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		中国计量大学
参赛队号		**23103560002**
队员姓名	**1.**	徐祺津
	2.	陈天驰
	3.	吴颖

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 U-Net 的强对流降水临近预报

摘　要：

强对流天气具有突发性、局地性强、持续时间短且灾害严重等特点，在天气预报中预测难度较高。因此，如何有效地利用观测数据进行强对流降水短临预报具有重要研究价值。本文利用现有的深度学习的模型，研究了双偏振雷达数据与强对流天气之间的关系，在已知雷达数据的情况下对降水做出了预测，同时优化了数据融合方式，提出了更高效地利用探测数据的模型，实现了数据在特征维度和通道维度的融合。

问题一要求在已知前一小时的雷达观测量（$\mathbf{Z_H}$、$\mathbf{Z_{DR}}$、$\mathbf{K_{DP}}$）的情况下，预测后一小时的 Z_H 预报。由于部分数据是在非强对流天气的时间段测得的，为提升对强对流天气的预测效果，本文首先对 **NJU-CPOL 双偏振雷达数据预**处理，根据**非降雨样本过滤**原则剔除了部分非强对流数据。其次，本文在 **FURENet**(Fusion and Reassignment Networks) 的基础上，构造了具有编码、高层语义特征融合、解码三阶段的临近预报 **U-Net** 网络，拟合出前一小时的雷达观测量与后一小时的 Z_H 之间的关系。在测试集中，本文的模型得到了较好的结果。其中：第 6 分钟 **CSI=0.62,MAE=0.70**,**RMSE=2.83**,**SSIM=0.95**,**PCC=0.34**。

针对问题二，需要改进现有模型因“回归到平均”问题导致的预报模糊问题。本文重点关注了损失函数——由于回归问题的损失函数，无论是 L_1（平均绝对误差）还是 L_2（平均平方误差）损失都会使得模型回归到平均，因此本文在原有的 L_1 损失基础上，分别尝试了对**强对流和非强对流区域固定加权的 L_1 损失、按强对流区域面积自适应分配权重的 L_1 损失、SSIM 损失、对抗损失**，并进行了**消融实验**。结果表明添加**固定加权的 L_1 损失和 SSIM 损失**对模糊效应缓解得较好。模型的 CSI 表明，该模型一定程度上缓解了时间对预测效果的影响。特别，存在一个**第 60 分钟的 CSI 为 0.52** 的结果。此外，本模型中的其他指标为：**MAE=0.86,RMSE=4.45,SSIM=0.96,PCC=0.2678**。

对于问题三，本文根据 **Z-R** 关系模型，利用非线性最小二乘拟合得到 Z-R 关系。由于拟合效果欠佳，本文尝试了对**数据归一化后的 Z-R 关系模型**，得到 $\mathbf{R = 29.1672 Z_H^{1.299} Z_{DR}^{5.0067}}$。剔除完拟合的异常数据后，该模型指标为：**NE=1.721**,**RMSE=0.0717**,**CC=0.682**。此外，利用神经网络的万能逼近性质和卷积网络提取邻域特征的优点，本文也构造了基于 **U-Net 的 Z-R 关系网络**，得到了比前者更好的性能：**NE=0.89,RMSE=2.96,CC=0.73**。

针对问题四，本文首先设计了**对照实验**，将不同的变量组合分别输入网络进行训练，发现在**输入三个变量时效果最好**，当输入变量较少时训练过程不稳定，“回归到平均”的问题更突出。其次，为了更充分地融合多种变量的特征，本文对 **FURENet** 的 **SE-BLOCK** 进行研究探讨，发现其注意力系数只关注了特征通道间的融合而忽略了特征内部的融合。为了在特征层面上融合各个变量，本文引入了 **Bi-GFF(双向门控特征融合) 模块**，在不同变量的特征内部互相融合，弥补了 **SE-BLOCK** 的不足，并得到了更优秀且能稳定训练的模型：**MAE=0.51,RMSE=2.78,SSIM=0.97,PCC=0.29**。

最后，对本文的模型评价与总结。本文基于 FURENet 框架建立了雷达观测量与强对流天气之间的关系，本文的特征融合模块，在通道和特征的维度上对特征进行了充分的融合，同时也增强了模型训练的稳定性，作为即插即用的模块具有一定的应用价值。

关键词： U-Net；FURENet；强对流临近预报；特征融合

目录

图录

表录

一、 问题重述

1.1 问题背景

雷暴是常见的对流天气，而强对流天气包括雷雨大风、龙卷风、短时强降水和冰雹等。这类天气突发性、局地性强、持续时间短且灾害严重，因此在天气预报中预测难度较高 [1]。

我国的国土辽阔，气候分布具有多样性 (北方的寒温带气候、华南地区的热带季风气候以及东北、华北和东部地区的温带季风气候等)，因此灾害性天气种类繁多，地区差异大。其中，由于热带季风气候和温带季风气候常常受到季风影响易发生强对流天气，如雷雨大风、冰雹、龙卷风、短时强降水等。再者，不同地区会有更为复杂的气候变化和特点，故我国强对流天气频发，常常导致重大人员伤亡和财产损失。

综上，因为强对流天气具有突发性和局地性强、灾害重等特点，所以对其进行**短时 (0 ~ 12 小时)** 和**临近 (0 ~ 2 小时)** 预报通常是天气预报工作中的难点。下面分两点对以往强对流天气临近预报工作进行总结：

(1) **传统的工作：**

传统的强对流天气临近预报主要使用雷达等观测资料，并结合风暴识别、追踪技术进行雷达外推预报。这种方法通过外推分析来获取未来一段时间内的雷达反射率因子，并根据**雷达反射率因子**与**降水**间的经验关系 (即 Z – R 关系)，进一步估计未来一段时间的降水量 [2]。雷暴识别追踪和外推预报技术可以分为三大类：持续性预报法、交叉相关法和单体质心法。例如，1978 年，Rinehart[3] 提出的 (Tracking Radar Echoesby Correlation, TREC) 算法是交叉相关法的代表。

(2) **基于深度学习的工作：**

随着大数据的不断积累和计算机计算能力的不断提升，人工智能及深度学习技术蓬勃发展。深度学习是数据驱动的一种常见方法，在理论上网络的性能会随着训练数据量的增加而提升。因此，深度学习方法非常适用于有大量雷达观测数据积累的**短临预报领域**。目前，基于深度学习的短临预报模型主要分两类：一类是基于卷积神经网络 (CNNs) 的模型，比如 U-Net 等；另一类是基于循环神经网络 (RNNs) 的模型，例如 ConvLSTM、DGMR 等。此外，**注意力机制**可以帮助模型更好地捕捉关键特征；**生成对抗网络** (GANs) 也被用于增强预报模型的表现。随着科技的不断进步，深度学习技术在短临预报领域的应用前景将更加广阔，并有望通过整合多种方法和模型来提高预报准确性和效果。

对于强对流预报的算法而言，降落中的雨滴是强对流降水临近预测的关键。雨滴在降落过程中受到空气阻力作用，其形状可能呈现扁球形或馒头形。这种不同的形状导致了雨滴对水平和垂直偏振的**电磁波反射特征**有所不同，并且雨滴的大小也与其形状相关。然而，传统雷达只能发送和接收单一偏振方向的电磁波。为了解决这一限制，新型的双偏振雷达应运而生。

双偏振雷达是一种先进的雷达技术，能够同时发送和接收**水平和垂直**两个偏振方向的电磁波。通过分析不同偏振方向上的回波特征，双偏振雷达可以获取降水粒子的大小、相态、含水量等**微物理信息**。这些信息对于强对流预报具有重要意义。

1.2 问题提出

短临预报是指对未来数小时内的天气变化进行预测，它强调**时间敏感性、空间局部性、强对流天气、数据融合和模型快速更新**等特点，以提供准确、及时的天气预报信息。且由

于短期预报领域有大量雷达观测数据积累，这一个特点使得数据驱动的方法在此领域得到发展。又因为深度学习具有**数据表示能力强、适应复杂非线性关系、处理大规模数据、可结合多源数据**等优势，使得深度学习可以更好的应用双偏振雷达数据来改进强对流降水短临预报，且可以提高短临天气预报的准确性和效率。现根据以上背景以及所提供数据完成以下任务：

问题一：提取微物理特征信息

为了改进强对流临近预报，核心目标是建立一个数学模型，以提取双偏振雷达资料中的微物理特征信息。模型的输入数据是前一小时（10 帧）的雷达观测量：水平反射率因子 (Z_H) 、差分反射率 (Z_{DR}) 、比差分相移 (K_{DP})。模型的输出是后续一小时（10 帧）的 Z_H 预报结果。

问题二：缓解强对流预报中的“回归到平均”问题

当前一些数据驱动的算法在强对流预报中存在”回归到平均”问题，导致预报结果趋于模糊和缺乏细节。为了解决这个问题，在问题一的基础上设计一个数学模型来缓解预报的模糊效应，以便提供更准确、更真实的预报结果。这个数学模型将针对强对流预报进行优化，以捕捉并保留雷达回波的细节信息，从而避免过度平均化的问题。

问题三：构建模型估计降水量

利用给定的 Z_H、Z_{DR} 和降水量数据 (不使用 K_{DP} 变量)，设计模型定量估计对应的降水量。模型的输入数据是 Z_H 和 Z_{DR}，输出数据是降水量。

问题四：评估数据贡献度和优化数据融合策略

为了更好地应对突发性和局地性强的强对流天气，需要设计一个数学模型来评估双偏振雷达资料在强对流降水临近预报中的贡献，并优化数据融合策略。

二、 总体技术路线图

本文总体技术路线图见图1：

三、 基本假设与符号说明

3.1 基本假设

假设一：假设存在 $Z-R$ 关系函数；

3.2 符号说明

符号	含义	单位
Z_H	真实样本的水平反射率因子	dBZ

$\overline{Z_H}$	真实样本的水平反射率因子的均值	/
$\widehat{\ }$	预测符号	/
$\mathrm{Z_{DR}}$	差分反射率	/
$\mathrm{K_{DP}}$	比差分相移	rad (弧度)
R	降水	mm/h
I	强对流矩阵	/
$\mathrm{I_{threshold}}$	强度阈值	dBZ
$\mathrm{A_{threshold}}$	面积阈值	km^2
$\mathrm{A_{evaluation}}$	预测面积	km^2
$\widehat{\mathrm{Z}}_{\mathrm{H}}$	预测的水平反射率因子	dBZ
$F(\cdot)$	网络模型	/
$F_{sq}(\cdot)$	squeeze 操作	/
$F_{ex}(\cdot)$	excitation 操作	/
$F_{scale}(\cdot,\cdot)$	excitation 操作	/
$\mathcal{L}$	损失函数	/

四、 数据预处理

为了更好的阐述本文提出的模型以及结果分析，本节首先介绍了模型使用到的两个数据集，其次对数据集中的数据进行预处理，使用阈值过滤的方法：通过过设定合适的阈值，可以选择性地丢弃部分数据。以此达到减少数据量和去除异常值的效果。

4.1 数据说明

数据集作为数据驱动模型的基础，它对于数据驱动的决策、模型训练和问题解决都至关重要。通过对数据集进行分析和挖掘，可以获取有关目标问题的有益见解和知识，由此更好地理解问题的本质，并从中提取有用的信息。本文的模型使用了两个数据：NJU – CPOL **双偏振雷达数据**和**降水格点数据**。

4.1.1 双偏振雷达 (NJU-CPOL) 数据集

NJU – CPOL 雷达的数据集收集了 2014 年至 2019 年期间共 258 个降水事件的观测数据：数据集具有较高的时间分辨率，每 6 – 7 分钟记录一次观测数据，即任意相邻两帧间隔 6 – 7 分钟。每个事件都提供了海拔高度为 1km、3km 和 7km 的空间分辨率，即记录雷达接收到的每个海拔高度的 $256 \times 256 km^2$ 空间范围内的三个变量：

1. **水平反射率因子** (Z_H)：通过分析 Z_H 的数值可以获得关于降水的信息，例如降雨的强度和密度。图2(a) 展示了某个样本在海拔 1km 处的水平反射率因子的分布图。
2. **差分反射率** (Z_{DR})：通过分析 Z_{DR} 的数值可以推断降水粒子的类型和大小分布情况，进而对降水形态和降水类型进行识别。

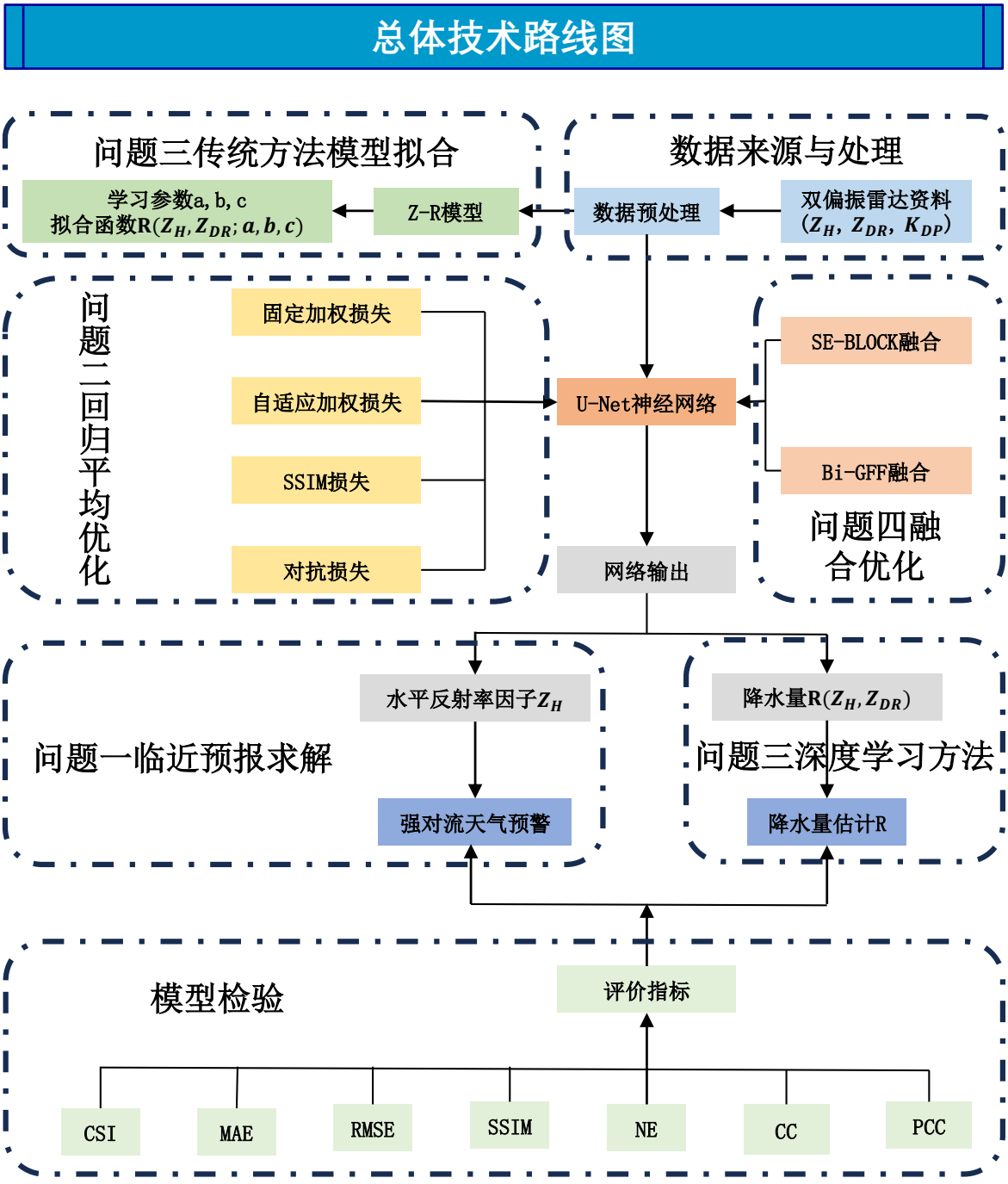
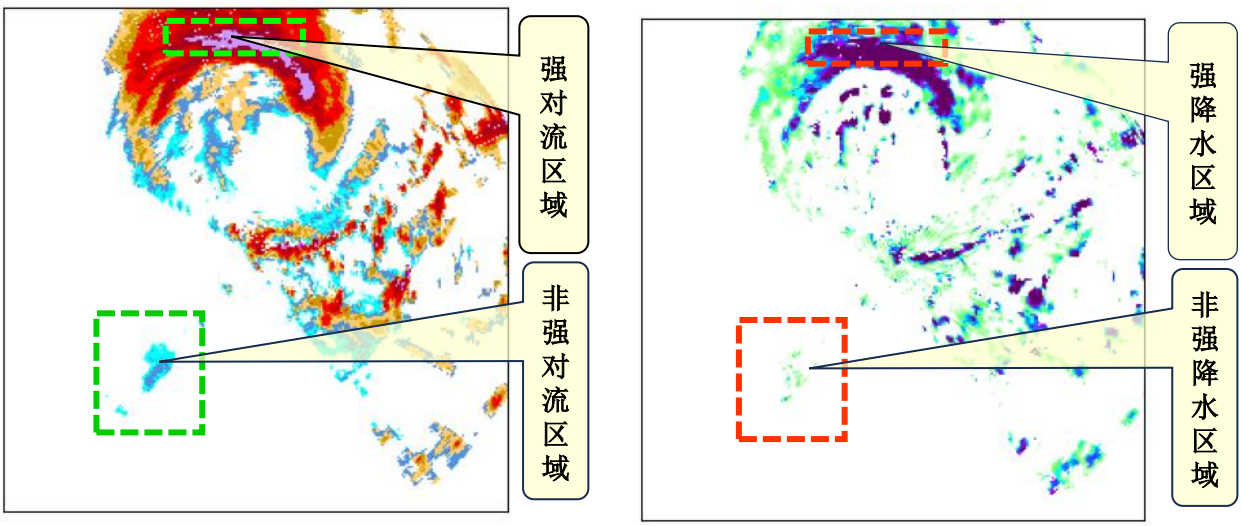

总体技术路线图
问题三传统方法模型拟合
学习参数a, b, c
拟合函数R(Z_H, Z_{DR}; a, b, c)
Z-R模型
数据来源与处理
数据预处理
双偏振雷达资料
(Z_H, Z_{DR}, K_{DP})
问题二回归平均优化
固定加权损失
自适应加权损失
SSIM损失
对抗损失
U-Net神经网络
网络输出
SE-BLOCK融合
Bi-GFF融合
问题四融合优化
水平反射率因子Z_H
强对流天气预警
问题一临近预报求解
降水量R(Z_H, Z_{DR})
降水量估计R
问题三深度学习方法
模型检验
评价指标
CSI
MAE
RMSE
SSIM
NE
CC
PCC

图 1: 总体技术路线图



(a) 水平反射率因子可视化　　(b) 降水量可视化

图 2: 某个时间点的水平反射率因子和降水量可视化

3. **比差分相移 (K_{DP})**：通过分析 K_{DP} 的数值可以了解降水过程中液态含水量的垂直分布、降水粒子的形状和密度等信息。

4.1.2 降水格点数据

该数据集包括了双偏振雷达 (NJU-CPOL) 数据集中对应 258 个降水事件的降水量，同样的任意相邻两帧间隔 6 – 7 分钟。这个数据集在空间上是以格点形式表示的，共有 258 个降水事件。每个降水事件都包含不同时间点的降水量信息。对于每个时间点，数据集将给出相应的降水量数值。图2(b) 展示了某个时间点在海拔 1km 处的降水量分布图。

每个降水事件的时间点可以根据时间间隔和总观测时间确定。例如，如果总观测时间为 1 小时，且时间间隔为 6 – 7 分钟，则每个降水事件包含约 8 – 10 个时间点。这样的降水格点数据集可以用于分析降水事件之间的时间演变和空间分布，以及进行降水量的统计和预测。

