表示输入的特征图，$\times$表示元素相乘操作。通过引入通道注意力和空间注意力，CBAM 模块能够自适应地调整通道和空间维度上的特征重要性，从而提升网络的性能和表

示能力。

在 UNet 中加入 CBAM 注意力机制的有效性和合理性在于，它可以帮助网络更加关注重要的特征，并减轻“回归到平均”问题。通道注意力模块可以自适应地调整不同通道的权重，使得网络更加关注重要的特征通道，而空间注意力模块可以自适应地调整不同空间位置的权重，使得网络更加关注重要的空间位置。这样，网络可以更好地捕捉到微物理特征的细节和差异，从而提高预测的准确性和清晰度。

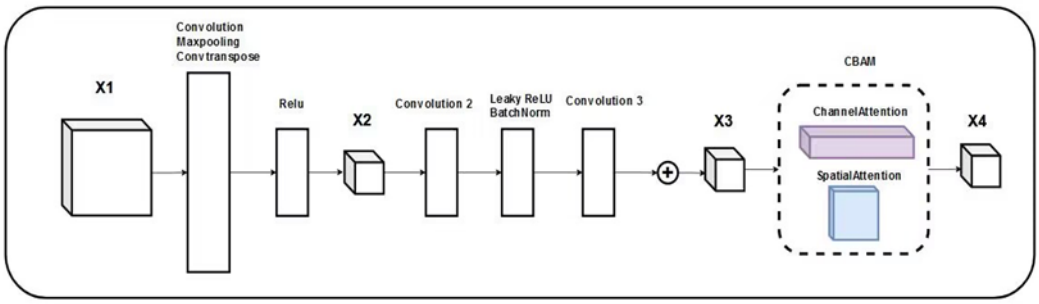


图 5.2 UNet- FURNet+CBAM 网络结构简图

5.2.2 基于多通道的 3D-UNET 模型

同时针对第一题的两个网络模型，ConvLSTM 与 UNet-FURNet。UNet-FURNet 进行多尺度特征融合保留了关键信息，可以更好地捕捉双偏振雷达资料中微物理特征的细节和上下文信息，从而提高预报的准确性。ConvLSTM 可以有效地捕捉时序特征和空间关系，并在预测时考虑时序信息。 具体的 ConvLSTM 要求数据格式为（batch，通道，时序，长，宽），而 UNet 要求数据格式为（batch，通道，长，宽）。采用 ConvLSTM 中时序维度的观念，融合到 UNet 中来，把时序维度设计成二维数据的高，构建了 3D 数据集，在 UNet 中使用了 3D 卷积（如图 5.3 所示）来行处理（代码见附录 6）。

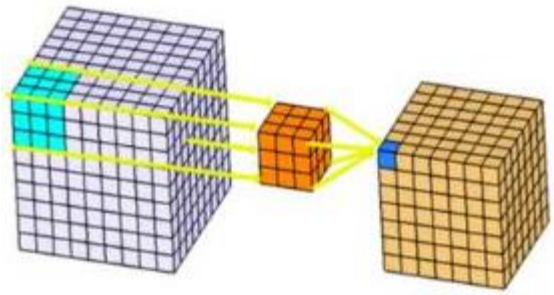


图 5.3 3D 卷积示意图

将输入数据从（N，30,256,256）变为（N，10,3,256,256）可以通过以下步骤实现：

（1）确定时间窗口大小和步长：首先，需要确定时间窗口大小和步长。时间窗口大小表示每个子序列的时间步数，步长表示相邻子序列之间的时间间隔。这里时间步长为 10。

（2）划分子序列：根据时间窗口大小和步长，将原始数据划分为多个连续的子序列。每个子序列的长度为时间窗口大小，相邻子序列之间的时间间隔为步长。

（3）组织子序列：将每个子序列组织为一个新的维度。在这种情况下，可以将子序

列作为新维度中的样本，时间步作为第一维，原先的通道数作为第二维，图像的高度和宽度保持不变。

（4）重塑数据：根据上述组织方式，将子序列重新排列成（N，10,3,256,256）的形状。其中 N 表示样本数，10 表示时间步数，3 表示新的通道数，256 表示图像的高度，256 表示图像的宽度。

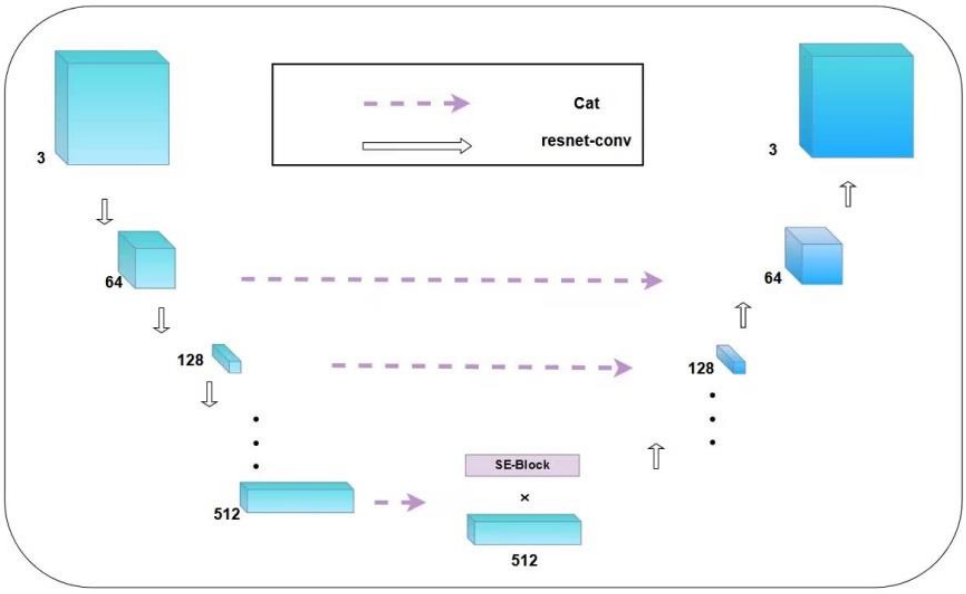


图 5.4 3D-UNET 模型示意图

将时间序列从 2D 变为 3D 可以使神经网络能够更好地利用时序信息，提高模型对时间序列的建模能力。并且可以有效地提取时间序列中的局部特征，将其作为子序列输入到神经网络中。这样可以增加模型对时间序列的敏感性，更好地捕捉序列中的规律和模式。还适用于各种不同长度的时间序列，可以根据任务需求和数据特点自由选择窗口大小。总之，将时间序列从 2D 变为 3D 的建模方法通过滑动窗口的方式，在神经网络中更好地利用了时序信息和局部特征，为时间序列建模提供了更强的表达能力。

5.3 模型求解与检验

CBAM 网络的损失结果图显示在图 5.5 中，右侧为局部放大图。在训练集上，模型的损失值由红色曲线表示，整体上随着训练步骤的进行而逐渐减小。初始阶段损失值较高，但从第 1 个 epoch 循环开始迅速下降。在 100 个 epoch 之后出现波动，在第 118 个 epoch 时达到最低点（用红点标记，最低值为 0.00238）。在测试集上，损失结果由蓝色曲线表示，出现一些局部震荡或波动，但总体上呈下降趋势。最终的损失值下降到 0.0026。值得注意的是，训练集和测试集 100 步以后振动幅度突然增加，说明已经过拟合，因此本文统一 epoch 为 100。

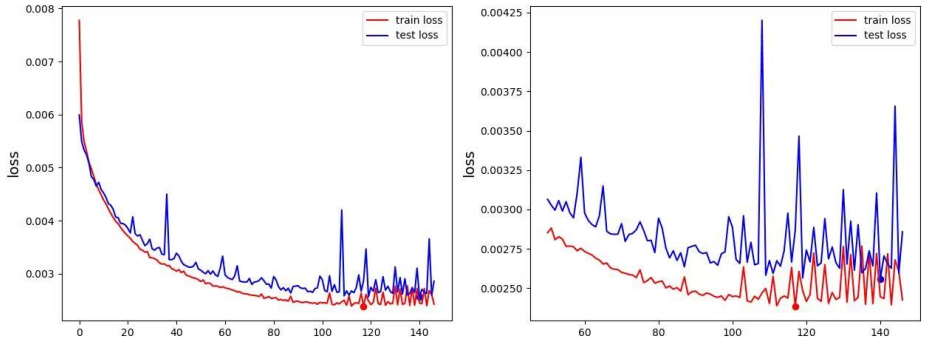


图 5.5 CBAM 网络的损失结果图

3D-UNET 网络的损失结果图（代码见附录 7）显示在图 5.6 中，右侧为局部放大图。在训练集上，模型的损失值由浅蓝曲线表示，整体上随着训练步骤的进行而逐渐减小。初始阶段损失值较高，但从第 1 个 epoch 循环开始迅速下降。在 20 个 epoch 之后开始收敛，在第 29 个 epoch 时达到最低点（用红点标记，最低值为 0.00238）。在测试集上，损失结果由黄色曲线表示，出现一些局部震荡或波动，但总体上呈下降趋势。最终的损失值下降到 0.0021。

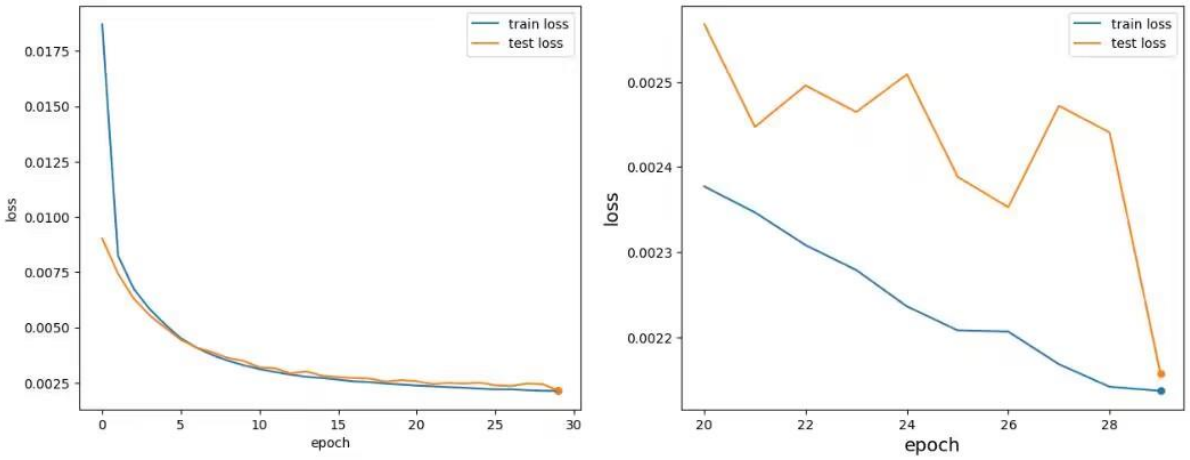


图 5.6 3D-UNet 网络训练损失变化图

在实际效果预测中，使用了三个网络模型对测试集进行了输出结果的可视化分析。图 5.7 展示了微物理特征信息的临近预报预测值的热力图。其中，ZH_True 表示雷达观测的真实序列，ZH_UNet 表示 FURNet 模型的预测序列，ZH_UNet_CBAM 表示引入 CBAM 注意力机制的 FURNet 模型的预测序列，ZH_3D_UNet 表示在 UNet 中融合了 ConvLSTM 时序维度观念的预测序列。

通过观察图 5.7 中的结果，可以得出以下结论：

（1）从真实序列 ZH_True 可以看出，对流系统在早期演化阶段降水较大

（2）从预测序列 Zh_UNet 可以看出，该模型不能很好地预测地区的对流形成，可能存在预测误差。

（3）相比之下，ZH_UNet_CBAM 和 ZH_3D_UNet 的预测序列能够准确预报风暴的形成。在空间范围和强度分布上，对地区风暴的预报也更为准确。

综上所述，引入 CBAM 注意力机制和将 ConvLSTM 时序维度观念融合到 UNet 中的模

型能够提高对临近预报的准确性，更好地预测地区的对流形成。

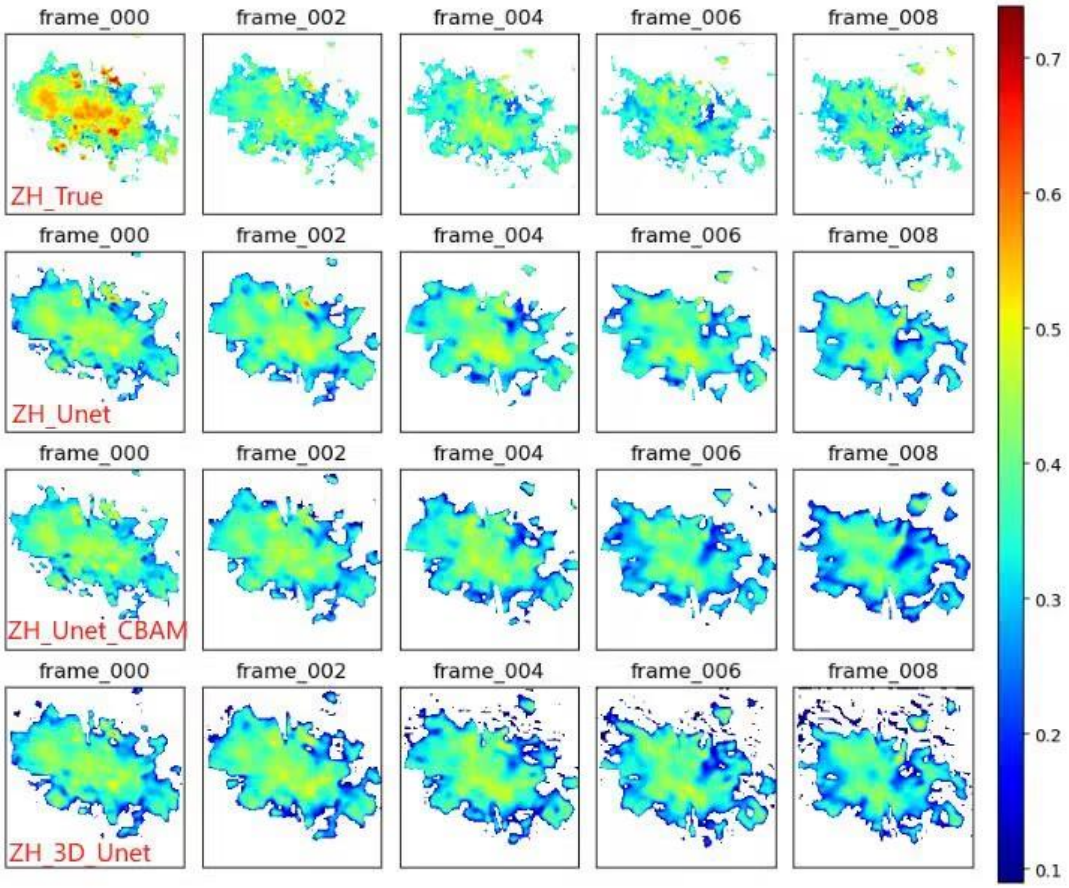


图 5.7 真实 ZH 与模型预测的对比图

为了验证模型的效果，同样使用问题一制定的评价指标进行评估。根据表中的数据，在 FURNet 模型中，相对误差为 0.4059。相比之下，CBAM 模型的预测误差较大，但比 3D-UNet 模型的预测误差要小。同时，CBAM 模型的比率偏差更接近于 1，说明其预测结果更接近实际情况，相比于 FURNet 模型更准确。虽然 CBAM 模型在相对误差、比率偏差等指标上表现优于 FURNet 模型。总之，在不同时刻的临近预报中，加入 CBAM 的 UNet 神经网络和 3D_UNet 神经网络都会出现了不同程度的细节信息，缓解了“回归到平均”问题。

