



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 华东师范大学

---

参赛队号 23102690048

---

1.张逸飞

---

队员姓名 2.周磊

---

3.陈鹏

---

**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：** 短临强降水情景下基于深度学习模型和双偏振雷达数据的未来降  
水预测与定量估计

**摘要：**

根据世界气象组织和政府间气候变化专门委员会的报告，人类引起的气候变化已经影响了全球每个地区的许多天气和气候极端事件。自 20 世纪 50 年代以来，强降雨事件的频率和强度已经增加，并预计这种情况将继续下去。全球每升温 1° C，日极端降水事件预计将加剧约 7%。

强对流降水是形成众多灾害性天气的主要原因，对其 0-2 小时的短时临近预报关系到日常生命财产保护，具有重要意义。然而，因其复杂度高、非线性强，强对流风暴短临预报仍是当前国际前沿热点难点问题。

在此背景下，本文以新型双偏振雷达观测数据和降水格点数据，进行数据预处理，构建了以 **ConvLSTM** 为基本框架的水平反射率因子  $Z_h$  深度学习预测模型 **DRPP-Net**，为解决预测结果的模糊效应，融合注意力机制模块 CBAM 和差值优化模块 DOM，构建了新型水平反射率因子  $Z_h$  深度学习预测模型 **SA-DRPP-DNet**，同时为了定量估计降水量，构建了以 **U-Net** 为基本框架，融合了 **swin transformer** 模块和 **ASPP** 模块的降水量预测模型 **PP-Net**。对比分析不同数据组合，验证了双偏振雷达观测数据对短临强降水预测具有高贡献度，融合多模态双偏振雷达观测数据，进一步提高模型精度。

针对问题一，首先对双偏振雷达观测数据进行数据预处理，计算了水平反射率因子与降水量之间的相关性，验证了基本假设；根据问题所给建议的雷达观测数据归一化范围，利用线性归一化方法，对双偏振雷达数据进行归一化处理；结合前人论文研究，确定水平反射率因子  $Z_h$  分割阈值变化范围，利用步长法和阈值分割法，以 35 dBZ 为分割阈值，将原始数据中低于 35 dBZ 的数据视为弱降水或无降水情景，不作为样本输入深度学习模型；随机划分训练样本和测试样本，构成样本数据集。然后以 **ConvLSTM** 为基本框架，构建水平反射率因子  $Z_h$  深度学习预测模型 **DRPP-Net**，选取两类具有代表性的降水事件，进行预测结果分析。利用气象预测评估指数 CSI、ETS、FAR、MAR、POD 和 RMSE，对模型精度进行评估，结果表明 **DRPP-Net** 模型取得较好的预测结果和较高的精度。

针对问题二，问题一中预测结果存在模糊效应，引入注意力机制 CBAM 模块和差值优化 DOM 模块，构建了新型水平反射率因子  $Z_h$  深度学习预测模型 **SA-DRPP-DNet**。同样对问题一选择的两类代表性降水事件，进行预测结果分析和模型精度评价，并对比问题一的

DRPP-Net 模型，结果表明 **SA-DRPP-DNet** 缓解了预测结果的模糊效应，提高了预测精度。

针对问题三，首先对降水格点数据进行数据空洞弥补，采用最邻近插值法，进行数值补全，然后以 U-Net 模型为基本框架，引入了 swin transformer 模块和 ASPP 模块，构建了降水量定量估计深度学习模型 PP-Net，然后对预测结果进行分析和模型的精度评定，结果表明，**PP-Net** 对降水量的估计，取得了较好的估计结果和不错的预测精度。

针对问题四，首先针对双偏振雷达观测数据对短临强降水预测的贡献度进行分析，设计了两类模型的不同数据输入组合的共六种组合方案，分别对这六种方案进行精度评定，结果表明双偏振雷达数据的输入有效提高了降水预测的精度，对降水预测具有高贡献度。其次，对不同等高面的双偏振雷达观测数据，进行加权数据融合，构建多模态的雷达数据集，输入 PP-Net 模型，结果表明多模态数据提高了预测精度，使模型取得更好的预测结果。

综上所述，**SA-DRPP-DNet** 和 **PP-Net** 模型对于短临强降水预测具有良好的精度和泛用性，预测结果符合预期。

**关键词：** 双偏振雷达数据、深度学习、短临强降水、强对流降水、ConvLSTM

# 目录

1 问题重述与数据说明.....	5
1.1 问题背景.....	5
1.2 问题提出.....	5
1.3 全文思路框架.....	5
1.4 数据说明.....	7
1.5 样本数据说明.....	7
2 基本假设 .....	8
2.1 关于数据的假设.....	8
2.2 关于模型的假设.....	8
3 符号说明 .....	9
4 问题一的分析与模型建立.....	10
4.1 问题一描述与分析.....	10
4.2 双偏振雷达原理.....	10
4.2.1 水平反射率因子 $Z_h$ .....	11
4.2.2 差分反射率 $Z_{DR}$ .....	11
4.2.2 比差分相移 $K_{DP}$ .....	11
4.3 数据预处理.....	11
4.4 深度学习 DRPP-Net 模型的构建.....	12
4.5 实验设计与结果分析.....	14
4.5.1 实验环境与模型参数.....	14
4.5.2 实验结果分析.....	14
4.5.2.1 降水事件一预测结果.....	14
4.5.2.2 降水事件二预测结果.....	16
4.5.3 模型精度评定指数.....	17
4.5.3.1 临界成功指数 CSI.....	17
4.5.3.2 公平技巧得分 ETS.....	17
4.5.3.3 空报率 FAR .....	17
4.5.3.4 漏报率 MAR.....	17
4.5.3.5 命中率 POD.....	17
4.5.3.6 均方根误差 RMSE.....	18
4.5.4 模型精度评定结果.....	18
4.5.4.1 降水事件一模型精度评定结果.....	18
4.5.4.2 降水事件二模型精度评定结果.....	18
5 问题二的分析与模型建立.....	20
5.1 问题二描述与分析.....	20
5.2 注意力机制结构的原理.....	20
5.3 差值优化模块 DOM 的结构与原理 .....	22
5.4 SA-DRPP-DNet 的结构与原理 .....	22
5.5 预测结果和模型精度对比.....	23
5.5.1 降水事件一预测结果与精度评定.....	23
5.5.2 降水事件二预测结果与精度评定 .....	25
5.5.3 降水事件一 DRPP-Net 与 SA-DRPP-DNet 对比分析 .....	27

5.5.4 降水事件二 DRPP-Net 与 SA-DRPP-DNet 对比分析 .....	29
6 问题三的分析与模型建立.....	32
6.1 问题三描述与分析.....	32
6.2 降水格点数据空洞弥补.....	32
6.3 传统降水定量预测方法.....	33
6.4 swin transformer 模块结构与原理.....	34
6.5 ASPP 模块结构与原理 .....	34
6.6 深度学习降水量定量预测模型 PP-Net .....	35
6.7 PP-Net 降水量预测结果与分析 .....	35
6.8 模型精度评定.....	36
7 问题四的分析与模型建立.....	37
7.1 问题四的描述与分析.....	37
7.2 双偏振雷达观测数据贡献度评定实验设计 .....	37
7.3 双偏振雷达观测数据贡献度评定实验结果与分析 .....	38
7.4 多模态降水预测模型的实验设计 .....	40
7.5 多模态降水量 $K_{DP\_Rain}$ 定量估计模型的实验结果与分析 .....	41
8 总结与评价 .....	42
8.1 模型的结论与评价.....	42
8.1.1 模型的优势.....	42
8.1.2 模型的劣势.....	42
8.2 模型的推广与改进.....	42
8.2.1 模型的推广.....	42
8.2.2 模型的改进.....	42
参考文献 .....	43
附录 .....	44

# 1 问题重述与数据说明

## 1.1 问题背景

进入 21 世纪，全球气候变化引发的极端天气气候事件愈加频繁，高温、洪水、飓风、寒潮等极端天气呈现常态化趋势，一是极端天气的频率、强度增加，时空分布模式发生变化；二是灾害的复杂性、极端性、不可预测性增加，容易引发灾害链效应和系统性风险<sup>[1]</sup>。

《中国气候变化蓝皮书（2023）》显示，中国气候风险指数呈升高趋势，表现为极端高温事件频发趋强，极端强降水事件增多，台风平均强度波动增强。

在全球变暖背景下，暴雨事件会变得更强、更频繁。例如，在全球 1.5°C 温升下，当前 20 年一遇的强降水事件发生频率将增加 10%，100 年一遇的强降水事件发生频率将增加 20%；在全球 2°C 温升下，当前 20 年一遇的强降水事件发生频率将增加 22%，100 年一遇的强降水事件发生频率将增加 45% 以上。

近年来，我国短临强降水纪录不断刷新，2021 年 7 月 20 日郑州最大小时降雨量达 201.9 毫米，突破我国大陆小时降雨量历史极值<sup>[2]</sup>。

随着我国新一代的气象观测设备如双偏振雷达、降水现象仪等的布设，更多的专家学者关注强降水物理过程机理和微物理特征分析的研究。其中双偏振雷达可以对不同降水类型的微物理特征进行全面的分析，通过对水平反射率因子、差分反射率、比差分相移、相关系数等变量的分析，可以得到有关降水粒子的大小、相态、含水量等信息，从而探究不同对流特征的降水的微物理特征及其演变规律<sup>[2,3]</sup>。

强降水短时（0~12 小时）和临近（0~2 小时）预报通常也是天气预报业务中的难点，传统强对流天气临近预报主要依靠雷达等观测资料，结合风暴识别、追踪技术进行雷达外推预报，即通过外推的方法得到未来时刻的雷达反射率因子，并进一步使用雷达反射率因子和降水之间的经验性关系（即 Z-R 关系）估计未来时刻的降水量<sup>[4,5]</sup>。而结合大量的气象数据和强大的计算机算力，运用深度学习或机器学习来预测未来降水量的研究发展迅速，短临强降水数据具有数据量大、时间序列完整等特点，大量深度学习模型被引入短临强降水领域，例如基于传统的卷积神经网络的 U-Net、ResNet 等模型，和基于循环神经网络的 LSTM、BRNN 等模型，提高了预测精度<sup>[6]</sup>。

本篇文章针对提出的问题，在给定的双偏振雷达观测数据和降水格点数据的基础上，引入深度学习模型，对未来时刻的水平反射率因子  $Z_h$  和降水量  $K_{DP\_Rain}$  进行预测，并根据真实值与预测值的差值，引入深度学习模型，对提出的算法模型进行优化。

## 1.2 问题提出

问题一：如何有效应用双偏振变量改进强对流预报，仍是目前气象预报的重点难点问题。请利用题目提供的数据，建立可提取用于强对流临近预报双偏振雷达资料中微物理特征信息的数学模型。临近预报的输入为前面一小时（10 帧）的雷达观测量 ( $Z_h$ 、 $Z_{DR}$ 、 $K_{DP}$ )，输出为后续一小时（10 帧）的  $Z_h$  预报。

问题二：当前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，因此预报总是趋于模糊。在问题一的基础上，请设计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。

问题三：请利用题目提供的  $Z_h$ 、 $Z_{DR}$  和降水量数据，设计适当的数学模型，利用  $Z_h$  及  $Z_{DR}$  进行定量降水估计。模型输入为  $Z_h$  和  $Z_{DR}$ ，输出为降水量。（注意：算法不可使用  $K_{DP}$  变量。）

问题四：请设计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，以便更好地应对突发性和局地性强的强对流天气。