4.2 数据预处理

数据预处理是数据挖掘中的重要步骤。由于临近预报领域雷达数据 6 分钟左右更新一次，故雷达资料有数据规模大的特点，而模型训练所需的计算资源有限。此时，需要通过丢弃部分数据来减少数据量，从而加快模型训练速度和降低计算成本。文献 [4] 的数据预处理方法很好的解决了这个问题，且本文通过这个数据预处理，优化了模型的训练过程，提高了建模的准确性和效率。预处理方法流程见图3：

以海拔 1km 的 258 个事件样本为例，过滤掉非强对流的样本后，数据集中还剩下 8910 张图片 (一张图片表示一帧)。由于图片间有时间上的先后顺序，故不能打乱数据集进行选取图片，本文选择前 8000 张图片作为训练数据集对模型进行训练，将后 910 张图片作为测试集对模型进行测试评估。

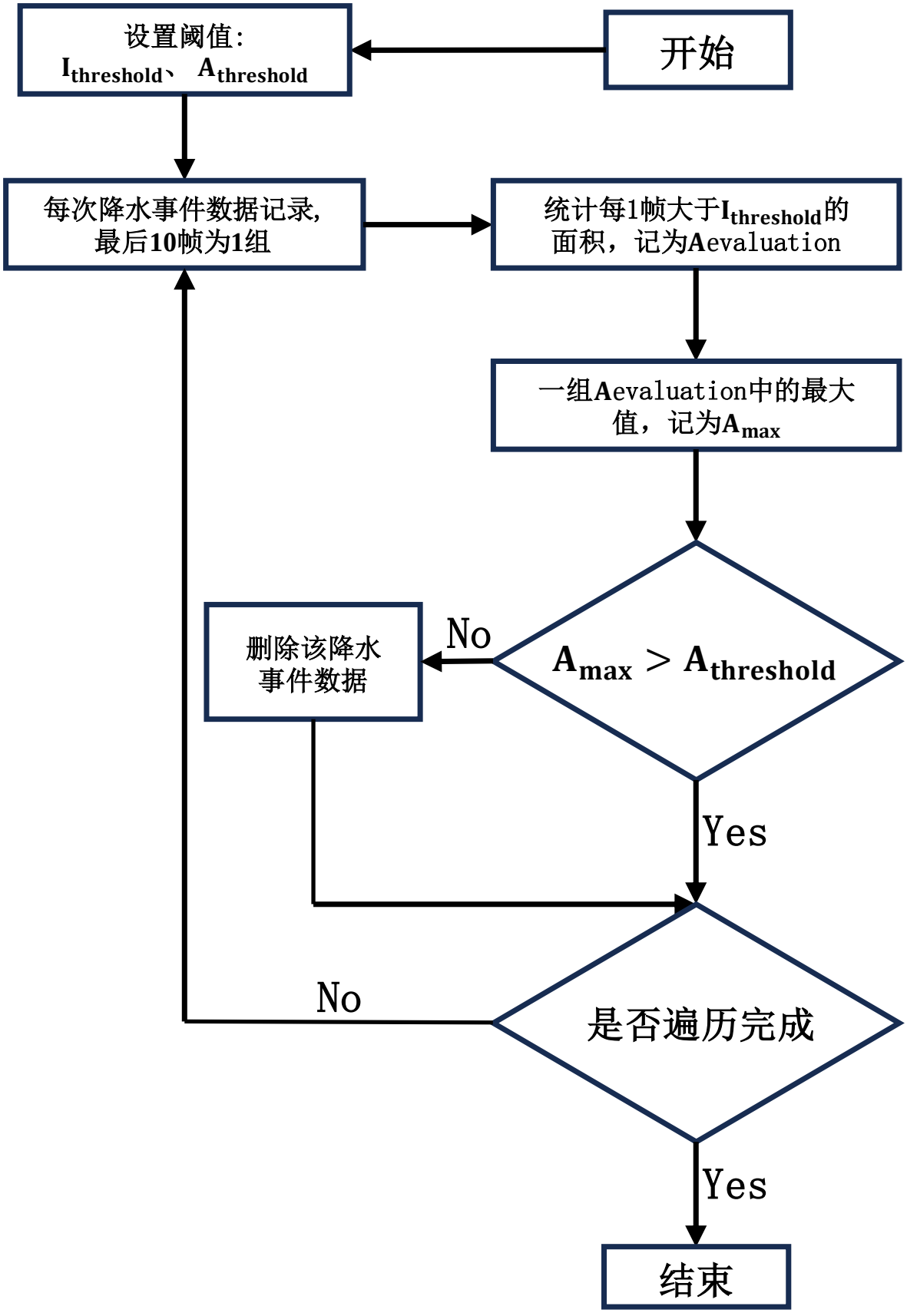


图 3: 对双偏振雷达 (NJU-CPOL) 数据集的预处理流程图

五、 问题一：基于 U-Net 的强对流临近预报模型

5.1 问题分析

问题重述中已指出，传统的方法在临近预报邻域的预测效果不理想，现阶段深度学习的方法在临近预报邻域已经开始广泛应用，预测效果有明显的提升。所以本文问题一的模型是基于深度学习的方法建立的，将图像领域常用的深度学习框架 U-Net[5] 用入模型中。理由如下：

1. **具有强大的特征提取能力：**

   U-Net 采用了**编码-解码结构**，其中编码器部分通过多个卷积层逐渐提取图像的抽象特征，而解码器部分则通过反卷积操作将这些特征映射回输入图像的尺寸。这种设计使 U-Net 能够在不同层级上提取并保留图像的局部和全局特征。

2. **对上下文信息能有效利用：**

   U-Net 在解码器中引入了**跳跃连接**，将编码器中较低层级的特征与解码器中对应的层级特征进行连接。这样可以有效地传递更多的上下文信息，帮助解码器更好地还原图像细节。

3. **结构简单且有良好的泛化能力：**

   U-Net 具备清晰的编码器-解码器结构和跳跃连接。这使得它易于理解和实现，并且能够更快地进行训练和推理。它能够适应不同形状和尺度的目标，适应少样本或不平衡样本的情况，并且在多种图像分割任务中表现出色。

故本文采用 U-Net 作为骨干网络，并设计了专门的机制来整合双偏振雷达变量（如 Z_H, Z_{DR}, K_{DP}）。双偏振雷达变量能够提供对流降水系统的重要微物理信息，通过将其作为输入进行训练，本文模型比传统模型在对强对流风暴的发生、发展和演变的预报中表现出显著的改进。

问题一的思路流程图如图4所示：

由于双偏振雷达数据集中的数据量过于庞大且夹杂着多网络不利的信息。所以为了减少数据量并提高数据的质量，本文先对原始数据进行一次筛选，筛选流程见**数据预处理**中的流程图3。

本题是基于数据驱动的卷积神经网络的模型，丰富的数据信息可以更好的提升网络的预测性能，故模型的输入是将数据集中提供的双偏振雷达常见的三个变量（Z_H, Z_{DR}, K_{DP}）全部作为**网络输入**，为网络提供更多的细节特征。

为了更好的将三种数据融合起来，提取更加丰富且细节的数据特征，模型使用三分支并行的编码结构分别对三种数据提取高层的语义信息，并嵌入 SE-BLOCK 对三分支并行编码结构的输出进行自适应重要性再分配。

5.2 模型建立

本文问题一的模型参考了文献 [4] 的模型（**Fusion and Reassignment Networks**, **FURENet**）。接下来，首先详细的描述了本文提出的网络总体框架（见图5），然后对本文模型的亮点进行阐述。

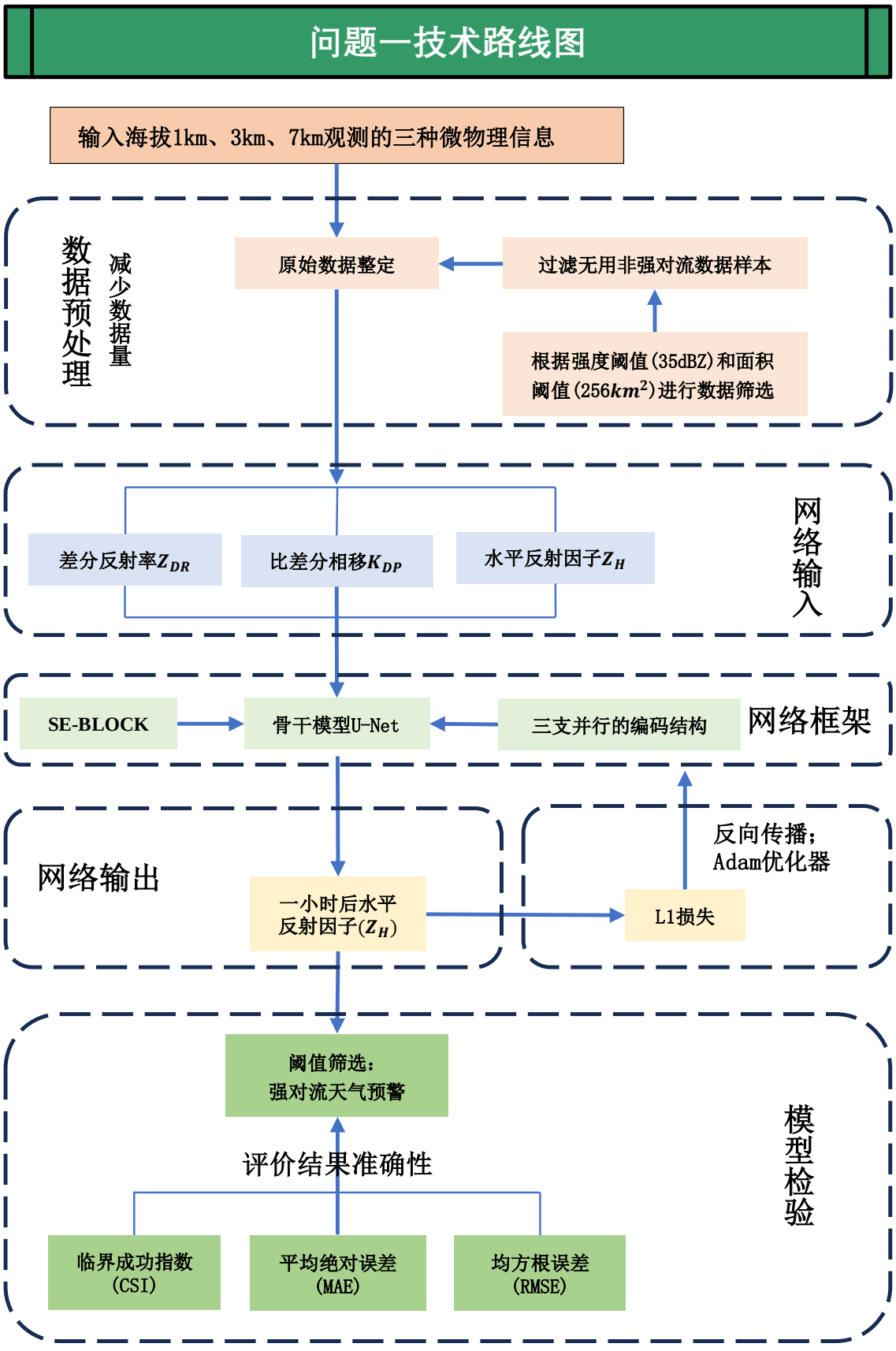

问题一技术路线图
输入海拔1km、3km、7km观测的三种微物理信息
数据预处理
减少数据量
原始数据整定
过滤无用非强对流数据样本
根据强度阈值(35dBZ)和面积
阈值(256km^2)进行数据筛选
网络输入
差分反射率Z_{DR}
比差分相移K_{DP}
水平反射因子Z_H
网络框架
SE-BLOCK
骨干模型U-Net
三支并行的编码结构
网络输出
一小时后水平
反射因子(Z_H)
反向传播;
Adam优化器
L1损失
阈值筛选:
强对流天气预警
评价结果准确性
临界成功指数
(CSI)
平均绝对误差
(MAE)
均方根误差
(RMSE)
模型检验

图 4: 问题一思路流程图

5.2.1 网络总体框架

模型的数学表示：

在数学上，可以将卷积神经网络看成多个卷积层和合适的激活函数的组合，记为 $F(\cdot)$。模型输入的雷达观测量有三种：Z_H, Z_{DR}, K_{DP}，那么模型可以用如下的数学公式表示：

$$\widehat{Z_{H;t+60}} = F(Z_{H;t}, Z_{DR;t}, K_{DP;t})$$

其中，模型的输入为前 60 分钟 (10 帧) 的三个雷达观测量 $(Z_{H;t}, Z_{DR;t}, K_{DP;t})$；模型的输出为 $\widehat{Z_{H;t+60}}$ 表示模型预测出的后续 60 分钟 (10 帧) 的水平反射率因子。

本模型的**重构损失为 l_1 损失**，其计算公式如下：

$$\mathcal{L}_{rec} = \left\| \mathrm{F}(Z_{H;t}, Z_{DR;t}, K_{DP;t}) - \mathrm{Z_{H;t+60}} \right\|_{l_1} \tag{5.1}$$

网络整体结构：

本题模型的主要架构图如图5所示。为了降低计算成本提高灵活性与可变性，本题采用了 U-Net 作为骨干模型。

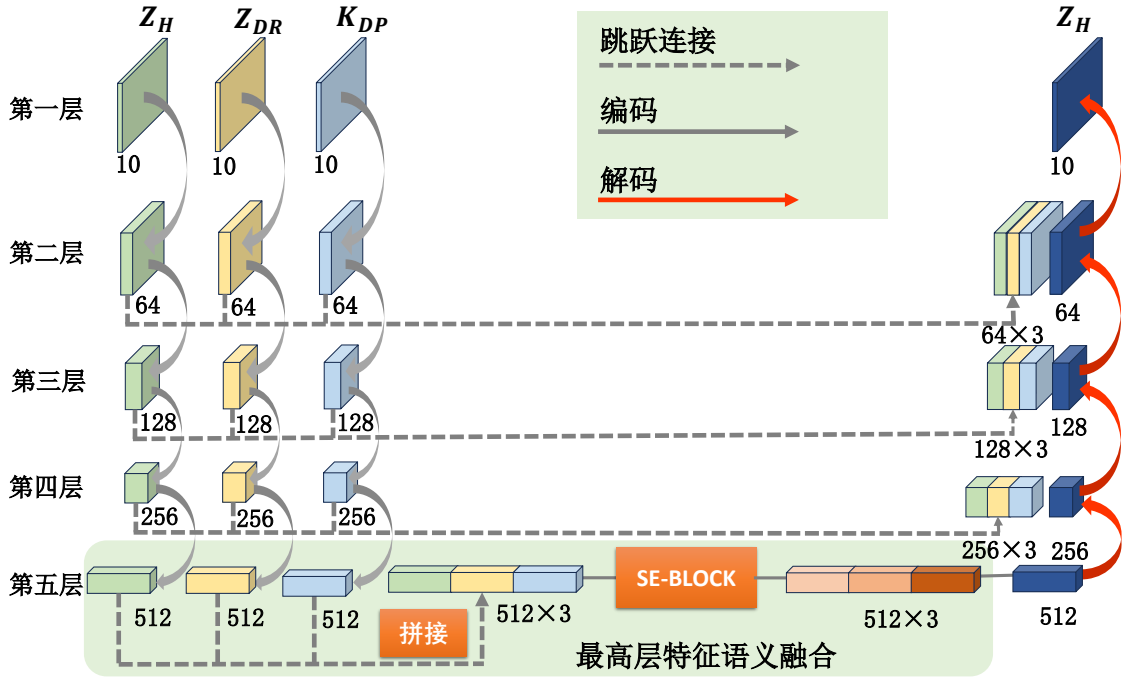


图 5: 问题一网络框架图

模型的总体框架是以 U-Net 网络框架为主进行改进，在下面一小节中将详细说明它们对模型带来的优势：

(1) **三支并行编码再共同解码**：

双偏振雷达的常用变量有三种，在数据说明中已详细阐述。它们各自带有雨滴的不同层面微物理特征信息，丰富的输入特征对神经网络的训练起着关键性的作用。

(2) **后期特征融合**:

文献 [4] 指出，当输入变量之前存在复杂的依赖关系 (不处于相同的语义级别) 时会产生信息纠缠反应 (Information Entanglement Effects)，而后期融合策略可以改变这一问题。

(3) **自适应重要性再分配**:

若对三种微物理信息特征拼接后，直接输入卷积层进行综合信息再提取，会出现特征提取不精确，预测 CSI 不高的问题，故我们引入自适应重要性再分配，让网络自己去控制哪种微物理信息对整体网络的贡献度，以提高临近预报的准确性。

编码器和解码器是 U-Net 结构的重要模块，本文的网络模型是由 4 个编码模块和 4 个解码模块组成。在图5中所示的线路中，编码器的中间状态张量 (用彩色立方体表示) 与解码器的相应级别的张量进行连接。这种连接通常被称为“跳跃连接”，它能够保留输入数据中的多尺度空间信息，从而能够更好地捕捉各种降水系统的演变特征。

在**编码模块**，网络产生一个对输入的微物理信息的多尺度的特征表示。如图6所示:

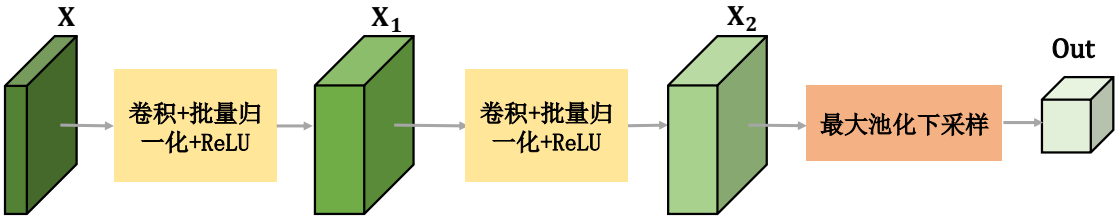


图 6: 编码模块示意图

每个**编码模块/解码模块**包含两次卷积层和一次**下采样层/上采样层**。以编码器第三支的第一个编码模块为例，其输入为前一小时的水平反射率因子 Z_H，将卷积操作记为 C(·)，将 ReLU 函数记为 R(·)，将下采样层的最大池化操作 (Max Pooling) 记为 M(·)，将第三支的第一层编码的输出记为 $\mathrm{Out}_{3,1}$，那么第一层编码模块的数学公式则可写为:

$$\mathrm{Out}_{3,1} = \mathrm{M}(\mathrm{R}(\mathrm{C}(\mathrm{R}(\mathrm{C}(Z_H))))) \tag{5.2}$$

相应的，在**解码模块**，网络进行信息还原、特征融合和空间数据还原等工作，解码模块的示意图如图7:

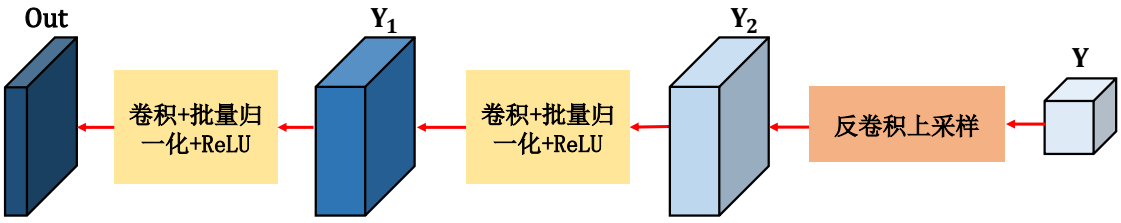


图 7: 解码模块示意图

综合而言，在强对流临近预报任务中，使用 U-Net 网络将编码和解码模块相互融合，能够有效地处理 NJU-CPOL 双偏振雷达数据，并提供准确可靠的。

5.2.2 模型亮点设计

虽然 U-Net 在多个方面都展现出良好的泛化能力，但由于数据质量、数据多样性以及任务特定的问题等因素的限制，在实际应用中，需要对模型进行验证和调优。在临近预报领域中直接引入 U-Net 后的预测效果显然也是不理想的。本文在采用 U-Net 作为主体框架的同时，进行了优化：**使用三支并行的编码结构、在最高层语义的时候拼接独立的三支特征图、加入 SE – Block 模块**，使得预测效果有了不错的提升。