表 5.1 改进模型的评价指标

误差 / 方法	R_E	R_{MSE}	B_{IAS}	C_C
FURNet+CBAM	0.3983	0.0548	0.8603	0.8656
UNet-FURENet	0.4059	0.0561	0.8414	0.8569
3D-UNet	0.5120	0.0538	0.7897	0.8645

六、问题三的分析与求解

6.1 问题分析

问题三目的在于利用 Z_H 和 Z_{DR} 数据设计数学模型，以实现对降水量的定量估计。这是一个 QPE 问题，在强对流系统中，降水强度可能会非常高，存在较大的降水粒子和较强的垂直运动。对于这种系统，QPE 的性能可能会受到一些挑战。雷达反射率与降水强度之间的关系可能会变得非线性，因为存在较大的降水粒子和不均匀的降水分布。此外，强对流系统中可能存在冰晶和雹子等非球形粒子，这也会对 QPE 的准确性造成影响。这里，不考虑冰晶和雹子这些非球形粒子产生的影响。在本题中，数据预处理方法与问题一相同，对原始 DSD 分钟数据中降水率小于 0.1 mm/h 或者降水粒子数小于 50 的进行剔除。根据题目已给数据，部分降水量的变化情况如图 6.1 所示。

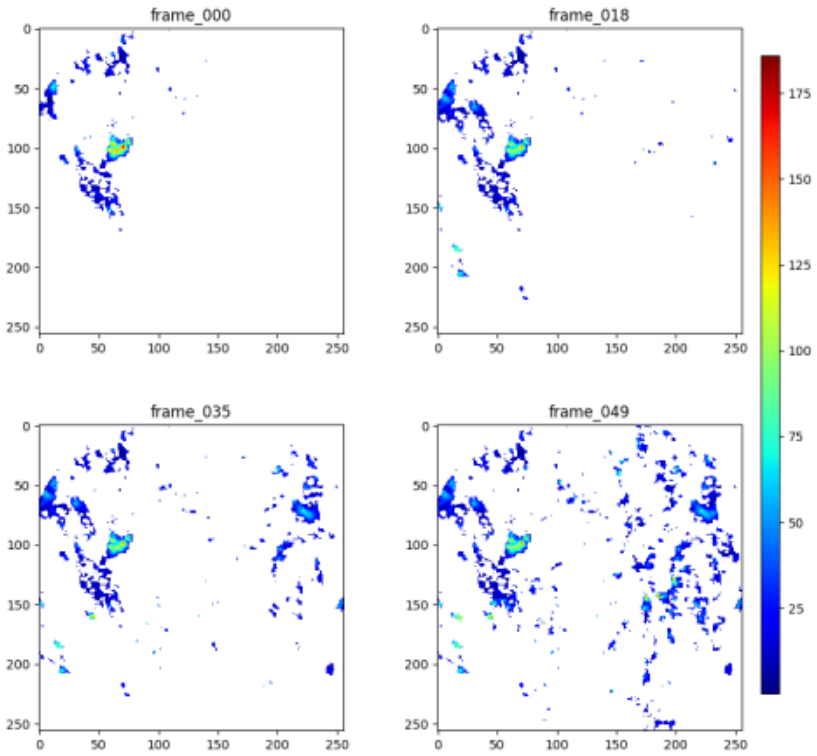


图 6.1 四个帧的降水累积密度图

本题主要建立两个模型来解决 QPE 问题，总体思路如图 6.2 所示。首先建立 CSU-HIDRO 模型，利用题目已知数据拟合出双偏振雷达两个微物理参数 Z_H 和 Z_{DR} 与降水量之间的关系式。传统经验模型容易受到降水粒子、雷达系统、仪器以及其他人为因素的影响。为了减少这些不确定性因素的影响，本文利用机器学习中的符号回归方法对原有模型进行改进优化，将 Z_H 和 Z_{DR} 作为输入量，通过学习这两个微物理参数和降水量之间的映射，输出降水量并对其进行定量分析。

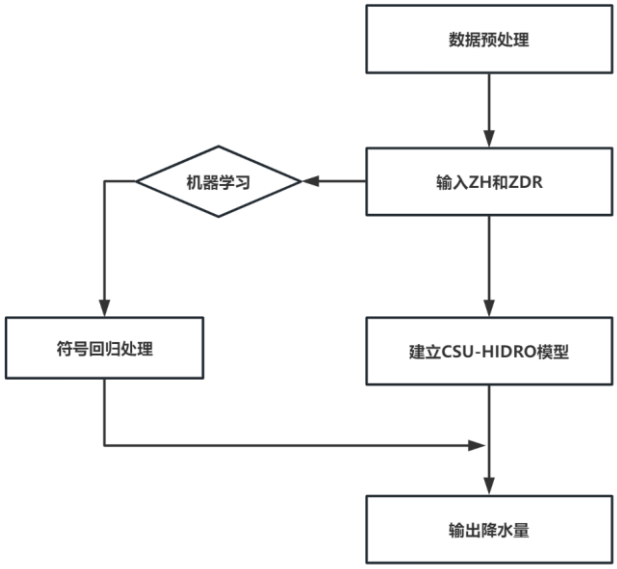


图 6.2 QPE 问题处理思路

6.2 数据预处理

6.2.1 3σ准则剔除异常数据

在 3σ 原则下，数值如超过 3 倍标准差，那么可以将其视为异常值。$P(|x-\mu| \le 3\sigma)$ 的概率是 99.7%，那么距离平均值 3σ 之外的值出现的概率为 $P(|x-\mu| \ge 3\sigma) = 0.03$，属于极个别的小概率事件，认为不会发生所以将 3$\sigma$ 之外的值剔除。本文画出三个变量的箱线图（如图 6.3 所示），降水量 R 的分布比较集中，Z_H 和 Z_{DR} 的分布比较离散，根据 3σ 原则，将三变量超出范围的值剔除（代码见附录 9）。

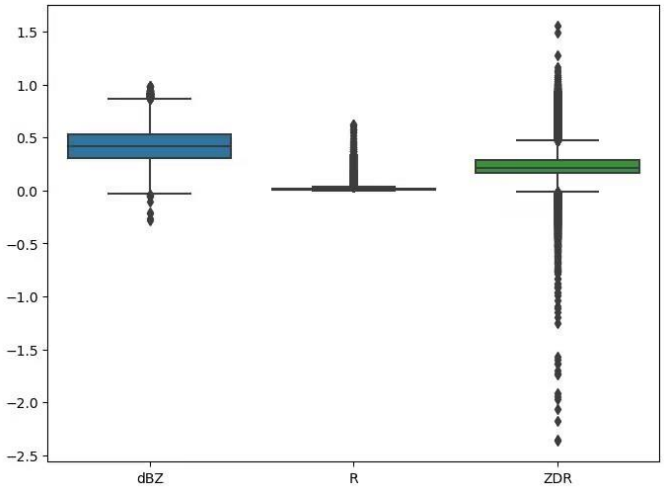


图 6.3 ZH，ZDR 和 R 变量的箱线图

6.2.2 综合数据处理

对剔除异常值的特征进行处理，按照问题一的数据处理，去除雷达反射率小于一定阈值的像素，去除强度不连续的区域（例如经过强降水的降雨区域），以及根据双偏振参数的一致性进行筛选。然后过滤非雨天样本，对于产生空值在相应坐标系下利用双线性插值填补空缺，最后采用最小-最大归一化将不同范围的数据缩放到相同的范围内，并消除量纲的影响。

6.2.3 变量相关性分析

本文利用皮尔逊相关系数，衡量 Z_H，Z_{DR} 和 R 变量之间的线性相关程度。它的取值范围从-1 到 1，其中 1 表示完全正相关，-1 表示完全负相关，0 表示没有线性相关性。如图 6.4 所示，颜色越红说明变量直接的发现三者之间线性相关的关系不明显，因此考虑三者之间存在非线性的关系。

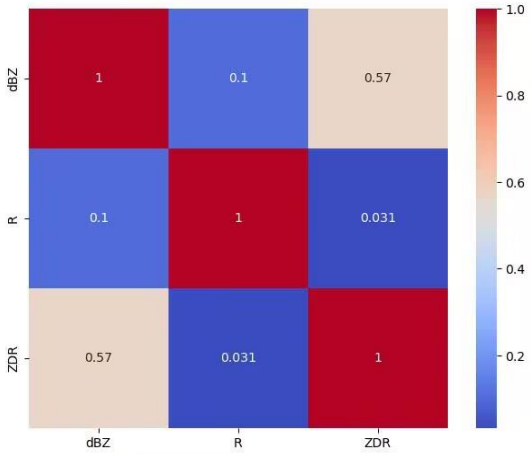


图 6.4 Z_H，Z_{DR} 和 R 变量的相关性热图

6.3 模型建立

6.3.1 基于 CSU-HIDRO 雷达模型定量降水估测

CSU-HDIRO 是由美国科罗拉多州立大学的 Cifelli 等人在 2011 年提出的一种双偏振雷达定量降水估测优化算法。该算法基于模糊逻辑法水凝物分类，并根据不同的降水类型将降水分为液态降水、固态降水和混合型降水三种类型。然后，根据 K_{DP}、Z_H 和 Z_{DR} 的阈值选择不同的降水率计算公式（代码见附录 10）。CSU-HDIRO 算法的计算公式如下：

$$R(Z_H)=a_1 Z_H^{b_1},$$
$$R(K_{DP})=a_2 K_{DP}^{b_2},$$
$$R(Z_H,Z_{DR})=a_3 Z_H^{b_3} 10^{C_3 Z_{DR}},$$
$$R(K_{DP},Z_{DR})=a_4 (K_{DP})^{b_4} 10^{c_4 Z_{DR}}。$$

本文所取的参数数据均在合理阈值范围内，并且双偏振雷达参数特征与降水量估测特征保持一致。在本文研究中，CSU-HDIRO 算法流程图如图 6.5 所示。

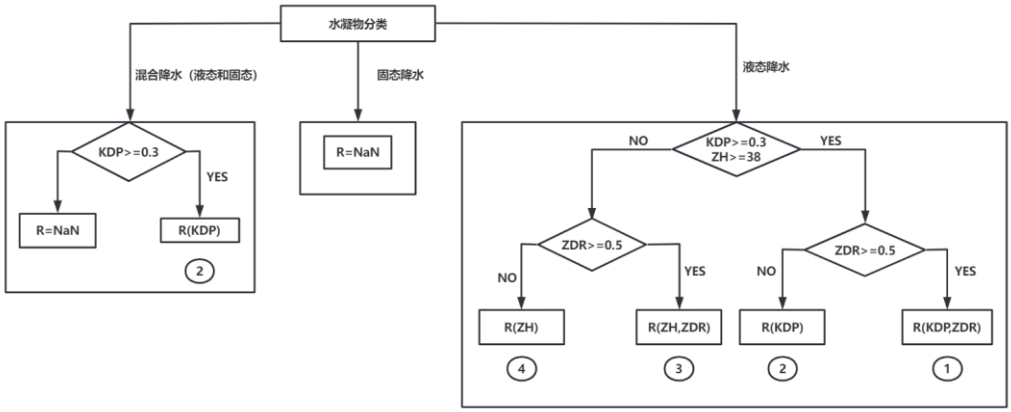


图 6.5 CSU-HDIRO 算法流程图

原始 DSD 数据会对拟合参数进行干扰，雷达反射率与降水之间的关系并不是线性的，而是非常复杂的。这是因为降水粒子的形状、分布、速度等因素都会影响雷达反射率的测量结果。此外，雷达反射率还受到雷达系统的配置、仪器误差、大气衰减等因素的影响。为了减少这些不确定性因素的影响，本文提出了一种基于双偏振雷达多个微物理参数分类拟合的方法，这里根据题目要求不考虑 KDP 的特征贡献，所以优化后模型为：

$$\begin{cases} R(Z_H) = a_1 10^{C_1 Z_H^{b_1}} \\ R(Z_H, Z_{DR}) = a_2 10^{C_2 Z_H^{b_2}} 10^{C_3 Z_{DR}^{b_3}} \end{cases},$$

将等效雷达反射率因子 Z_H 定义为：

$$Z_H = 10\log_{10}\left(\frac{4\lambda^4}{\pi^4 |K_W|^2} \int_{D_{\min}}^{D_{\max}} \left| f_{HH,VV}(\pi, D) \right|^2 N(D) dD \right),$$

其中，D 为当量直径，$N(D)$为单位体积空气中、单位尺寸区间内雨滴的个数浓度；λ 为雷达波长；$D_{\min}$ 和 D_{max} 分别为实际降水粒子（DSD）的最小和最大直径；K_W 为水的介电常数因子，常温状态下，$K_W = 78.5 + 4.9i$；$f_{HH,VV}(\pi, D)$是水平或垂直偏振时的后向散射幅度。

6.3.2 基于遗传算法的符号回归模型定量降水估测

符号回归（Symbolic Regression）是一种机器学习方法，用于在给定的数据集中寻找数学表达式或函数来描述变量之间的关系（代码见附录 8）。它的目标是通过自动发现数学公式或函数来建立一个符号表达式模型，而不需要预先设定模型的形式。

符号回归则通过搜索和优化算法来自动发现最优的数学表达式或函数形式，以最好地拟合给定的数据。通常使用遗传算法、遗传编程、粒子群优化等进化算法来搜索最优的数学表达式。这些算法通过不断迭代和优化，逐步改进和调整表达式的形式和参数，以找到最佳的拟合结果。

一个重要特点是，它可以通过将表达式表示为二叉树来进行处理。二叉树是一种树状结构，其中每个节点最多有两个子节点。在符号回归中，每个节点代表一个操作符或变量，并且每个叶子节点代表一个常量或输入特征（如图 6.6 所示）。通过组合不同的操作符和变量，可以构建出复杂的符号表达式。

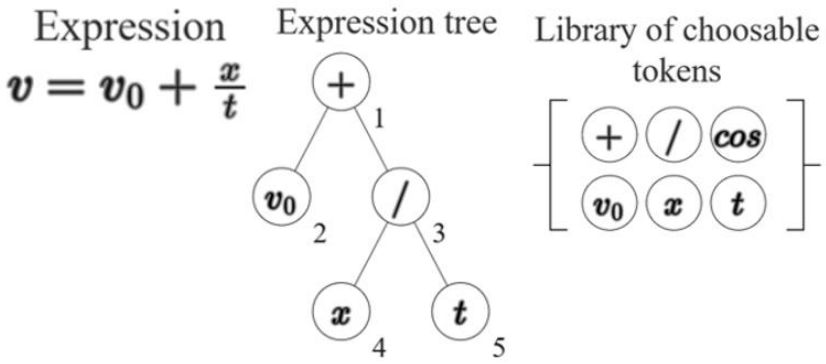


图 6.6 符号回归的二叉树表示

值得注意的是，表达式的总体是随机抽样的。在这个子样本中，进行比赛，并选择获胜者进行育种:通过突变，交叉，简化或显式优化（如图 6.7 所示）。将遗传算法嵌入到“进化-简化-优化”循环中。“进化”是基于一系列突变的基因选择进化的重复应用。换句话说，这是一个种群的整个进化过程。这里的“简化”指的是相等简化-在循环过程中，使用一组代数等价将方程简化为等价形式(由于这种情况很少发生，因此通常不会通过限制搜索空间而损害发现)。这部分算法显著改善了包含实常数方程的发现，对表达式中的数值常数进行局部梯度搜索可以极大地提高性能——紧密地将其优化和随机版本集成在一起。在继续进行简化优化阶段之前执行几个突变(以及交叉)的原因是因为一些方程只能通过冗余中间状态访问。以下是符号回归算法。