## 1.3 全文思路框架

全文的思路框架图如下所示：

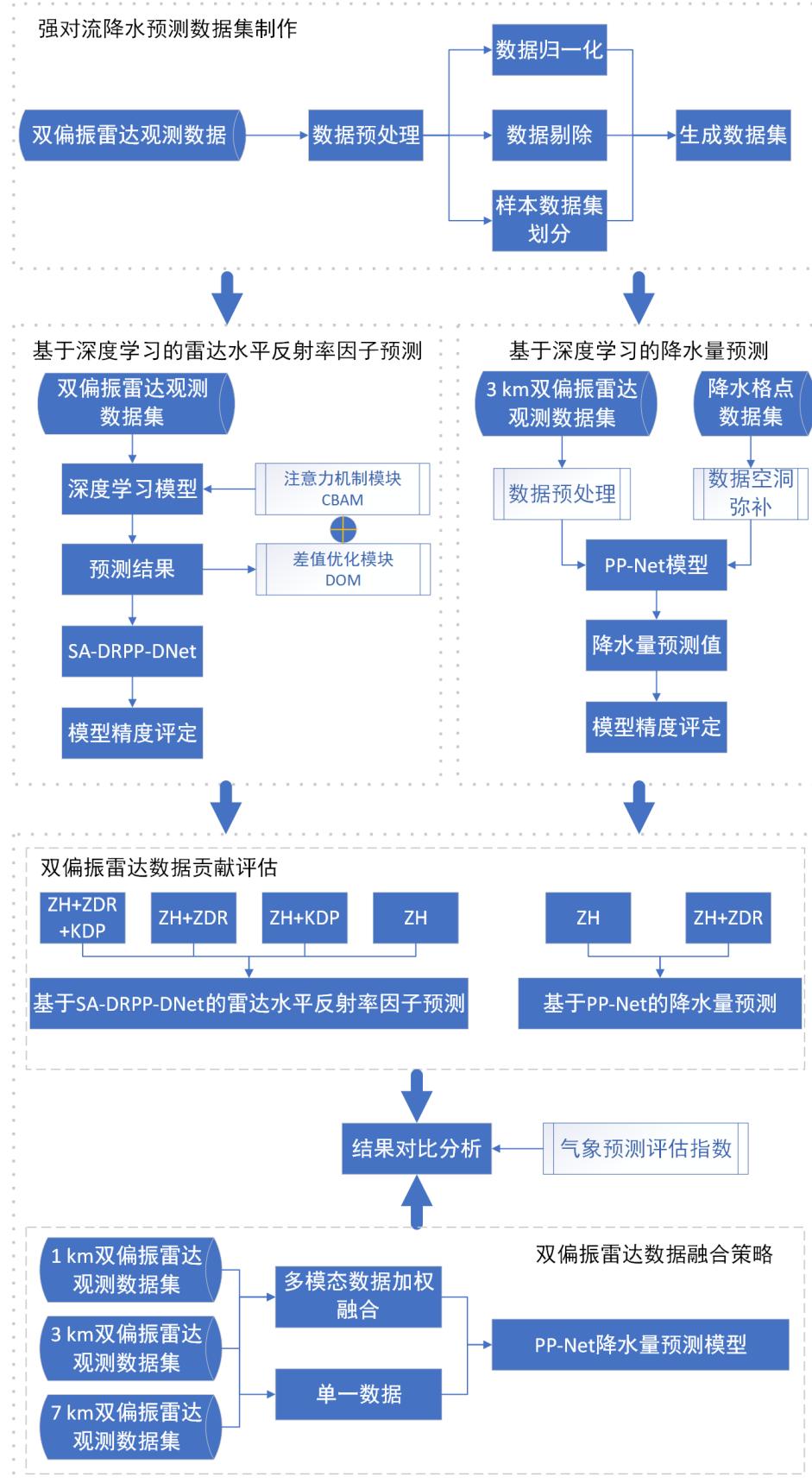


图 1-1 全文思路框架流程图

## 1.4 数据说明

本文使用的数据来自于南京大学获取的 NJU-CPOL 数据集<sup>[7]</sup>, 包含 258 个降水事件。此数据集已经完成质量控制, 且已转换到笛卡尔坐标系。其幅宽为 256 km\*256 km, 空间分辨率为 1 km, 时间分辨率为 6 分钟。数据集中包含了三种等高面 (1 km、3 km、7 km) 的三种双偏振雷达观测数据, 水平反射率因子 $Z_h$ 、差分反射率 $Z_{DR}$ 和比差分相移 $K_{DP}$ 。

## 1.5 样本数据说明

表 1-1 1 km 等高面数据预测水平反射率因子 $Z_h$ 模型各样本数量(0 值代表未输入)

样本	$Z_h$	$Z_{DR}$	$K_{DP}$	$K_{DP\_Rain}$
训练样本	9103	9103	9103	0
测试样本	1968	1968	1968	0

表 1-2 3 km 等高面数据预测水平反射率因子 $Z_h$ 模型各样本数量(0 值代表未输入)

样本	$Z_h$	$Z_{DR}$	$K_{DP}$	$K_{DP\_Rain}$
训练样本	11692	11692	11692	0
测试样本	1968	1968	1968	0

表 1-3 7 km 等高面数据预测水平反射率因子 $Z_h$ 模型各样本数量(0 值代表未输入)

样本	$Z_h$	$Z_{DR}$	$K_{DP}$	$K_{DP\_Rain}$
训练样本	6351	6351	6351	0
测试样本	1968	1968	1968	0

表 1-4 1 km 等高面数据预测降水量 $K_{DP\_Rain}$ 模型各样本数量(0 值代表未输入)

样本	$Z_h$	$Z_{DR}$	$K_{DP}$	$K_{DP\_Rain}$
训练样本	7963	7963	0	7963
测试样本	3412	3412	0	3412

表 1-5 3 km 等高面数据预测降水量 $K_{DP\_Rain}$ 模型各样本数量(0 值代表未输入)

样本	$Z_h$	$Z_{DR}$	$K_{DP}$	$K_{DP\_Rain}$
训练样本	10126	10126	0	10126
测试样本	4340	4340	0	4340

表 1-6 7 km 等高面数据预测降水量 $K_{DP\_Rain}$ 模型各样本数量(0 值代表未输入)

样本	$Z_h$	$Z_{DR}$	$K_{DP}$	$K_{DP\_Rain}$
训练样本	5715	5715	0	5715
测试样本	2449	2449	0	2449

## 2 基本假设

### 2.1 关于数据的假设

- 1、数据有效性假设：获取到的 NJU-CPOL 双偏振雷达观测数据和降水格点数据已经完成质量控制，其数值是准确的，没有噪声或失真，异常值已经得到剔除。
- 2、数据独立性假设：各降水事件之间无任何关联，彼此独立。

### 2.2 关于模型的假设

- 1、降水过程抽象为数值的时间序列，不考虑其他因素的影响，如地形、风向、风速、温度等。
- 2、所有降水事件皆处于统一稳定的大气系统，不存在其他气象因素的影响。
- 3、模型得到充分训练。
- 4、短临强降水事件具有可预测性，可以利用双偏振雷达观测数据进行预测。

### 3 符号说明

表 3-1 符号说明

符号	含义
$Z_h$	水平反射率因子
$Z_{DR}$	差分反射率
$K_{DP}$	比差分相移
$dBZ$	水平反射率因子的单位
$dB$	差分反射率的单位
${}^\circ km^{-1}$	比差分相移的单位
RNN	循环神经网络
LSTM	长短期记忆神经网络
ConvLSTM	卷积长短期记忆神经网络
DRPP-Net	双偏振雷达水平反射率因子预测网络
$K_{DP\_Rain}$	降水量
NJU-CPOL	双偏振雷达观测数据集
CSI	临界成功指数
ETS	公平技巧得分
FAR	空报率
MAR	漏报率
POD	命中率
RMSE	均方根误差
Hits	命中
False Alarms	误警
Misses	漏报
Correct negatives	正确否定
SA	注意力机制
CBAM	卷积注意力机制模块
CAM	通道注意力机制模块
SAM	空间注意力机制模块
DOM	差值优化模块
SA-DRPP-DNet	融合注意力机制与差值模块的双偏振雷达水平反射率因子预测网络
Default Relationship	传统默认 Z-R 关系
Rosenfeld tropical Realtionship	罗斯菲尔德热带 Z-R 关系
Marshall-Palmer Realtionship	马歇尔·帕尔默 Z-R 关系
swin transformer	swin transformer 模型
Windows Multi-Head Self-Attention	窗口多头注意力机制模块
Shifted Windows Multi-Head Self-Attention	滑动窗口多头注意力机制模块
MLP	多层感知机
ASPP	空洞空间金字塔池化模块
U-Net	U-Net 网络结构
PP-Net	降水量预测网络

## 4 问题一的分析与模型建立

### 4.1 问题一描述与分析

根据题目的描述，本问题要求根据题目提供的双偏振雷达观测数据建立数学模型，输入不同等高面的前 10 帧（一小时）的  $Z_h$ 、 $Z_{DR}$  和  $K_{DP}$  数据，预测未来 10 帧（一小时）的不同等高面的  $Z_h$  值。

问题一属于时间序列大数据预测问题，数据文件中给出了 258 个降水事件的双偏振雷达观测数据，总数据量超过 78 Gigabytes，传统的雷达回波降水预测方法难以处理大数据量的雷达数据，因此本文引入最新的深度学习模型，旨在输入多个雷达参数指标，从中提取有效特征以进行未来时刻的水平反射率因子  $Z_h$  预测。

第一步：数据预处理，首先读取数据集中的  $Z_h$  数据，设计了不同的阈值分割算法，对数据集中降水较少或无降水的降水事件进行剔除，得到符合强降水的要求的数据。

第二步：建立深度学习模型

第三步：模型精度评定

问题一的技术流程图 4-1 如下：

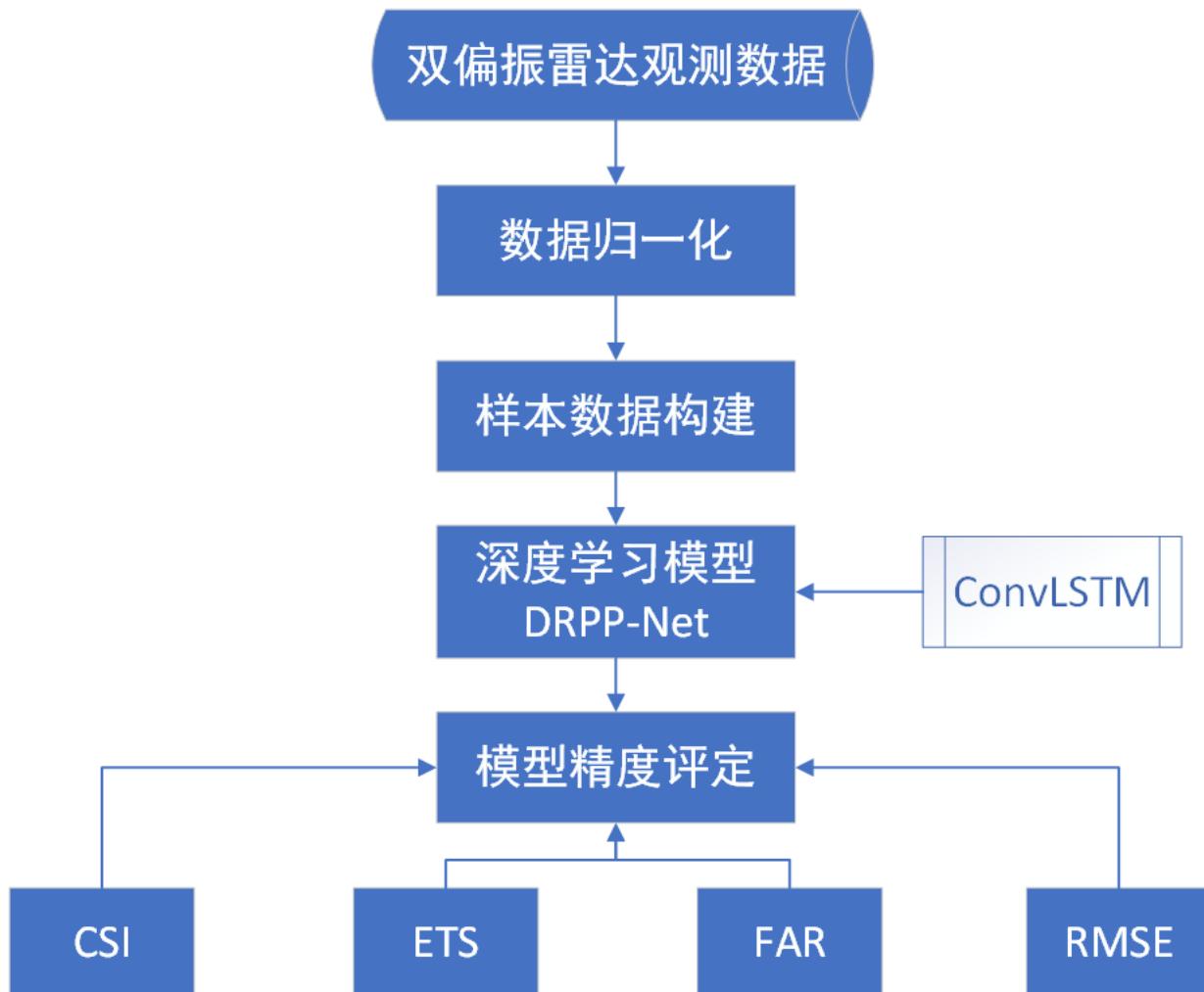


图 4-1 问题一技术流程图

### 4.2 双偏振雷达原理

传统雷达仅能发射和接收一个偏振方向上的电磁波，而新型的双偏振雷达可同时发射和接收在水平和垂直两个偏振方向的电磁波，可以根据两个偏振方向上的回波的强度差别、

相位关系等信息获得降水粒子的大小、相态、含水量等信息。

#### 4.2.1 水平反射率因子 $Z_h$

水平反射率因子 $Z_h$ 与粒子数浓度和等体积直径的六次方成正比，粒子直径越大，粒子数浓度越高，则水平反射率因子 $Z_h$ 越大，通常能够反映降水的强度，其数学公式如下：

$$Z_h = \frac{4\lambda^4}{\pi^4 |K_w|^2} \int_{D_{min}}^{D_{max}} |f_{hh}(\pi, D)|^2 N(D) dD \quad (4-1)$$