- **三支并行的编码结构：**

双偏振雷达数据提高了其常见的三个变量，本题将三个变量分三支单独输入网络中。不同的雷达参数提供了不同方面的降雨信息，并且多通道的输入又可以提供更多的特征信息，所以在模型的输入层，考虑三支并行编码结构。它能更全面地捕捉降雨的特征，从而提高模型的表达能力和预测准确性。又因为这三个变量之间存在相关性，所以在网络中需要将提取到的特征拼接起来。

- **最高层特征语义融合：**

由图8所示，最高层特征融合示图由两部分组成：**对三支并行解码结构输出的特征进行拼接**和 **SE – BLOCK**。第一步生成融合对象，第二步对融合对象的重要性进行自适应分配再融合。

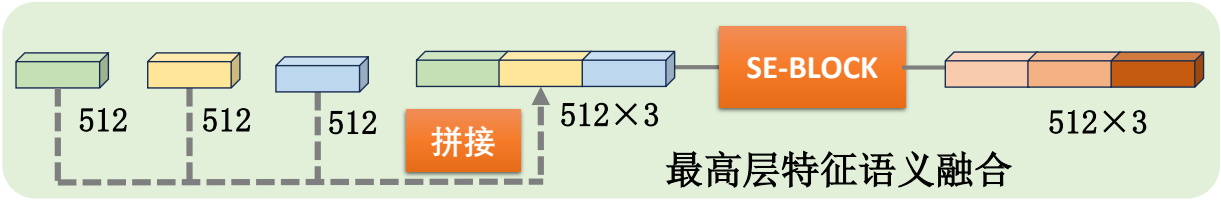


图 8: 最高层特征融合示图

A. **拼接**

文献 [4] 指出，在高层特征语义的时候融合比在低层语义的时候融合更能提高网络性能，故本文在最第 4 次编码模块之后对三支提取到的特征语义进行拼接 (见问题一网络框架图9左下角)。由此可以捕捉不同变量之间的依赖关系。具体来说，首先三支并行编码结构可以对每个变量进行编码，以获取其重要的特征信息。接着，在其最高特征语义级别上，将这些信息进行拼接以生成全面准确的特征表示。

将三支不同变量的输入经过 4 次编码模块后的输出分别记为 $\mathrm{Out}_{1,4}$、$\mathrm{Out}_{2,4}$、$\mathrm{Out}_{3,4}$。同时它们也是后面特征融合阶段的融合对象，本文的拼接操作，拼接机制的结果记为：

$$\left[\mathrm{Out}_{1,4}, \mathrm{Out}_{2,4}, \mathrm{Out}_{3,4}\right]$$

B. **SE – Block**[6]

相比于在输入时候拼接数据进行融合 (简称为" 早期融合")，在最高层特征语义拼接再进行融合 (简称为" 后期融合") 能够更好地处理不同变量之间的非线性依存关系，从而提高整体模型的表现。

值得注意的是，双偏振雷达的这三种变量都是通过测量降水粒子在不同方向上的电磁波的反射情况而来，相互之间可能存在某种联系。在模型中加入自注意力的模块，可以通过**自适应重要性**再分配来提升网络对拼接后的三种特征信息的自主提取能力。

模型如果能够自适应地重新分配多个变量的重要性，即考虑提高有利于预测的特征权重，降低破坏预测的特征权重。对于这个想法，本文加入的 SE – Block 是向多变量的后期融合步骤中添加 **"Squeeze & Excitation 操作"**，图如9所示。

具体来说，SE – Block 使用全局池化的方式对融合后的最高特征语义张量进行 Squeeze 操作。在这个操作中，SE – Block 对张量在其空间维度 (宽度 × 高度) 上取平均值，将其简化为一个全局通道特征。通过这种方式，可以得到了一个能够代表整体特征。

Squeeze 操作的数学公式如下：

$$F_{sq}(\mathbf{X}_{\text{in};c}) = \frac{1}{\text{W} \times \text{H}} \sum_{i=1}^{\text{W}} \sum_{j=1}^{\text{H}} \mathbf{X}_{\mathbf{in};c}(i,j), \quad c = 1, 2, ..., C \tag{5.3}$$

其中，$\mathbf{X}_{\mathbf{in};c} \in \mathbf{R}^{\text{W}\times\text{H}}$ 是 SE-Block 输入张量 $\mathbf{X}_{\mathbf{in}} \in \mathbf{R}^{\text{W}\times\text{H}\times\text{C}}$ 的通道维度切片，W 和 H 分别是张量 $\mathbf{X}_{\mathbf{in}}$ 的宽度和高度，$F_{sq}(\cdot)$ 表示 Squeeze 操作。

Excitation 操作利用双层全连接感知器神经网络和自适应门控机制，生成一个学习权重的向量 S_c(大小为 $1 \times C$)，表示原始特征在重新分配中的重要性。Excitation 操作的数学公式如下：

$$F_{ex}(F_{sq}(\mathbf{X}_{\mathbf{in}}), \mathbf{W}) = \sigma(\mathbf{W_2}(\text{ReLU}(\mathbf{W_1} F_{sq}(\mathbf{X}_{\mathbf{in}})))) \tag{5.4}$$

其中，$F_{ex}(\cdot, \mathbf{W})$ 表示 Excitation 操作，$\mathbf{W}$ 表示双层全连接层的权重，σ 表示 Sigmoid 函数。

最后，**Scale 操作**将学习到的权重与最初拼接好的待融合对象相乘，以获得最终的重要性重新分配张量。具体计算公式如下：

$$\mathbf{X}_{\text{out};\mathbf{c}} = F_{\text{scale}}(\mathbf{S_c}, \mathbf{X}_{\text{in};\mathbf{c}}) = \mathbf{S_c} \cdot \mathbf{X}_{\text{in};\mathbf{c}} \tag{5.5}$$

其中，$F_{scale}(\cdot,\cdot)$ 表示 Scale 操作，也就是给输入的 $\mathbf{X}_{\mathbf{in}}$ 的第 c 个通道维切片乘上学习到的权重。$\mathbf{X}_{\mathbf{out};\text{c}}$ 表示 SE – Block 的最终输出结果。

综上所述，SE-BLOCK 的整体数学表达式如下：

$$\mathbf{X}_{out} = \mathbf{SE}(\mathbf{X}_{in}) = F_{scale}(F_{ex}(F_{sq}(\mathbf{X}_{in}), \mathbf{W}), \mathbf{X}_{in}) \tag{5.6}$$

其中，$\mathbf{SE}(\cdot)$ 表示 SE-BLOCK 操作。

网络总体表达式为：

$$\begin{aligned} \widehat{Z}_H &= \mathbf{F_{Net1}}(Z_H, Z_{DR}, K_{DP}) \\ &= \mathbf{Decode}(\mathbf{SE}(\mathbf{Encode}(Z_H, Z_{DR}, K_{DP}))) \end{aligned} \tag{5.7}$$

其中，$\mathbf{F_{Net1}}(\cdot)$ 表示问题一的网络，**Encode**(·) 表示编码操作，**Decode**(·) 表示解码操作。

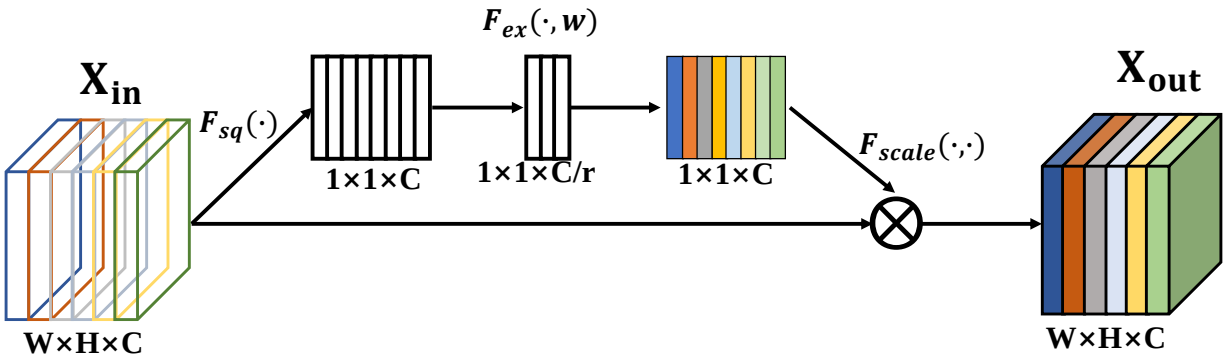


图 9: SE – Block 示图

5.3 模型求解与结果分析

5.3.1 实验设置

本文的模型设置如表2所示：

表 2: 问题一的实验设置细节表

实验设置	
工作环境	Windows10, Python=3.7
深度学习框架	PyTorch=1.11.0
骨干网络	U-Net
编码层数	4
各层卷积输出通道数	64,128,256,512
优化器	Adam(动量衰减系数 0.9, 0.99, 其余参数为默认值)
学习率	0.001
损失函数	l_1 损失
批量	1
SE 模块降维率	16

5.3.2 评价指标

(1) **临界成功指数** (CSI)**:**

临界成功指数 (Critical Success Index，CSI)，也称为 TS 评分 (Threat Score)。CSI 是一种用于评估强对流天气预报准确性的指标，它综合考虑了预报中的命中、漏报和误报三个要素，以量化预报的成功性。在临近预报领域内，对应的临界成功指标计算过程：

- **强对流矩阵：**

  强对流矩阵 I 中强对流天气表示为 1，非强对流天气表示 0，整体的数学公式如下：

$$\mathrm{I} = \begin{cases} 1 & , Z_H \geq \mathrm{I}_{Threshold} \\ 0 & , Z_H < \mathrm{I}_{Threshold} \end{cases} \tag{5.8}$$

- **预测强对流矩阵：**

  $\hat{\mathrm{I}}$ 是模型输出的对应区域是否是强对流的预测，整体的数学公式表示为：

$$\hat{\mathrm{I}} = \begin{cases} 1 & , \widehat{Z}_H \geq \mathrm{I}_{Threshold} \\ 0 & , \widehat{Z}_H < \mathrm{I}_{Threshold} \end{cases} \tag{5.9}$$

- **水平反射率因子的阈值：**

  由于水平反射率因子 (Z_H) 主要反映的是区域降水的强弱，故可以使用区域 Z_H 的大小来判定该区域是否处于强对流天气，由文献 [4] 得，强度阈值设为 $\mathrm{I}_{Threshold} = 35$。

- **区域遭受强对流天气：**

  若预测出的区域的 $\hat{\mathrm{I}} \geq 35$ 即该区域被判定为后续一个小时即将受到强对流天气影响。

由上述内容可以计算出评估一个模型对某区域后续一个小时是否遭到强对流预测准确性的指标 (CSI) 计算方式如下：

$$\mathrm{CSI} = \frac{\sum_{i,j} \mathrm{Hits}_{ij}}{\sum_{i,j} (\mathrm{Hits}_{i,j} + \mathrm{Misses}_{i,j} + \mathrm{False\ Alarms}_{i,j})} \tag{5.10}$$

* Hits : 预报正确的次数，即观测和预报都正确判断为强对流。此时 $\mathrm{I}_{i,j} = 1$ 且 $\hat{\mathrm{I}}_{i,j} = 1$；
* Misses : 表示预报漏报的次数。即观测为强对流但预报未能正确判断为强对流，此时 $\mathrm{I}_{i,j} = 1$ 且 $\hat{\mathrm{I}}_{i,j} = 0$；
* False Alarms : 表示预报误报的次数，即预报为强对流但观测并非强对流。此时 $\mathrm{I}_{i,j} = 0$ 且 $\hat{\mathrm{I}}_{i,j} = 1$。

CSI 的取值范围为 0 到 1，数值越接近 1 代表预报准确性越高，而接近 0 则表示预报准确性较低。

临界成功指数综合了**命中率、漏报率**和**误报率**。它不仅考虑了预报对真实事件的检测能力，还考虑了误报的情况，并且对漏报和误报进行了平衡考虑。因此，CSI 更全面地评估了强对流天气预报的性能，是一种常用且有效的评估指标。

值得注意的是，CSI 作为一种指标，仅关注预报是否是强对流天气，并不考虑强对流天气的等级。因此，在实际应用中，还需要结合其他指标和方法，综合评估对流天气预报的全面性能。

(2) **平均绝对误差** (MAE)

平均绝对误差 (Mean Absolute Error, MAE) 是一种常用的统计指标，用于评估预测结果与实际观测值之间的误差大小。它的计算方法是将预测值与观测值之间的差的绝对值加起来，然后求出其平均值。数学公式如下：

$$\mathrm{MAE} = (\frac{1}{\mathrm{N}}) \sum_{i=1}^{\mathrm{N}} (\left| \widehat{\mathrm{Z}_{\mathrm{H};i}} - \mathrm{Z}_{\mathrm{H};i} \right|) \tag{5.11}$$

其中，N 表示成对的预测和真实水平反射率 Z_H 的数量。MAE 越小，说明模型的预测误差越小，即预测结果越接近真实观测值。MAE 广泛应用于各个领域中，例如金融、能源、气候等，用于评估与比较预测模型的性能。

(3) **均方根误差** (RMSE)**:**

对于涉及到降水预报的对流天气，可以使用定量降水预报的评估指标，如均方根误差 (Root Mean Square Error, RMSE)、标准化绝对误差 (NE) 和相关系数 (CC) 等来评估预报结果与观测结果之间的一致性和准确性。其中，均方根误差 (RMSE) 度量了预测值与真实值之间的差异程度。

RMSE 计算过程中先对误差进行平方，然后再取平均值的平方根。RMSE 值越小表示预测结果越接近真实值。计算公式如下：

$$\mathrm{RMSE} = \left[\frac{1}{\mathrm{N}} \sum_{i=1}^{\mathrm{N}} (\widehat{Z}_{H;,i} - \mathrm{Z}_{H;i})^2 \right]^{\frac{1}{2}} \tag{5.12}$$

5.3.3 结果分析

根据数据预处理后的海拔 1km 高度时的数据集经过训练后模型预测的结果图 (如图10所示)，可以观察到模型对未来时间段的变化趋势进行了合理的预测。预测结果显示了随着时间的推移，海拔 1km 高度的变化情况。

各行分别表示： 前一个小时 Z_H 的地面真值 (第一行)；后续一个小时 Z_H 的地面真值 (第二行)；后续一个小的 Z_H 的地面预测值 (第三行)；强对流天气降雨强度的比色卡，从左到右强度逐渐增加 (第四行)。

各列分别表示： 对应的 6 分钟 (第一列)、12 分钟 (第二列)；18 分钟 (第三列)；24 分钟 (第四列) 和 30 分钟 (第五列) 预测的临近预报。图中一格表示 25km 的距离。

图11是测试集某一小时内 CSI 变化图。子图 (a) 是文献 [4] 的结果图；子图 (b) 是本文的模型结果图。从整体上观察，预测 CSI 随着预测时间跨度的增加而递减，CSI 曲线呈下降趋势。

由于 CSI 未考虑强对流天气的等级，故需要结合其他指标 (MAE、RMSE、SSIM 和 PCC) 一起评价，评价结果见表3。其中，结构相似性指标 (SSIM) 和皮尔逊相关系数 (PCC) 的详细介绍见问题二的评价指标。

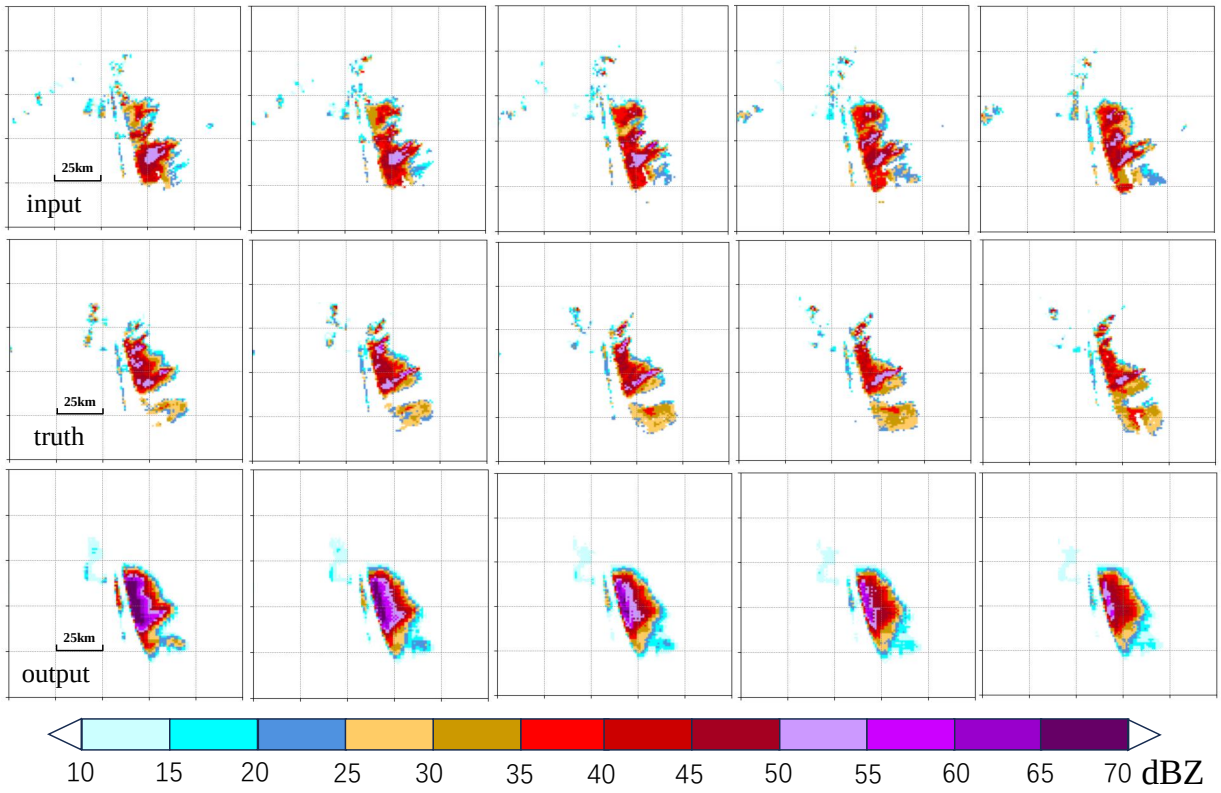


图 10: 问题一模型预测结果可视化

从上到下分别是前一个小时的真实样本和后续一小时的真实样本、预测样本

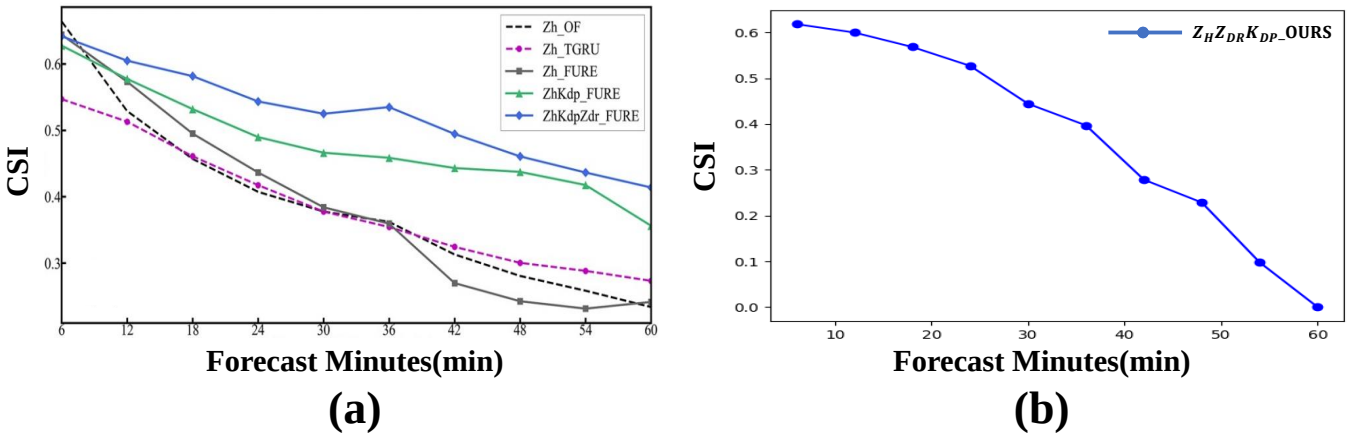


图 11: 一个小时内问题一模型的临近预报 CSI 效果图

(a) 文献 [4] 中 35dBZ 阈值的 CSI; (b) 本文的 CSI

表 3: 问题一模型的评价表

评价指标	评价结果
MAE↓	0.70
RMSE↓	2.83
SSIM↑	0.95
PCC↑	0.34

六、 问题二：构建不同损失缓解“模糊效应”