符号回归算法：

输入：要查找表达式的数据集
输出：每个复杂度下的最佳表达式
参数：种群数量，每个种群中的表达式数量，表达式的替换分数 1，表达式的替换分数 2

1. 假设有一群个体、一个适应度函数和一组突变算子。
2. 从总体中随机选择一个 ns 大小的个体子集。
3. 通过评估该子集中每个个体的适合度来进行比较。
4. 选择最适合的个体作为获胜者，概率为 p 。否则，从子集中删除该个体并再次重复此步骤。如果还剩下一个，选择它。因此，概率大致为 $p, p(1-p), p(1-p)^2, \cdots$ ，然后依此类推。
5. 创建该选定个体的副本，并从一组可能的突变中应用随机选择的突变。
6. 用突变个体替换种群中的一个成员。通常替换最弱的群体或子集。

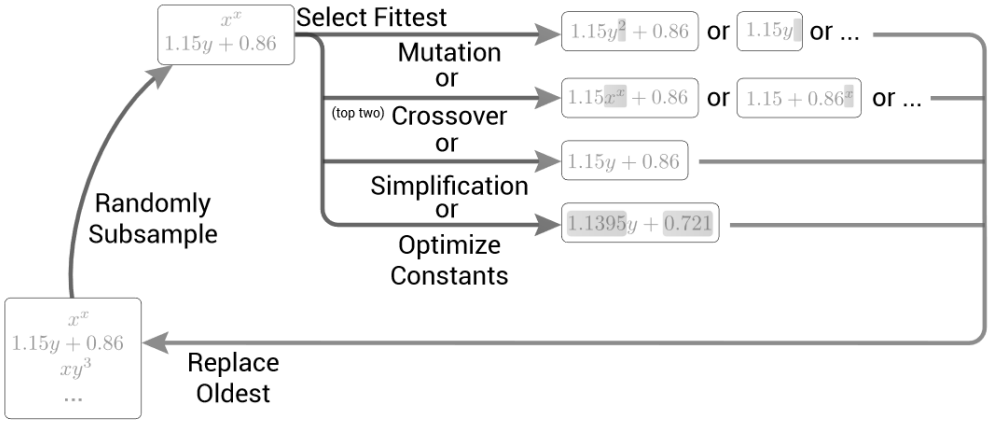


图 6.7 基于遗传算法的符号回归逻辑

6.4 模型求解与检验

基于优化的 CSU-HIDRO 雷达模型的拟合和 GANG CHEN、KUN ZHAO 等人[2]进行的拟合形式相同，得到反射率和降水之间的函数关系：

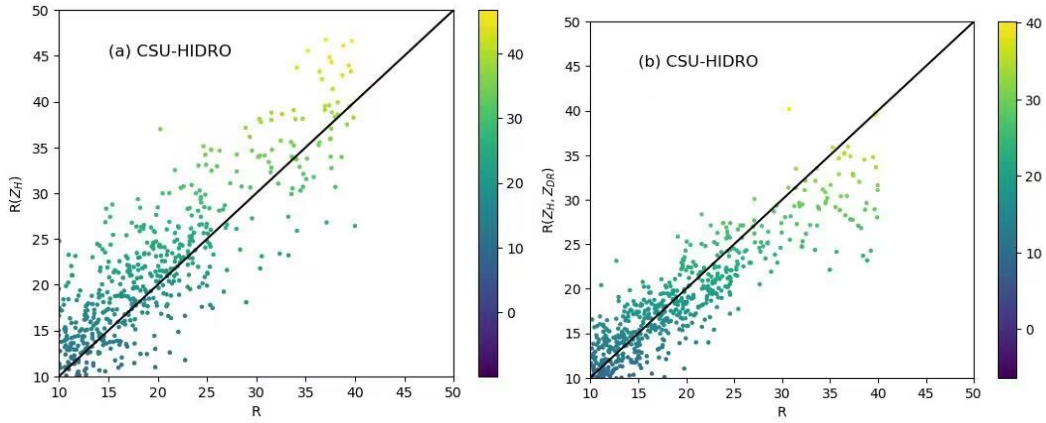
$$R(Z_H)=1.675\times Z_H^{0.855},$$
$$R(Z_H,Z_{DR})=1.8742\times Z_H^{0.7685}\times 10^{0.00482Z_{DR}}。$$

基于遗传算法的符号回归模型，得到反射率和降水之间的函数关系：

$$R(Z_H)=1.5879\times Z_H^{0.7745},$$
$$R(Z_H,Z_{DR})=1.5438\times Z_H^{0.6742}\times 10^{0.0052Z_{DR}}。$$

发现符号回归得到的模型与优化的 CSU-HIDRO 雷达模型模型相似，但符号回归拟合的速度更加快速。

利用题目已给部分数据，对该拟合关系式进行验证，可以得到降水量和雷达反射率（Z_H,Z_{DR}）之间的关系图，将双偏振雷达的估计量与实际降水量进行比较，如图 6.8 所示。



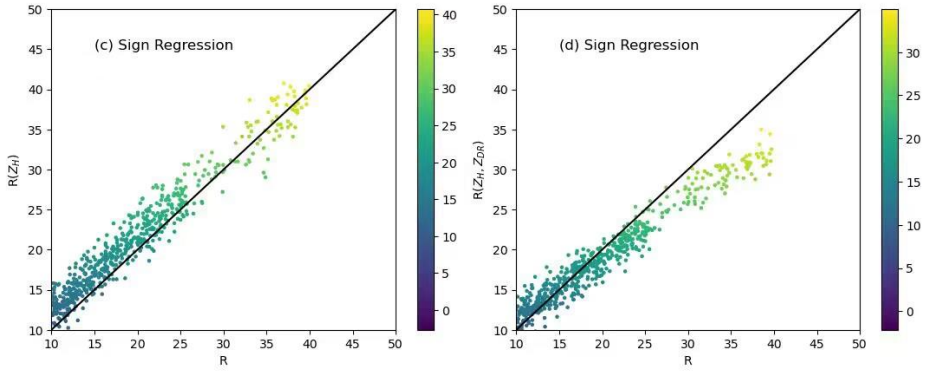


图 6.8 不同模型预测值与雷达观测值的对比图

在 CSU-HDIRO 模型中，单个 Z_H 拟合出来的相对误差为 0.2543，相对于 Z_H, Z_{DR} 组合出来的误差较大，同时 Z_H，Z_{DR} 组合拟合的均方根误差、比率偏差都比单个 Z_H 拟合的结果更趋向 0，而皮尔逊相关系数也更趋向 1，这表明组合拟合结果比单一拟合结果更接近实际。符号回归模型拟合出来的误差也表明组合的形式优于单一拟合，具体数值如表 6.1 所示。

另外，由表 6.1 可以看出，符号回归模型中单个 Z_H 拟合误差为 0.1676，这比 CSU-HDIRO 模型中的单个拟合误差小，同时组合误差也更接近于 0。均方根误差、比率偏差以及皮尔逊相关系数都优于传统 CSU-HDIRO 模型。因此，符号回归拟合出的结果要优于传统经验模型。

表 6.1 CSU-HDIRO 与符号回归模型的评价指标表

误差 / 方法	模型	R_E	R_{MSE}	B_{IAS}	C_C
CSU-HDIRO	$R(Z_H)$	0.2543	0.2608	0.9027	0.9806
	$R(Z_H, Z_{DR})$	0.1500	0.1576	0.9542	0.9892
符号回归	$R(Z_H)$	0.1676	0.1725	0.9655	0.9904
	$R(Z_H, Z_{DR})$	0.0872	0.0749	0.9861	0.9978

七、问题四的分析与求解

7.1 问题分析

评估双偏振雷达资料在强对流降水临近预报中的贡献以及优化数据融合策略是一个复杂的问题，涉及到大量的数据处理、模型设计和性能评估。双偏振雷达资料包括多种参数（如 Z_H、Z_{DR}、K_{DP} 等），这些参数对于不同类型的强对流天气有不同的重要性。
流性降水迅速、短暂和高度非线性动力学，使得精确预测变得非常具有挑战性。强对流天气区别于其他对流天气最主要的两个特征为"突发性"和"局地性强"。

突发性：指强对流天气情况的变动速度非常快，往往在很短的时间内出现强降水等极

端状况。这种突发性的特点使得准确预报强对流天气变得具有挑战性。具体地，在利用前10 帧的微物理特征信息对后 10 帧进行预测时，后 10 帧的某一帧可能会突然出现强降雨天气，这时候前 10 帧的情况通常是不足够来进行突发性的降水预测， 因为短时间内的变化很难被准确捕捉和预测。

局地性强：指强对流天气的发生范围通常比较小，降雨的覆盖面比较集中，局限在某些特定的地区或区域，表现为局部强降雨现象。这种局地性强的特点使得准确预测强对流天气的具体位置和范围变得困难，因为强对流天气往往只影响到局部地区，而其他地区可能没有强对流天气。

数据融合方式方面，包括了多源数据集的融合、加权平均或集成学习的模型融合、以及构造更高相关性的特征融合、多尺度特征信息融合等情况（代码见附录 11）。

（1）突发性天气往往涉及不同时间和空间尺度上的变化。通过在模型中引入多尺度的特征提取，可以更好地捕捉不同时间和空间尺度上的突发性变化。UNet 系列网络可以处理不同尺度的特征，然后将它们进行融合（多尺度特征融合），以获取全局和局部的特征信息。突发性天气通常与某些局部区域或特定特征相关（局地性强）。

（2）注意力机制可以根据历史观测数据的重要性动态地调整模型的权重，使其更加注重突发性天气的预测。通过引入 CBAM 注意力机制，通过在特征的空间维度和通道维度赋予不同的权重，可以使模型更加关注与突发性天气相关的区域或特征，从而提高对突发性天气的感知能力。与现有的 SE 注意力机制区别，CBAM 注意力机制同时实现了空间位置的加权，在此任务中将有助于应对“局地性强”的问题。

（3）多源数据融合，我们了解到现有的数据包括了 1km、3km、7km 等高面的数据。在对 3km 等高面的强对流天气进行预报时，我们可以同时利用 1km、3km、7km 的三种数据，因为 1km、7km 可能包含有许多潜在的信息，这样充分利用各个位置之间的相关性与空间位置信息能够帮助我们应对“突发性”、“局地性强”等问题。

（4）特征数据集融合构造，我们把时序数据的时序维度看作 3D 卷积的通道维，结合 UNet 实现了 3D 数据的构造与预测，其中数据格式为（batch，通道，高，长，宽）。

（5）多网络模型融合，我们利用 GNN 提取多源数据特征然后充分利用 UNet 网络优势进行预报。

因为强对流天气具有高度的时空变化性，需要实时或近实时的预测。通过使用以上方法（具体流程见图 7.1），结合双偏振雷达资料中微物理特征信息，我们可以相对准确的实现了强对流降水临近预报的降水预测。

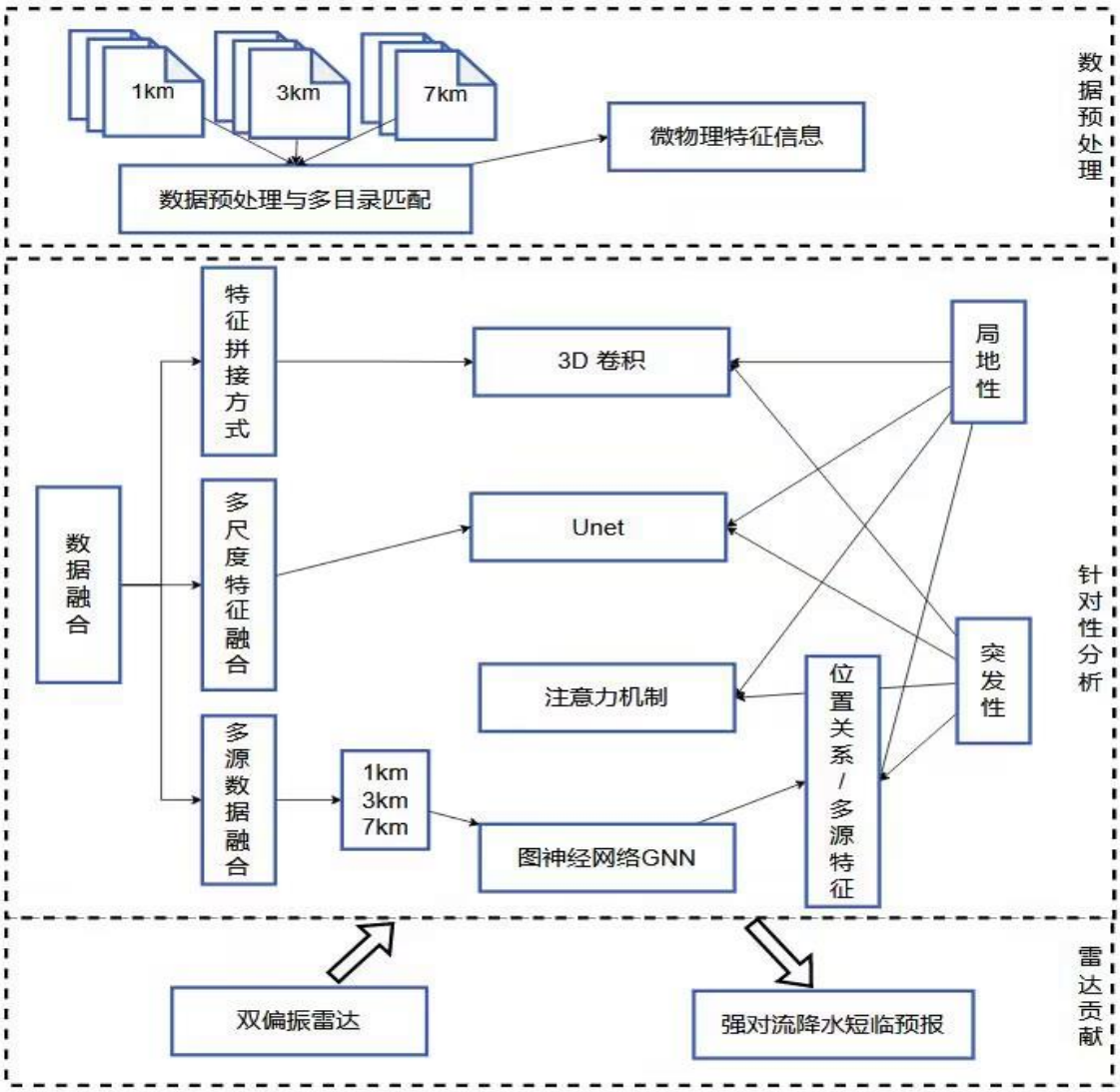


图 7.1 问题四的总体思路

7.2 模型建立

对流性降水迅速、短暂和高度非线性动力学，使得精确预测变得非常具有挑战性。强对流天气区别于其他对流天气最主要的两个特征为"突发性"和"局地性强"。其中突发性是指强对流天气情况的变动速度非常快，往往在很短的时间内出现强降水等极端状况。这种突发性的特点使得准确预报强对流天气变得具有挑战性。具体地，在利用前 10 帧的微物理特征信息对后 10 帧进行预测时，后 10 帧的某一帧可能会突然出现强降雨天气，这时候前 10 帧的情况通常是不足够来进行突发性的降水预测，因为短时间内的变化很难被准确捕捉和预测。而局地性强：指强对流天气的发生范围通常比较小，降雨的覆盖面比较集中，局限在某些特定的地区或区域，表现为局部强降雨现象。这种局地性强的特点使得准确预测强对流天气的具体位置和范围变得困难，因为强对流天气往往只影响到局部地区，而其他地区可能没有强对流天气。