其中 $D$ 是粒子的直径，单位是 $mm$ ； $N(D)$ 是单位体积空气、单位尺寸内的粒子数浓度，单位是 $m^{-3}mm^{-1}$ ； $\lambda$ 是雷达波长，单位是 $mm$ ； $D_{max}$ 和 $D_{min}$ 是粒子直径的最大值和最小值； $K_w$ 是粒子的介电常数； $f_{hh}(\pi, D)$ 是水平极化方向上的后向散射幅度；水平反射率因子 $Z_h$ 的单位是 $dBZ$ 。

#### 4.2.2 差分反射率 $Z_{DR}$

差分反射率为水平极化方向和垂直极化方向回波的反射率因子的比值的对数，主要反映了观测区域的降水粒子大小。 $Z_{DR}$ 取决于粒子的形状、大小、密度和相态组成，与产生后向散射的粒子浓度无关，其数学公式如下：

$$Z_{DR} = 10 \log_{10} \frac{Z_h}{Z_v} \quad (4-2)$$

其中 $Z_v$ 是垂直反射率因子；差分反射率 $Z_{DR}$ 的单位是 $dB$ 。

#### 4.2.2 比差分相移 $K_{DP}$

比差分相移为单位距离上降水粒子导致的水平和垂直方向回波的相位差，主要反映了液态含水量。 $K_{DP}$ 几乎与降水率呈线性关系，取决于粒子浓度、大小和组成成分（即粒子介电常数），不受雷达定标不准、衰减和差分衰减，波束部分阻挡等因素的影响，其数学公式如下：

$$K_{DP} = 10^{-3} \frac{180}{\pi} \lambda Re \left\{ \int_{D_{min}}^{D_{max}} [f_{hh}(0, D) - f_{vv}(0, D)] N(D) dD \right\} \quad (4-3)$$

其中 $Re$ 是积分部分的实部值； $f_{hh}(0, D)$ 和 $f_{vv}(0, D)$ 是水平极化方向和垂直极化方向上的前向散射幅度，单位是 $mm$ ；比差分相移 $K_{DP}$ 的单位是 $^\circ km^{-1}[8]$ 。

### 4.3 数据预处理

首先对数据进行归一化处理，依据给出的数据说明文件，其中水平反射率因子的归一化范围为[0,65]，差分反射率的归一化范围为[-1,5]，比差分相移的归一化范围为[-1,6]。

其次依据假设，水平反射率因子 $Z_h$ 与降水量之间有一定的相关性，采用相关性比较、卡方比较和巴氏距离比较，对数据集中的水平反射率因子和降水格点数据进行相关性分析，验证了假设的成立，然后对弱降水和无降水情况进行数据剔除。

依据相关资料，部分学者认为水平反射率因子 $Z_h$ 小于 $30 dBZ$ 为弱降水，大于 $35 dBZ$ 且小于 $40 dBZ$ 为较强降水，大于 $40 dBZ$ 为强降水<sup>[9]</sup>，具体阈值分割情况如表 4-1：

表 4-1 水平反射率因子与降水强度类型的关系

水平反射率因子 ( $dBZ$ )	降水强度类型
$Z_h < 35$	弱降水
$35 < Z_h < 40$	较强降水
$40 < Z_h < 60$	强降水
$Z_h > 60$	冰雹

也有部分学者认为水平反射率因子大于  $45 \text{ dBZ}$  为强降水，具体阈值分割情况如表 4-2 所示：

表 4-2 水平反射率因子与降水强度类型和降水强度的关系

水平反射率因子 ( $\text{dBZ}$ )	降水强度类型	降水强度 ( $\text{mm/hr}$ )
$15 < Z_h < 30$	小雨	0.2-2.0
$30 < Z_h < 40$	中雨	2.0-10
$40 < Z_h < 46$	大雨	10-30
$46 < Z_h < 50$	暴雨	30-50
$50 < Z_h < 55$	大暴雨	50-100
$56 < Z_h < 70$	特大暴雨	>100

因此，针对强降水的水平反射率因子的阈值，不同地区有不同的值。对于本数据集，采用步长法，以  $5 \text{ dBZ}$  为步长，以  $[30,50]$  为水平反射率因子的范围，进行弱降水或无降水情景的剔除，以减少数据量和保证样本量的原则，得出  $35 \text{ dBZ}$  为最优分割阈值。

数据预处理的流程图如图 4-2 所示：

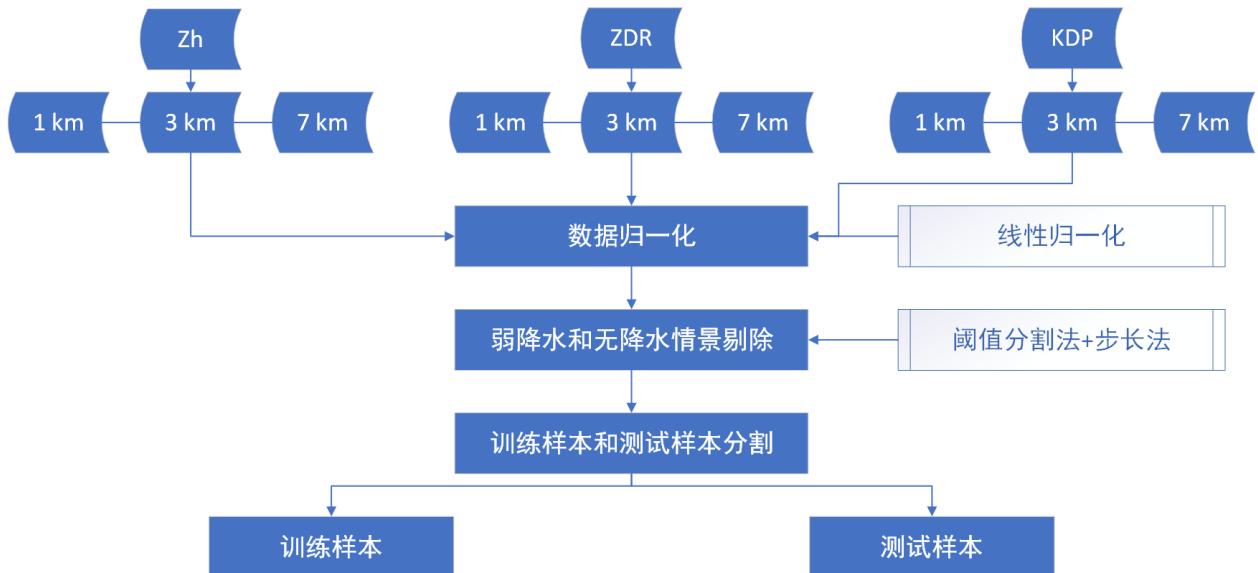


图 4-2 数据预处理流程图

#### 4.4 深度学习 DRPP-Net 模型的构建

传统的降水预测模型是基于雷达数据和地面站测量数据，利用特征分析和数据同化等进行数据预处理，结合物理经验方程，输出预测值，但短临强降水，降水量大，变化速度快，传统的数值模型的预测不确定性高。

有专家学者提出了雷达回波外推法，主要包括单体质心跟踪法和交叉相关法，即将降水云团视为实体，计算其运动矢量，结合雷达回波估计算法，实现雷达回波的时空预测，但由于其复杂的运算过程，对于短临强降水的处理效率和累计误差无法达到平衡，预测准确性无法保证<sup>[10]</sup>。

随着大数据时代的到来，专家学者开始研究基于深度学习的预测算法，将雷达回波的外推问题转换为雷达回波的时空预测的问题，即以过去时间截的雷达回波图像作为输入，预测未来时间截的雷达回波图像。

循环神经网络(Recurrent neural network, RNN)由于其循环结构，对于在时间维度上高相关的序列数据具有高适应性，然而 RNN 在训练过程中存在梯度爆炸或消失问题，且难以捕捉长时间序列的相关性，长短期记忆神经网络 LSTM 和门控循环单元 GRU 解决了这个问题，已经广泛应用于降水雷达回波预测问题。

但尽管 LSTM 已经广泛应用于时间序列数据的预测，但它依赖于基于密集矩阵乘法运算的全连接层，导致网络训练需要更多参数，且没有突出预测过程中的重要特征信息，普遍出现训练效率低、预测精度不理想的问题。

ConvLSTM 是一个具有卷积结构的 RNN。它通过当前输入和过去状态确定未来状态。与 LSTM 相比，它具有更少的网络参数，并借助卷积运算代替全连接运算，提高了模型的非线性建模能力，因此，与 LSTM 相比，所提出的具有更快的收敛速度和更好的预测精度 [11,12]。

本部分以 Conv-LSTM 为基本深度学习框架，构建了新型的雷达回波水平反射率因子深度学习预测模型，即 DRPP-Net(Dual-Polarization Radar Precipitation Prediction Net)。

其数学公式如下：

$$f_t = \sigma(W_f * [x_t, h_{t-1}] + b_f) \quad (4-4)$$

$$i_t = \sigma(W_i * [x_t, h_{t-1}] + b_i) \quad (4-5)$$

$$o_t = \sigma(W_o * [x_t, h_{t-1}] + b_o) \quad (4-6)$$

$$\tilde{c}_t = \tanh(W_c * [x_t, h_{t-1}] + b_c) \quad (4-7)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \quad (4-8)$$

$$h_t = o_t \odot \tanh(c_t) \quad (4-9)$$

其中，\*是卷积运算； $f_t$ 、 $i_t$ 、 $c_t$ 、 $\tilde{c}_t$  和  $o_t$  是 ConvLSTM 中的门控机制； $W$  和  $b$  是卷积算子的权重和偏差（例如， $W_f$  是门控机制  $o_t$  的卷积核的权重算子）； $x_t$  是输入数据； $h_t$  和  $h_{t-1}$  是当前 ConvLSTM 单元的循环状态和上一个 ConvLSTM 单元输出的循环状态。

其结构框架如图所示：

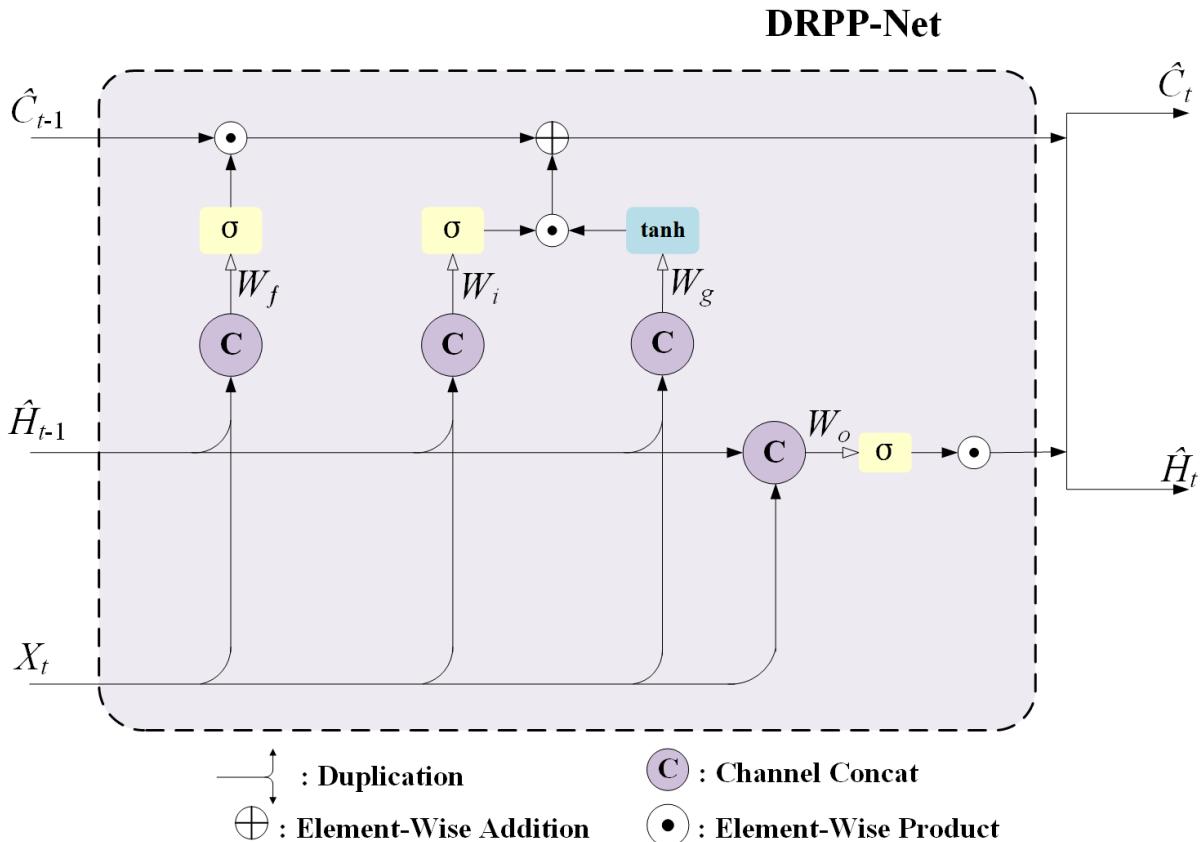


图 4-3 DRPP-Net 结构框架

## 4.5 实验设计与结果分析

### 4.5.1 实验环境与模型参数

实验是在 Window 10 专业版系统安装 Anaconda 3.0，编程语言为 Python 3.8，完成实验训练与结果验证。具体的计算机参数如表 4-3 所示：