6.1 问题分析

根据文献 [7] 提供的结论，在基于深度学习的临近预报邻域使用 l_1 损失作为网络的损失函数会误导模型的评估，使得预报结果出现“回归到平均”问题，因此预报结果趋于模糊，问题二中称为：“模糊效应”，这对强对流临近预报任务来说是一个难题。

在基于深度学习方法的图像修复领域 [8, 9]，模糊效应也是一个常见的问题，在该领域中使用图像质量度量可以显着减少图像模糊问题。若将每帧 256×256 的张量看成 256×256 的图像，再将图像修复领域的方法迁移到临近预报邻域，是一个解决问题的方案。所以针对问题一模型中原有的损失函数增加图像质量度量来优化损失。

本题从以下两个方面进行考虑图像质量度量的选取：

1. **使模型更加关注强对流天气：**

   根据文献 [8]，图像修复领域给待修复图像加入掩码可以明确标记待修复区域，限制修复算法的作用范围，并引导模型更好地处理缺失区域。

   迁移该方法至临近预报中，引入**基于掩码的加权损失函数**：引导模型更专注地学习目标区域的特征和纹理。进一步，对于临近预报邻域，需要重点关注的是强对流天气的范围。

   此外，除了简单的引入掩码的固定值加权损失，还可以考虑**自适应的掩码面积加权损失**，以此给网络提供更加有目的性的自适应约束。

2. **使模型增强细节的真实性：**

   - **结构相似性 (SSIM) 损失**
     文献 [7] 指出，将结构相似性指标 (SSIM) 作为新增的损失函数与 MAE 和 MSE 损失函数结合起来使用，可以有效提高预报网络提取细节特征的真实性。
   - **对抗损失**
     对抗损失是一种生成对抗网络 (GANs) 常用的损失函数，它的目标是使生成器可以生成逼真的样本，判别器无法准确区分生成样本和真实样本。将对抗损失引入临近预报邻域可以使得生成的预测样本在视觉上更加真实合理。

本文重点关注损失函数对模型预测细节充分性、真实性的作用。问题二的技术路线见图12：

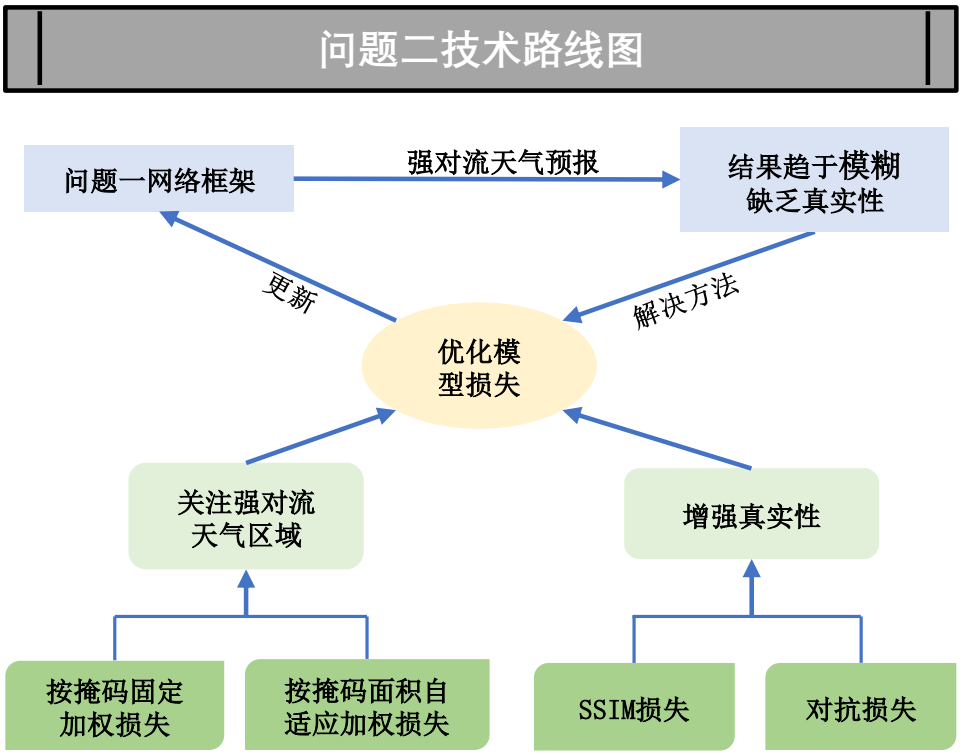


图 12: 问题二的技术路线图

6.2 模型建立

优化模型损失的方案有 8 种，为提高效率本文采用消融实验的方式对损失的作用进行讨论：分别是 l_1 加权损失 (基于掩码的固定加权和按掩码面积的自适应加权)；l_1 加权损失与 SSIM 损失结合；l_1 加权损失、SSIM 损失和对抗损失结合。随后使用评价指标 (MAE、RMSE、SSIM 和 PCC(Pearson 相关系数)) 来判断哪种方案是最优的。

6.2.1 基于强对流区域掩码的加权损失函数的模型

为了使网络能够更多关注强对流天气的区域，首先考虑将掩码的思想引入临近预报中，应用加权损失对问题一求解中的重构损失进行修改。

将掩码思想引入模型可以使其更加关注样本中的强对流天气区域：以将强对流天气的数值设置为 1，表示重点关注区域；非强对流天气的数值设置为 0，表示弱关注区域。那么掩码在本题中可以被重新定义：一个标记是否为强对流或非强对流的二值图像。本题选择的掩码为问题一的强对流天气判定矩阵 I。

Case1：按掩码固定加权的损失函数

考虑固定比值的掩码加权损失函数：

$$\mathcal{L}_{wrec1} = \omega_1 \left\| \mathrm{I} \odot (\mathrm{F}(\mathrm{Z}_{\mathrm{DR;t}}, \mathrm{K}_{\mathrm{DP;t}}, \mathrm{Z}_{\mathrm{H;t}}) - \mathrm{Z}_{\mathrm{H;t+60}}) \right\|_{l_1} + \omega_2 \left\| (1 - \mathrm{I}) \odot (\mathrm{F}(\mathrm{Z}_{\mathrm{DR;t}}, \mathrm{K}_{\mathrm{DP;t}}, \mathrm{Z}_{\mathrm{H;t}}) - \mathrm{Z}_{\mathrm{H;t+60}}) \right\|_{l_1} \tag{6.1}$$

其中，ω_1、ω_2 表示固定权重。

Case2：按强对流面积自适应的掩码加权损失函数

为了让模型自适应地关注强对流天气区域，本文考虑按面积自适应比值的掩码加权损失函数：

$$\mathcal{L}_{wrec2} = \omega_1 \frac{\left\| \mathrm{I} \odot (\mathrm{F}(\mathrm{Z}_{\mathrm{DR;t}}, \mathrm{K}_{\mathrm{DP;t}}, \mathrm{Z}_{\mathrm{H;t}}) - \mathrm{Z}_{\mathrm{H;t+60}}) \right\|_{l_1}}{S_1 + 1} + \omega_2 \frac{\left\| (1 - \mathrm{I}) \odot (\mathrm{F}(\mathrm{Z}_{\mathrm{DR;t}}, \mathrm{K}_{\mathrm{DP;t}}, \mathrm{Z}_{\mathrm{H;t}}) - \mathrm{Z}_{\mathrm{H;t+60}}) \right\|_{l_1}}{S_2 + 1} \tag{6.2}$$

其中，S_1 表示强对流区域的面积，S_2 表示非强对流区域的面积。在后续叠加损失函数的模型中，本题使用**固定权重的掩码加权损失**。

6.2.2 按掩码固定加权损失和 SSIM 损失的组合模型

在关注强对流天气区域的同时，对于预测的强对流区域等级情况也应当要符合真实情况，为此，再次引入结构相似性指标 SSIM 作为损失，以增强预测结果的准确性。SSIM 的具体描述和数学表达式见本小节的评价指标。

为了在神经网络训练过程中使用这些测量作为损失函数 (最大限度地降低训练成本)，预测图像 $\widehat{\mathrm{Y}}$ 和真实图像 Y 的 SSIM 损失函数数学表达式：

$$\mathcal{L}_{\mathrm{SSIM}} = 1 - \mathrm{SSIM}(\widehat{\mathrm{Y}}, \mathrm{Y}) \tag{6.3}$$

Case3：按掩码固定加权和 SSIM 的组合损失函数

$$\mathcal{L}_{case3} = \lambda_1 \mathcal{L}_{wrec1} + \lambda_2 \mathcal{L}_{\mathrm{SSIM}} \tag{6.4}$$

6.2.3 按掩码固定加权损失、SSIM 损失和对抗损失的组合模型

对抗损失的特点是关注输出图像的视觉真实性、合理性，而不关注与真实图像之间的差异。因此，本题加入对抗损失希望网络的输出 (预测数据) 更加真实、合理。

对抗损失是基于生成对抗网络的思想，包括一个生成器网络和一个判别器网络，前者负责生成伪造样本，后者负责判断真假。对抗损失旨在使生成样本愈发难以被准确分类，从而生成更逼真的样本。

生成器 G 损失函数的数学表达式：

$$\mathcal{L}_{adv} = \mathbb{E}\left[\log\left(1 - D\left(\widehat{\mathrm{Y}}\right)\right)\right] \tag{6.5}$$

判别器 D 损失函数的数学表达式：

$$\mathcal{L}_{\mathrm{D}} = \mathbb{E}_{\mathrm{Y}}\left[\log\left(D\left(\mathrm{Y}\right)\right)\right] + \mathbb{E}_{\widehat{\mathrm{Y}}}\left[\log\left(1 - D\left(\widehat{\mathrm{Y}}\right)\right)\right] \tag{6.6}$$

Case4：按掩码固定加权、SSIM 和对抗组合的损失函数

由简到繁，case4 将上述的模型融合起来。对于预测图像 $\widehat{\mathrm{Y}}$ 和真实图像 Y 形成一个组合损失函数：

$$\mathcal{L}_{Case4}(\widehat{\mathrm{Y}},\mathrm{Y}) = \sum_{l} \lambda_l \mathcal{L}_{\mathrm{l}}(\widehat{\mathrm{Y}},\mathrm{Y}) \tag{6.7}$$

其中，l 是在集合 $wrec1$, $wrec2$, $SSIM$, adv 中选择的分量 λ_l 是每个分量的权值。

6.2.4 其他策略

问题二的模型主要以优化损失函数为主。在此基础上，模型还**加深了网络层数** (增加模型复杂度) 以及采取了**提前停止策略**。为了得到更深层的特征语义，模型加深了网络层数 (模型复杂度增加)。采取提前停止策略：通过监控验证集上的性能，并在性能不再提升时停止训练，可以有效避免模型在训练集上过拟合。

6.3 模型求解和结果分析

6.3.1 实验设置

问题二的模型改变了问题一模型的损失函数，详细的介绍在模型建立中已经给出。除此之外，其他设置：

$$\omega_1 = 6,\ \omega_2 = 1\ \lambda_1 = \lambda_2 = 1,\ \lambda_3 = 0.2,\ \lambda_4 = 0.01$$

本题的其他实验设置于问题一相同，详情见表2。

6.3.2 评价指标

在问题二中使用了评价指标：平均绝对误差 (MAE)、均方根误差 (RMSE)、结构相似性指标 (SSIM) 和皮尔逊相关系数 (PCC)。前两个指标在问题一中已经详细说明，故下面对 SSIM 和 PCC 进行介绍。

(1) **结构相似性指标 (SSIM)：**

SSIM 是一种用于衡量两张图片相似度的指标，它考虑了人类视觉系统的特征，包括亮度、对比度和结构等因素，以更好地反映图像质量的变化。故将 SSIM 损失引入网络中，即可以使网络的输出更加符合人类主观感受，又可以使网络更准确地判断图像的相似性与质量。

测量两个图像 x 和 y 之间的相似性的 SSIM 方法定义如下：

$$\mathrm{SSIM}(x,y) = l(x,y)\cdot c(x,y)\cdot s(x,y) = \left(\frac{2\mu_x\mu_y + \mathrm{C}_1}{\mu_x^2 + \mu_y^2 + \mathrm{C}_1}\right)\cdot\left(\frac{2\sigma_x\sigma_y + \mathrm{C}_2}{\sigma_x^2 + \sigma_y^2 + \mathrm{C}_2}\right)\cdot\left(\frac{\sigma_{xy} + \mathrm{C}_3}{\sigma_x\sigma_y + \mathrm{C}_3}\right) \tag{6.8}$$

其中，$l(x,y)$ 是亮度相似性，$c(x,y)$ 是对比度相似性，$s(x,y)$ 是结构相似性。μ_x 和 μ_y 分别是 x 和 y 的平均值，而 σ_x 和 σ_y 分别是 x 和 y 的标准差。σ_{xy} 是 x 和 y 的互相关 (协方差)。C_1、C_2 和 C_3 是小的正常数，用于避免零除法和数值不稳定性。

(2) **皮尔逊相关系数 (PCC)：**

皮尔逊相关系数 (PCC) 是衡量向量相似度的一种方式。输出范围为-1 到 +1，其中 0 代表无相关性，负值代表负相关，正值代表正相关。计算公式如下：

$$\text{Pearson} = \frac{\sum_{i=1}^{n}(\mathrm{X}_i - \overline{\mathrm{X}})(\mathrm{Y}_i - \overline{\mathrm{Y}})}{\sqrt{\sum_{i=1}^{n}(\mathrm{X}_i - \overline{\mathrm{X}})^2}\sqrt{\sum_{i=1}^{n}(\mathrm{Y}_i - \overline{\mathrm{Y}})^2}} \tag{6.9}$$

皮尔逊相关系数 (PCC) 适用于：

· 两个变量之间是线性关系，都是连续数据；

· 两个变量的总体是正态分布，或接近正态的单峰分布；

· 两个变量的观测值是成对的，每对观测值之间相互独立。

6.3.3 结果分析

• 模型预测强对流天气的结果：

下面是 **Case1** 和 **Case3** 的模型对海拔高度 1km 的事件样本由前一个小时的双偏振雷达变量 ($\mathrm{Z_H}$, $\mathrm{Z_{DR}}$, $\mathrm{K_{DP}}$) 预测后续一个小时的水平反射率因子 ($\widehat{\mathrm{Z}}_{\mathrm{H}}$, 反映降水的强弱)。水平反射率因子的大小被分为多个等级。它用于展示强对流天气降雨强度，从左到右强度逐渐增加。

首先，**Case1**(固定加权的掩码损失函数) 的模型预测结果见图13第 $1-3$ 行；**Case3**(固定加权的掩码损失和 SSIM 损失的组合函数) 的模型预测结果见图13第 $4-6$ 行。

其次，由设定的强对流阈值 $\mathrm{I}_{Threshold} = 35$ 可见，在比色卡中红色向右的颜色表明是强对流天气，此时模型需要对该区域的天气预报发出强对流预警消息。如图可见，两种方案的模型都能在后续一小时对强对流区域给出及时的预警，故两个模型预测 CSI 高，都有效缓解了“模糊效应”。

• 模型评价指标评分

对本题提出的所有模型 (用 **Case** 表示) 都用多个评价指标进行评，结果见表4。其中，MAE 和 PCC 指标都是越低模型预测的结果越好，而 RMSE 和 SSIM 指标都是越高越好，每个指标的最佳值已经加粗。对比表格的指标得：Case4 的模型的性能 (MAE、RMSE 和 SSIM) 优于其他方案的模型。其中：

(1) **Case1:** 固定加权的掩码损失函数模型；

(2) **Case2:** 按强对流面积自适应加权的掩码损失函数模型；

(3) **Case3:** 固定加权的掩码损失和 SSIM 损失的组合模型；

(4) **Case4:** 固定加权的掩码损失、SSIM 损失和对抗损失的组合模型；

• 后续一小时内的模型临近预报 CSI

图14比较得，按掩码固定加权的损失函数模型 (**Case1**) 对强对流天气的预报 CSI 在 $50-60$min 时比问题一模型的预报 CSI 高很多，可见基于问题一的优化模型 (加入按掩码固定加权的损失函数) 有效缓解了“模糊效应”。值得注意的是，测试集中有部分的样本临近预报 CSI 整体呈上升的趋势，后期预报 CSI 趋近于 60%，见图15。

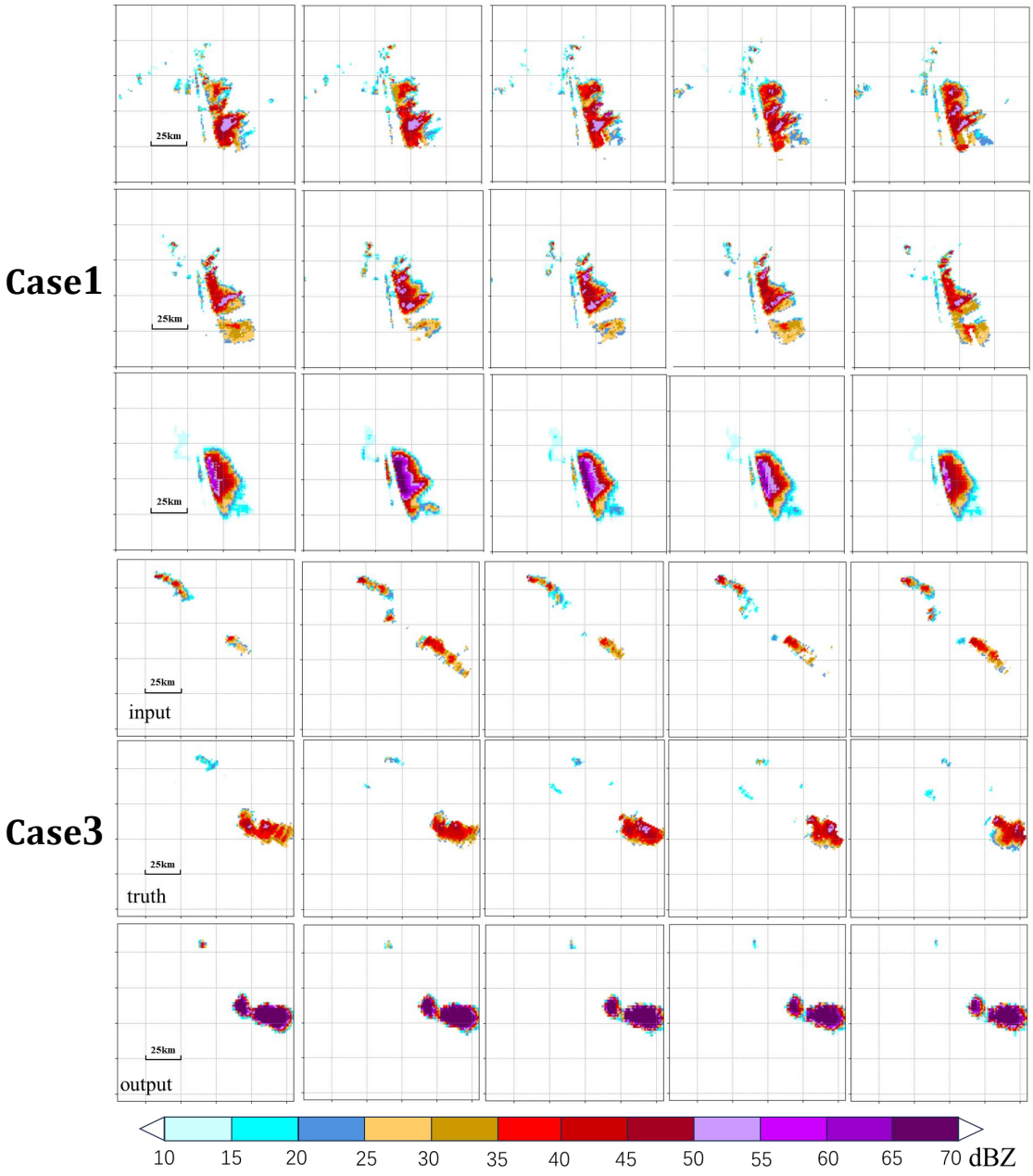


图 13: **Case1** 和 **Case3** 的模型预测结果可视化图

以 **Case1**(上三行) 为例：前一个小时 Z_H 的地面真值 (第一行)；后续一个小时 Z_H 的地面真值 (第二行)；后续一个小时的 Z_H 的地面预测值 (第三行)。对应的 6 分钟 (第一列)、12 分钟 (第二列)；18 分钟 (第三列)；24 分钟 (第四列) 和 30 分钟 (第五列) 预测的临近预报。图中一格表示 25km 的距离。