对于 1km,3km,7km 的等高线数据，首先我们认为不同等高线数据之间存在关联，并且

不同的等高线数据可能含有不同的潜在降水特征。运用不同等高线的数据构建了图结构数据。其中每一个数据为（3，256，256），最终构建成的图结构数据为（256×256，3），其含义为有 256×256 个位置的数据点，每个数据点包含了 1km，3km，7km 的 3 个特征。同时我们认为不同位置之间的存在着潜在联系，通过图神经网络来进行不同位置之间的交互实现消息传递来更新结构。网络输入（256×256，3）的三等高线微物理特征信息，输出（256×256，1）的单等高线微物理特征信息作为新的 3km 处的特征。

对于以上问题，先进行多源数据融合，我们了解到现有的数据包括了 1km、3km、7km 等高面的数据。在对 3km 等高面的强对流天气进行预报时，我们可以同时利用 1km、3km、7km 的三种数据，因为 1km、7km 可能包含有许多潜在的信息，这样充分利用各个位置之间的相关性与空间位置信息能够帮助我们应对“突发性”、“局地性强”等问题。数据融合方式方面包括了多源数据集的融合、加权平均或集成学习的模型融合、以及构造更高相关性的特征融合、多尺度特征信息融合等情况。

接着特征数据集融合构造，把时序数据的时序维度看作 3D 卷积的通道维，结合 UNet 实现了 3D 数据的构造与预测，其中数据格式为（batch，通道，高，长，宽）。
然后通过在模型中引入多尺度的特征提取，可以更好地捕捉不同时间和空间尺度上的突发性变化。UNet 系列网络可以处理不同尺度的特征，然后将它们进行融合（多尺度特征融合），以获取全局和局部的特征信息。需要注意的是，突发性天气通常与某些局部区域或特定特征相关（局地性强）。

并且引入 CBAM 注意力机制，通过在特征的空间维度和通道维度赋予不同的权重，可以使模型更加关注与突发性天气相关的区域或特征，从而提高对突发性天气的感知能力。CBAM 注意力机制同时实现了空间位置的加权（与现有的 SE 注意力机制区别），在此任务中将有助于应对“局地性强”的问题。

最后多网络模型融合，利用 GNN 提取多源数据特征并且充分利用 UNet 网络优势进行预报。因为强对流天气具有高度的时空变化性，需要实时或近实时的预测。通过使用以上方法，结合双偏振雷达资料中微物理特征信息，我们可以相对准确的实现了强对流降水临近预报的降水预测。

当涉及到图神经网络（如图 7.2）时，常用的计算公式包括图卷积操作和消息传递操作。

（1）图卷积操作（Graph Convolution）：图卷积操作是图神经网络中的核心操作，用于在图结构数据上进行特征提取和传播。下面是图神经网络的计算公式：

$$H^{(l+1)}=\sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)})\,,$$

其中，$H^{(l)}$ 表示第 l 层的节点特征矩阵，$\tilde{A}$ 表示邻接矩阵加上自连接的修正邻接矩阵，$\tilde{D}$ 表示度矩阵加上自连接的修正度矩阵，$W^{(l)}$ 表示第 l 层的权重矩阵,表示激活函数。

（2）消息传递操作(Message Passing):消息传递操作用于在图结构数据上进行节点之间的信息交互和传递。其计算公式如下:

$$\begin{aligned} m_{ij}^{(l+1)} &= \mathrm{ReLU}(\sum_{k\in N(i)} MLP^{(l)}(h_i^{(l)},h_j^{(l)},x_i,x_j)) \\ h_i^{(l+1)} &= GRU(h_i^{(l)},AGGREGATE(\{m_{ij}^{(l+1)}\mid j\in N(i)\})) \end{aligned},$$

其中 $m_{ij}^{(l+1)}$ 主元节占 i 和节占 j 少间的消息传递结果，$h_i^{(l)}$ 表示节点 i 在第 l 层的隐藏状态，x_i 表示节点 i 的输入特征，$MLP^{(l)}$ 表示多层感知机，AGGREGATE 表示聚合函数，GRU 表示门控循环单元。

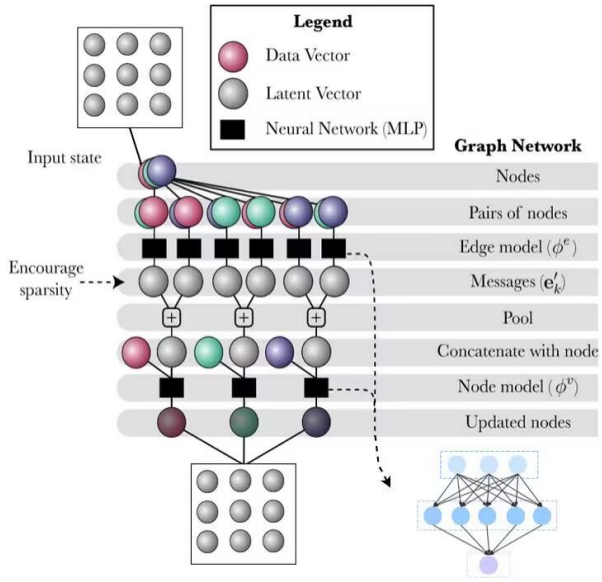


图 7.2 神经网络结构示意图

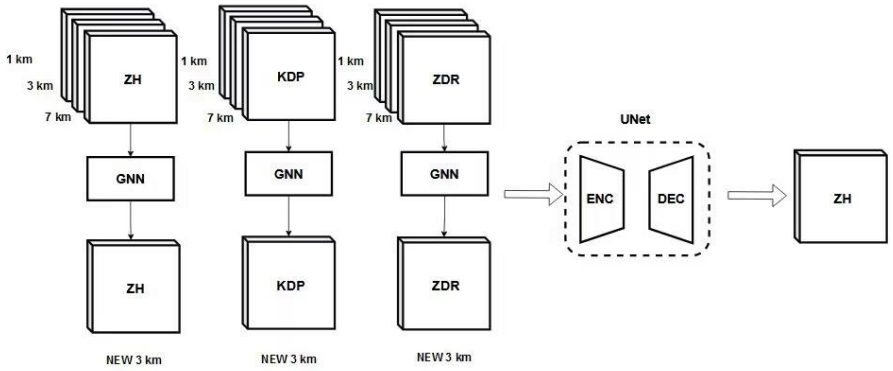


图 7.3 融合数据传输图

对于 1km,3km,7km 的等高线数据，首先我们认为不同等高线数据之间存在关联，并且不同的等高线数据可能含有不同的潜在降水特征。区别于单纯的在通道维度继续添加其他等高线数据，我们运用不同等高线的数据构建了图结构数据来使用图学习替代。其中每一个数据为（3，256，256），最终构建成的图结构数据为（256*256，3），其含义为有 256*256 个位置的数据点，每个数据点包含了 1km，3km，7km 的 3 个特征。

同时我们假设不同位置之间的存在着潜在联系，通过图神经网络来进行不同位置之间的交互实现消息传递来更新图的信息。我们规定了三层的消息传递网络，在第一层某个位置只与自己相邻的位置进行消息传递，随着消息传递层数的增加，当前位置考虑的特征信息范围会更加广阔。

网络输入（256*256，3）的三等高线微物理特征信息，输出（256*256，1）的单等高线微物理特征信息作为新的 3km 处的特征。并没有使用单纯的多特征于通道拼接的方式，而是使用了图神经网络融合多等高线数据，得到初步的训练数据，之后再通过第二题中改进的 UNet 系列网络进行临近预报。下图（图 7.4）为网络结构整体示意图。

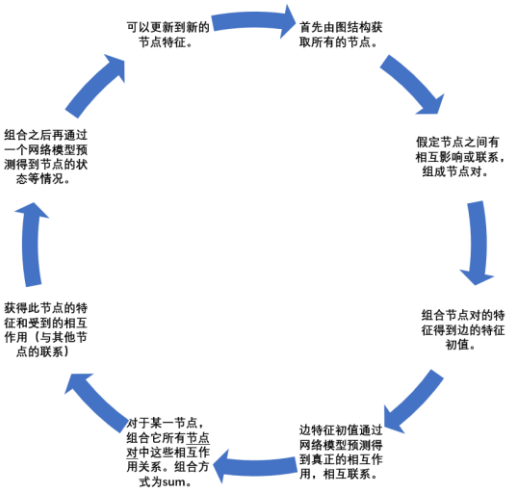


图 7.4 神经网络消息传递与训练机制

7.3 模型求解与检验

将 GNNUNet 网络和 FURNet 网络的损失结果进行对比（如图 4.7），右侧局部放大图显示了更详细的结果（代码见附录 12）。在训练集上，FURNet 模型的损失值用红色表示，GNNUNet 网络的损失值用浅蓝色曲线表示。两者整体上随着训练步骤的进行而逐渐减小。初始时 FURNet 的损失结果几乎是 GNNUNet 的 1.5 倍，从第 20 个 epoch 之后，FURNet 损失结果接近于 GNNUNet。在测试集上，损失结果 FURNet 模型的损失值用蓝色表示，GNNUNet 网络的损失值用黄色曲线表示，总体上呈下降趋势，但从图中可以看出，GNNUNet 模型损失值约为 FURNet 的损失的 1.06。

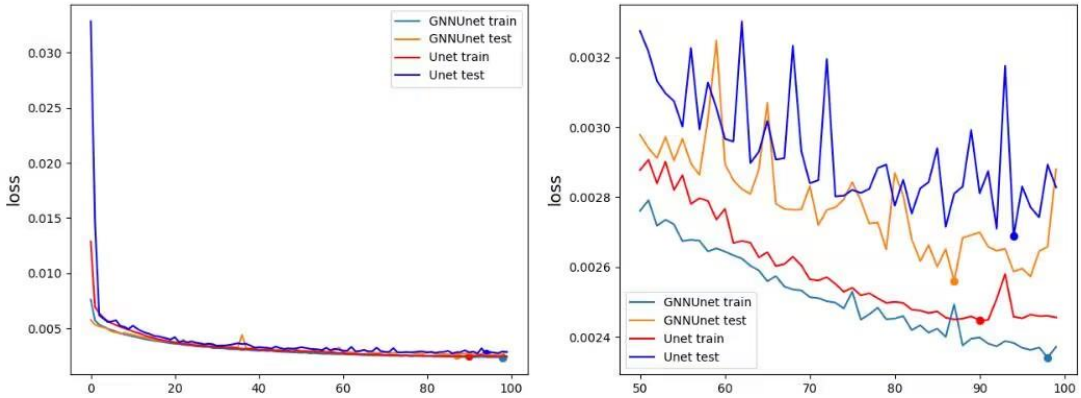


图 7.5 GNNUNet 网络和 FURNet 网络的损失对比图

在实际效果预测中，使用两个网络模型在测试集上的输出结果绘制了微物理特征信息的临近预报预测值的热力图，将预测出的结果进行可视化分析，如图 4.8 所示，给出了测试数据集中的一个代表性案例来说明建立的两个模型如何进行临近预报。图 4.8 的对流数据 ZH_True、ZH_UNet、分别表示一小时内雷达观测的 ZH_True，FURNet 模型预测。为简洁起见，我们每两帧显示雷达图。从真实序列 ZH_True 可以看出，对流系统早期演化阶段降水较大。从预测序列 ZH_UNet 可以看出，该模型不能较好地预测地区的对流形成。而 ZH_GNN_UNet 预测序列能准确预报风暴的形成，在空间范围和强度分布上对地区风暴

的预报也更为准确。

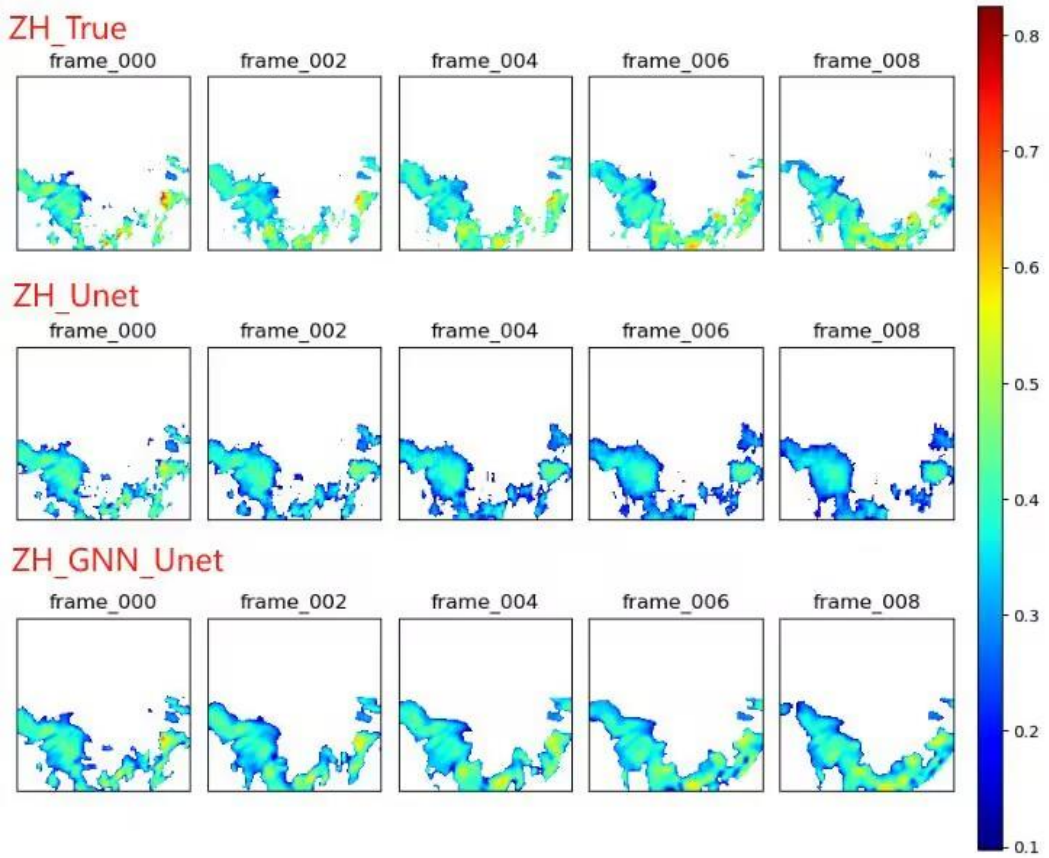


图 7.8 真实 Z_H 与模型预测的对比图

由表 7.1 可以看出，ConvLSTM 的相对误差、均方根误差都比问题一中的结果更接近 0，而比率偏差、皮尔逊相关系数则更接近于 1。此外，FURNet+CBAM、UNet-FURNet 和 3D-UNet 的结果也都优于问题二中的结果，所以融合后的模型比任何单一模型更加符合实际。并且新建立的 GNN+UNet 模型应用到融合后的数据更有优势。

表 7.1 融合模型的评价指标对比

误差 / 方法	R_E	R_{MSE}	B_{IAS}	C_C
ConvLSTM	0.8001	0.0822	0.6047	0.5214
UNet-FURNet	0.4059	0.0561	0.8414	0.8569
FURNet+CBAM	0.3983	0.0548	0.8603	0.8656
3D-UNet	0.3720	0.0538	0.7900	0.8645
GNN+UNet	0.353	0.041	0.735	0.881

八、模型评价与改进

8.1 模型的优点

（1）FURNet 采用了 U-Net 结构，具有编码器和解码器的对称结构，适用于处理多尺度的雷达观测数据。同时，加入了 SE 注意力机制和 ConvLSTM，增强了模型对关键信息和时空特征的敏感性。

（2）CBAM 模块引入了通道和空间注意力机制，提升模型对重要特征的关注能力。3D-FURNet 结合了 FURNet 和 ConvLSTM，适用于处理时序数据和空间特征的联合建模。

（3）本文提出的遗传算法中的符号回归算法可以根据现有数据自动学习各参数之间的映射关系，提高模型的拟合能力和预测准确性。

（4）利用图神经网络综合考虑地图不同位置之间的相关性，将地图位置信息作为节点特征，通过多源数据融合提高对强对流天气的预测准确性。这样的方法可以更好地利用地图数据的特点，提高模型的性能和预测能力。

8.2 模型的不足

（1）ConvLSTM 的复杂结构需要大量计算资源和时间进行训练和推理，且对处理不同尺度特征信息相对较弱。FURNet 的计算复杂度也较高，且通道注意力机制仅在解码器的最底层使用，可能导致预测结果区域平均化现象。对于时间序列特征，仅拼接到通道维度，缺乏对数据集时序性的考虑。

（2）引入 CBAM 模块会增加计算复杂度，可能导致过拟合风险和丢失细节信息。3DFURNet 构建了 3D 数据集输入模式，但在降采样和上采样时需要特殊设计卷积核大小和步长，且高度维度与长、宽不匹配时需要额外操作来保持训练稳定。

（3）问题三和问题四中，CSU-HDIRO 模型和符号回归拟合数据集仅考虑了特定地区的数据，导致结果只适用于该特定区域。图神经网络在处理大规模地图数据时计算复杂度高，且性能受限于输入数据质量和可靠性，需要额外设计节点之间的边。

8.3 模型的改进

（1）对多个地区进行空间质量控制，选取一些具有代表性的区域进行基于 CSU-HDIRO 模型和符号回归的参数拟合，使结果更加符合现实情况。

（2）考虑多源数据融合、集成学习联合投票模式来进一步提高模型的准确度。

（3）选用更优模型，减少模型复杂度，降低运行内存。

参考文献

[1] Huang, H., and Coauthors, 2018: Quantitative Precipitation Estimation with Operational Polarimetric Radar Measurements in Southern China: A Differential Phase–Based Variational Approach. J. Atmos. Oceanic Technol., 35, 1253–1271.