表 4-3 实验平台具体参数

设备	设备参数
CPU	AMD Ryzen Threadripper PRO 5965 WX
GPU	NVIDIA RTX A6000 48GB
内存	128G

深度学习超参数设置如表 4-4 所示：

表 4-4 深度学习具体超参数

超参数名称	超参数数值
Batch_size	6
学习率	0.001
衰减策略	每迭代 20 次将学习率减少为原来的 0.8 倍
迭代次数	200
优化器	Adam

### 4.5.2 实验结果分析

#### 4.5.2.1 降水事件一预测结果

**Truth**

**Prediction**

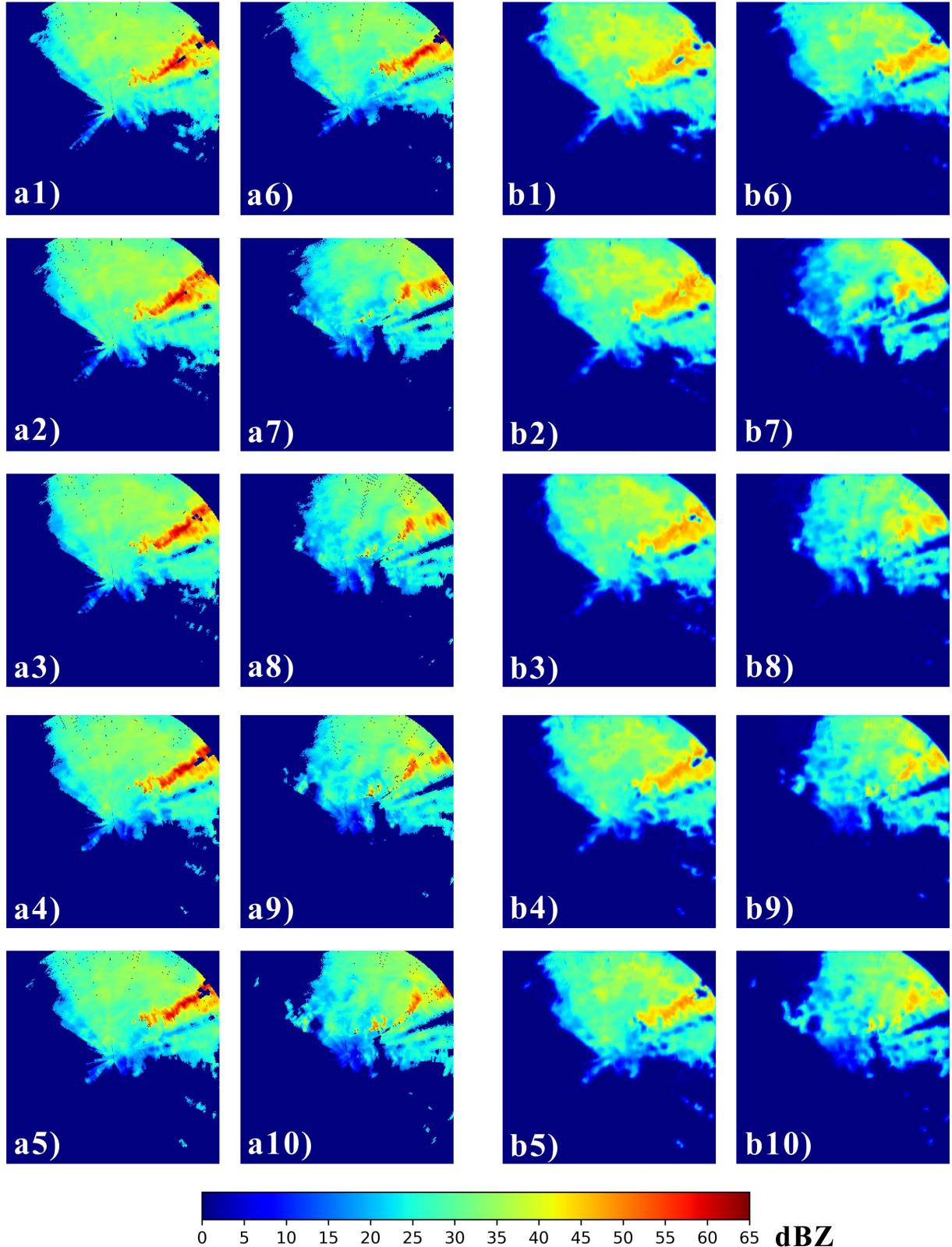


图 4-4 第 178 个降水事件的水平反射率因子  $Z_h$  真值(a1-a10)和基于 DRPP-Net 的水平反射率因子  $Z_h$  预测值(b1-b10), 分别对应于 6 分钟、12 分钟、18 分钟、24 分钟、30 分钟、36 分钟、42 分钟、48 分钟、54 分钟和 60 分钟

#### 4.5.2.2 降水事件二预测结果

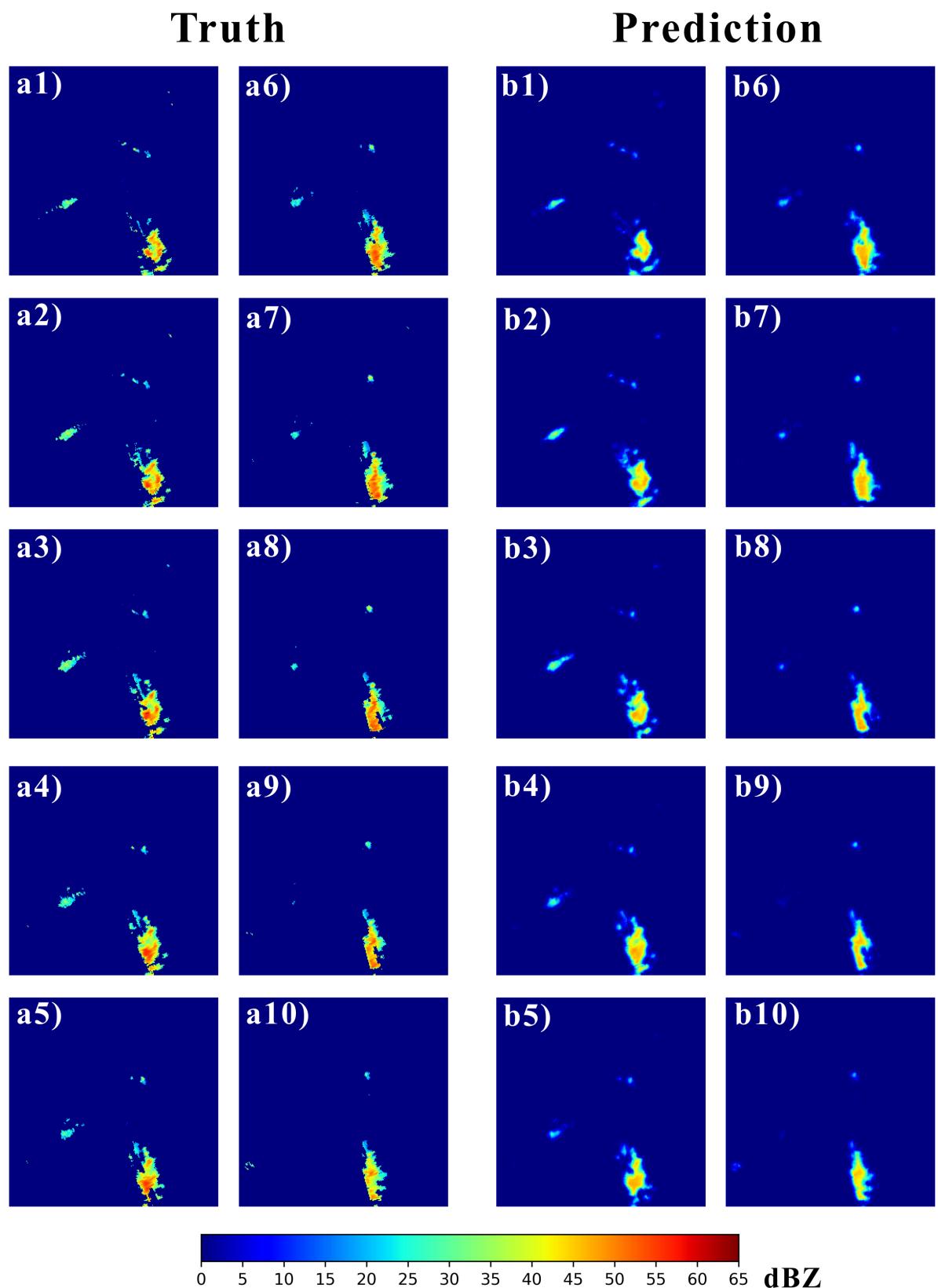


图 4-5 第 229 个降水事件的水平反射率因子  $Z_h$  真值(a1-a10)和基于 DRPP-Net 的水平反射率因子  $Z_h$  预测值(b1-b10), 分别对应于 6 分钟、12 分钟、18 分钟、24 分钟、30 分钟、36 分钟、42 分钟、48 分钟、54 分钟和 60 分钟

### 4.5.3 模型精度评定指数

本文引入气象学中的临界成功指数(Critical Success Index, CSI)、公平技巧得分(Equitable Threat Score, ETS)、空报率(False Alarm Rate, FAR)、漏报率(Miss Alarm Rate, MAR)、命中率(Probability of Detection, POD)和均方根误差(Root Mean Square Error, RMSE)来评价模型预测结果的精度。

指数中输入真实值、预测值和阈值，本文针对强降水情景，设置阈值为  $35 \text{ dBZ}$ 。

#### 4.5.3.1 临界成功指数 CSI

CSI 是一种用于评估分类预报准确性的常用统计指标，用于评估降水、风暴、降雪等天气事件的预测准确性<sup>[13]</sup>。CSI 的计算基于四个不同的预测分类：命中(Hit)、误警(False Alarm, FA)、漏报(Miss)和正确否定(Correct Negative, CN)。CSI 值的范围在 0 到 1 之间，较高的 CSI 值表示分类预报准确性较高。

其数学公式如下所示：

$$CSI = \frac{\text{hits}}{\text{hits} + \text{misses} + \text{false alarms}} \quad (4 - 10)$$

表 4-5 预测值与真实值混淆矩阵

		预测值 $Y_{Prd}$	
		正例	反例
真实值 $Y_{Tru}$	正例	命中(hits)	漏报(misses)
	反例	误警(false alarms)	正确否定(correct negatives)

#### 4.5.3.2 公平技巧得分 ETS

公平技巧评分用于衡量对流尺度集合预报的预报效果。ETS 评分表示在预报区域内满足某降水阈值的降水预报结果相对于满足同样降水阈值的随机预报的预报技巧。

ETS 评分是对 CSI 评分的改进，能对空报或漏报进行惩罚，使评分相对后者更加公平。

其数学公式如下所示：

$$ETS = \frac{\text{hits} - Dr}{\text{hits} + \text{misses} + \text{false alarms} - Dr} \quad (4 - 11)$$

$$Dr = \frac{\text{Num}}{\text{Den}} \quad (4 - 12)$$

$$\text{Num} = (\text{hits} + \text{false alarms}) * (\text{hits} + \text{misses}) \quad (4 - 13)$$

$$\text{Den} = \text{hits} + \text{false alarms} + \text{misses} + \text{correct negatives} \quad (4 - 14)$$

#### 4.5.3.3 空报率 FAR

FAR 指的是在预报降水区域中实际没有降水的区域占总预报降水区域的比重。

其数学公式如下所示：

$$FAR = \frac{\text{false alarms}}{\text{hits} + \text{false alarms}} \quad (4 - 15)$$

#### 4.5.3.4 漏报率 MAR

MAR 指的是实际降水区域中漏报的区域占据全部实际降水区域的比重。

其数学公式如下所示：

$$MAR = \frac{\text{misses}}{\text{hits} + \text{misses}} \quad (4 - 16)$$

#### 4.5.3.5 命中率 POD

POD 指的是预测出的实际的降水区域占据全部实际降水区域的比重。

其数学公式如下所示：

$$POD = \frac{hits}{hits + misses} = 1 - MAR \quad (4-17)$$

#### 4.5.3.6 均方根误差 RMSE

均方根误差（RMSE）是常见的统计量，用于衡量预测值与真实值之间的差异。在机器学习和数据分析领域，RMSE 通常被用作回归模型性能的指标。RMSE 是通过先计算平均误差的平方，再计算平均值，最后将结果取平方根得到的。