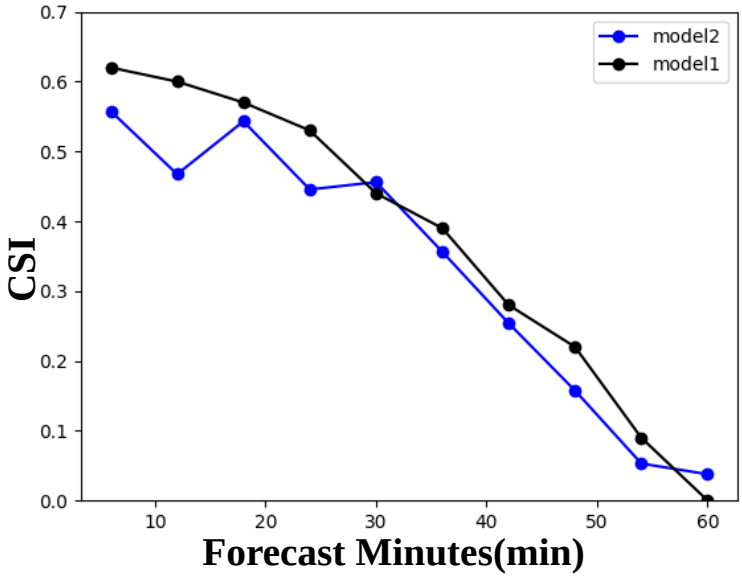


图 14: 问题一模型和问题二 (Case3) 模型临近预报 CSI 对比

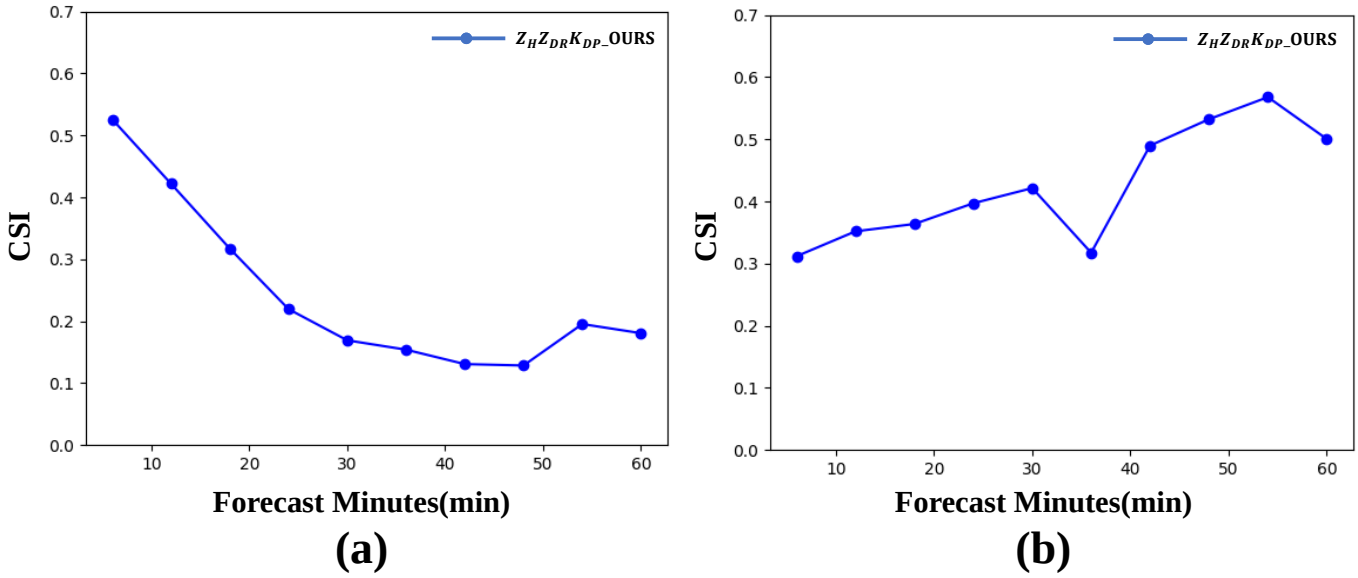


图 15: **Case1** 的模型临近预报 CSI

(a) 一般样本 CSI; (b) 特殊样本预测的 CSI 结果图

由图14可见，按掩码固定加权的损失函数和 SSIM 损失函数的组合模型 **(Case3)** 在**第** 0 – 30 **分钟**时的临近预报 CSI 高于只按掩码固定加权的损失函数模型 **(Case1)**。且更加稳定，**Case3** 的模型在短期强对流预报领域的应用价值高于 **Case1**。

表 4: 问题二的多方案模型评价表

评价指标 / 方案	MAE↓	RMSE↓	SSIM↑	PCC↑
Case1	0.89	4.35	**0.96**	**0.2801**
Case2	2.68	3.98	0.57	0.0043
Case3	0.86	4.45	**0.96**	0.2678
Case4	**0.61**	**3.24**	**0.96**	0.0017

七、 问题三：基于传统模型和深度学习方法的 $Z-R$ 关系

7.1 问题分析

NJU-CPOL 双偏振雷达数据集提供了丰富的样本数据 (Z_H, Z_{DR})，降水格点数据提供了对应样本的降水量。本题分别使用传统方法和深度学习方法来构建模型，利用 Z_H 和 Z_{DR} 进行定量的降水估计，即找到变量 Z_H 和 Z_{DR} 和降水量之间的关系用于估计未来时刻的降水量。

由文献 [2] 中可以得到经验性关系 (Z – R 关系)，那么通过对应的数据可以通过非线性最小二乘的方法拟合出 Z – R 关系，它能在一定程度上通过雷达反射率反映出降水量:

$$R(Z_H, Z_{DR}) = a \cdot Z_H^b \cdot Z_{DR}^c$$

由于神经网络具有逼近任意函数的优秀性质，可以假设存在函数 $f(\cdot)$，可以将 Z_H 和 Z_{DR} 映射到降水量 r 上，可以表示为:

$$r = f(Z_H, Z_{DR})$$

。神经网络模型可以很好地拟合出雷达反射率与降水量之间隐含的关系。同时考虑到数据集的特点 (单通道图像数据)，数据之间具有非常强的空间关联性。因此，本题的第二个模型基于神经网络建立双输入模型拟合函数 $f(\cdot)$，并与传统的非线性最小二乘的方法进行对比。

综上所述，问题三的技术路线图见图16:

7.2 模型建立

由问题分析可知，问题三有两个模型：传统模型和神经网络模型。

7.2.1 传统模型——非线性最小二乘模型

本文使用非线性最小二乘的方法构建模型，用于拟合 Z – R 关系函数 $R(\cdot)$。

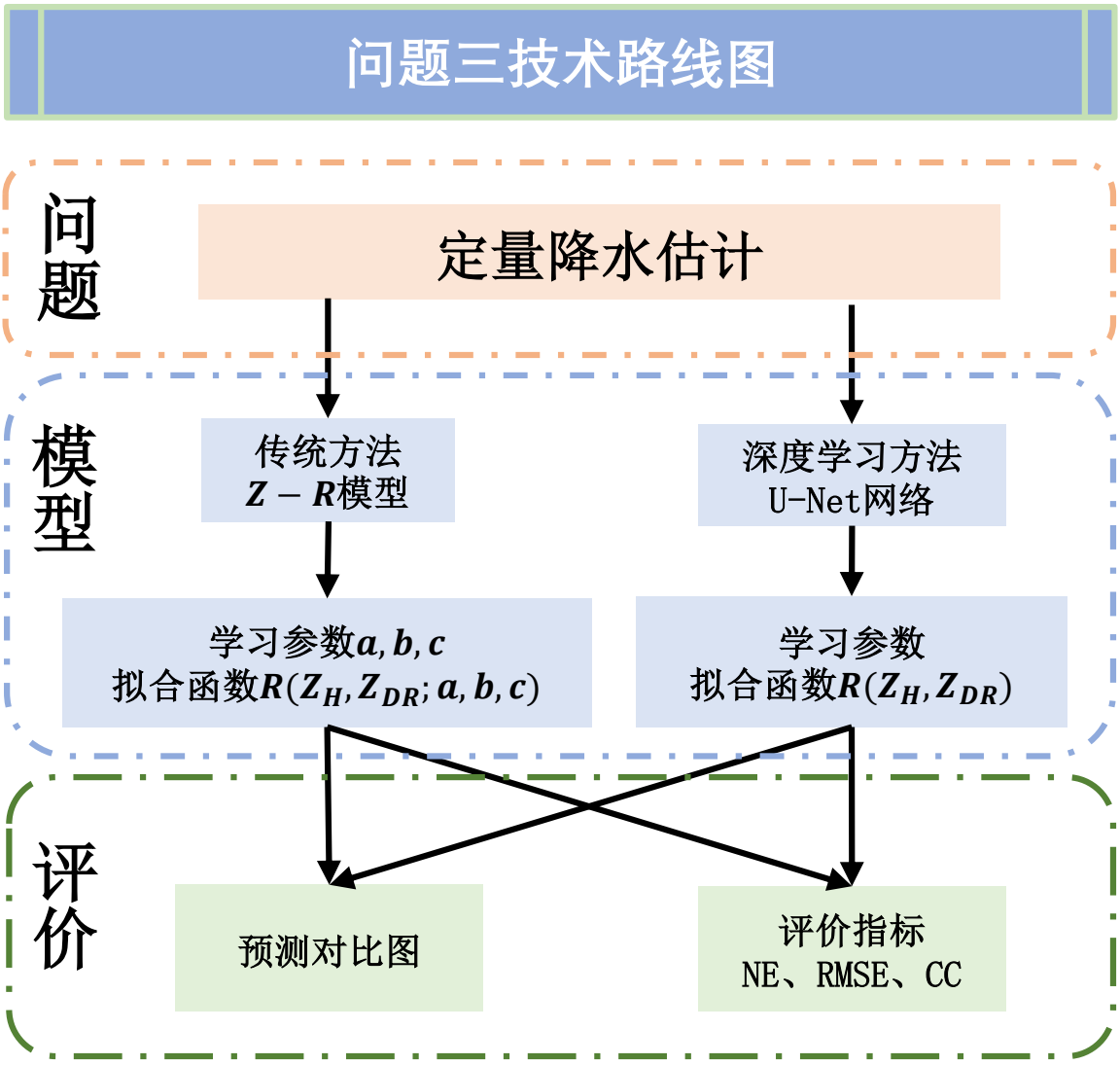


图 16: 问题三的思路流程图

待拟合函数为：

$$R(Z_H, Z_{DR}) = a \cdot Z_H^b \cdot Z_{DR}^c \tag{7.1}$$

目标函数：

$$\min_{a,b,c} = \sum_i \left[r_{truth}^{(i)} - R(Z_H^{(i)}, Z_{DR}^{(i)}; a, b, c) \right]^2 \tag{7.2}$$

其中，实际降水量用 r_{truth} 表示，待拟合函数为 $R(\cdot)$.

7.2.2 神经网络模型

问题三的神经网络模型是基于问题一的模型进行适当修改得到的。网络结构由双支并行的编码结构、SE-BLOCK 和解码结构组成，框架图见图17。网络的详细介绍见问题一的模型建立。问题三的网络表达式为：

$$\mathbf{F_{Net2}}(Z_H, Z_{DR}) = \mathbf{Decode}(\mathbf{SE}(\mathbf{Encode}(Z_H, Z_{DR}))) \tag{7.3}$$

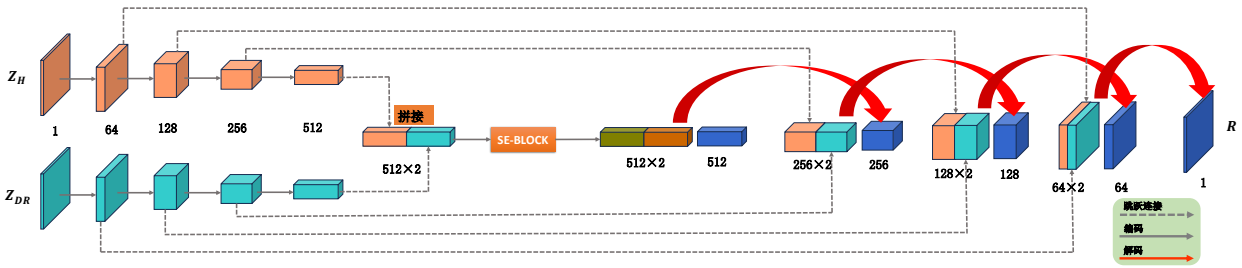


图 17: 问题三模型框架

7.3 模型求解和结果分析

7.3.1 非线性最小二乘模型的求解与结果分析

将 NJU-CPOL 双偏振雷达数据和降水格点数据中海拔高度为 1km 的变量 (Z_H, Z_{DR}) 一一对应起来，通过迭代优化算法，更新参数以减小观测数据与模型之间的误差。

最终模型拟合的效果图见图18。子图 (a) 表示输入数据未进行归一化的效果图，子图 (b) 表示对数据进行归一化后再作为输入的效果图。可见对数据进行归一化的操作可以使模型更好的拟合关系函数 $R(\cdot)$。数据归一化可以将所有特征的取值范围映射到给定区间，进而消除不同特征之间的量纲差异，避免因此而导致拟合的偏差。

7.3.2 神经网络模型的求解与结果分析

将同样的数据输入神经网络模型中，通过反向传播算法，使得网络的输出更加贴近真实值。神经网络模型预测效果见图19。对比传统模型的预测效果图18，神经网络模型的预测结果更准确。

图20的效果图分别是一个样本的水平反射率因子 Z_H(子图 a)、真实降雨量 (子图 b) 和模型预测出的降水量 (子图 c) 可视化。通过色彩对比可见，模型对真实降雨量的估计相近，可以临近预报对流天气以及它的降雨量。

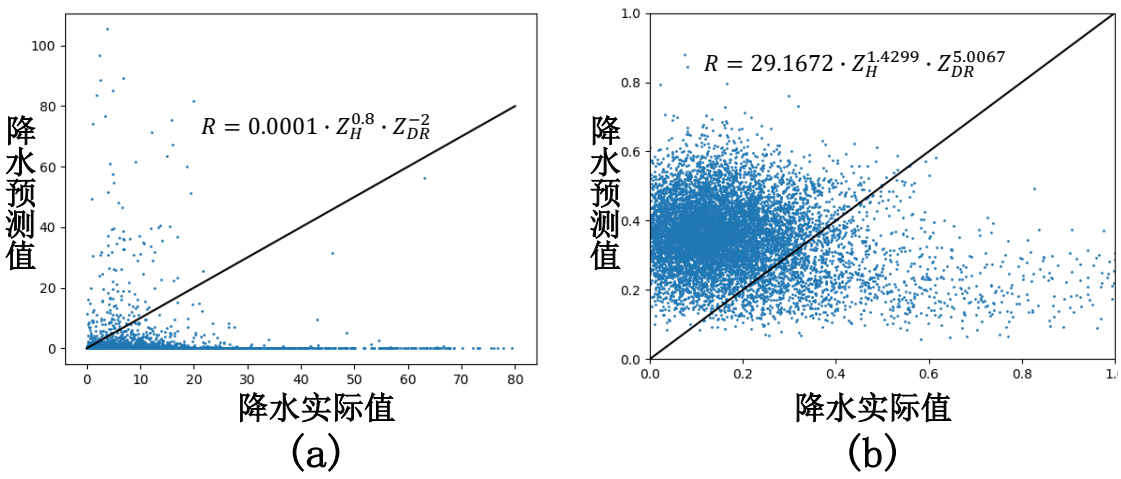


图 18: 非线性最小二乘拟合效果图。
(a) 未归一化数据的效果；(b) 数据归一化后的效果

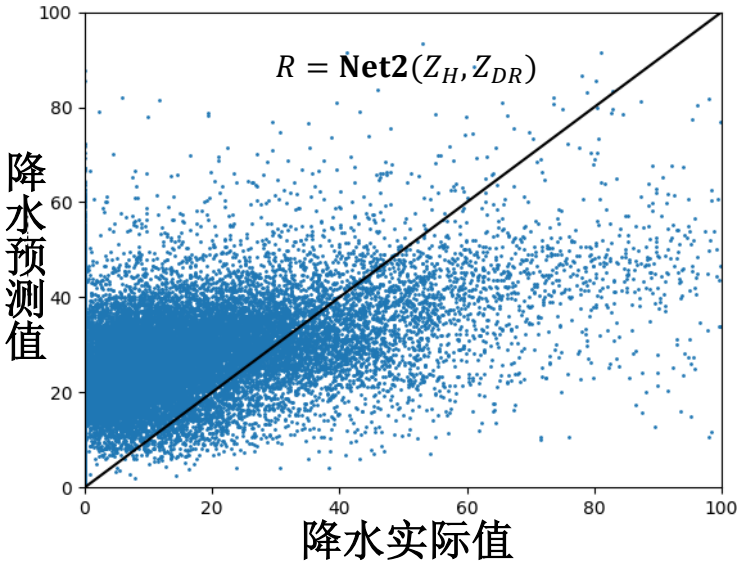


图 19: 神经网络模型预测效果图

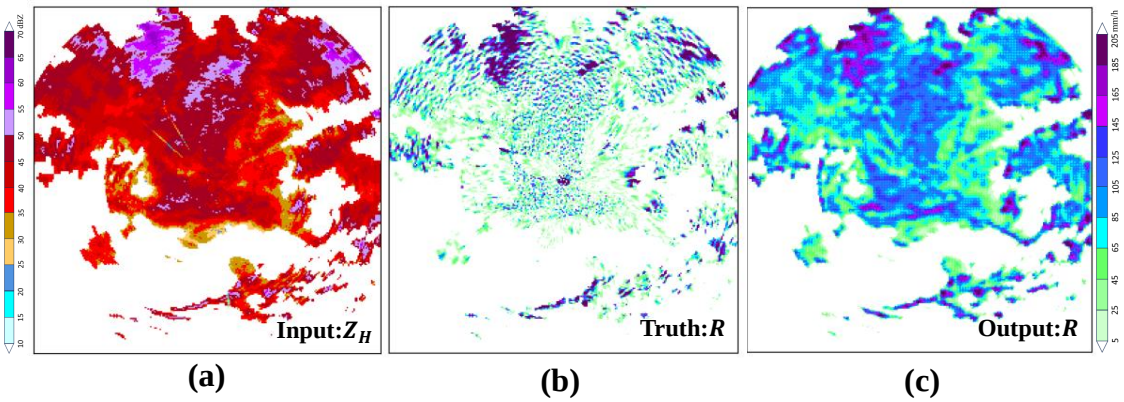


图 20: 神经网络模型预测效果可视化

7.3.3 评价指标

问题三使用的评价指标包括：标准化平均绝对误差 (NE)、均方根误差 (RMSE) 以及相关系数 (CC)。在问题一的评价指标中已经给出了均方根误差 (RMSE) 的详细介绍，下面介绍一下另外两个指标。

(1) **相关系数** (CC)：

在临近预报领域，相关系数 (Correlation Coefficient, CC) 是一种常用的统计指标，用于评估模型预测值与真实值之间的线性相关程度。它可以衡量预报结果与观测数据之间的线性相关程度，从而判断预报的准确性和可靠性，具体解释如下：