[2] Chen, G., Zhao, K., Zhang, G., Huang, H., Liu, S., Wen, L., ... & Zhu, W. (2017). Improving polarimetric C-band radar rainfall estimation with two-dimensional video disdrometer observations in Eastern China. Journal of Hydrometeorology, 18(5), 1375-1391.

[3] Zhang, Y., Long, M., Chen, K., Xing, L., Jin, R., Jordan, M. I., & Wang, J. (2023). Skilful nowcasting of extreme precipitation with NowcastNet. Nature, 619, 526–532.

[4] 皇甫江,胡志群,郑佳锋等.利用深度学习开展偏振雷达定量降水估测研究[J].气象学报,2022,80(04):565-577.

[5] 张永华. 基于深度学习的华南登陆台风偏振雷达定量降水估测研究[D].南京信息工程大学,2022.DOI:10.27248/d.cnki.gnjqc.2022.000280.

[6] 赵畅,魏鸣,刘红亚等.雷达定量估测降水量中双偏振参量的算法[J].科学技术与工程,2019,19(18):40-46.

[7] Lei W,Yuanchang D,Chenghong Z, et al. Extreme and severe convective weather disasters: A dual-polarization radar nowcasting method based on physical constraints and a deep neural network model[J]. Atmospheric Research,2023,289.

[8] 刘红亚,杨引明,张晶等.一次冰雹天气的 WSR-88D 双偏振雷达特征分析[J].气象与环境科学,2020,43(02):1-10.

[9] 汪舵, 刘黎平, 吴翀. 2017. 基于相态识别的 S 波段双线偏振雷达最优化定量降水估测方法研究 [J]. 气象, 43(9): 1041−1051.

[10] 吕健,刘圣楠,王俊人等.一次秋季冰雹过程的环境场和双偏振雷达特征分析[J].沙漠与绿洲气象,2023,17(04):53-60.

[11] 曹伟华,南刚强,陈明轩等.基于深度学习的京津冀地区精细尺度降水临近预报研究[J].气象学报,2022,80(04):546-564.

[12] 赵玮, 基于深度学习方法的多模式集成降水预报客观算法研究. 北京市,北京市气象台,2022-06-01.

[13] 杨绚,代刊,朱跃建.深度学习技术在智能网格天气预报中的应用进展与挑战[J].气象学报,2022,80(05):649-667.

[14] Pan, X., Lu, Y., Zhao, K., Huang, H., Wang, M., & Chen, H. (2021). Improving Nowcasting of Convective Development by Incorporating Polarimetric Radar Variables Into a Deep-Learning Model. Geophysical Research Letters, 48(21), e2021GL095302.

附录

附录 1	数据拼接读取与预处理

```
import os
data_dir_dBZ="/home /vscode/NJU_CPOL_update2308/dBZ/3.0km"
data_dir_KDP="/home/vscode/NJU_CPOL_update2308/KDP/3.0km"
data_dir_ZDR="/home /vscode/NJU_CPOL_update2308/ZDR/3.0km"
save_dir = "/home/vscode/data/"
norm_param = {
                'dBZ': [0, 65],
                'ZDR': [-1, 5],
                'KDP': [-1, 6],}
mmin_dBZ, mmax_dBZ = norm_param["dBZ"]
mmin_KDP, mmax_KDP = norm_param["KDP"]
mmin_ZDR, mmax_ZDR = norm_param["ZDR"]

# 设置输入和输出的帧数
input_frames = 10
output_frames = 10

folder_names_dBZ = sorted(os.listdir(data_dir_dBZ), key=lambda x: int(x.split('_')[2]))
folder_names_KDP = sorted(os.listdir(data_dir_KDP), key=lambda x: int(x.split('_')[2]))
folder_names_ZDR = sorted(os.listdir(data_dir_ZDR), key=lambda x: int(x.split('_')[2]))

numberk=0
# 遍历所有文件夹
for folder_name_dBZ,folder_name_KDP,folder_name_ZDR in zip(folder_names_dBZ,folder_names_KDP,folder_names_ZDR):
    print("----",folder_name_dBZ,folder_name_KDP,folder_name_ZDR)
    folder_path_dBZ = os.path.join(data_dir_dBZ, folder_name_dBZ)
#'/home/chuyangkkk/vscode/shuxue/NJU_CPOL_update2308/dBZ/3.0km/data_dir_254'
    folder_path_KDP = os.path.join(data_dir_KDP, folder_name_KDP)
    folder_path_ZDR = os.path.join(data_dir_ZDR, folder_name_ZDR)
    if not os.path.isdir(folder_path_dBZ):
        continue
    # 获取文件夹中的所有 npy 文件路径
    npy_files_dBZ = sorted([file for file in os.listdir(folder_path_dBZ) if file.endswith('.npy')], key=lambda x: int(x.split('_')[1].split('.')[0]))
    npy_files_KDP = sorted([file for file in os.listdir(folder_path_KDP) if file.endswith('.npy')], key=lambda x: int(x.split('_')[1].split('.')[0]))
    npy_files_ZDR= sorted([file for file in os.listdir(folder_path_ZDR) if file.endswith('.npy')], key=lambda x: int(x.split('_')[1].split('.')[0]))
```

```
# 读取每个降雨数据的多帧测量数据
frames_dBZ = []  #npy 读取后的保存位置
for file_name in npy_files_dBZ:
    file_path = os.path.join(folder_path_dBZ, file_name)
    frame_data = np.load(file_path)
    frames_dBZ.append(frame_data)
frames_KDP = []  #npy 读取后的保存位置
for file_name in npy_files_KDP:
    file_path = os.path.join(folder_path_KDP, file_name)
    frame_data = np.load(file_path)
    frames_KDP.append(frame_data)
frames_ZDR = []  #npy 读取后的保存位置
for file_name in npy_files_ZDR:
    file_path = os.path.join(folder_path_ZDR, file_name)
    frame_data = np.load(file_path)
    frames_ZDR.append(frame_data)
assert (len(frames_dBZ)==len(frames_KDP)) and (len(frames_dBZ)==len(frames_ZDR))
# 确定可以用于预测的帧数范围
num_frames = len(frames_dBZ)
input_range = range(num_frames - input_frames - output_frames + 1)
# 划分每个降雨数据的样本
for i in input_range:
    input_samples_dBZ = np.array(frames_dBZ[i:i+input_frames])
    output_samples_dBZ = np.array(frames_dBZ[i+input_frames:i+input_frames+output_frames])
    input_samples_KDP = np.array(frames_KDP[i:i+input_frames])
    output_samples_KDP = np.array(frames_KDP[i+input_frames:i+input_frames+output_frames])
    input_samples_ZDR = np.array(frames_ZDR[i:i+input_frames])
    output_samples_ZDR = np.array(frames_ZDR[i+input_frames:i+input_frames+output_frames])
    # #对矩阵中的 nan 数据进行双线性插值
    # #无 nan 数据
    # 获取原始矩阵的行数和列数
    #rows, cols = data.shape
    # 获取包含 nan 值的索引
    # nan_indices = np.argwhere(np.isnan(data))
    # 创建一个插值函数
    #f = interp2d(range(cols), range(rows), data, kind='linear')
    #是否是非降水数据
    # 设置阈值
    Ithreshold = 35
    Athreshold = 256
    # 计算每个样本的 Aevaluation
    Aevaluation = np.sum(np.array(output_samples_dBZ) > Ithreshold, axis=(1, 2))
    # 获取评估序列的最大值
```

```
            max_Aevaluation = np.max(Aevaluation)
            # 判断样本是否为非降水样本
            if max_Aevaluation < Athreshold:
                # print("该样本为非降水样本，将被剔除")
                pass
            if
Data_Quality_Control(input_samples_dBZ ,output_samples_dBZ ,input_samples_KDP ,output_samples_KDP
 ,input_samples_ZDR,output_samples_ZDR):
                pass
            else:
                input_samples_dBZ = (input_samples_dBZ - mmin_dBZ) / (mmax_dBZ - mmin_dBZ)
                output_samples_dBZ = (output_samples_dBZ - mmin_dBZ) / (mmax_dBZ - mmin_dBZ)
                input_samples_KDP = (input_samples_KDP - mmin_KDP) / (mmax_KDP - mmin_KDP)
                output_samples_KDP = (output_samples_KDP - mmin_KDP) / (mmax_KDP - mmin_KDP)
                input_samples_ZDR = (input_samples_ZDR - mmin_ZDR) / (mmax_ZDR - mmin_ZDR)
                output_samples_ZDR = (output_samples_ZDR - mmin_ZDR) / (mmax_ZDR - mmin_ZDR)

                endinput_samples = np.concatenate((np.array(input_samples_dBZ),
np.array(input_samples_KDP),np.array(input_samples_ZDR)), axis=0)
                endoutput_samples = output_samples_dBZ
                np.save(save_dir+'input/input_{}.npy'.format(numberk), endinput_samples)
                np.save(save_dir+'output/output_{}.npy'.format(numberk), endoutput_samples)
                numberk=numberk+1
```

附录 2	UNet- FURENet 网络结构

```
class Residual_Convolution(nn.Module):
    def __init__(self, input_channels, out_channels, strides=1):
        super().__init__()
        self.conv1 = nn.Conv2d(input_channels, out_channels, kernel_size=3, stride=strides, padding=1)
        self.act1 = nn.ReLU()
        self.maxpool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1)
        self.act2 = nn.LeakyReLU()
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.conv3 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1)

    def forward(self, X):
        # 下采样卷积
        X = self.conv1(X)
        X = self.maxpool(X)
        X = self.act1(X)
        # 进入 resnet
        Y = self.conv2(X)
        Y = self.act2(Y)
        Y = self.bn2(Y)
        Y = self.conv3(Y)
        # 跳跃连接
        Y = torch.add(Y, X)
        #     Y=self.cbam(Y)
        return Y

class Residual_Convolution_up(nn.Module):
    def __init__(self, input_channels, out_channels, strides=1):
        super(Residual_Convolution_up,self).__init__()
        self.conv1 = nn.Conv2d(input_channels, out_channels, kernel_size=3, stride=1, padding=1)
        self.UpConv = nn.ConvTranspose2d(out_channels, out_channels, kernel_size=2, stride=2)
        self.act1 = nn.ReLU()
        #self.maxpool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1)
        self.act2 = nn.LeakyReLU()
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.conv3 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1)

    def forward(self, X):
        # 上采样卷积
        X = self.conv1(X)
```

```
            X = self.UpConv(X)
            X = self.act1(X)
            # 进入 resnet
            Y = self.conv2(X)
            Y = self.act2(Y)
            Y = self.bn2(Y)
            Y = self.conv3(Y)
            # 跳跃连接 sel
            Y = torch.add(Y, X)
            #     Y=self.cbam(Y)
            return Y

class SEBlock(nn.Module):
    def __init__(self, in_channels, reduction_ratio=16):
        super(SEBlock, self).__init__()

        self.squeeze = nn.AdaptiveAvgPool2d(1)
        self.excitation = nn.Sequential(
            nn.Linear(in_channels, in_channels // reduction_ratio),
            nn.ReLU(inplace=True),
            nn.Linear(in_channels // reduction_ratio, in_channels),
            nn.Sigmoid()
        )

    def forward(self, x):
        batch_size, channels, height, width = x.size()
        # Squeeze
        squeeze_output = self.squeeze(x).view(batch_size, channels)
        # Excitation
        excitation_output = self.excitation(squeeze_output).view(batch_size, channels, 1, 1)
        # Scale
        scale_output = x * excitation_output.expand_as(x)
        return scale_output

class UNet(nn.Module):
    def __init__(self):
        super(UNet, self).__init__()

        self.resnet1_dBZ = Residual_Convolution(input_channels=10, out_channels=64)
        self.resnet2_dBZ = Residual_Convolution(input_channels=64, out_channels=128)
        self.resnet3_dBZ = Residual_Convolution(input_channels=128, out_channels=256)
        self.resnet4_dBZ = Residual_Convolution(input_channels=256, out_channels=512)
        self.resnet5_dBZ = Residual_Convolution(input_channels=512, out_channels=512)
```

```
        self.resnet6_dBZ = Residual_Convolution(input_channels=512, out_channels=512)
        self.resnet7_dBZ = Residual_Convolution(input_channels=512, out_channels=512)

        self.resnet1_KDP = Residual_Convolution(input_channels=10, out_channels=64)
        self.resnet2_KDP = Residual_Convolution(input_channels=64, out_channels=128)
        self.resnet3_KDP = Residual_Convolution(input_channels=128, out_channels=256)
        self.resnet4_KDP = Residual_Convolution(input_channels=256, out_channels=512)
        self.resnet5_KDP = Residual_Convolution(input_channels=512, out_channels=512)
        self.resnet6_KDP = Residual_Convolution(input_channels=512, out_channels=512)
        self.resnet7_KDP = Residual_Convolution(input_channels=512, out_channels=512)

        self.resnet1_ZDR = Residual_Convolution(input_channels=10, out_channels=64)
        self.resnet2_ZDR = Residual_Convolution(input_channels=64, out_channels=128)
        self.resnet3_ZDR = Residual_Convolution(input_channels=128, out_channels=256)
        self.resnet4_ZDR = Residual_Convolution(input_channels=256, out_channels=512)
        self.resnet5_ZDR = Residual_Convolution(input_channels=512, out_channels=512)
        self.resnet6_ZDR = Residual_Convolution(input_channels=512, out_channels=512)
        self.resnet7_ZDR = Residual_Convolution(input_channels=512, out_channels=512)

        self.allpool = nn.AdaptiveMaxPool2d((1, 1))
        self.SEBlock = SEBlock(512*3)

        self.resnet7_up = Residual_Convolution_up(input_channels=512*3, out_channels=512)
        self.resnet6_up = Residual_Convolution_up(input_channels=512*4, out_channels=512)
        self.resnet5_up = Residual_Convolution_up(input_channels=512*4, out_channels=512)
        self.resnet4_up = Residual_Convolution_up(input_channels=512*4, out_channels=256)
        self.resnet3_up = Residual_Convolution_up(input_channels=256*4, out_channels=128)
        self.resnet2_up = Residual_Convolution_up(input_channels=128*4, out_channels=64)
        self.resnet1_up = Residual_Convolution_up(input_channels=64*4, out_channels=10)

    def forward(self, x):

        dBZ = x[:,:10]
        KDP = x[:,10:20]
        ZDR = x[:,20:30]

        dBZ1 = self.resnet1_dBZ(dBZ)
        dBZ2 = self.resnet2_dBZ(dBZ1)
        dBZ3 = self.resnet3_dBZ(dBZ2)
        dBZ4 = self.resnet4_dBZ(dBZ3)
        dBZ5 = self.resnet5_dBZ(dBZ4)
        dBZ6 = self.resnet6_dBZ(dBZ5)
        dBZ7 = self.resnet7_dBZ(dBZ6
```

```
KDP1 = self.resnet1_KDP(KDP)
KDP2 = self.resnet2_KDP(KDP1)
KDP3 = self.resnet3_KDP(KDP2)
KDP4 = self.resnet4_KDP(KDP3)
KDP5 = self.resnet5_KDP(KDP4)
KDP6 = self.resnet6_KDP(KDP5)
KDP7 = self.resnet7_KDP(KDP6)

ZDR1 = self.resnet1_ZDR(ZDR)
ZDR2 = self.resnet2_ZDR(ZDR1)
ZDR3 = self.resnet3_ZDR(ZDR2)
ZDR4 = self.resnet4_ZDR(ZDR3)
ZDR5 = self.resnet5_ZDR(ZDR4)
ZDR6 = self.resnet6_ZDR(ZDR5)
ZDR7 = self.resnet7_ZDR(ZDR6)

dBZ_KDP_ZDR7 = torch.cat((dBZ7, KDP7, ZDR7), dim=1)
dBZ_KDP_ZDR7 = self.SEBlock(dBZ_KDP_ZDR7)

dBZ_KDP_ZDR6_1 = self.resnet7_up(dBZ_KDP_ZDR7) #512*3-512

dBZ_KDP_ZDR6 = torch.cat((dBZ6, KDP6, ZDR6,dBZ_KDP_ZDR6_1), dim=1)
dBZ_KDP_ZDR5_1 = self.resnet6_up(dBZ_KDP_ZDR6)#512+512*3-512

dBZ_KDP_ZDR5 = torch.cat((dBZ5, KDP5, ZDR5,dBZ_KDP_ZDR5_1), dim=1)
dBZ_KDP_ZDR4_1 = self.resnet5_up(dBZ_KDP_ZDR5)#512+512*3-512

dBZ_KDP_ZDR4 = torch.cat((dBZ4, KDP4, ZDR4,dBZ_KDP_ZDR4_1), dim=1)
dBZ_KDP_ZDR3_1 = self.resnet4_up(dBZ_KDP_ZDR4)#512+512*3-512

dBZ_KDP_ZDR3 = torch.cat((dBZ3, KDP3, ZDR3,dBZ_KDP_ZDR3_1), dim=1)
dBZ_KDP_ZDR2_1 = self.resnet3_up(dBZ_KDP_ZDR3)#256+256*3-128
#print(dBZ_KDP_ZDR3.shape,dBZ_KDP_ZDR2_1.shape)

dBZ_KDP_ZDR2 = torch.cat((dBZ2, KDP2, ZDR2,dBZ_KDP_ZDR2_1), dim=1)
dBZ_KDP_ZDR1_1 = self.resnet2_up(dBZ_KDP_ZDR2)#128+128*3-64

dBZ_KDP_ZDR1 = torch.cat((dBZ1, KDP1, ZDR1,dBZ_KDP_ZDR1_1), dim=1)
dBZ = self.resnet1_up(dBZ_KDP_ZDR1)#64+64*3-10
return dBZ
```