其数学公式如下：

$$RMSE = \sqrt{\frac{1}{m} \sum_1^m (Y_{Prd} - Y_{Tru})^2} \quad (4-18)$$

其中  $m$  是样本数量， $Y_{Prd}$  是预测值， $Y_{Tru}$  是真实值。

#### 4.5.4 模型精度评定结果

##### 4.5.4.1 降水事件一模型精度评定结果

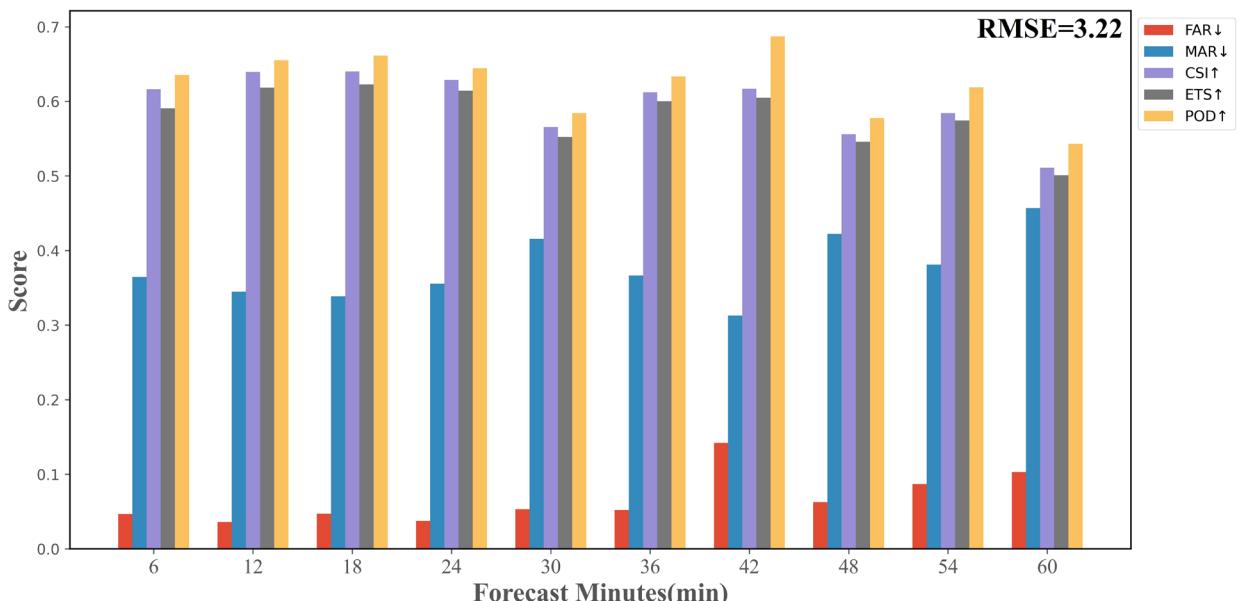


图 4-6 第 178 个降水事件 DRPP-Net 模型一小时预测的气象评估指数得分柱状图(↑代表数值越大越好，↓代表数值越小越好)，以 35 dBZ 为阈值

对于降水事件一，我们从图 4-6 中，可以观察得出 CSI 和 ETS 得分都在 0.5 以上，POD 得分在 0.6 以上，FAR 控制在 0.15 以下，整体 RMSE 为 3.22，预测精度良好。

##### 4.5.4.2 降水事件二模型精度评定结果

对于降水事件二，我们从图 4-7 中，可以观察得出 CSI 和 POD 得分在 0.5 以上，ETS 得分在 0.4 以上，FAR 得分控制在 0.1 以下，预测精度良好。

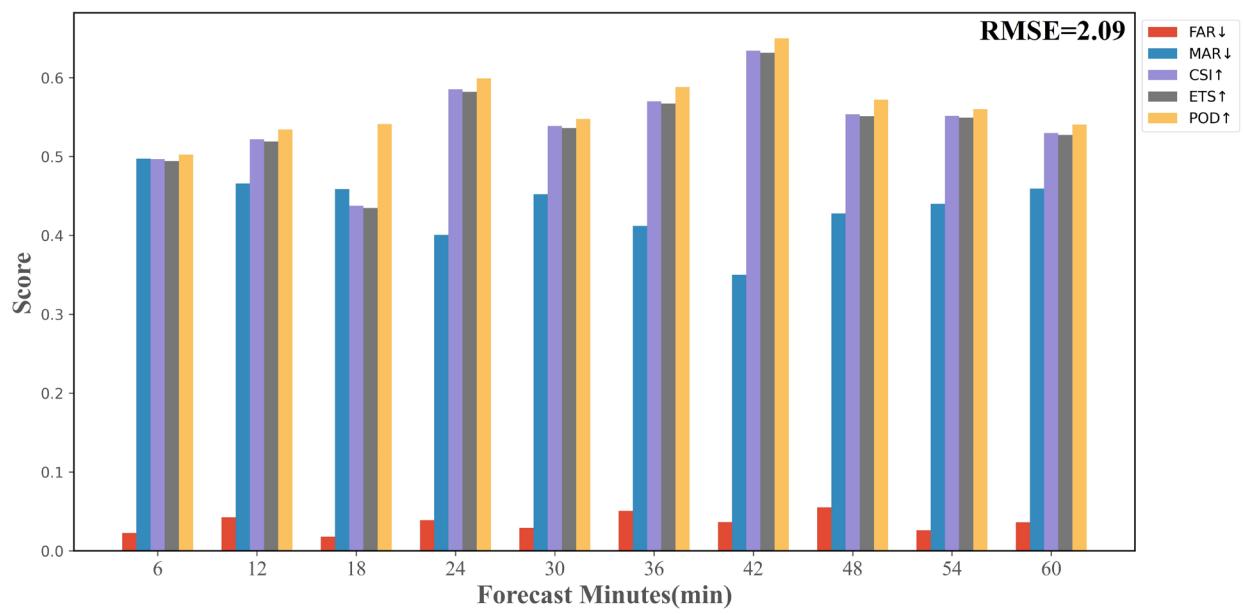


图 4-7 第 229 个降水事件 DRPP-Net 模型一小时预测的气象评估指数得分柱状图(↑ 代表数值越大越好, ↓ 代表数值越小越好), 以 35 dBZ 为阈值

## 5 问题二的分析与模型建立

### 5.1 问题二描述与分析

根据题目的描述，当前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，即存在回归到平均的问题，需要优化问题一的模型，缓解预测过程中的模糊效应。

问题二属于模型优化问题，出现回归到平均的问题主要有 1) 数据不平衡，即极端事件占比小，正常事件占比大，模型对极端事件的学习不充分，无法提取足够的极端事件特征；2) 数据预处理不正确，存在过度平滑和缩放问题，损失了极端事件的特征；3) 损失函数选择不正确，常见的损失函数，如均方误差对极端值的惩罚较小，对中间值的惩罚较大；4) 模型架构不够复杂，对于极端值的特征不够敏感。

本文主要针对最后一种情形，进行实验。对于模型架构不够复杂的问题，引入注意力机制架构，同时在原模型的基础上，对于差值信息进行训练，修正预测结果的趋于平均的问题。

具体步骤如下：

- 1、在原模型中引入注意力机制结构和差值结构，提高模型结构复杂度。
- 2、对比两个模型的精度。

问题二的技术流程如图 5-1 所示：

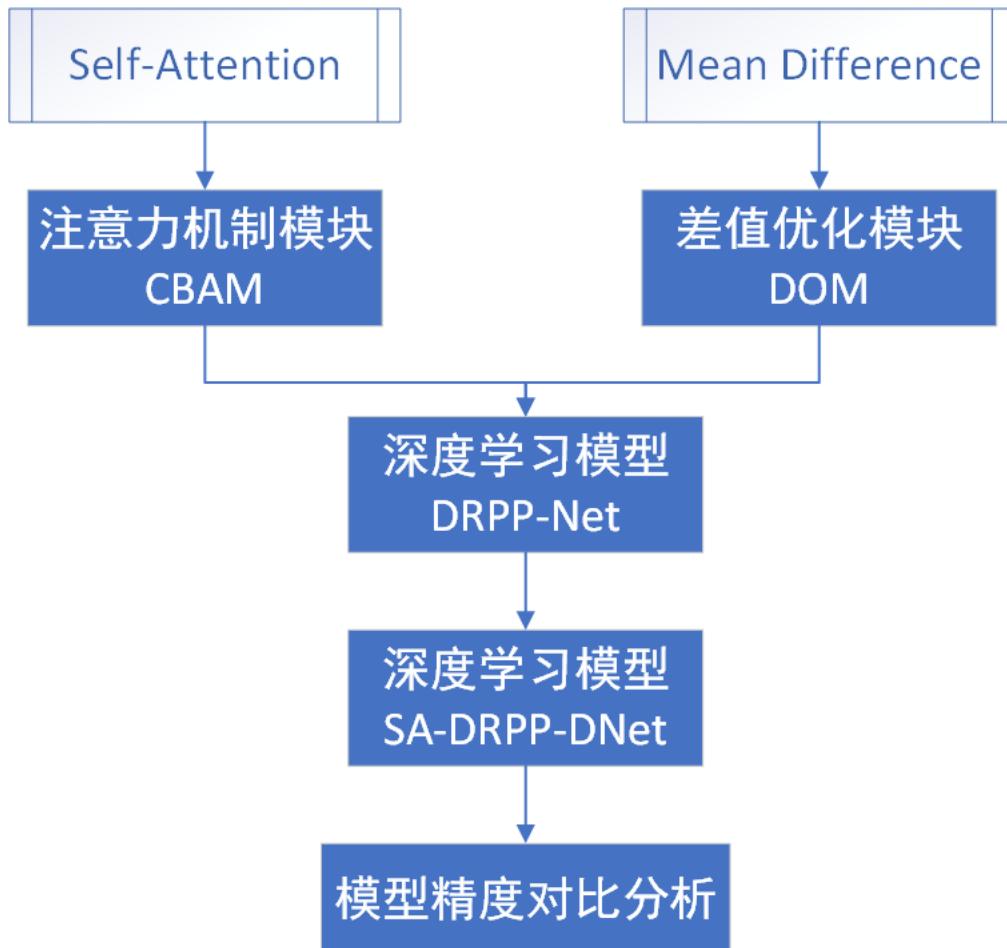


图 5-1 问题二技术流程图

### 5.2 注意力机制的原理

注意力机制可以使网络关注对目标有帮助的信息，从而增强网络的性能。

结合 ConvLSTM 和注意力机制的优点，并且为了使注意力系数能够随着网络的更新而自适应更新，将 LSTM 发展为自注意力 ConvLSTM。在中，ConvLSTM 继承了卷积算子的优点（例如稀疏连接和参数共享），保留了 LSTM 捕获长期记忆的优点，同时还基于全连接结构减少了模型的冗余，提高了模型的冗余度。网络的非线性建模能力，采用注意力机制来关注对预测结果有利的特征数据，同时抑制不太有用的数据。其目标是根据生成的预测数据与传入的降级数据之间的相似性，自适应地为降级数据分配权重。

为了剔除无关的冗余特征信息，提升模型的特征提取能力。因此，本方案设计并添加卷积注意力机制模块（Convolutional Block Attention Module, CBAM）注意力模块，CBAM 注意力机制整体流程如图 5-2 所示，由通道注意力机制与空间注意力机制组成，协同学习影像中的关键局部细节信息，为神经网络特征图中降水区域分配较高的权重，背景分配较低的权重，提高神经网络对影像中降水区域的关注度，进而提升网络的特征学习和表达能力<sup>[14]</sup>。

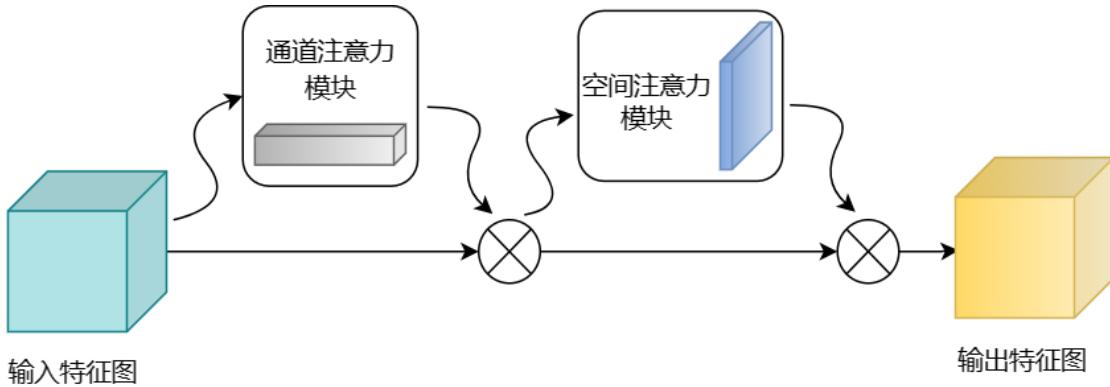


图 5-2 CBAM 模块结构图

其数学公式如下所示：

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}} + B\right)V \quad (5-1)$$