* 当 CC 接近于 1 时，表示预报结果与观测数据之间存在较强的正相关关系，这表明预报结果能够很好地反映实际观测情况；
* 当 CC 接近于 -1 时，表示预报结果与观测数据之间存在较强的负相关关系，这意味着预报结果与实际观测数据之间存在相反的变化趋势。
* 当 CC 接近于 0 时，表示预报结果与观测数据之间没有线性关系或关系非常弱。这可能说明预报结果无法准确地反映观测数据的变化，或者两者之间存在非线性的关联关系。

计算公式如下：

$$\mathrm{CC} = \frac{\sum_{i=1}^{\mathrm{N}}(\widehat{Z}_{H;i} - \overline{\widehat{Z}_H})(\mathrm{Z}_{H;i} - \overline{Z_H})}{\sqrt{\sum_{i=1}^{\mathrm{N}}(\widehat{Z}_{H;i} - \overline{\widehat{Z}_H})^2}\sqrt{\sum_{i=1}^{\mathrm{N}}(Z_{H;i} - \overline{Z_H})^2}} \tag{7.4}$$

其中，$\widehat{Z}_{H;,i}$ 表示预测的水平反射率因子；$\overline{\widehat{Z}_H}$ 表示预测的水平反射率因子的均值，$\overline{Z_H}$ 表示真实的水平反射率因子的均值。

(2) **标准化平均绝对误差**：(NE)：

标准化平均绝对误差 (Normalized Absolute Error, NE) 是绝对误差与真实值平均值的比值，它能够评估模型预测结果的平均误差程度。NE 的取值范围为 0 到正无穷，越接近于 0 表示预测误差越小。计算公式如下：

$$\mathrm{NE} = \frac{(\frac{1}{\mathrm{N}})\sum_{i=1}^{\mathrm{N}}(\left|\widehat{Z}_{H;,i} - \overline{Z_H}\right|)}{\mathrm{Z}_{H;i}} \tag{7.5}$$

NE 仅考虑了预报误差的相对大小,并没有考虑误差的方向。一般与均方根误差 (RMSE) 或相关系数 (CC) 同时使用。

7.3.4 总结

问题三通过给定的数据建立模型，以定量估计降水量为目标。本文通过效果图和评价指标来对比非线性最小二乘模型和神经网络模型的性能进行评估。

对比神经网络模型预测的效果图19和非线性最小二乘模型的效果图18，得出结论：神经网络模型比非线性最小二乘模型对降水量的预测更加精确。

使用指标 NE、RSM 和 CC 对问题三的两个模型进行评价，评价结果见表5。得出结论：

(1) 对数据进行归一化后再应用于模型中是必需的；

(2) 卷积神经网络的拟合效果更好。

表 5: 问题三的模型评价表

评价指标 / 模型	NE↓	RMSE↓	CC↑
$Z-R$ 关系模型拟合（未归一化）	0.9976	35.43	-0.0226
Z-R 关系模型拟合 (归一化)	1.721	**0.0717**	0.6829
卷积网络拟合	**0.89**	2.96	**0.73**

八、 问题四：基于 Bi-GFF 模块的特征层面信息融合

8.1 问题分析

在 **SE – BLOCK** 中，通道注意力产生了通道赋权的融合效果：这样操作使得整体特征在通道层面上有了一致的更新，但三个变量的信息融合在特征图的尺度上比较单一：对于一个特征图而言，各个位置的特征同时乘以了相同的系数。这样的信息融合并不充分。

以 Z_H 为例，不同位置的 Z_H 相同 (eg:$Z_H(1,1) = Z_H(1,2) = 1$)，则不论该位置的其它变量是否相等，Z_H 在 **SE – BLOCK** 融合后信息仍相同 (eg:$Z_H(1,1) = Z_H(1,2) = a$)，操作示图见21。这样的结果说明 **SE – BLOCK** 并不能将 Z_H 与其他变量信息充分融合。而 **Bi – GFF** 模块较好地解决了这一问题。

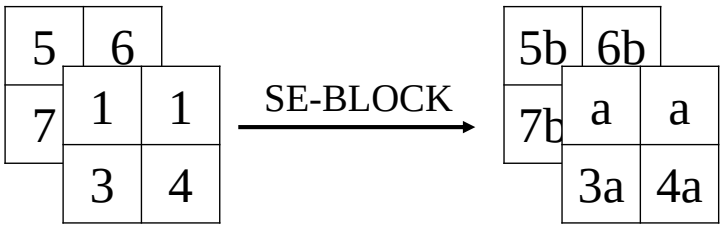


图 21: SE – BLOCK 输入输出示意图

模型四的思路见流程图22所示。**双偏振雷达三个变量的贡献分析：**首先针对网络输入设计一组对照实验，其次使用问题二的网络训练，最后使用评价指标对测试结果进行评估进而分析 Z_H, Z_{DR}, K_{DP} 的贡献。**本文优化数据融合的策略：**对问题一模型中的原始数据融合策略进行改进，引入 **Bi – GFF** 模块得到改进后的问题四模型，训练后观察和评估其预测值。

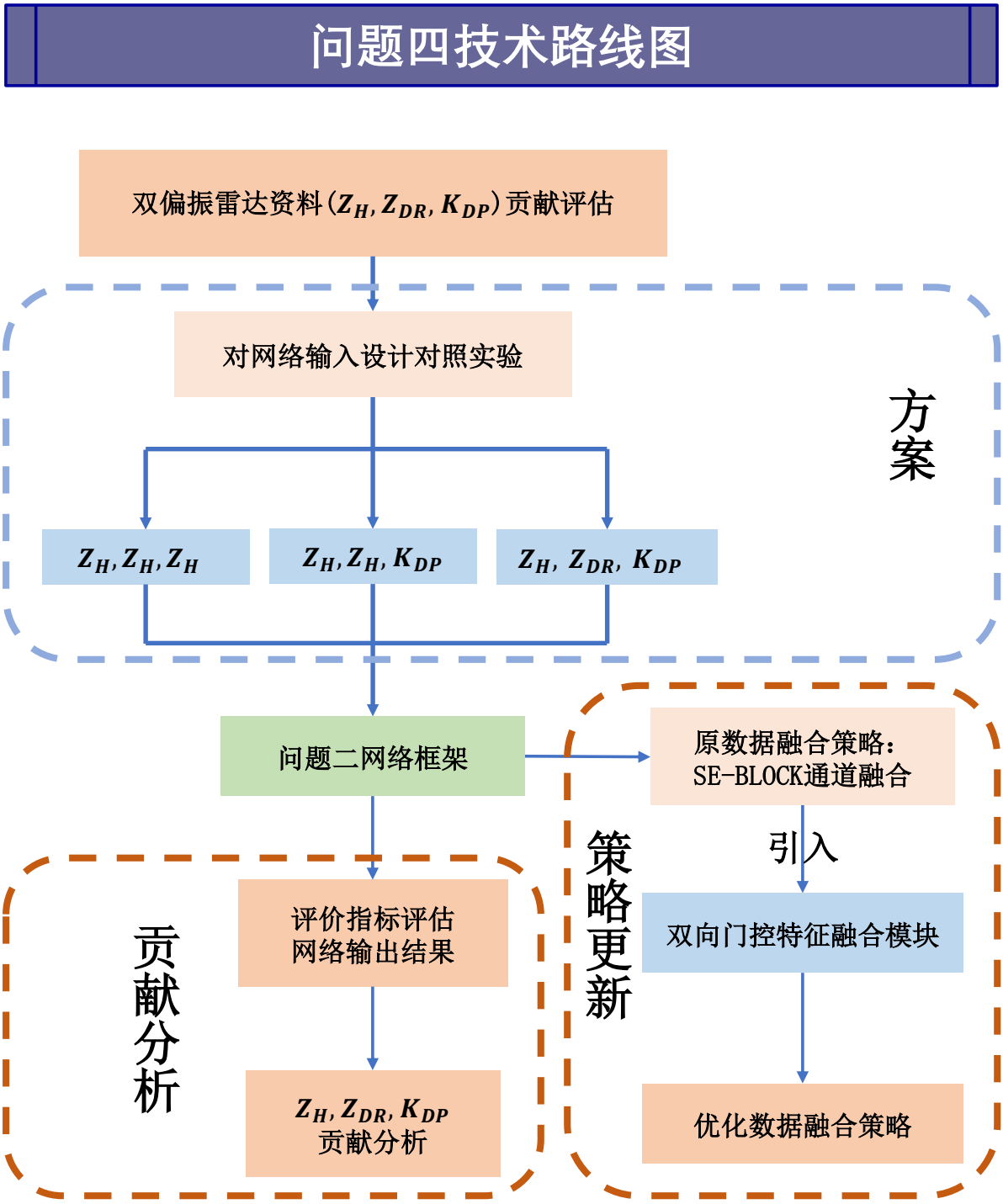


图 22: 模型四的技术路线图

8.2 模型建立

为了更好地融合多个变量的特征，文献 [9] 提出了双向门控特征融合 (Bi – directional Gated Feature 模块见23，对于两个变量 F_1, F_2，该模块在两个特征堆叠后，学习双向的软门 (**soft gate**) 并输出残差信息，与原特征融合。

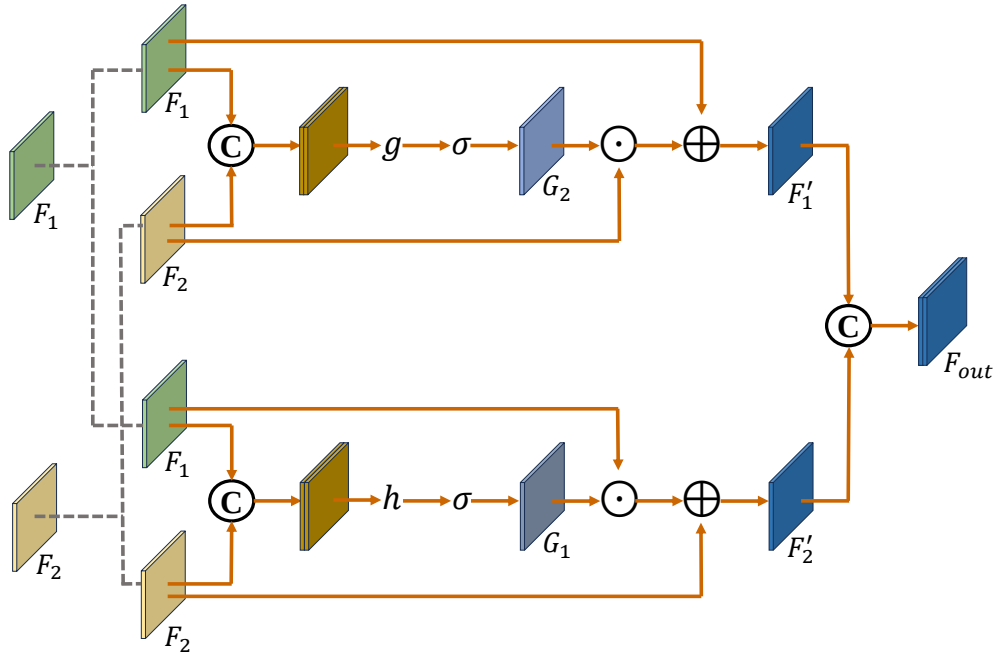


图 23: Bi – GFF 细节示意图

以 F_1 为例，首先将堆叠的特征输入线性层和激活层得到软门：

$$G_2 = \sigma(h(\text{Concat}(F_1, F_2))) \tag{8.1}$$

将软门与 F_2 做点积之后可得到 F_2 的信息，将此信息乘以可学习的权重 α，加到 F_1 上即可完成特征层面上的信息融合:

$$F_1^{'} = F_1 + \alpha(G_2 \odot F_2) \tag{8.2}$$

综上所述，本文的 Bi – GFF 模型为：

$$\begin{aligned} F_1^{'} &= F_1 + \alpha(\sigma(h(\text{Concat}(F_1, F_2)))) \odot F_2 \\ F_2^{'} &= F_2 + \beta(\sigma(g(\text{Concat}(F_1, F_2)))) \odot F_1 \end{aligned} \tag{8.3}$$

值得一提的是，Bi – GFF 在通道层面并未做更多处理，因此 Bi – GFF 和 SE – BLOCK 可以较好地互补，使得融合模型即关注通道间的信息，也关注特征层面的信息。SE – BLOCK 操作示例见24。

问题四的模型就是在问题一模型的基础上嵌入了三个 Bi – GFF 模块。图25展示了三次 Bi – GFF 模块的细节，前两个 Bi – GFF 负责对输入的三个变量的高层语义特征进行交错融合，最后一个 Bi – GFF 则负责将两个不同的融合特征进行二次融合。

最终，问题四的整体模型框架见26，下方深灰色区域突出了问题四模型的新颖之处。

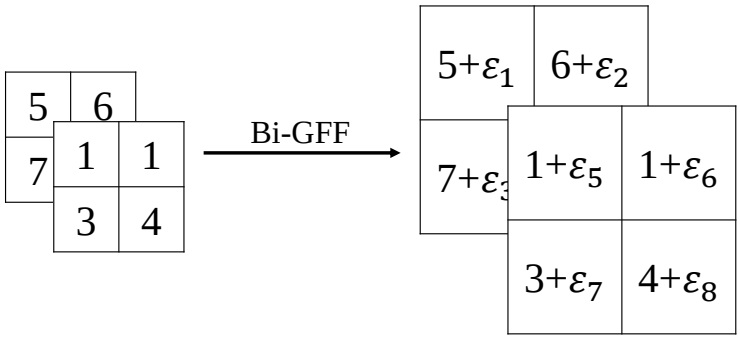


图 24: Bi – GFF 输入输出示意图

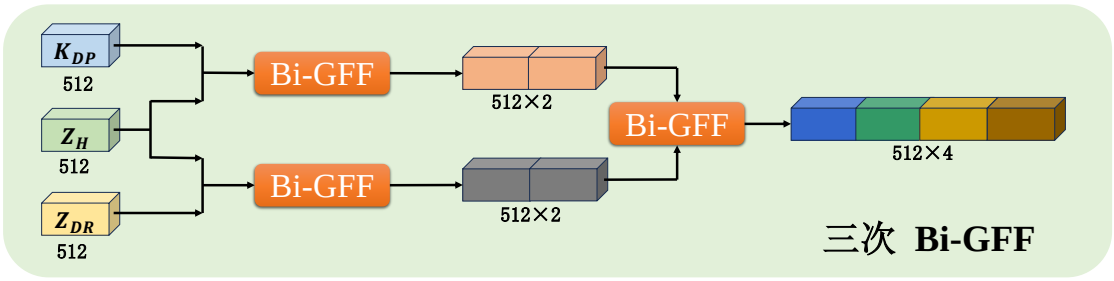


图 25: 三次 Bi – GFF 模块的细节图

三次 Bi – GFF 对输入网络的三个变量在最高层特征语义处进行融合，最后将融合的特征输入 SE – BLOK。

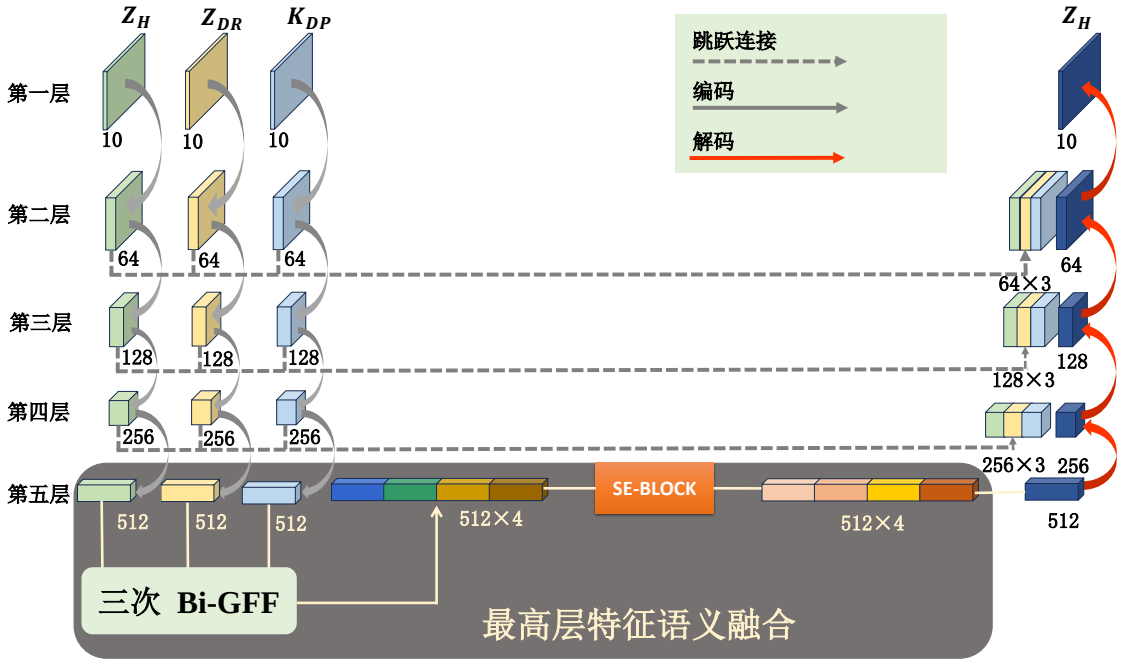


图 26: 问题四整体网络框架

8.3 模型求解和结果分析

为了严格控制影响结果的因素，本模型每个实验结构、参数量保持一致。因此，问题四设置了一组对照实验，即依据网络输入的不同分三组：

(1) 网络输入为：Z_H, Z_H, Z_H;

(2) 网络输入为：Z_H, Z_H, Z_{DP};

(3) 网络输入为：Z_H, Z_{DP}, K_{DP};

三组对照实验使用问题二的模型 (按掩码固定加权损失模型) 进行训练，某个时间点的临近预报可视化见图27。根据子图 (b),(c),(d) 中颜色分布可以看出使用特征融合的模型预测结果更贴近真实样本，融合双偏振雷达变量在强对流降水临近预报中起着关键性作用。

为了更加合理的评价模型，将实验结果绘成了折线图，折线图简明的展示了模型在临近预报的效果。图28中特征融合的模型临近预报是最准确的，其次是未特征融合的模型。由于训练不稳定，当网络只输入两个变量 (Z_H 和 K_{DP}) 时，评价指标 CSI 为 0.