附录 3	ConvLSTM 网络结构

```
class ConvLSTMCell(nn.Module):
    def __init__(self, input_dim, hidden_dim, kernel_size, bias):
        super(ConvLSTMCell, self).__init__()
        self.input_dim  = input_dim
        self.hidden_dim = hidden_dim
        self.kernel_size = kernel_size
        self.bias = bias
        self.conv = nn.Conv2d(in_channels=self.input_dim + self.hidden_dim,
                              out_channels=4 * self.hidden_dim,
                              kernel_size=self.kernel_size,
                              padding=self.kernel_size // 2,
                              bias=self.bias)

    def forward(self, input_tensor, cur_state):
        h_cur, c_cur = cur_state
        combined = torch.cat([input_tensor, h_cur], dim=1)
        combined_conv = self.conv(combined)
        cc_i, cc_f, cc_o, cc_g = torch.split(combined_conv, self.hidden_dim, dim=1)
        i = torch.sigmoid(cc_i)
        f = torch.sigmoid(cc_f)
        o = torch.sigmoid(cc_o)
        g = torch.tanh(cc_g)
        c_next = f * c_cur + i * g
        h_next = o * torch.tanh(c_next)
        return h_next, c_next

# 定义端到端的 ConvLSTM 模型
class ConvLSTM(nn.Module):
    def __init__(self, input_dim, hidden_dim, kernel_size, bias):
        super(ConvLSTM, self).__init__()
        self.hidden_dim = hidden_dim  # 保存为类属性
        self.encoder = ConvLSTMCell(input_dim, hidden_dim, kernel_size, bias)
        self.decoder = ConvLSTMCell(hidden_dim, hidden_dim, kernel_size, bias)
        self.final_conv = nn.Conv2d(64, 1, kernel_size=1)

    def forward(self, input_tensor, hidden_state=None):
        # 输入形状：(b, t, c, h, w)
        b, t, c, h, w = input_tensor.size()

        # 初始化隐藏状态
        if hidden_state is None:
            hidden_state = (torch.zeros(b, self.hidden_dim, h, w).to(input_tensor.device),
```

```
                                torch.zeros(b, self.hidden_dim, h, w).to(input_tensor.device))

        # 编码阶段（Encoder）
        encoder_states = []
        for i in range(t):
            hidden_state = self.encoder(input_tensor[:, i, :, :], hidden_state)
            encoder_states.append(hidden_state[0])

        # 解码阶段（Decoder 或 Predictor）
        decoder_states = []
        decoder_input = encoder_states[-1]
        for i in range(10):  # 假设预测未来 10 帧
            hidden_state = self.decoder(decoder_input, hidden_state)
            decoder_states.append(hidden_state[0])
            decoder_input = hidden_state[0]
        decoder_states = [self.final_conv(state) for state in decoder_states]

        return torch.stack(decoder_states, dim=1)

ConvLSTM(input_dim=3, hidden_dim=64, kernel_size=3, bias=True)
```

附录 4	问题 1 训练代码

ConvLSTM

```
def train(self, epoch, optimizer,sched, batch_size, device, train_loader, myloss,threshold):
    train_loss = 0    # 损失
    train_number = 0
    # 模型变成训练模式
    self.train()
    for feature, label in train_loader:
        print("train_number:",train_number)
        train_number+=1
        input_all = []
        for file in feature:
            input_data = np.load(os.path.join(path, file))
            input_all.append([input_data])
        input_all = np.concatenate(input_all, axis=0)

        label_all = []
        for file in label:
            label_data = np.load(os.path.join(path2, file))
            label_all.append([label_data])
        label_all = np.concatenate(label_all, axis=0)

        feature = torch.from_numpy(input_all).float()
        feature = feature.unsqueeze(2)
        label = torch.from_numpy(label_all).float()
        label = label.unsqueeze(2)

        feature1 = feature[:,:10]
        feature2 = feature[:,10:20]
        feature3 = feature[:,20:]
        feature = torch.cat((feature1,feature2,feature3),axis=2)
        feature = feature.to(device)
        label = label.to(device)
        optimizer.zero_grad()
        outputs = self(feature)
        losst=myloss(outputs, label)
        losst.backward()
        optimizer.step()
        sched.step()
        train_loss += losst.item()
    avg_loss = train_loss / len(train_loader.dataset)
```

UNet：

```
def train(self, epoch, optimizer,sched, batch_size, device, train_loader, myloss,threshold):
    train_loss = 0    # 损失
    train_number = 0
    # 模型变成训练模式
    self.train()
    for feature, label in train_loader:
        print("train_number:",train_number)
        train_number+=1
        input_all = []
        for file in feature:
            input_data = np.load(os.path.join(path, file))
            input_all.append([input_data])
        input_all = np.concatenate(input_all, axis=0)

        label_all = []
        for file in label:
            label_data = np.load(os.path.join(path2, file))
            label_all.append([label_data])
        label_all = np.concatenate(label_all, axis=0)

        feature = torch.from_numpy(input_all).float()
        label = torch.from_numpy(label_all).float()

        feature = feature.to(device)
        label = label.to(device)

        optimizer.zero_grad()
        outputs = self(feature)
        losst=myloss(outputs, label)

        losst.backward()
        optimizer.step()
        sched.step()

        train_loss += losst.item()


    avg_loss = train_loss / len(train_loader.dataset)
```

附录 5	Cbam 网络结构

```
class ChannelAttention(nn.Module):
    def __init__(self, in_channels, reduction_ratio=16):
        super(ChannelAttention, self).__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.max_pool = nn.AdaptiveMaxPool2d(1)
        self.fc1 = nn.Conv2d(in_channels, in_channels // reduction_ratio, kernel_size=1, stride=1, padding=0)
        self.relu = nn.ReLU()
        self.fc2 = nn.Conv2d(in_channels // reduction_ratio, in_channels, kernel_size=1, stride=1, padding=0)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        avg_out = self.fc2(self.relu(self.fc1(self.avg_pool(x))))
        max_out = self.fc2(self.relu(self.fc1(self.max_pool(x))))
        out = avg_out + max_out
        return self.sigmoid(out)


class SpatialAttention(nn.Module):
    def __init__(self, kernel_size=7):
        super(SpatialAttention, self).__init__()
        padding = kernel_size // 2
        self.conv = nn.Conv2d(2, 1, kernel_size, padding=padding)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        avg_out = torch.mean(x, dim=1, keepdim=True)
        max_out, _ = torch.max(x, dim=1, keepdim=True)
        out = torch.cat([avg_out, max_out], dim=1)
        out = self.conv(out)
        return self.sigmoid(out)


class CBAM(nn.Module):
    def __init__(self, in_channels, reduction_ratio=16, kernel_size=7):
        super(CBAM, self).__init__()
        if in_channels<reduction_ratio:
            reduction_ratio=2
        self.channel_attention = ChannelAttention(in_channels, reduction_ratio)
        self.spatial_attention = SpatialAttention(kernel_size)
```

```
    def forward(self, x):
        out = self.channel_attention(x) * x
        out = self.spatial_attention(out) * out
        return out

class Residual_Convolution(nn.Module):
    def __init__(self, input_channels, out_channels, strides=1):
        super().__init__()

        self.conv1 = nn.Conv2d(input_channels, out_channels, kernel_size=3, stride=strides, padding=1)
        self.act1 = nn.ReLU()
        self.maxpool = nn.MaxPool2d(kernel_size=2, stride=2)

        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1)
        self.act2 = nn.LeakyReLU()
        self.bn2 = nn.BatchNorm2d(out_channels)

        self.conv3 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1)

        self.cbam = CBAM(in_channels=out_channels)

    def forward(self, X):
        # 下采样卷积
        X = self.conv1(X)
        X = self.maxpool(X)
        X = self.act1(X)
        # 进入 resnet
        Y = self.conv2(X)
        Y = self.act2(Y)
        Y = self.bn2(Y)
        Y = self.conv3(Y)

        # 跳跃连接
        Y = torch.add(Y, X)
        Y = self.cbam(Y)
        return Y

class Residual_Convolution_up(nn.Module):
    def __init__(self, input_channels, out_channels, strides=1):
        super(Residual_Convolution_up,self).__init__()
        self.conv1 = nn.Conv2d(input_channels, out_channels, kernel_size=3, stride=1, padding=1)
        self.UpConv = nn.ConvTranspose2d(out_channels, out_channels, kernel_size=2, stride=2)
        self.act1 = nn.ReLU()
```

```
        #self.maxpool = nn.MaxPool2d(kernel_size=2, stride=2)

        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1)
        self.act2 = nn.LeakyReLU()
        self.bn2 = nn.BatchNorm2d(out_channels)

        self.conv3 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1)
        self.cbam = CBAM(in_channels=out_channels)

    def forward(self, X):
        # 上采样卷积
        X = self.conv1(X)
        X = self.UpConv(X)
        X = self.act1(X)

        # 进入 resnet
        Y = self.conv2(X)
        Y = self.act2(Y)
        Y = self.bn2(Y)
        Y = self.conv3(Y)

        # 跳跃连接 sel
        Y = torch.add(Y, X)
        Y = self.cbam(Y)

        return Y
```