其中  $Q$  为查询， $K$  为地址， $V$  为值， $B$  为可学习的相对位置编码， $d_k$  为值  $V$  的维度。

因此，引入通道注意力机制模块（Channel Attention Module, CAM），建模特征图通道间的相互关系，有区别地对待不同通道的特征信息。通道注意力模块如图 5-3 所示，输入特征图记为  $F \in R^{C \times H \times W}$ ,  $H$  和  $W$  分别为输入特征图的高和宽， $C$  为输入特征图的通道数。通过全局最大池化和全局平均池化对特征图  $F$  的全局空间信息进行压缩，生成两个尺寸为  $C \times 1 \times 1$  的特征图  $S1$  和  $S2$ 。为了充分利用压缩操作提取的特征信息， $S1$  和  $S2$  经过全连接层和 LeakyReLU 非线性激活函数操作来获取深层次特征。对两个一维特征图按通道进行求和操作后使用 Sigmoid 函数进行归一化，得到尺寸为  $C \times 1 \times 1$  的通道权重特征图。

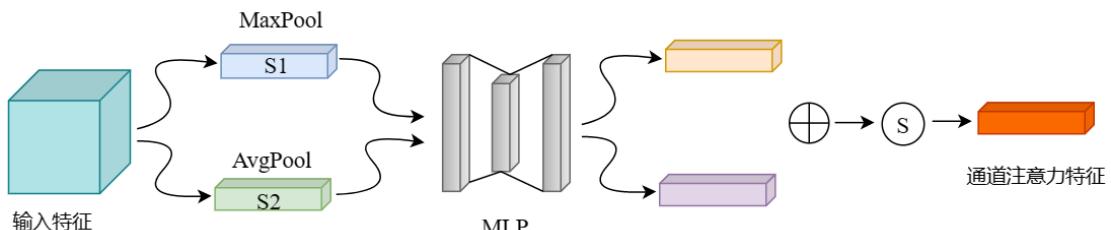


图 5-3 CAM 模块结构图

空间注意力机制模块（Spatial Attention Module, SAM）以通道特征重标定后得到的特

征图 $F'$ 作为输入，在通道维度上分别做全局最大池化和平均池化操作，得到 $P1 \in R^{1 \times H \times W}$ 、 $P2 \in R^{1 \times H \times W}$ 。通过通道级联的方式得到特征描述 $P3 \in R^{2 \times H \times W}$ ，利用卷积层对 $P3$ 中不同位置的信息进行编码和融合，并使用 Sigmoid 非线性函数进行归一化操作，得到空间权重特征图，其可用于区分图像不同空间位置的重要程度。其结构图如图 5-4 所示：

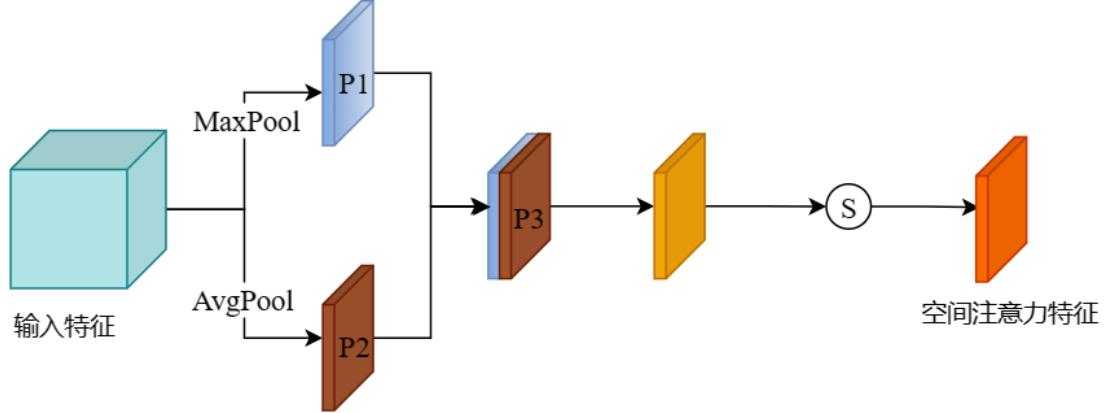


图 5-4 SAM 模块结构图

### 5.3 差值优化模块 DOM 的结构与原理

输入双偏振雷达数据，输出为预测值与真值的差值，优化训练差值，融合 CBAM 模块，对预测结果进行修正，其结构图如图 5-5 所示：

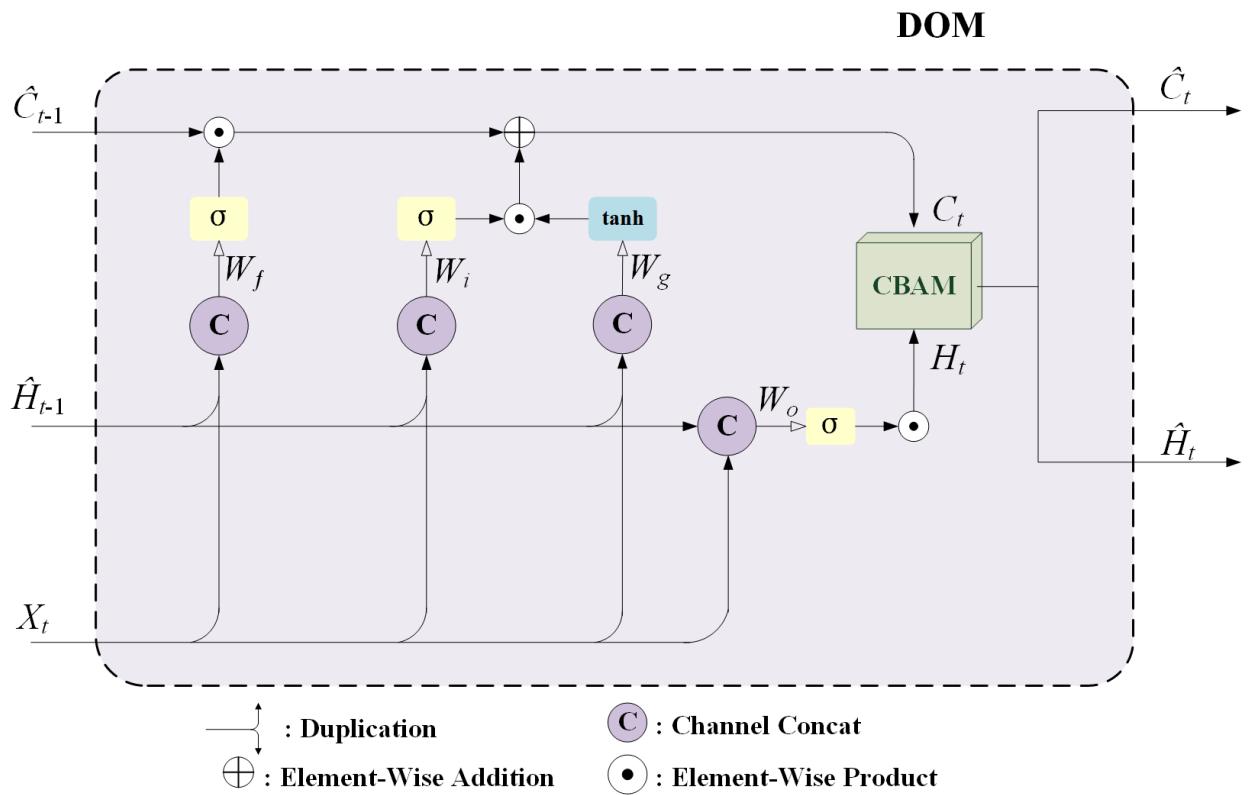
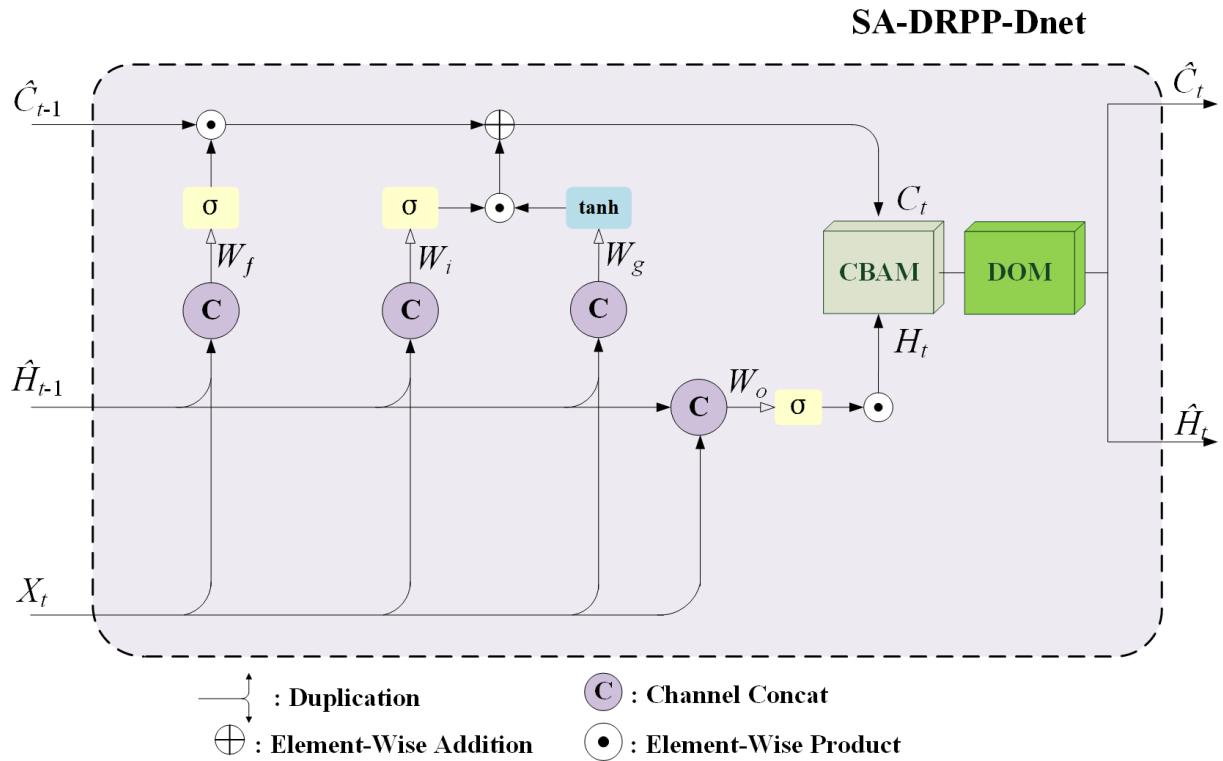


图 5-5 DOM 模块的结构图

### 5.4 SA-DRPP-DNet 的结构与原理

本部分以 DRPP-Net 为基本深度学习框架，融合了 CBAM 模块和 DOM 模块，构建了优化后的雷达回波水平反射率因子深度学习预测模型，即 SA-DRPP-DNet(Self-Attention Dual-Polarization Radar Precipitation Prediction Difference Net)。

其结构框架如图 5-6 所示：



## 5.5 预测结果和模型精度对比

选用测试集中的第 178 个降水事件和第 229 个降水事件作为代表性降水事件。

### 5.5.1 降水事件一预测结果与精度评定

第 178 个降水事件属于大范围强降水事件。图 5-7 为第 178 个降水事件的水平反射率因子的真值和预测值，输入数据为第 178 个降水事件前一个小时的水平反射率因子  $Z_h$ 、差分反射率  $Z_{DR}$  和比差分相移  $K_{DP}$ 。我们从结果中观察到，预测值和真值的趋势是一致的，数值上也是接近的，但也存在一些数值上的低估情况出现，如第 42 分钟的预测值，出现了明显的低估。但总体上，预测效果是很好的。