8.3.1 总结

通过对问题四模型的求解和分析得出表6，分析表6可以得到以下两个结论：

(1) **对比 PCC 可以发现，当输入变量种类较少时，训练不稳定，因此多种双偏振雷达变量的设置对提高网络性能起到至关重要的作用。**

(2) **双向门控特征融合模型可以更充分的融合三种变量的信息，因此可以大大的提升网络的性能和稳定性。**

表 6: 问题四模型的评价表

评价指标 / 方案	MSE↓	RMSE↓	SSIM↑	PCC↑
Z_H, Z_H, Z_H	0.75	3.24	0.96	0.0055
Z_H, Z_H, K_{DR}	0.62	3.24	0.96	0.0060
Z_H, Z_{DR}, K_{DP}	0.89	4.35	0.96	0.2801
双向门控特征融合	**0.51**	**2.78**	**0.97**	**0.29**

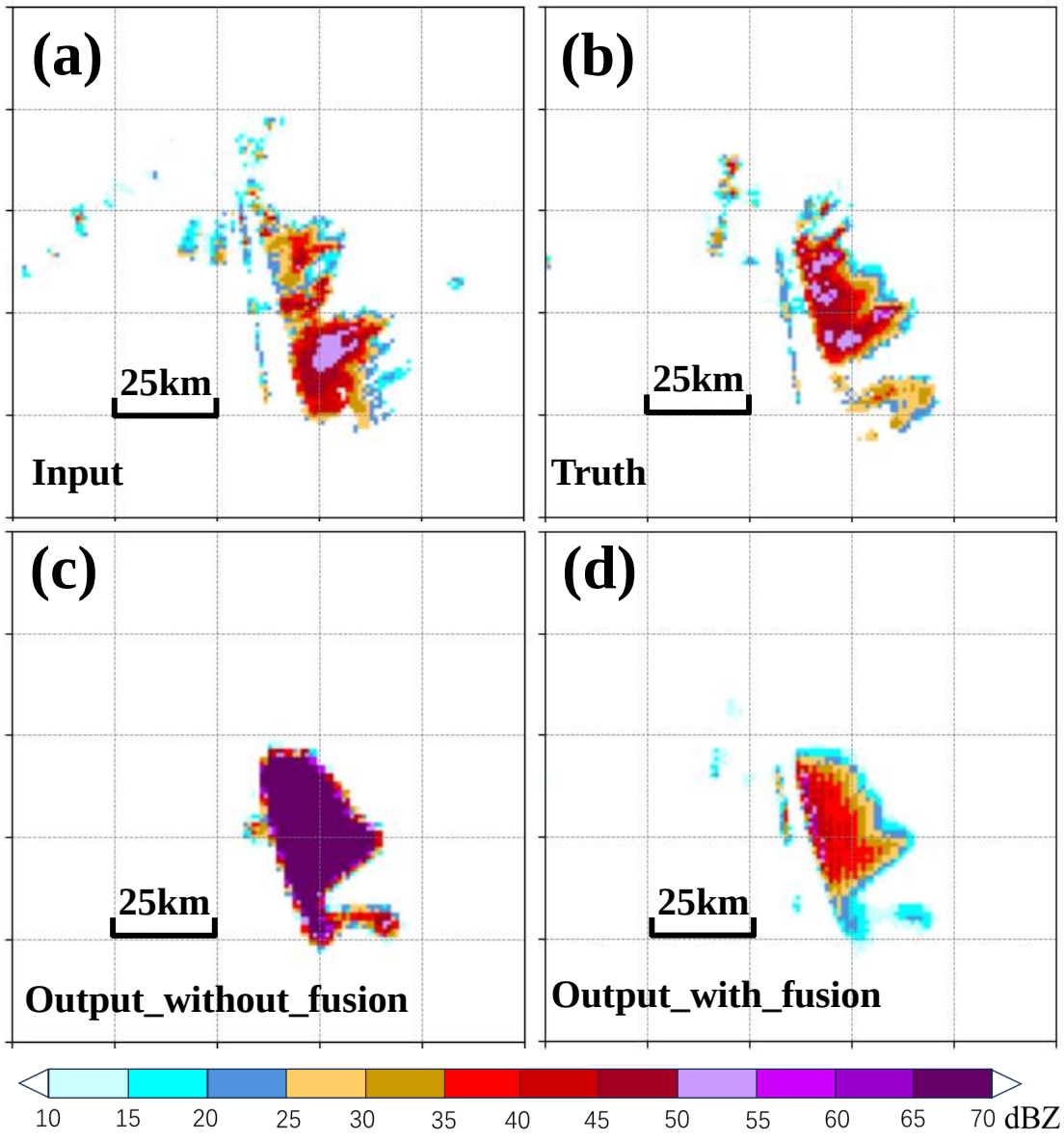


图 27: 问题四模型某个时间点的临近预报可视化

(a) 前一个小时 Z_H 可视化；(b) 后续一个小时的 Z_H 可视化；(c) 未使用特征融合模型的后续一小时 Z_H 预测可视化；(d) 使用了特征融合模型的后续一小时 Z_H 预测可视化

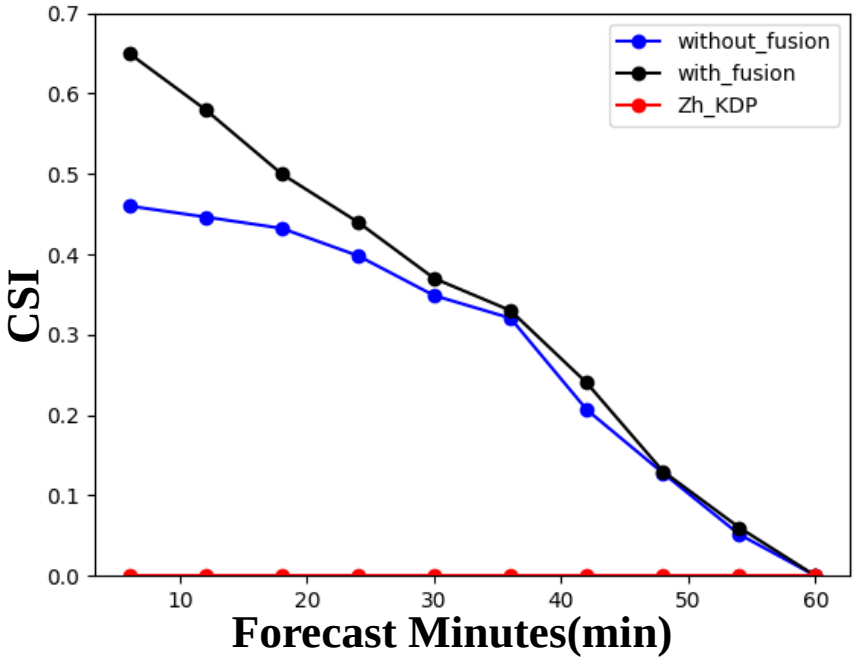


图 28: 问题四模型临近预报结果对比图

九、 模型评价

9.1 模型优点

(1) U-net 网络是由编码器和解码器组合而成。这种编码解码一体化的方法，使模型在处理 NJU-CPOL 双偏振雷达数据时具有更强的表达能力和预测准确性。

(2) 编码器通过特征的提取将 NJU-CPOL 双偏振雷达数据中的关键特征进行提取。利用卷积和池化操作，编码器可以捕捉不同尺度的特征，如降雨的强度、类型、粒子形态以及液态水含量等，从而增强模型对降雨现象的感知能力。

(3) 解码器运用反卷积和上采样操作，将特征融合与分析，将低层次的雷达回波特征和高层次的降水类型信息结合起来。这种特征融合有助于提升模型对不同降雨现象的感知能力，为降雨临近预报任务提供全面、多角度的降水分析结果。

(4) 本文的加权损失模型提升了对强对流天气的预警效果和预测结果的真实性。

(5) 本文的融合模型也充分地融合了不处于相同语义上的特征。

9.2 模型缺点

(1) 由于时间限制，本文的网络模型层数较浅，并且各超参数的取值仍待讨论。

(2) 在加入 **Bi – GFF** 模块前，模型训练的稳定性一般，一般需要多次训练并微调超参数。加入 **Bi – GFF** 模块后，模型的参数量大幅上升，减缓了训练的速度。

9.3 模型的推广

尽管参数量增加，本文交替的 **Bi – GFF** 模块和 **SE – BLOCK** 在多输入的特征融合方面仍然展现出强大的性能。作为即插即用模块，可以用于很多图像领域的任务。本文同时也验证了 **Bi – GFF** 模块并不局限于两个特征的输入，对于多特征的输入也能起到效果。

参考文献

[1] 郑永光, 张小玲, 周庆亮等. 强对流天气短时临近预报业务技术进展与挑战 [J]. 气象,2010,36(07):33-42.

[2] Chen G, Zhao K, Zhang G, et al. Improving polarimetric C-band radar rainfall estimation with two-dimensional video disdrometer observations in Eastern China[J]. Journal of Hydrometeorology, 2017, 18(5): 1375-1391.

[3] Rinehart R E, Garvey E T. Three-dimensional storm motion detection by conventional weather radar[J]. Nature, 1978, 273(5660): 287-289.

[4] Pan X, Lu Y, Zhao K, et al. Improving Nowcasting of convective development by incorporating polarimetric radar variables into a deep-learning model[J]. Geophysical Research Letters, 2021, 48(21): e2021GL095302.

[5] Ronneberger O, Fischer P, Brox T. U-net: Convolutional networks for biomedical image segmentation[C]//Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18. Springer International Publishing, 2015: 234-241.

[6] Hu J, Shen L, Sun G. Squeeze-and-excitation networks[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2018: 7132-7141.

[7] Tran Q K, Song S. Computer vision in precipitation nowcasting: Applying image quality assessment metrics for training deep neural networks[J]. Atmosphere, 2019, 10(5): 2 4.

[8] Xiang H, Zou Q, Nawaz M A, et al. Deep learning for image inpainting: A survey[J]. Pattern Recognition, 2023, 134: 109046.

[9] Guo X, Yang H, Huang D. Image inpainting via conditional texture and structure dual generation[C]//Proceedings of the IEEE/CVF International Conference on Computer Vision. 2021: 14134-14143.