附录 6	3D-UNet 网络结构

```
class Residual_Convolution(nn.Module):
    def __init__(self, input_channels, out_channels,kernel_size=(1,2,2), strides=(1,2,2), output_padding=0):
        super().__init__()

        self.conv1 = nn.Conv3d(input_channels, out_channels, kernel_size=3, stride=1, padding=1)
        self.act1 = nn.ReLU()
        self.maxpool = nn.MaxPool3d(kernel_size=kernel_size, stride=strides)

        self.conv2 = nn.Conv3d(out_channels, out_channels, kernel_size=3, stride=1, padding=1)
        self.act2 = nn.LeakyReLU()
        self.bn2 = nn.BatchNorm3d(out_channels)
        self.conv3 = nn.Conv3d(out_channels, out_channels, kernel_size=3, stride=1, padding=1)

    def forward(self, X):
        # 下采样卷积
        X = self.conv1(X)
        X = self.maxpool(X)
        X = self.act1(X)
        # 进入 resnet
        Y = self.conv2(X)
        Y = self.act2(Y)
        Y = self.bn2(Y)
        Y = self.conv3(Y)
        # 跳跃连接
        Y = torch.add(Y, X)
        #     Y=self.cbam(Y)
        return Y

class Residual_Convolution_up(nn.Module):
    def __init__(self, input_channels, out_channels,kernel_size=(1,2,2), strides=(1,2,2),output_padding = 0):
        super(Residual_Convolution_up,self).__init__()
        self.conv1 = nn.Conv3d(input_channels, out_channels, kernel_size=3, stride=1, padding=1)
        self.UpConv = nn.ConvTranspose3d(out_channels, out_channels, kernel_size=kernel_size,
stride=strides, output_padding=output_padding)
        self.act1 = nn.ReLU()
        #self.maxpool = nn.MaxPool3d(kernel_size=2, stride=2)

        self.conv2 = nn.Conv3d(out_channels, out_channels, kernel_size=3, stride=1, padding=1)
        self.act2 = nn.LeakyReLU()
        self.bn2 = nn.BatchNorm3d(out_channels)

        self.conv3 = nn.Conv3d(out_channels, out_channels, kernel_size=3, stride=1, padding=1)
```

```
    def forward(self, X):
        # 上采样卷积
        X = self.conv1(X)
        X = self.UpConv(X)
        X = self.act1(X)
        # 进入 resnet
        Y = self.conv2(X)
        Y = self.act2(Y)
        Y = self.bn2(Y)
        Y = self.conv3(Y)

        # 跳跃连接 sel
        Y = torch.add(Y, X)
        #     Y=self.cbam(Y)
        return Y

class SEBlock(nn.Module):
    def __init__(self, in_channels, reduction_ratio=16):
        super(SEBlock, self).__init__()
        self.avg_pool = nn.AdaptiveAvgPool3d(1)
        self.fc = nn.Sequential(
            nn.Linear(in_channels, in_channels // reduction_ratio),
            nn.ReLU(inplace=True),
            nn.Linear(in_channels // reduction_ratio, in_channels),
            nn.Sigmoid()
        )

    def forward(self, x):
        b, c, d, h, w = x.size()
        y = self.avg_pool(x).view(b, c)
        y = self.fc(y).view(b, c, 1, 1, 1)
        return x * y.expand_as(x)

class UNet(nn.Module):
    def __init__(self):
        super(UNet, self).__init__()

        self.resnet1 = Residual_Convolution(input_channels=3, out_channels=64)
        self.resnet2 = Residual_Convolution(input_channels=64, out_channels=128)
        self.resnet3 = Residual_Convolution(input_channels=128, out_channels=256)
        self.resnet4 = Residual_Convolution(input_channels=256, out_channels=512)
        self.resnet5 = Residual_Convolution(input_channels=512, out_channels=512)
```

```
        self.resnet6 = Residual_Convolution(input_channels=512, out_channels=512,kernel_size=(2,2,2), strides=(2,2,2))
        self.resnet7 = Residual_Convolution(input_channels=512, out_channels=512,kernel_size=(2,2,2), strides=(2,2,2))
        self.allpool = nn.AdaptiveMaxPool3d((1, 1))
        self.SEBlock = SEBlock(512)

        self.resnet7_up = Residual_Convolution_up(input_channels=512, out_channels=512,kernel_size=(2,2,2), strides=(2,2,2), output_padding=(1, 0, 0))
        self.resnet6_up = Residual_Convolution_up(input_channels=512*2, out_channels=512,kernel_size=(2,2,2), strides=(2,2,2))
        self.resnet5_up = Residual_Convolution_up(input_channels=512*2, out_channels=512)
        self.resnet4_up = Residual_Convolution_up(input_channels=512*2, out_channels=256)
        self.resnet3_up = Residual_Convolution_up(input_channels=256*2, out_channels=128)
        self.resnet2_up = Residual_Convolution_up(input_channels=128*2, out_channels=64)
        self.resnet1_up = Residual_Convolution_up(input_channels=64*2, out_channels=1)

    def forward(self, x):
        x1 = self.resnet1(x)
        x2 = self.resnet2(x1)
        x3 = self.resnet3(x2)
        x4 = self.resnet4(x3)
        x5 = self.resnet5(x4)
        x6 = self.resnet6(x5)
        x7 = self.resnet7(x6)

        y6_pre = self.resnet7_up(x7) #512*3-512
        y6 = torch.cat((x6,y6_pre), dim=1)
        y5_pre = self.resnet6_up(y6)#512+512*3-512

        y5 = torch.cat((x5,y5_pre), dim=1)
        y4_pre = self.resnet5_up(y5)#512+512*3-512

        y4 = torch.cat((x4,y4_pre), dim=1)
        y3_pre = self.resnet4_up(y4)#512+512*3-512
        y3 = torch.cat((x3,y3_pre), dim=1)
        y2_pre = self.resnet3_up(y3)#256+256*3-128
        y2 = torch.cat((x2,y2_pre), dim=1)
        y1_pre = self.resnet2_up(y2)#128+128*3-64
        y1 = torch.cat((x1,y1_pre), dim=1)
        end = self.resnet1_up(y1)#64+64*3-10
        return torch.squeeze(end, dim=1)
```

附录 7	3D-UNet 训练代码

```
def train(self, epoch, optimizer,sched, batch_size, device, train_loader, myloss,threshold):
    train_loss = 0    # 损失
    train_number = 0
    # 模型变成训练模式
    self.train()
    for feature, label in train_loader:
        print("train_number:",train_number)
        train_number+=1
        input_all = []
        for file in feature:
            input_data = np.load(os.path.join(path, file))
            input_all.append([input_data])
        input_all = np.concatenate(input_all, axis=0)

        label_all = []
        for file in label:
            label_data = np.load(os.path.join(path2, file))
            label_all.append([label_data])
        label_all = np.concatenate(label_all, axis=0)

        feature = torch.from_numpy(input_all).float()
        label = torch.from_numpy(label_all).float()

        dBZ = feature[:,:10]
        KDP = feature[:,10:20]
        ZDR = feature[:,20:30]
        feature = torch.zeros(dBZ.shape[0], 3, 10, 256, 256)
        for i in range(10):
            slice_tensor = torch.cat((dBZ[:, i:i+1, :, :], KDP[:, i:i+1, :, :], ZDR[:, i:i+1, :, :]), dim=1)
            feature[:, :, i, :, :] = slice_tensor
        feature = feature.to(device)
        label = label.to(device)
        optimizer.zero_grad()
        outputs = self(feature)
        losst=myloss(outputs, label)

        losst.backward()
        optimizer.step()
        sched.step()
        train_loss += losst.item()

    avg_loss = train_loss / len(train_loader.dataset)
```

附录 8	符号回归方法

```
import pysr
import sympy
import numpy as np
from matplotlib import pyplot as plt
from pysr import PySRRegressor
from sklearn.model_selection import train_test_split
import pandas as pd

def print_model_latex(model):
    #print(model)
    print(r"\begin{align*}")
    print("best:",r"\\")
    print("\phi^e~&=~")
    print(model.latex())
    print("&score:",np.max(model.equations_["score"]))
    print(r"\\")
    print(r"\\")
    print("all:",r"\\")

    for i in range(len(model.equations_)):
        print("\phi^e~&=~")
        print(model.latex(i))
        print("&score:",model.equations_.loc[i,"score"])
        print(r"\\")
    print("\end{align*}")
def plt_model(model):
    colors = []
    import random
    for i in range(len(model.equations_["complexity"])):
        colors.append('#%06X' % random.randint(0, 0xFFFFFF))
    plt.bar(model.equations_["complexity"],model.equations_["score"],edgecolor='black',color=colors)
    plt.xlabel("Complexity")
    plt.ylabel("Score = δ Accuracy / δ Complexity")

data=pd.read_csv("/home/chuyangkkk/vscode/shuxue/data_quest3_求和_横向预测.csv")
data[:3]
def SR_model(data, target_col=None):
    X = data[["dBZ"]]#,"ZDR"
    y = data["R"]
    model = PySRRegressor(
        model_selection="best",
        niterations=300,
```

```
            binary_operators=["*", "+", "-","^", ],
            complexity_of_operators={"+": 1, "*": 1, "-": 1,
                                                "^": 1,},
            # binary_operators=["*", "/", "+", "-","^", "div", "mult", "plus", "sub"],
            # unary_operators=[ "log", "neg", "abs"],
            # complexity_of_operators={"+": 1, "*": 1, "-": 1, "/": 1,"div":1,"mult":1,"plus":1,"sub":1,
            #                                     "^": 1, "exp":3, "log":3, "neg":3, "abs":3},
            population_size=20,
            loss="L1DistLoss()",

#equation_file="/home/chuyangkkk/code/carnmer2020/symbolic_deep_learning/pySR_data/idea/log",
            #tempdir="/home/chuyangkkk/code/carnmer2020/symbolic_deep_learning/pySR_data/idea/log/"
        )
        model.fit(X, y)
        return model

data2 = data.sample(n=5000, random_state=42).reset_index(drop=True) #[:500]#

models=[]

model = SR_model(data2)
models.append(model)
print_model_latex(models[0])
```

附录 9	第三题数据处理代码

```
data_dir_R="/home/vscode/shuxue/NJU_CPOL_kdpRain"
folder_names = sorted(os.listdir(data_dir_R), key=lambda x: int(x.split('_')[2]))
numberk=0
min_npy_data_r = []
max_npy_data_r = []
# 遍历所有文件夹
for folder_name in folder_names:
    print("----",folder_name)
    folder_path = os.path.join(data_dir_R, folder_name) #'/home/vscode/NJU_CPOL_update2308/dBZ/3.0km/data_dir_254'
    if not os.path.isdir(folder_path):
        continue
    # 获取文件夹中的所有 npy 文件路径
    npy_files = sorted([file for file in os.listdir(folder_path) if file.endswith('.npy')], key=lambda x: int(x.split('_')[1].split('.')[0]))
    # 读取每个降雨数据的多帧测量数据
    for file_name in npy_files:
        file_path = os.path.join(folder_path, file_name)
        frame_data = np.load(file_path)
        max_r = np.max(frame_data)
        min_r = np.min(frame_data)
        min_npy_data_r.append(min_r)
        max_npy_data_r.append(max_r)
np.min(min_npy_data_r),np.max(max_npy_data_r)
#(0.0    1225.7087)

# 设置输入和输出的帧数
input_frames = 10
output_frames = 10

folder_names_dBZ = sorted(os.listdir(data_dir_dBZ), key=lambda x: int(x.split('_')[2]))
folder_names_R = sorted(os.listdir(data_dir_R), key=lambda x: int(x.split('_')[2]))
folder_names_ZDR = sorted(os.listdir(data_dir_ZDR), key=lambda x: int(x.split('_')[2]))

data_quest3=[]



numberk=0
# 遍历所有文件夹
for folder_name_dBZ,folder_name_R,folder_name_ZDR in zip(folder_names_dBZ,folder_names_R,folder_names_ZDR):
```

```
    print("----",folder_name_dBZ,folder_name_R,folder_name_ZDR)
    folder_path_dBZ = os.path.join(data_dir_dBZ, folder_name_dBZ) #'/home/chuyangkkk/vscode/shuxue/NJU_CPOL_update2308/dBZ/3.0km/data_dir_254'
    folder_path_R = os.path.join(data_dir_R, folder_name_R)
    folder_path_ZDR = os.path.join(data_dir_ZDR, folder_name_ZDR)
    if not os.path.isdir(folder_path_dBZ):
        continue
    # 获取文件夹中的所有 npy 文件路径
    npy_files_dBZ = sorted([file for file in os.listdir(folder_path_dBZ) if file.endswith('.npy')], key=lambda x: int(x.split('_')[1].split('.')[0]))
    npy_files_R = sorted([file for file in os.listdir(folder_path_R) if file.endswith('.npy')], key=lambda x: int(x.split('_')[1].split('.')[0]))
    npy_files_ZDR= sorted([file for file in os.listdir(folder_path_ZDR) if file.endswith('.npy')], key=lambda x: int(x.split('_')[1].split('.')[0]))
    # 读取每个降雨数据的多帧测量数据
    frames_dBZ = []  #npy 读取后的保存位置
    for file_name in npy_files_dBZ:
        file_path = os.path.join(folder_path_dBZ, file_name)
        frame_data = np.load(file_path)
        frames_dBZ.append(frame_data)
    frames_R = []  #npy 读取后的保存位置
    for file_name in npy_files_R:
        file_path = os.path.join(folder_path_R, file_name)
        frame_data = np.load(file_path)
        frames_R.append(frame_data)
    frames_ZDR = []  #npy 读取后的保存位置
    for file_name in npy_files_ZDR:
        file_path = os.path.join(folder_path_ZDR, file_name)
        frame_data = np.load(file_path)
        frames_ZDR.append(frame_data)
    assert (len(frames_dBZ)==len(frames_R)) and (len(frames_dBZ)==len(frames_ZDR))
    # 确定可以用于预测的帧数范围
    num_frames = len(frames_dBZ)
    input_range = range(num_frames - input_frames - output_frames + 1)
    # 划分每个降雨数据的样本
    for i in input_range:
        input_samples_dBZ = np.array(frames_dBZ[i:i+input_frames])
        output_samples_dBZ = np.array(frames_dBZ[i+input_frames:i+input_frames+output_frames])
        input_samples_R = np.array(frames_R[i:i+input_frames])
        output_samples_R = np.array(frames_R[i+input_frames:i+input_frames+output_frames])
        input_samples_ZDR = np.array(frames_ZDR[i:i+input_frames])
        output_samples_ZDR = np.array(frames_ZDR[i+input_frames:i+input_frames+output_frames])
        #是否是非降水数据
```

```
            # 设置阈值
            Ithreshold = 35
            Athreshold = 256
            # 计算每个样本的 Aevaluation
            Aevaluation = np.sum(np.array(output_samples_dBZ) > Ithreshold, axis=(1, 2))
            # 获取评估序列的最大值
            max_Aevaluation = np.max(Aevaluation)
            # 判断样本是否为非降水样本
            if max_Aevaluation < Athreshold:
                # print("该样本为非降水样本，将被剔除")
                  pass
            else:

                  # input_samples_dBZ = (input_samples_dBZ - mmin_dBZ) / (mmax_dBZ - mmin_dBZ)
                  # output_samples_dBZ = (output_samples_dBZ - mmin_dBZ) / (mmax_dBZ - mmin_dBZ)
                  # input_samples_R = (input_samples_R - mmin_R) / (mmax_R - mmin_R)
                  # output_samples_R = (output_samples_R - mmin_R) / (mmax_R - mmin_R)
                  # input_samples_ZDR = (input_samples_ZDR - mmin_ZDR) / (mmax_ZDR - mmin_ZDR)
                  # output_samples_ZDR = (output_samples_ZDR - mmin_ZDR) / (mmax_ZDR - mmin_ZDR)

                  dBZ_sum = np.sum(input_samples_dBZ, axis=0).reshape(-1,1)
                  R_sum = np.sum(input_samples_R, axis=0).reshape(-1,1)
                  ZDR_sum    = np.sum(input_samples_ZDR, axis=0).reshape(-1,1)



                  # dBZ_col = np.concatenate((np.array(input_samples_dBZ), np.array(output_samples_dBZ)),
axis=0).reshape(-1,1)
                  #    R_col    =    np.concatenate((np.array(input_samples_R),    np.array(output_samples_R)),
axis=0).reshape(-1,1)
                  # ZDR_col = np.concatenate((np.array(input_samples_ZDR), np.array(output_samples_ZDR)),
axis=0).reshape(-1,1)
                  # 拼接三个数组
                  combined_data = np.concatenate((dBZ_sum, R_sum, ZDR_sum), axis=1)
                  # 随机抽取 20 行数据
                  random_rows                    =                    np.random.choice(combined_data.shape[0],
size=int(combined_data.shape[0]/200), replace=False)

                  # 提取抽取到的行数据
                  selected_data = combined_data[random_rows, :]
                  data_quest3.append(selected_data)
                  numberk+=1
```

```
import pandas as pd
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

data = pd.read_csv('/home/chuyangkkk/vscode/shuxue/data_quest3_非 0 非归一化 dzb 不为 0.csv')
random_data = data.sample(n=5000)
# x1 = data['ZDR']
x_data = data['dBZ']
y_data = data['R']

# ZH,ZDR 组合
# def linear_func(x, x1, a, b):
#       return a * x* np.power(10, b*x1)

#ZH 单个
def linear_func(x, x1, a, b):
    return a * np.power(x, b)

params, covariance = curve_fit(linear_func, x_data, y_data)
print(params,covariance)

random_data = data.sample(n=2000)
x_data1 = random_data['dBZ']
y_data1 = random_data['R']

plt.scatter(x_data1, y_data1, label='Data')  # 绘制数据点
plt.plot(y_data, linear_func(x_data, *params), 'r', label='R(ZH)')  # 绘制拟合曲线

plt.xlabel('ZH')
plt.ylabel('R')

 # 添加颜色条
plt.colorbar()
# 设置坐标轴范围
plt.xlim(10, 50)  # 根据需要设置 x 轴范围
plt.ylim(10, 50)  # 根据需要设置 y 轴范围
plt.xlabel('R')
plt.ylabel('R($Z_{H}$)')
plt.show()
```