从图 5-8 上看，POD 得分超过 0.7，CSI 和 ETS 得分都超过了 0.6，FAR 得分控制在 0.2 以内，整体 RMSE 为 3.07。

**Truth**

**Prediction**

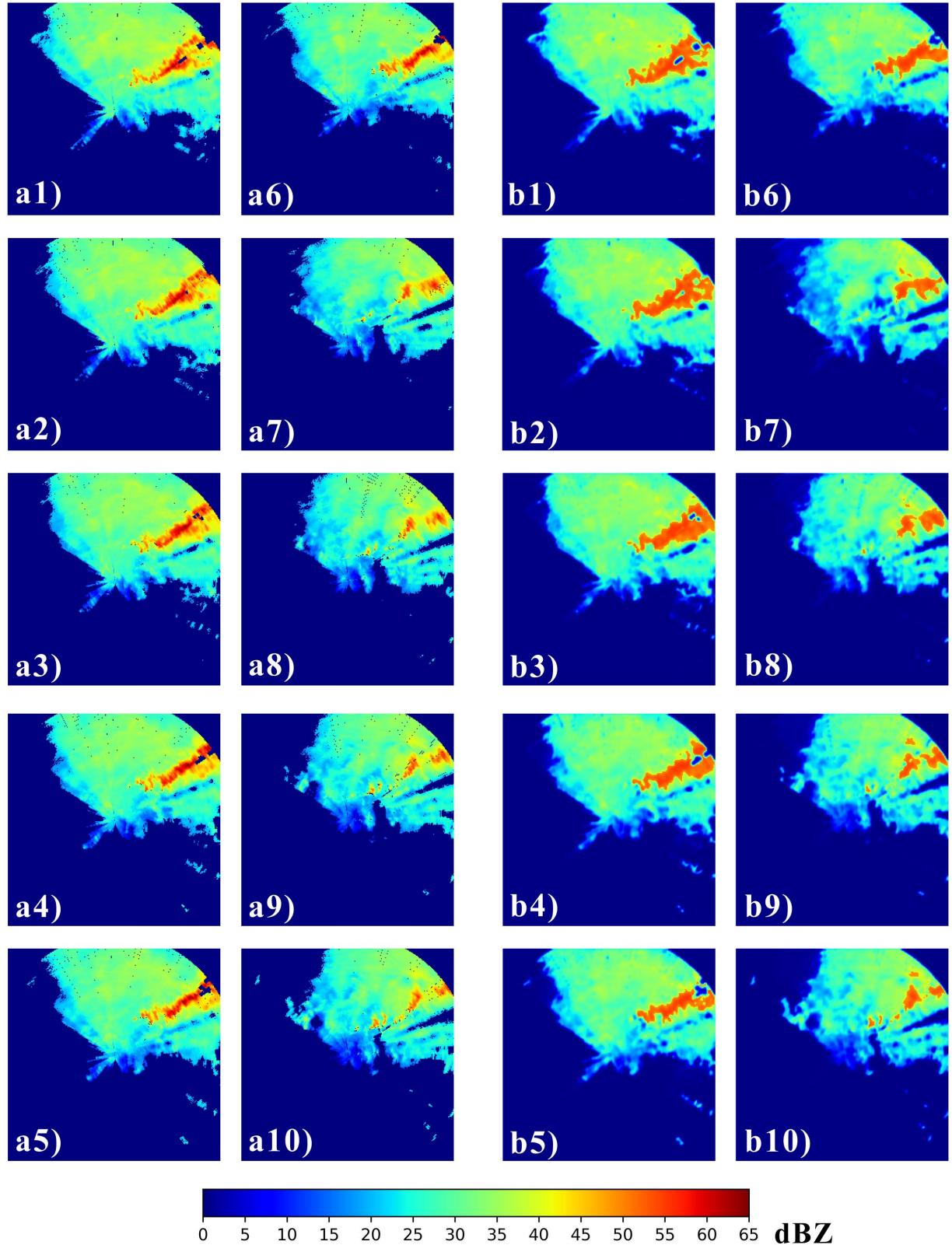


图 5-7 第 178 个降水事件的水平反射率因子  $Z_h$  真值(a1-a10)和 SA-DRPP-DNet 的水平反射率因子  $Z_h$  预测值(b1-b10), 分别对应于 6 分钟、12 分钟、18 分钟、24 分钟、30 分钟、36 分钟、42 分钟、48 分钟、54 分钟和 60 分钟

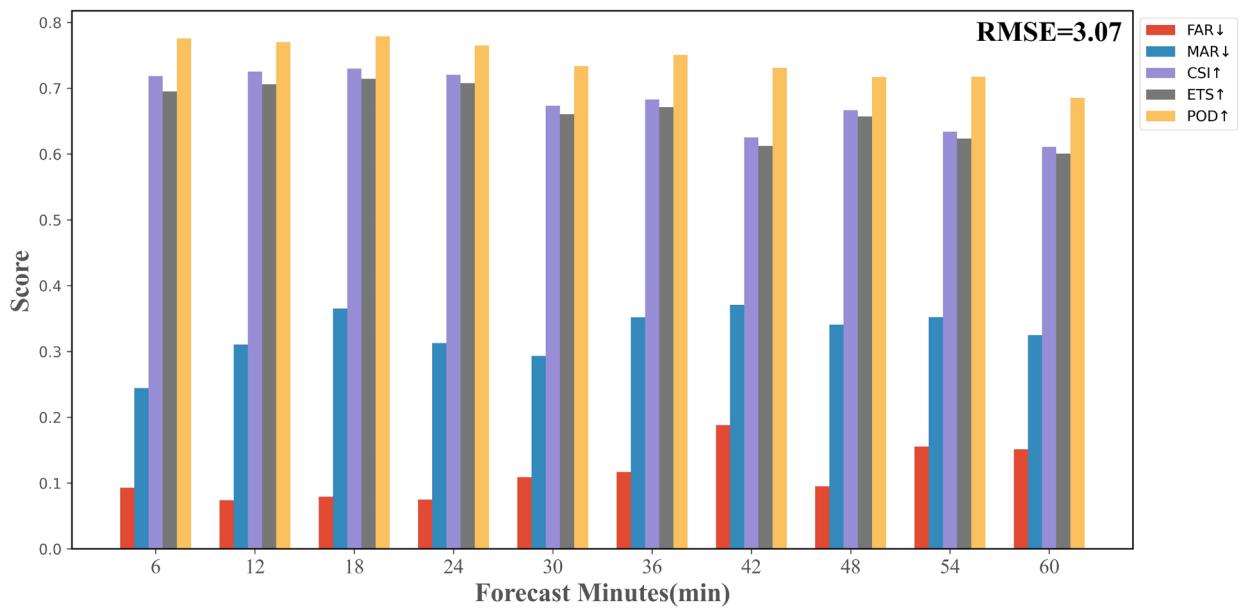


图 5-8 第 178 个降水事件 SA-DRPP-DNet 模型一小时预测的气象评估指数得分柱状图(↑代表数值越大越好, ↓代表数值越小越好), 以 35 dBZ 为阈值

### 5.5.2 降水事件二预测结果与精度评定

第 229 个降水事件属于小范围强降水事件。图 5-10 为第 229 个降水事件的水平反射率因子的真值和预测值, 输入数据为第 229 个降水事件前一个小时的水平反射率因子  $Z_h$ 、差分反射率  $Z_{DR}$  和比差分相移  $K_{DP}$ 。我们从结果上观察, 预测值和真值的趋势是一致的, 数值上也是接近的, 但依然存在模糊效应, 但已经得到了一定的改善。

从图 5-9 可以看出, CSI、ETS 和 POD 得分都超过了 0.6, FAR 得分控制在 0.1 之内, 整体 RMSE 为 2.03。

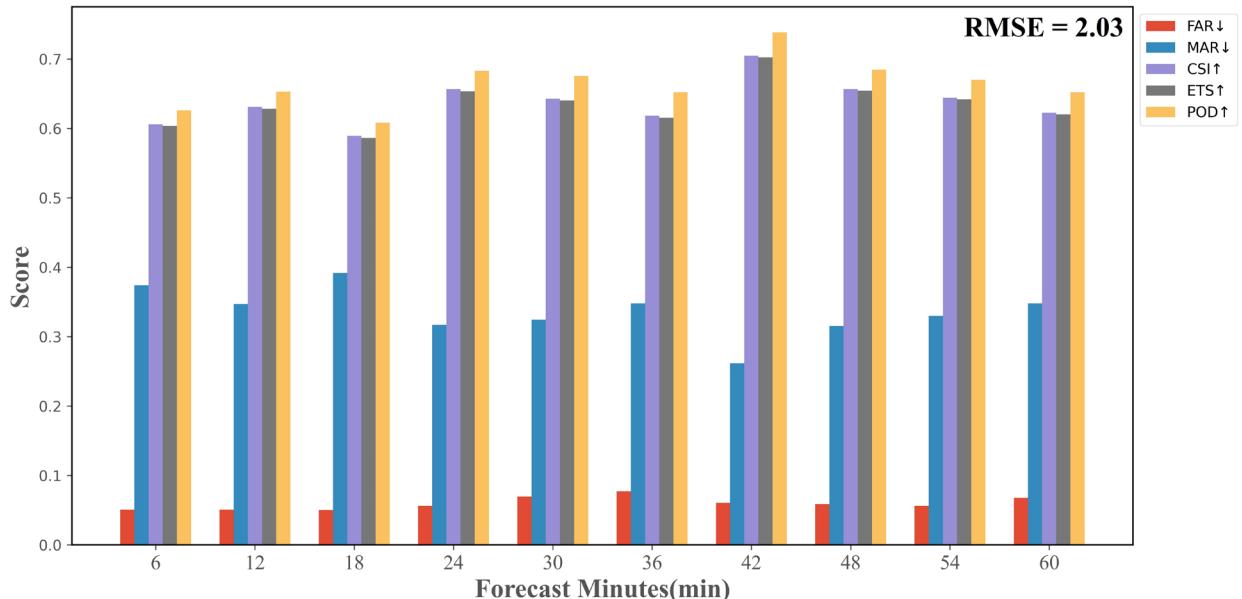


图 5-9 第 229 个降水事件 SA-DRPP-DNet 模型一小时预测的气象评估指数得分柱状图(↑代表数值越大越好, ↓代表数值越小越好), 以 35 dBZ 为阈值

## Truth

## Prediction

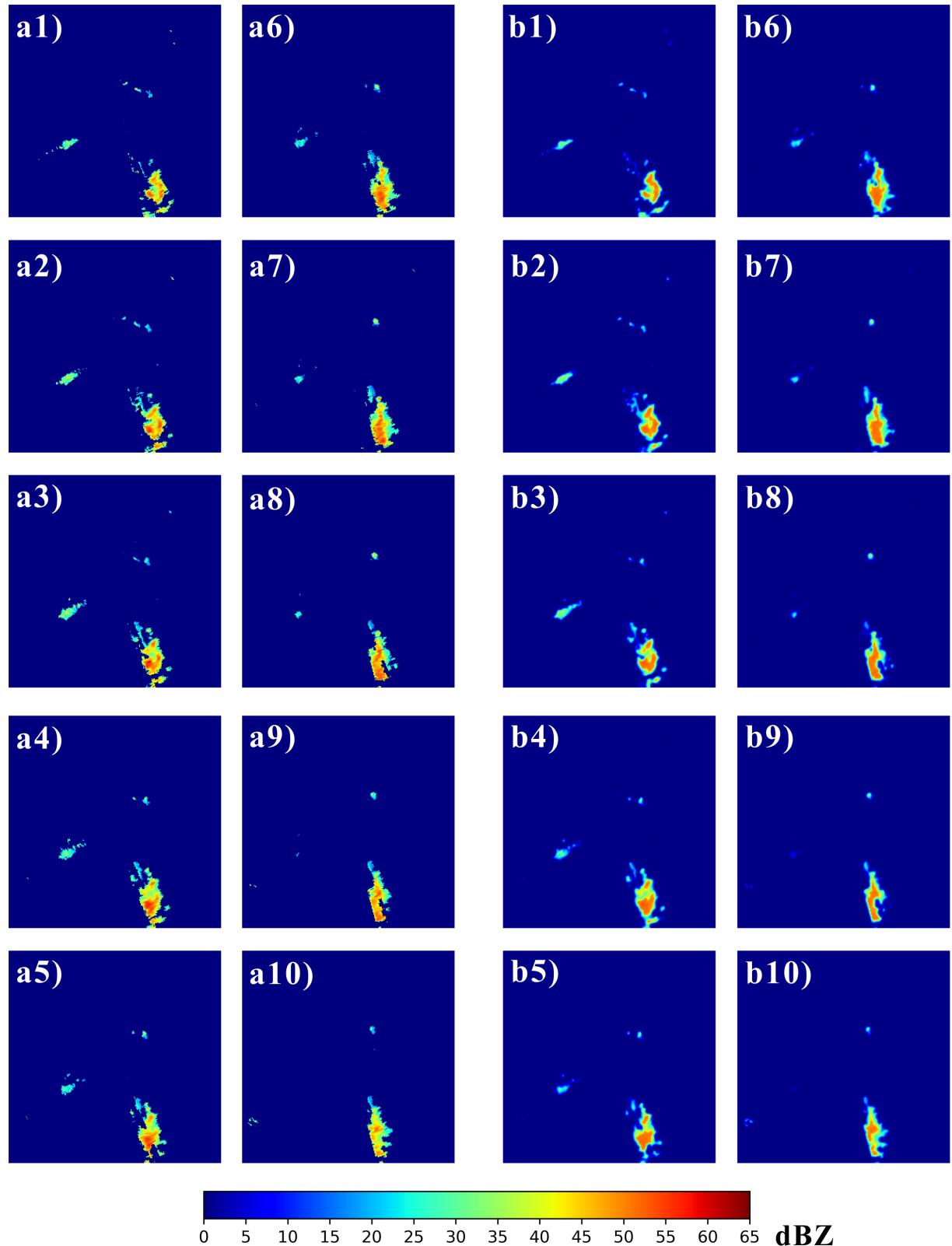


图 5-10 第 229 个降水事件的水平反射率因子  $Z_h$  真值(a1-a10)和 SA-DRPP-DNet 的水平反射率因子  $Z_h$  预测值(b1-b10), 分别对应于 6 分钟、12 分钟、18 分钟、24 分钟、30 分钟、36 分钟、42 分钟、48 分钟、54 分钟和 60 分钟