附录 A 主程序源代码

1.1 代码：

```
"""
Created on Mon Sep 25 19:43:34 2023

@author: Administrator
"""


import torch
import torch.nn as nn
import os
import numpy as np
import matplotlib.pyplot as plt
import torch.nn.functional as F
import shutil
import torchvision

import math

from PIL import Image
# 定义 Squeeze-and-Excitation Block
class SEBlock(nn.Module):
def __init__(self, in_channels, reduction_ratio=16):
super(SEBlock, self).__init__()
self.avg_pool = nn.AdaptiveAvgPool2d(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, _, _ = x.size()
y = self.avg_pool(x).view(b, c)  # 全局平均池化
y = self.fc(y).view(b, c, 1, 1)  # 全连接层
return x * y.expand_as(x)  # 乘法操作，将通道注意力应用
   到输入特征图上
class BiGFF(nn.Module):
'''Bi-directional Gated Feature Fusion.'''

def __init__(self, in_channels, out_channels):
super(BiGFF, self).__init__()

self.structure_gate = nn.Sequential(
nn.Conv2d(in_channels=in_channels + in_channels,
   out_channels=out_channels, kernel_size=3, stride=1,
```

```
            padding=1),
        nn.Sigmoid()
        )
        self.texture_gate = nn.Sequential(
        nn.Conv2d(in_channels=in_channels + in_channels,
            out_channels=out_channels, kernel_size=3, stride=1,
            padding=1),
        nn.Sigmoid()
        )
        self.structure_gamma = nn.Parameter(torch.zeros(1))
        self.texture_gamma = nn.Parameter(torch.zeros(1))

        def forward(self, texture_feature, structure_feature):

        energy = torch.cat((texture_feature, structure_feature)
            , dim=1)

        gate_structure_to_texture = self.structure_gate(energy)
        gate_texture_to_structure = self.texture_gate(energy)

        texture_feature = texture_feature + self.texture_gamma
            * (gate_structure_to_texture * structure_feature)
        structure_feature = structure_feature + self.
            structure_gamma * (gate_texture_to_structure *
            texture_feature)

        return torch.cat((texture_feature, structure_feature),
            dim=1)
        # 定义 UNet 模型
        class UNet(nn.Module):
        def __init__(self, in_channels=10, out_channels=10):
        super(UNet, self).__init__()

        # 编码器部分
        self.encoder1_dBZ = nn.Sequential(
        nn.Conv2d(in_channels, 64, kernel_size=3, stride=1,
            padding=1),
        # nn.BatchNorm2d(64),
        nn.ReLU(inplace=True),
        nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
        # nn.BatchNorm2d(64),
        nn.ReLU(inplace=True),
        nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.encoder2_dBZ = nn.Sequential(
        nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
        # nn.BatchNorm2d(128),
        nn.ReLU(inplace=True),
        nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1)
            ,
```

```
            # nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2)
            )
            self.encoder3_dBZ = nn.Sequential(
            nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1)
                ,
            # nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1)
                ,
            # nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2)
            )
            self.encoder4_dBZ = nn.Sequential(
            nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1)
                ,
            # nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1)
                ,
            # nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2)
            )

            self.encoder1_KDP = nn.Sequential(
            nn.Conv2d(in_channels, 64, kernel_size=3, stride=1,
               padding=1),
            # nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
            # nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2)
            )
            self.encoder2_KDP = nn.Sequential(
            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            # nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1)
                ,
            # nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2)
            )
            self.encoder3_KDP = nn.Sequential(
            nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1)
                ,
```

```
                # nn.BatchNorm2d(256),
                nn.ReLU(inplace=True),
                nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1)
                    ,
                # nn.BatchNorm2d(256),
                nn.ReLU(inplace=True),
                nn.MaxPool2d(kernel_size=2, stride=2)
                )
                self.encoder4_KDP = nn.Sequential(
                nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1)
                    ,
                # nn.BatchNorm2d(512),
                nn.ReLU(inplace=True),
                nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1)
                    ,
                # nn.BatchNorm2d(512),
                nn.ReLU(inplace=True),
                nn.MaxPool2d(kernel_size=2, stride=2)
                )

                self.encoder1_ZDR = nn.Sequential(
                nn.Conv2d(in_channels, 64, kernel_size=3, stride=1,
                    padding=1),
                # nn.BatchNorm2d(64),
                nn.ReLU(inplace=True),
                nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
                # nn.BatchNorm2d(64),
                nn.ReLU(inplace=True),
                nn.MaxPool2d(kernel_size=2, stride=2)
                )
                self.encoder2_ZDR = nn.Sequential(
                nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
                # nn.BatchNorm2d(128),
                nn.ReLU(inplace=True),
                nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1)
                    ,
                # nn.BatchNorm2d(128),
                nn.ReLU(inplace=True),
                nn.MaxPool2d(kernel_size=2, stride=2)
                )
                self.encoder3_ZDR = nn.Sequential(
                nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1)
                    ,
                # nn.BatchNorm2d(256),
                nn.ReLU(inplace=True),
                nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1)
                    ,
                # nn.BatchNorm2d(256),
                nn.ReLU(inplace=True),
                nn.MaxPool2d(kernel_size=2, stride=2)
```

```
        )
        self.encoder4_ZDR = nn.Sequential(
        nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1)
            ,
        # nn.BatchNorm2d(512),
        nn.ReLU(inplace=True),
        nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1)
            ,
        # nn.BatchNorm2d(512),
        nn.ReLU(inplace=True),
        nn.MaxPool2d(kernel_size=2, stride=2)
        )


        # 解码器部分
        #1. 融合ZH、KDP
        #2. 融合ZH、ZDR
        #3. 融合1,2

        self.GFF_1 = BiGFF(512, 512)
        self.GFF_2 = BiGFF(512, 512)
        self.GFF_3 = BiGFF(1024, 1024)

        self.out_up1=nn.Sequential(SEBlock(in_channels=512*4),
        nn.Conv2d(512*4, 512, kernel_size=1),
        )
        self.decoder_1 = nn.Sequential(
        nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1)
            ,
        nn.ReLU(inplace=True),
        nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1)
            ,
        nn.ReLU(inplace=True),
        nn.ConvTranspose2d(512, 256, kernel_size=2, stride=2)
        )
        self.decoder_2 = nn.Sequential(
        nn.Conv2d(256*4, 256*4, kernel_size=3, stride=1,
            padding=1),
        nn.ReLU(inplace=True),
        nn.Conv2d(256*4, 256*4, kernel_size=3, stride=1,
            padding=1),
        nn.ReLU(inplace=True),
        nn.ConvTranspose2d(256*4, 128, kernel_size=2, stride=2)
        )
        self.decoder_3 = nn.Sequential(
        nn.Conv2d(128*4, 128*4, kernel_size=3, stride=1,
            padding=1),
        nn.ReLU(inplace=True),
        nn.Conv2d(128*4, 128*4, kernel_size=3, stride=1,
            padding=1),
```

```
            nn.ReLU(inplace=True),
            nn.ConvTranspose2d(128*4, 64, kernel_size=2, stride=2)
            )
            self.decoder_4 = nn.Sequential(
            nn.Conv2d(64*4, 64*4, kernel_size=3, stride=1, padding
               =1),
            nn.ReLU(inplace=True),
            nn.Conv2d(64*4, 64*4, kernel_size=3, stride=1, padding
               =1),
            nn.ReLU(inplace=True),
            nn.ConvTranspose2d(64*4, 10, kernel_size=2, stride=2)
            )

            def forward(self, dBZ, ZDR, KDP):
            # 编码器
            dBZ1 = self.encoder1_dBZ(dBZ)
            dBZ2 = self.encoder2_dBZ(dBZ1)
            dBZ3 = self.encoder3_dBZ(dBZ2)
            dBZ4 = self.encoder4_dBZ(dBZ3)

            ZDR1 = self.encoder1_ZDR(ZDR)
            ZDR2 = self.encoder2_ZDR(ZDR1)
            ZDR3 = self.encoder3_ZDR(ZDR2)
            ZDR4 = self.encoder4_ZDR(ZDR3)

            KDP1 = self.encoder1_KDP(KDP)
            KDP2 = self.encoder2_KDP(KDP1)
            KDP3 = self.encoder3_KDP(KDP2)
            KDP4 = self.encoder4_KDP(KDP3)

            fusion_1 = self.GFF_1(dBZ4,KDP4)
            fusion_2 = self.GFF_2(dBZ4,ZDR4)
            fusion_3 = self.GFF_3(fusion_1, fusion_2)

            out = self.out_up1(fusion_3)
            out = self.decoder_1(out)
            concat_skip = torch.cat((dBZ3, ZDR3, KDP3, out), dim=1)
            out = self.decoder_2(concat_skip)
            concat_skip = torch.cat((dBZ2, ZDR2, KDP2, out), dim=1)
            out = self.decoder_3(concat_skip)
            concat_skip = torch.cat((dBZ1, ZDR1, KDP1, out), dim=1)
            out = self.decoder_4(concat_skip)



            return out

            #定义判别器
            class Discriminator(nn.Module):
            def __init__(self, in_channels=10):
```

```
        super(Discriminator, self).__init__()
        self.main = nn.Sequential(
        nn.Conv2d(in_channels, 32, kernel_size=4, stride=2,
            padding=1), #feature_size = 128
        nn.LeakyReLU(),
        nn.Conv2d(32, 64, kernel_size=4, stride=2, padding=1),
            #feature_size = 64
        nn.LeakyReLU(),
        nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1),
             #feature_size = 32
        nn.LeakyReLU(),
        nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1)
            , #feature_size = 16
        nn.LeakyReLU(),
        nn.Conv2d(256, 512, kernel_size=4, stride=2, padding=1)
            , #feature_size = 8
        nn.LeakyReLU(),
        nn.Conv2d(512, 1, kernel_size=8, stride=1, padding=0),
            #feature_size = 1
        nn.Sigmoid())

        def forward(self, x):
        x = self.main(x)
        return x



        def count_hits(tensor1, tensor2, threshold=35): #hits计
            数
        tensor1_tf = (tensor1[0]>=threshold).type(torch.int)
        tensor2_tf = (tensor2[0]>=threshold).type(torch.int)
        hits = (tensor1_tf==1) & (tensor2_tf==1)
        return hits.type(torch.int)

        def count_misses(tensor1, tensor2, threshold=35):
        tensor1_tf = (tensor1[0]>=threshold).type(torch.int)
        tensor2_tf = (tensor2[0]>=threshold).type(torch.int)
        misses = (tensor1_tf==1) & (tensor2_tf==0)
        return misses.type(torch.int)

        def count_false_alarms(tensor1, tensor2, threshold=35):
        tensor1_tf = (tensor1[0]>=threshold).type(torch.int)
        tensor2_tf = (tensor2[0]>=threshold).type(torch.int)
        false_alarms = (tensor1_tf==0) & (tensor2_tf==1)

        return false_alarms.type(torch.int)

        def calculate_CSI(tensor1, tensor2, threshold=35):#按全
            图计算CSI
        #tensor 1 是gt, tensor2是预测
```

```
csi=[]
for i in range(10):
tensor1_tf = (tensor1[0][i]>=35).type(torch.int)
tensor2_tf = (tensor2[0][i]>=35).type(torch.int)
hits = torch.sum((tensor1_tf==1) & (tensor2_tf==1))
misses = torch.sum(tensor1_tf)-hits
false_alarms = torch.sum(tensor2_tf)-hits
csi.append((hits/(hits + misses + false_alarms + torch.
   finfo(torch.float32).eps)).item())

return csi

def cal_ppc(x, y):
# 将张量扩展为相同的形状
x = x.view(1, -1)
y = y.view(1, -1)

# 计算张量的均值
x_mean = torch.mean(x)
y_mean = torch.mean(y)

# 减去均值
x = x - x_mean
y = y - y_mean

# 计算归一化后的余弦相似度
cos_sim = F.cosine_similarity(x, y, dim=1)

# 将余弦相似度转换为皮尔逊相关系数
pearson_corr = cos_sim
return pearson_corr
# def calculate_CSI(tensor1, tensor2, threshold=35):
#     #tensor 1 是gt, tensor2是预测

#     csi=np.array([])
#     for i in range(10):
#         tensor1_tf = (tensor1[0][i]>=35).type(torch.
   int)
#         tensor2_tf = (tensor2[0][i]>=35).type(torch.
   int)
#         hits = torch.sum((tensor1_tf==1) & (
   tensor2_tf==1))

#         misses = torch.sum(tensor1_tf)-hits
#         false_alarms = torch.sum(tensor2_tf)-hits
#         csi = np.hstack((csi, np.array([(hits/(hits +
    misses + false_alarms)).item()])))

#     return csi
```

```
root = 'D:/数模研赛/研赛/2023年中国研究生数学建模竞赛赛
    题/F题/NJU_CPOL_update2308'

root1 = os.listdir(root)    # 读取文件夹里面的所有文件
    名
root2 = os.listdir(os.path.join(root, root1[0]))    #
    海拔
root3 = os.listdir(os.path.join(root, root1[0], root2
    [0]))    # 样本

# 数据处理，将非强对流天气过滤掉

def is_rainy(root4_dBZ,Ithreshold=35, Athreshold=256):
dBZ_file = os.listdir(root4_dBZ)
dBZ=torch.cat([torch.from_numpy(np.load(os.path.join(
    root4_dBZ,dBZ_file[len(dBZ_file)-10+j] )).astype(np.
    float16)).unsqueeze(0) for j in range(10)], dim=0)
A_evaluation_list = []
for i in range(10):
A_evaluation_list.append(torch.sum(dBZ[i]>=Ithreshold).
    numpy())
if max(A_evaluation_list)>=Athreshold:
return True
else:
return False

for i in range(len(root3),0,-1):
root_4_dBZ=os.path.join(root, root1[0], root2[0], root3
    [i-1])
root_4_KDP=os.path.join(root, root1[1], root2[0], root3
    [i-1])
root_4_ZDR=os.path.join(root, root1[2], root2[0], root3
    [i-1])
if not is_rainy(root_4_dBZ):

if shutil.os.path.exists(root_4_dBZ):
try:

shutil.rmtree(root_4_dBZ)
shutil.rmtree(root_4_KDP)
shutil.rmtree(root_4_ZDR)
# print("文件夹删除成功")
except OSError as e:
print(f"删除文件夹时出错: {e}")
else:
print("文件夹不存在")
# 使用shutil模块的rmtree()函数删除文件夹及其内容
# 请确保在删除文件夹之前备份或移动重要文件
root3 = os.listdir(os.path.join(root, root1[0], root2
    [0]))    # 样本
```

```
dBZ_dataset=[]
ZDR_dataset=[]
KDP_dataset=[]
gt_dataset=[]
for i in range(len(root3)):
root4_dBZ=os.path.join(root, root1[0], root2[0], root3[
    i])
dBZ_file = os.listdir(root4_dBZ)
if len(dBZ_file)>=20: #当文件数大于20时开始读取三种变量
    的文件
root4_ZDR=os.path.join(root, root1[2], root2[0], root3[
    i])
ZDR_file = os.listdir(root4_ZDR)
root4_KDP=os.path.join(root, root1[1], root2[0], root3[
    i])
KDP_file = os.listdir(root4_KDP)

for i in range (len(dBZ_file)//10-1):
dBZ_dataset.append(torch.cat([torch.from_numpy(np.load(
    os.path.join(root4_dBZ,dBZ_file[10*i+j] )).astype(np
    .float16)).unsqueeze(0).unsqueeze(0) for j in range
    (10)], dim=1))
ZDR_dataset.append(torch.cat([torch.from_numpy(np.load(
    os.path.join(root4_ZDR,ZDR_file[10*i+j] )).astype(np
    .float16)).unsqueeze(0).unsqueeze(0) for j in range
    (10)], dim=1))
KDP_dataset.append(torch.cat([torch.from_numpy(np.load(
    os.path.join(root4_KDP,KDP_file[10*i+j] )).astype(np
    .float16)).unsqueeze(0).unsqueeze(0) for j in range
    (10)], dim=1))
gt_dataset.append(torch.cat([torch.from_numpy(np.load(
    os.path.join(root4_dBZ,dBZ_file[10*(i+1)+j] )).
    astype(np.float16)).unsqueeze(0).unsqueeze(0) for j
    in range(10)], dim=1))


#定义加权损失函数和优化器
# lossfunc = torch.nn.MSELoss()
# l1 = nn.L1Loss(reduction='sum')
l1 = nn.MSELoss(reduction='sum')

#定义ssim损失, 权重为1.2(l1损失的1/5)
# def compute_ssim(tensor1, tensor2):
#     # 将图像范围调整为 [0, 1]
#     tensor1 = tensor1 / tensor1.max()
#     tensor2 = tensor2 / tensor2.max()

#     # 计算每个通道的 SSIM
#     ssim_scores = []
```

```
#     for i in range(tensor1.shape[1]):
#         channel_ssim = torchvision.functional.ssim(
    tensor1[:, i, :, :], tensor2[:, i, :, :], data_range
    =1.0)
#         ssim_scores.append(channel_ssim)

#     # 求平均得到最终的 SSIM 分数
#     avg_ssim = torch.mean(torch.stack(ssim_scores))

#     return avg_ssim
def ssim(tensor1, tensor2, window_size=11, sigma=1.5,
    data_range=255):
K1 = 0.01
K2 = 0.03

if not tensor1.size() == tensor2.size():
raise ValueError("输入张量的形状必须相同")

_, channels, height, width = tensor1.size()
window = create_window(window_size, channels).to(
    tensor1.device)

mu1 = F.conv2d(tensor1, window, padding=window_size//2,
     groups=channels)
mu2 = F.conv2d(tensor2, window, padding=window_size//2,
     groups=channels)

mu1_sq = mu1.pow(2)
mu2_sq = mu2.pow(2)
mu1_mu2 = mu1 * mu2

sigma1_sq = F.conv2d(tensor1 * tensor1, window, padding
    =window_size//2, groups=channels) - mu1_sq
sigma2_sq = F.conv2d(tensor2 * tensor2, window, padding
    =window_size//2, groups=channels) - mu2_sq
sigma12 = F.conv2d(tensor1 * tensor2, window, padding=
    window_size//2, groups=channels) - mu1_mu2

C1 = (K1 * data_range) ** 2
C2 = (K2 * data_range) ** 2

ssim_map = ((2 * mu1_mu2 + C1) * (2 * sigma12 + C2)) /
    ((mu1_sq + mu2_sq + C1) * (sigma1_sq + sigma2_sq +
    C2))

return ssim_map.mean()


def gaussian(window_size, sigma):
gauss = torch.Tensor([math.exp(-(x - window_size//2)
```

```
    **2/float(2*sigma**2)) for x in range(window_size)])
return gauss / gauss.sum()


def create_window(window_size, channels):
_1D_window = gaussian(window_size, 1.5).unsqueeze(1)
_2D_window = _1D_window.mm(_1D_window.t()).float().
    unsqueeze(0).unsqueeze(0)
window = _2D_window.expand(channels, 1, window_size,
    window_size).contiguous()
return window
# CLASS torch.nn.CrossEntropyLoss(weight: Optional[
    torch.Tensor] = None, size_average=None,
# ignore_index: int = -100, reduce=None, reduction: str
     = 'mean')
if torch.cuda.is_available():
# 设置设备为第一块 GPU
device = torch.device("cuda:0")
else:
device = torch.device("cpu")

#定义模型、判别器、感知器
model=UNet().to(device)
dis_model = Discriminator().to(device)



Learning_rate=0.001
# optimizer = torch.optim.SGD(params = model.parameters
    (), lr = Learning_rate)
# opt_Momentum = torch.optim.SGD(model.parameters(),lr=
    Learning_rate,momentum=0.8,nesterov=True)
# #nesterov拿着上一步的速度先走一小步，再看当前的梯度然
    后再走一步。如果=false就是简单冲量
# opt_RMSprop = torch.optim.RMSprop(model.parameters(),
    lr=Learning_rate,alpha=0.9)#1
optimizer = torch.optim.Adam(model.parameters(),lr=
    Learning_rate,betas=(0.9,0.99))#2
d_optim = torch.optim.Adam(dis_model.parameters(), lr
    =0.0001)
dis_loss_fn= torch.nn.BCELoss()
# opt_Adagrad = torch.optim.Adagrad(model.parameters(),
    lr=Learning_rate)#3
#123都是逐参数适应学习率的方法，adam像是RMSprop的冲量版
#在实际操作中，我们推荐Adam作为默认的算法，一般而言跑起
    来比RMSProp要好一点。但是也可以试试SGD+Nesterov动
    量。

#为了使用torch.optim，你需要构建一个optimizer对象。这个
    对象能够保持当前参数状态并基于计算得到的梯度进行参数
```

```
    更新。
# 开始训练
for epoch in range(20):#epochs=10

model.train(True)
loss_sets=[]
for i in range(800):
dBZ, KDP, ZDR, gt = dBZ_dataset[i].to(device,dtype=
    torch.float32), KDP_dataset[i].to(device,dtype=torch
    .float32), ZDR_dataset[i].to(device,dtype=torch.
    float32),gt_dataset[i].to(device,dtype=torch.float32
    )


optimizer.zero_grad()   # 清空上一步的残余更新参数值,梯
    度属于参数，因此清空的是参数的梯度。
output = model(dBZ, ZDR, KDP)    # 得到预测值
#计算损失并更新判别器
# d_optim.zero_grad()
# real_output = dis_model(gt)
# d_real_loss = dis_loss_fn(real_output, torch.
    ones_like(real_output))
# d_real_loss.backward()
# fake_output = dis_model(output.detach())
# d_fake_loss = dis_loss_fn(fake_output,torch.
    zeros_like(fake_output))
# d_fake_loss.backward()
# d_optim.step()
# ---------
# hole loss
# ---------
mask = (gt<35).to(torch.int32)
loss_hole = l1((1-mask) * output, (1-mask) * gt)

# ----------
# valid loss
# ----------
loss_valid = l1(mask * output, mask * gt)
# #固定权重
# valid_weight = 1
# hole_weight = 6
#根据面积计算权重并归一化
valid_weight = torch.sum(mask)
hole_weight = torch.sum(1-mask)
# #防止权重归零，导致损失变为0
# valid_weight =(valid_weight+1)/2
# hole_weight =(hole_weight+1)/2
loss = 1*loss_valid/valid_weight + 6*hole_weight/(
    hole_weight+1)
```

```
            # loss = loss/256**2
            #添加ssim损失
            loss_ssim = ssim(output, gt)
            loss += 200*(1-loss_ssim)
            #添加对抗损失
            # fake_output = dis_model(output)
            # g_loss = dis_loss_fn(fake_output,torch.ones_like(
               fake_output))
            # loss += 0.01*g_loss
            #添加感知损失
            # vgg_output = extractor(output)
            # vgg_gt = extractor(gt)
            # for i in range(3): #vgg 提取三个尺度的特征
            #     loss += l1(vgg_output[i], vgg_gt[i])
            loss.backward()          # 误差反向传播, 计算参数更新值
            optimizer.step()         # 将参数更新值施加到 net 的
               parameters 上


            # opt_Momentum.step()
            # opt_RMSprop.step()
            # opt_Adam.step()
            # opt_Adagrad.step()
            if i % 100 ==0:#每隔100个输出1次结果
            print('loss=',loss.item())
            loss_sets.append(loss.item())

            # 每遍历一遍数据集, 测试一下准确率
            # 指定保存的文件路径
            model_path = 'model/q4_trained_model_3fusion_'+str(
               epoch)+'.pth'

            # 保存模型的状态字典及其他信息
            torch.save(model.state_dict(), model_path)
            # test()
            # model=UNet().to(device)
            # checkpoint_path = "q2/q2_1km结果2(添加掩码加权损失和
               ssim损失)/q2_trained_model_1km_9.pth"#注意, 这个模型
               用了q1的UNET训练, 没有BN层。
            # model.load_state_dict(torch.load(checkpoint_path))
            model.train(False)

            with torch.no_grad():  # 训练集中不需要反向传播
            #torch.no_grad() impacts the autograd engine and
               deactivate it.
            #It will reduce memory usage and speed up computations
               but you won't be able to backprop
            avg_mae=0

            avg_rmse=0
```

```
avg_ssim=0
avg_ppc=0
for i in range(800,891):
dBZ, KDP, ZDR, gt = dBZ_dataset[i].to(device,dtype=
    torch.float32), KDP_dataset[i].to(device,dtype=torch
    .float32), ZDR_dataset[i].to(device,dtype=torch.
    float32), gt_dataset[i].to(device,dtype=torch.
    float32)


output = model(dBZ, ZDR, KDP)
# 计算归一化绝对误差（MAE）
mae = F.l1_loss(gt,output)
# print("MAE:", mae.item())
avg_mae +=mae

# 计算均方根误差（RMSE）
mse = F.mse_loss(gt,output)
rmse = torch.sqrt(mse)
# print("RMSE:", rmse.item())
avg_rmse +=rmse

#计算ssim
result_ssim=ssim(gt,output)
avg_ssim+=result_ssim
#计算ppc
ppc = cal_ppc(gt,output)
avg_ppc+=ppc

############################ 计算逐点csi, 阈值取35
#     hits_all += count_hits(gt, output, threshold=35)
#     misses_all += count_misses(gt, output, threshold
    =35)
#     false_alarms_all += count_false_alarms(gt, output
    , threshold=35)
# csi = hits_all / (hits_all + misses_all +
    false_alarms_all + torch.finfo(torch.float32).eps)

# csi=csi.to('cpu')


# 找到最大值的位置
# max_index = torch.argmax(csi)

# # 将 Tensor 转换为 NumPy 数组
# tensor_np = csi.numpy()

# # 获取最大值所在的各个维度的坐标
# max_coords = np.where(tensor_np == tensor_np.flatten
    ()[max_index])
```

```
            # for i in range(256):
            #     for j in range(256):
            #         csi_result = csi[:, i, j].numpy()

            #         # 将数组写入 txt 文件
            #         with open("q1_output_逐点历史csi.txt", "a")
                as f:

            #             # 将每一行转换为字符串，并使用空格分隔元
                素
            #             line = " ".join("{:.2f}".format(x) for x
                in csi_result)
            #             f.write(line + "\n")
            #         print(csi_result)
            ############################## 计算每个面的CSI，阈值取35
            csi = calculate_CSI(gt, output)
            with open("q4_output_区域csi_1km_3_fusion.txt", "a") as
                f:

            # 将每一行转换为字符串，并使用空格分隔元素
            line = " ".join("{:.2f}".format(x) for x in csi)
            f.write(line + "\n")
            print(csi)

            avg_mae = avg_mae.item()/float(91)
            avg_ssim = avg_ssim.item()/float(91)
            avg_rmse = avg_rmse.item()/float(91)
            avg_ppc = avg_ppc.item()/float(91)
            with open("q4/metrics.txt", "a") as f:

            # 将每一行转换为字符串，并使用空格分隔元素
            line = "".join("3_fusion_input_mae = {:.2f}".format(
                avg_mae))
            f.write(line + "\n")
            line = "".join("rmse = {:.2f}".format(avg_rmse))
            f.write(line + "\n")
            line = "".join("ssim = {:.2f}".format(avg_ssim))
            f.write(line + "\n")
            line = "".join("ppc = {:.2f}".format(avg_ppc))
            f.write(line + "\n")
            # mean_csi = np.mean(csi, axis=0)
            print('MAE, RMSE of the network on the test images:
                {:.2f}, {:.2f}, {:.2f}, {:.2f}'.format(avg_mae,
                avg_rmse, avg_ssim, avg_ppc ))

            print('average csi of one points:')

            # 找到最大值的位置
            # max_index = torch.argmax(torch.sum(csi,dim=0))
            # # 将 Tensor 转换为 NumPy 数组
```

```
# tensor_np = torch.sum(csi,dim=0).numpy()
# # 获取最大值所在的各个维度的坐标
# max_coords = np.where(tensor_np == tensor_np.flatten
    ()[max_index])

def tensor_to_rgb(tensor):
rgb_tensor = torch.zeros(256,256,3, dtype=torch.uint8)
for i, row in enumerate(tensor):
for j, element in enumerate(row):
if element < 10:
rgb_tensor[i,j,:] = torch.tensor([255, 255, 255])
elif element <=15:
rgb_tensor[i,j,:] = torch.tensor([204, 255, 255])
elif element <=20:
rgb_tensor[i,j,:] = torch.tensor([0, 255, 255])
elif element <=25:
rgb_tensor[i,j,:] = torch.tensor([79, 146, 224])
elif element <=30:
rgb_tensor[i,j,:] = torch.tensor([255, 204, 102])
elif element <=35:
rgb_tensor[i,j,:] = torch.tensor([204, 153, 0])
elif element <=40:
rgb_tensor[i,j,:] = torch.tensor([255, 0, 0])
elif element <=45:
rgb_tensor[i,j,:] = torch.tensor([204, 0, 0])
elif element <=50:
rgb_tensor[i,j,:] = torch.tensor([165, 0, 33])
elif element <=55:
rgb_tensor[i,j,:] = torch.tensor([204, 153, 255])
elif element <=60:
rgb_tensor[i,j,:] = torch.tensor([204, 0, 255])
elif element <=65:
rgb_tensor[i,j,:] = torch.tensor([153, 0, 204])
elif element >65:
rgb_tensor[i,j,:] = torch.tensor([102, 0, 102])
return rgb_tensor
def plot_dBZ(tensor):
#     tensor = tensor_to_rgb(tensor.to(torch.int64))
#     image = Image.fromarray(tensor.numpy())

# # 显示图像
#     image.show()
tensor = tensor_to_rgb(tensor.to(torch.int64))

# 创建图像
fig, ax = plt.subplots()

# 显示图像
ax.imshow(tensor)
```

```
# 添加网格线
# 添加网格线
x_ticks = range(0, tensor.shape[1], 25)
y_ticks = range(0, tensor.shape[0], 25)
ax.set_xticks(x_ticks)
ax.set_yticks(y_ticks)
ax.set_xticklabels([])
ax.set_yticklabels([])
ax.grid(True, color='gray', linestyle='--', linewidth
    =0.5)
ax.set_xlim([75, 200])
ax.set_ylim([225, 100])

# 保存图像到文件

plt.show()




#结果取834和866位置
dBZ = dBZ_dataset[890]
gt = gt_dataset[890]
KDP = KDP_dataset[890]
ZDR = ZDR_dataset[890]
model.train(False)
with torch.no_grad():  # 训练集中不需要反向传播
dBZ_hat = model(dBZ.to(device,dtype=torch.float32), ZDR
    .to(device,dtype=torch.float32), KDP.to(device,dtype
    =torch.float32))
for i in range(1):
plot_dBZ(dBZ[0][i])
plt.savefig('q4/fusion_input_'+str(i)+'.png')
plot_dBZ(gt[0][i])
plt.savefig('q4/fusion_truth_'+str(i)+'.png')
plot_dBZ(dBZ_hat[0][i])
plt.savefig('q4/fusion_output_'+str(i)+'.png')
csi = calculate_CSI(gt.to(device,dtype=torch.float32),
    dBZ_hat)
with open("csi_by_different_input.txt", "a") as f:

# 将每一行转换为字符串，并使用空格分隔元素
line = '3fusion_csi:'+" ".join("{:.2f}".format(x) for x
     in csi)
f.write(line + "\n")

x = [6, 12, 18, 24, 30, 36, 42, 48, 54, 60]
y = csi
plt.figure()
plt.plot(x, y, 'bo-')
```

```
                plt.ylim([0, 0.7])
                plt.xlabel('forecast minutes(min)')
                plt.ylabel('CSI')

                plt.show()
                plt.savefig('q4/fusion_CSI.png')

                # dBZ=dBZ_dataset[800]
                # for i in range(10):
                #     plot_dBZ(dBZ[0][i])
```