附录 11	问题 4 图结构数据处理与图网络结构

```
def get_edge_indices(matrix):
    n = len(matrix)  # 行数
    m = len(matrix[0])  # 列数

    edge_indices = []

    for i in range(n):
        for j in range(m):
            # 上方节点
            if i - 1 >= 0:
                edge_indices.append((i, j, i - 1, j))
            # 下方节点
            if i + 1 < n:
                edge_indices.append((i, j, i + 1, j))
            # 左方节点
            if j - 1 >= 0:
                edge_indices.append((i, j, i, j - 1))
            # 右方节点
            if j + 1 < m:
                edge_indices.append((i, j, i, j + 1))

    return edge_indices
class GN(MessagePassing):
    def __init__(self, n_f, msg_dim, ndim, hidden=300, aggr='add'):
        super(GN, self).__init__(aggr=aggr)  # "Add" aggregation.
        self.msg_fnc = Seq(
            Lin(2*n_f, hidden),
            ReLU(),
            Lin(hidden, hidden),
            ReLU(),
            Lin(hidden, hidden),
            ReLU(),
            ##(Can turn on or off this layer:)
#             Lin(hidden, hidden),
#             ReLU(),
            Lin(hidden, msg_dim)
        )

        self.node_fnc = Seq(
            Lin(msg_dim+n_f, hidden),
            ReLU(),
            Lin(hidden, hidden),
```

```
                ReLU(),
                Lin(hidden, hidden),
                ReLU(),
#                   Lin(hidden, hidden),
#                   ReLU(),
                Lin(hidden, ndim)
            )

        #[docs]
        def forward(self, x, edge_index):
            #x is [n, n_f]
            x = x
            return self.propagate(edge_index, size=(x.size(0), x.size(0)), x=x)

        def message(self, x_i, x_j):
            # x_i has shape [n_e, n_f]; x_j has shape [n_e, n_f]
            tmp = torch.cat([x_i, x_j], dim=1)    # tmp has shape [E, 2 * in_channels]
            return self.msg_fnc(tmp)

        def update(self, aggr_out, x=None):
            # aggr_out has shape [n, msg_dim]

            tmp = torch.cat([x, aggr_out], dim=1)
            return self.node_fnc(tmp) #[n, nupdate]
class Residual_Convolution(nn.Module):
        def __init__(self, input_channels, out_channels,kernel_size=(1,2,2), strides=(1,2,2), output_padding=0):
            super().__init__()
            self.conv1 = nn.Conv3d(input_channels, out_channels, kernel_size=3, stride=1, padding=1)
            self.act1 = nn.ReLU()
            self.maxpool = nn.MaxPool3d(kernel_size=kernel_size, stride=strides)
            self.conv2 = nn.Conv3d(out_channels, out_channels, kernel_size=3, stride=1, padding=1)
            self.act2 = nn.LeakyReLU()
            self.bn2 = nn.BatchNorm3d(out_channels)
            self.conv3 = nn.Conv3d(out_channels, out_channels, kernel_size=3, stride=1, padding=1)
        def forward(self, X):
            # 下采样卷积
            X = self.conv1(X)
            X = self.maxpool(X)
            X = self.act1(X)
            Y = self.conv2(X)
            Y = self.act2(Y)
            Y = self.bn2(Y)
            Y = self.conv3(Y)
```

```
            Y = torch.add(Y, X)
            return Y

class Residual_Convolution_up(nn.Module):
    def __init__(self, input_channels, out_channels,kernel_size=(1,2,2), strides=(1,2,2),output_padding = 0):
            super(Residual_Convolution_up,self).__init__()
            self.conv1 = nn.Conv3d(input_channels, out_channels, kernel_size=3, stride=1, padding=1)
            self.UpConv = nn.ConvTranspose3d(out_channels, out_channels, kernel_size=kernel_size,
stride=strides, output_padding=output_padding)
            self.act1 = nn.ReLU()
            self.conv2 = nn.Conv3d(out_channels, out_channels, kernel_size=3, stride=1, padding=1)
            self.act2 = nn.LeakyReLU()
            self.bn2 = nn.BatchNorm3d(out_channels)
            self.conv3 = nn.Conv3d(out_channels, out_channels, kernel_size=3, stride=1, padding=1)
    def forward(self, X):
            X = self.conv1(X)
            X = self.UpConv(X)
            X = self.act1(X)
            Y = self.conv2(X)
            Y = self.act2(Y)
            Y = self.bn2(Y)
            Y = self.conv3(Y)
            Y = torch.add(Y, X)
            return Y

class SEBlock(nn.Module):
    def __init__(self, in_channels, reduction_ratio=16):
            super(SEBlock, self).__init__()
            self.avg_pool = nn.AdaptiveAvgPool3d(1)
            self.fc = nn.Sequential(
                nn.Linear(in_channels, in_channels // reduction_ratio),
                nn.ReLU(inplace=True),
                nn.Linear(in_channels // reduction_ratio, in_channels),
                nn.Sigmoid()
            )
    def forward(self, x):
            b, c, d, h, w = x.size()
            y = self.avg_pool(x).view(b, c)
            y = self.fc(y).view(b, c, 1, 1, 1)
            return x * y.expand_as(x)

class UNet(nn.Module):
    def __init__(self):
```

```
        super(UNet, self).__init__()
        self.resnet1 = Residual_Convolution(input_channels=3, out_channels=64)
        self.resnet2 = Residual_Convolution(input_channels=64, out_channels=128)
        self.resnet3 = Residual_Convolution(input_channels=128, out_channels=256)
        self.resnet4 = Residual_Convolution(input_channels=256, out_channels=512)
        self.resnet5 = Residual_Convolution(input_channels=512, out_channels=512)
        self.resnet6 = Residual_Convolution(input_channels=512, out_channels=512,kernel_size=(2,2,2), strides=(2,2,2))
        self.resnet7 = Residual_Convolution(input_channels=512, out_channels=512,kernel_size=(2,2,2), strides=(2,2,2))
        self.allpool = nn.AdaptiveMaxPool3d((1, 1))
        self.SEBlock = SEBlock(512)
        self.resnet7_up = Residual_Convolution_up(input_channels=512, out_channels=512,kernel_size=(2,2,2), strides=(2,2,2), output_padding=(1, 0, 0))
        self.resnet6_up = Residual_Convolution_up(input_channels=512*2, out_channels=512,kernel_size=(2,2,2), strides=(2,2,2))
        self.resnet5_up = Residual_Convolution_up(input_channels=512*2, out_channels=512)
        self.resnet4_up = Residual_Convolution_up(input_channels=512*2, out_channels=256)
        self.resnet3_up = Residual_Convolution_up(input_channels=256*2, out_channels=128)
        self.resnet2_up = Residual_Convolution_up(input_channels=128*2, out_channels=64)
        self.resnet1_up = Residual_Convolution_up(input_channels=64*2, out_channels=1)
    def forward(self, x):
        x1 = self.resnet1(x)
        x2 = self.resnet2(x1)
        x3 = self.resnet3(x2)
        x4 = self.resnet4(x3)
        x5 = self.resnet5(x4)
        x6 = self.resnet6(x5)
        x7 = self.resnet7(x6)
        y6_pre = self.resnet7_up(x7) #512*3-512
        y6 = torch.cat((x6,y6_pre), dim=1)
        y5_pre = self.resnet6_up(y6)#512+512*3-512
        y5 = torch.cat((x5,y5_pre), dim=1)
        y4_pre = self.resnet5_up(y5)#512+512*3-512
        y4 = torch.cat((x4,y4_pre), dim=1)
        y3_pre = self.resnet4_up(y4)#512+512*3-512
        y3 = torch.cat((x3,y3_pre), dim=1)
        y2_pre = self.resnet3_up(y3)#256+256*3-128
        y2 = torch.cat((x2,y2_pre), dim=1)
        y1_pre = self.resnet2_up(y2)#128+128*3-64
        y1 = torch.cat((x1,y1_pre), dim=1)
        end = self.resnet1_up(y1)#64+64*3-10
        return torch.squeeze(end, dim=1)
```

附录 12	图像绘制代码

```
data=pd.read_csv("")
data=data[:110]
data[:3]
fig,(ax1,ax2)=plt.subplots(nrows=1,ncols=2,figsize=(15,5))
fig.subplots_adjust(hspace=0.5, top=0.9, bottom=0.05)
ax1.plot(data["epoch"],data["train_loss"],label="train loss")
ax1.plot(data["epoch"],data["val_loss"],label="test loss")
ax1.scatter(data.loc[data["train_loss"].idxmin(),"epoch"],data.loc[data["train_loss"].idxmin(),"train_loss"],s=25)
ax1.scatter(data.loc[data["val_loss"].idxmin(),"epoch"],data.loc[data["val_loss"].idxmin(),"val_loss"],s=25)
ax1.set_xlabel("epoch")
ax1.set_ylabel("loss")
ax2.xaxis.label.set_fontsize(14)   # 设置 x 轴标签字体大小为 14
ax2.yaxis.label.set_fontsize(14)   # 设置 y 轴标签字体大小为 14
ax1.legend()
data2=data[20:]
ax2.plot(data2["epoch"],data2["train_loss"],label="train loss")
ax2.plot(data2["epoch"],data2["val_loss"],label="test loss")
ax2.scatter(data2.loc[data2["train_loss"].idxmin(),"epoch"],data2.loc[data2["train_loss"].idxmin(),"train_loss"],s=25)
ax2.scatter(data2.loc[data2["val_loss"].idxmin(),"epoch"],data2.loc[data2["val_loss"].idxmin(),"val_loss"],s=25)
ax2.set_xlabel("epoch")
ax2.set_ylabel("loss")
ax2.xaxis.label.set_fontsize(14)   # 设置 x 轴标签字体大小为 14
ax2.yaxis.label.set_fontsize(14)   # 设置 y 轴标签字体大小为 14
ax2.legend()
plt.savefig("train.png")
plt.show()

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import Normalize
number = 3
for number in range(81,101):

    data_frame1 = targets_list2[number][0][0]
    data_frame2 = targets_list2[number][0][2]
    data_frame3 = targets_list2[number][0][4]
    data_frame4 = targets_list2[number][0][6]
    data_frame5 = targets_list2[number][0][8]
```

```
data_frame11 = (pred_list2[number][0])[0]
data_frame22 = (pred_list2[number][0])[2]
data_frame33 = (pred_list2[number][0])[4]
data_frame44 = (pred_list2[number][0])[6]
data_frame55 = (pred_list2[number][0])[8]



data_frame111 = (pred_list1[number][0])[0]
data_frame222 = (pred_list1[number][0])[2]
data_frame333 = (pred_list1[number][0])[4]
data_frame444 = (pred_list1[number][0])[6]
data_frame555 = (pred_list1[number][0])[8]

data_frame1111 = (pred_list3[number][0])[0]
data_frame2222 = (pred_list3[number][0])[2]
data_frame3333 = (pred_list3[number][0])[4]
data_frame4444 = (pred_list3[number][0])[6]
data_frame5555 = (pred_list3[number][0])[8]

print(data_frame1.shape,data_frame2.shape)
# 创建一个图例标签
legend_label = ['Data 1', 'Data 2', 'Data 3', 'Data 4']
data_frame1[data_frame1 < 0.0019] = np.nan
data_frame2[data_frame2 < 0.0019] = np.nan
data_frame3[data_frame3 < 0.0019] = np.nan
data_frame4[data_frame4 < 0.0019] = np.nan
data_frame5[data_frame5 < 0.0019] = np.nan

data_frame11[data_frame11 < 0.0019] = np.nan
data_frame22[data_frame22 < 0.0019] = np.nan
data_frame33[data_frame33 < 0.0019] = np.nan
data_frame44[data_frame44 < 0.0019] = np.nan
data_frame55[data_frame55 < 0.0019] = np.nan

data_frame111[data_frame111 < 0.0019] = np.nan
data_frame222[data_frame222 < 0.0019] = np.nan
data_frame333[data_frame333 < 0.0019] = np.nan
data_frame444[data_frame444 < 0.0019] = np.nan
data_frame555[data_frame555 < 0.0019] = np.nan

data_frame1111[data_frame1111 < 0.0019] = np.nan
data_frame2222[data_frame2222 < 0.0019] = np.nan
```

```
data_frame3333[data_frame3333 < 0.0019] = np.nan
data_frame4444[data_frame4444 < 0.0019] = np.nan
data_frame5555[data_frame5555 < 0.0019] = np.nan
# 绘制 4 个热图
fig, axes = plt.subplots(4, 5, figsize=(10, 10))
axes[0,0].imshow(data_frame1,vmax=1,vmin=0, cmap='jet', origin='upper')
axes[0,0].set_title('frame_000')
axes[0,1].imshow(data_frame2,vmax=1,vmin=0, cmap='jet', origin='upper')
axes[0,1].set_title('frame_002')
axes[0,2].imshow(data_frame3,vmax=1,vmin=0, cmap='jet', origin='upper')
axes[0,2].set_title('frame_004')
axes[0,3].imshow(data_frame4,vmax=1,vmin=0, cmap='jet', origin='upper')
axes[0,3].set_title('frame_006')
axes[0,4].imshow(data_frame5,vmax=1,vmin=0, cmap='jet', origin='upper')
axes[0,4].set_title('frame_008')

axes[1,0].imshow(data_frame11,vmax=1,vmin=0, cmap='jet', origin='upper')
axes[1,0].set_title('frame_000')
axes[1,1].imshow(data_frame22,vmax=1,vmin=0, cmap='jet', origin='upper')
axes[1,1].set_title('frame_002')
axes[1,2].imshow(data_frame33,vmax=1,vmin=0, cmap='jet', origin='upper')
axes[1,2].set_title('frame_004')
axes[1,3].imshow(data_frame44,vmax=1,vmin=0, cmap='jet', origin='upper')
axes[1,3].set_title('frame_006')
axes[1,4].imshow(data_frame55,vmax=1,vmin=0, cmap='jet', origin='upper')
axes[1,4].set_title('frame_008')

axes[2,0].imshow(data_frame111,vmax=1,vmin=0, cmap='jet', origin='upper')
axes[2,0].set_title('frame_000')
axes[2,1].imshow(data_frame222,vmax=1,vmin=0, cmap='jet', origin='upper')
axes[2,1].set_title('frame_002')
axes[2,2].imshow(data_frame333,vmax=1,vmin=0, cmap='jet', origin='upper')
axes[2,2].set_title('frame_004')
axes[2,3].imshow(data_frame444,vmax=1,vmin=0, cmap='jet', origin='upper')
axes[2,3].set_title('frame_006')
axes[2,4].imshow(data_frame555,vmax=1,vmin=0, cmap='jet', origin='upper')
axes[2,4].set_title('frame_008')

axes[3,0].imshow(data_frame1111,vmax=1,vmin=0, cmap='jet', origin='upper')
axes[3,0].set_title('frame_000')
axes[3,1].imshow(data_frame2222,vmax=1,vmin=0, cmap='jet', origin='upper')
axes[3,1].set_title('frame_002')
axes[3,2].imshow(data_frame3333,vmax=1,vmin=0, cmap='jet', origin='upper')
```

```
axes[3,2].set_title('frame_004')
axes[3,3].imshow(data_frame4444,vmax=1,vmin=0, cmap='jet', origin='upper')
axes[3,3].set_title('frame_006')
axes[3,4].imshow(data_frame5555,vmax=1,vmin=0, cmap='jet', origin='upper')
axes[3,4].set_title('frame_008')
# 统一图例
# 隐藏坐标轴
# for ax in axes.flat:
#         ax.axis('off')
# 隐藏刻度但显示坐标轴
for ax in axes.flat:
        ax.set_xticks([])
        ax.set_yticks([])
        ax.spines['top'].set_visible(True)
        ax.spines['right'].set_visible(True)
        ax.spines['bottom'].set_visible(True)
        ax.spines['left'].set_visible(True)

cax = fig.add_axes([0.92, 0.15, 0.02, 0.7])   # 调整图例位置
cbar = fig.colorbar(axes[0,0].imshow(data_frame1, cmap='jet', origin='upper'), cax=cax)

# 调整子图之间的间距
plt.subplots_adjust(wspace=0.1, hspace=-0.3)
plt.savefig("cnn-UNet.png")
# 显示图形
plt.show()
```