### 5.5.3 降水事件一 DRPP-Net 与 SA-DRPP-DNet 对比分析

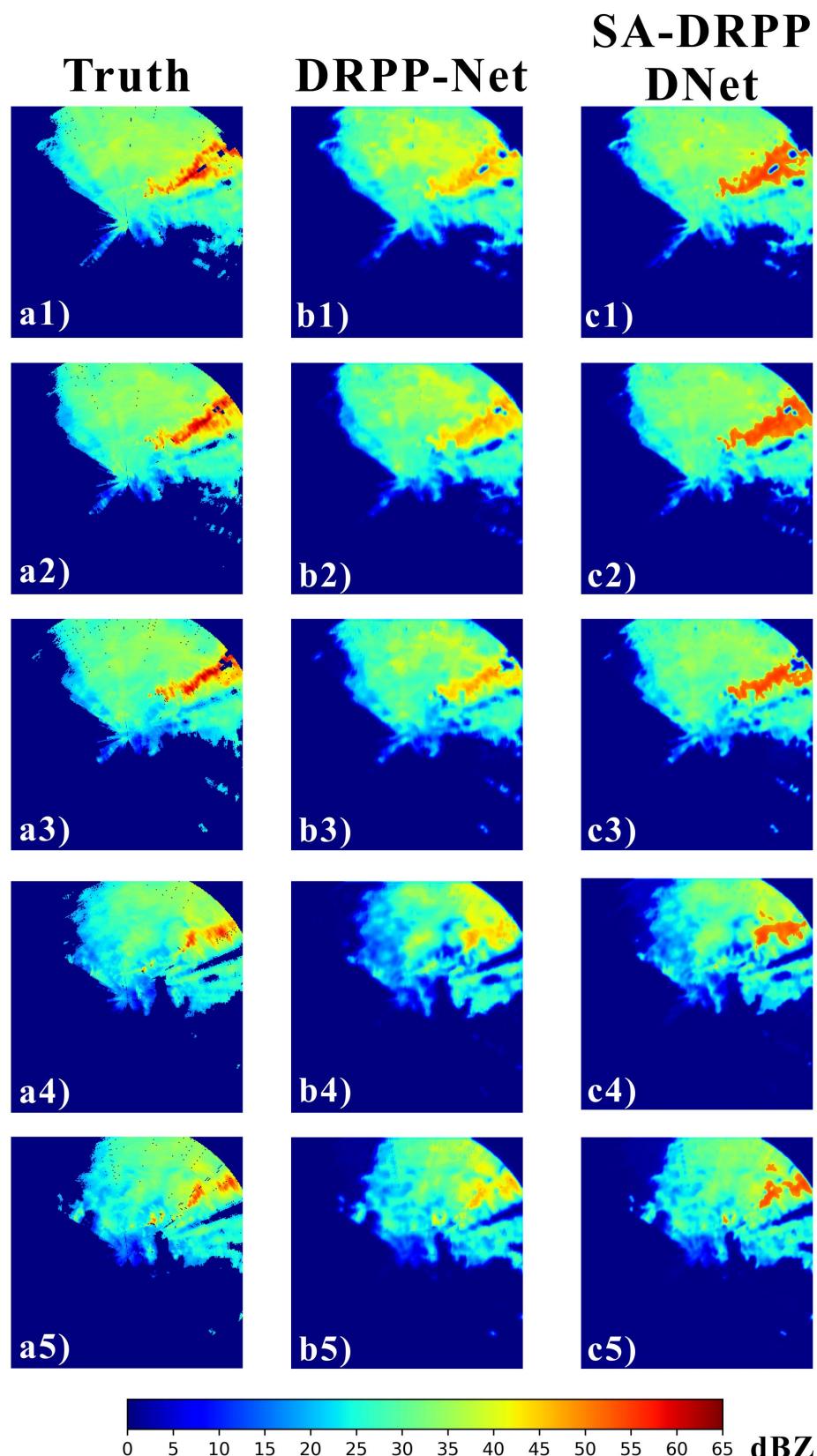


图 5-11 第 178 个降水事件的水平反射率因子 $Z_h$ 的真值(a1-a5)、DRPP-Net 预测值(b1-b5)和 SA-DRPP-DNet 预测值(c1-c5)，分别对应于 6 分钟、18 分钟、30 分钟、42 分钟和 54 分钟

表 5-1 第 178 个降水事件一小时的 DRPP-Net 气象评估指数得分与 SA-DRPP-DNet 气象评估指数得分(CSI、ETS 和 POD), 数值越大代表模型精度越高

预测时间	CSI DRPP-Net	CSI SA-DRPP-DNet	ETS DRPP-Net	ETS SA-DRPP-DNet	POD DRPP-Net	POD SA-DRPP-DNet
6	0.616	<b>0.718</b>	0.591	<b>0.695</b>	0.635	<b>0.776</b>
12	0.640	<b>0.725</b>	0.619	<b>0.706</b>	0.655	<b>0.770</b>
18	0.640	<b>0.730</b>	0.623	<b>0.714</b>	0.661	<b>0.779</b>
24	0.629	<b>0.720</b>	0.614	<b>0.708</b>	0.644	<b>0.765</b>
30	0.566	<b>0.673</b>	0.552	<b>0.660</b>	0.584	<b>0.734</b>
36	0.612	<b>0.683</b>	0.600	<b>0.671</b>	0.633	<b>0.751</b>
42	0.617	<b>0.625</b>	0.605	<b>0.612</b>	0.687	<b>0.731</b>
48	0.556	<b>0.667</b>	0.546	<b>0.657</b>	0.578	<b>0.717</b>
54	0.584	<b>0.634</b>	0.574	<b>0.624</b>	0.619	<b>0.718</b>
60	0.511	<b>0.679</b>	0.501	<b>0.601</b>	0.543	<b>0.685</b>

表 5-2 第 178 个降水事件一小时的 DRPP-Net 气象评估指数得分与 SA-DRPP-DNet 气象评估指数得分(FAR、MAR 和 RMSE), 数值越小代表模型精度越高

预测时间	FAR DRPP-Net	FAR SA-DRPP-DNet	MAR DRPP-Net	MAR SA-DRPP-DNet	RMSE DRPP-Net	RMSE SA-DRPP-DNet
6	<b>0.047</b>	0.093	0.365	<b>0.244</b>	3.276	<b>3.164</b>
12	<b>0.036</b>	0.074	0.345	<b>0.311</b>	3.140	<b>3.023</b>
18	<b>0.047</b>	0.079	<b>0.339</b>	0.365	3.240	<b>3.078</b>
24	<b>0.038</b>	0.075	0.356	<b>0.313</b>	3.319	<b>3.165</b>
30	<b>0.053</b>	0.109	0.416	<b>0.293</b>	3.370	<b>3.258</b>
36	<b>0.052</b>	0.117	0.367	<b>0.352</b>	3.112	<b>2.996</b>
42	<b>0.142</b>	0.188	<b>0.313</b>	0.371	3.599	<b>3.254</b>
48	<b>0.063</b>	0.095	0.422	<b>0.341</b>	3.075	<b>2.916</b>
54	<b>0.087</b>	0.155	0.381	<b>0.352</b>	2.953	<b>2.850</b>
60	<b>0.103</b>	0.151	0.457	<b>0.325</b>	3.102	<b>2.974</b>

表 5-3 第 178 个降水事件一小时的 DRPP-Net 气象评估指数得分与 SA-DRPP-DNet 气象评估指数得分的差值, 已经做了逆指标一致化处理, 正值代表有提升, 负值代表无提升

预测时间	FAR	MAR	CSI	ETS	POD	RMSE
6	-0.046	0.120	0.102	0.104	0.140	0.112
12	-0.038	0.034	0.086	0.088	0.115	0.117
18	-0.032	-0.026	0.090	0.091	0.118	0.163
24	-0.037	0.043	0.092	0.093	0.120	0.154
30	-0.056	0.122	0.108	0.108	0.149	0.112
36	-0.065	0.015	0.071	0.071	0.117	0.116
42	-0.046	-0.058	0.008	0.007	0.044	0.345
48	-0.032	0.082	0.111	0.111	0.139	0.158
54	-0.068	0.029	0.050	0.049	0.099	0.103
60	-0.048	0.132	0.100	0.100	0.142	0.128

从图 5-11, 我们可以看出对于降水事件一, 预测结果的模糊效应得到改善, 预测结果

更接近于真值。

从表 5-1、表 5-2 和表 5-3, 我们可以发现 SA-DRPP-DNet 的整体精度高于 DRPP-Net, 其中 CSI 平均提升了 0.082, ETS 平均提升了 0.082, POD 平均提升了 0.118, MAR 平均降低了 0.049, RMSE 平均降低了 0.151。因此, 对于降水事件一, SA-DRPP-DNet 的精度更高, CBAM 模块和 DOM 模块的引入提高了预测精度, 减轻了模糊效应。

#### 5.5.4 降水事件二 DRPP-Net 与 SA-DRPP-DNet 对比分析

表 5-4 第 229 个降水事件一小时的 DRPP-Net 气象评估指数得分与 SA-DRPP-DNet 气象评估指数得分(CSI、ETS 和 POD), 数值越大代表模型精度越高

预测时间	CSI DRPP-Net	CSI SA-DRPP-DNet	ETS DRPP-Net	ETS SA-DRPP-DNet	POD DRPP-Net	POD SA-DRPP-DNet
6	0.497	<b>0.606</b>	0.494	<b>0.604</b>	0.503	<b>0.626</b>
12	0.522	<b>0.631</b>	0.519	<b>0.628</b>	0.534	<b>0.653</b>
18	0.438	<b>0.589</b>	0.435	<b>0.586</b>	0.541	<b>0.608</b>
24	0.585	<b>0.656</b>	0.582	<b>0.654</b>	0.599	<b>0.683</b>
30	0.539	<b>0.643</b>	0.536	<b>0.641</b>	0.548	<b>0.676</b>
36	0.570	<b>0.618</b>	0.567	<b>0.615</b>	0.588	<b>0.652</b>
42	0.634	<b>0.705</b>	0.632	<b>0.702</b>	0.650	<b>0.738</b>
48	0.554	<b>0.657</b>	0.551	<b>0.654</b>	0.572	<b>0.685</b>
54	0.552	<b>0.644</b>	0.549	<b>0.642</b>	0.560	<b>0.670</b>
60	0.530	<b>0.623</b>	0.527	<b>0.620</b>	0.541	<b>0.652</b>

表 5-5 第 229 个降水事件一小时的 DRPP-Net 气象评估指数得分与 SA-DRPP-DNet 气象评估指数得分(FAR、MAR 和 RMSE), 数值越小代表模型精度越高

预测时间	FAR DRPP-Net	FAR SA-DRPP-DNet	MAR DRPP-Net	MAR SA-DRPP-DNet	RMSE DRPP-Net	RMSE SA-DRPP-DNet
6	<b>0.023</b>	0.051	0.497	<b>0.374</b>	2.177	<b>2.148</b>
12	<b>0.043</b>	0.051	0.466	<b>0.347</b>	2.311	<b>2.267</b>
18	<b>0.018</b>	0.050	0.459	<b>0.392</b>	2.329	<b>2.232</b>
24	<b>0.039</b>	0.056	0.401	<b>0.317</b>	2.203	<b>2.094</b>
30	<b>0.029</b>	0.070	0.452	<b>0.324</b>	2.185	<b>2.109</b>
36	<b>0.051</b>	0.077	0.412	<b>0.348</b>	2.100	<b>2.041</b>
42	<b>0.036</b>	0.061	0.350	<b>0.262</b>	1.949	<b>1.906</b>
48	<b>0.055</b>	0.059	0.428	<b>0.315</b>	2.028	<b>1.978</b>
54	<b>0.026</b>	0.056	0.440	<b>0.330</b>	1.835	<b>1.815</b>
60	<b>0.036</b>	0.068	0.459	<b>0.348</b>	1.765	<b>1.701</b>

从图 5-12, 我们可以看出对于降水事件二, 预测结果的模糊效应得到改善, 预测结果更接近于真值。

从表 5-4、表 5-5 和表 5-6, 我们可以发现 SA-DRPP-DNet 的整体精度高于 DRPP-Net, 其中 CSI 平均提升了 0.095, ETS 平均提升了 0.095, POD 平均提升了 0.101, MAR 平均降低了 0.101, RMSE 平均降低了 0.059。因此, 对于降水事件二, SA-DRPP-DNet 的精度更高, CBAM 模块和 DOM 模块的引入提高了预测精度, 减轻了模糊效应。

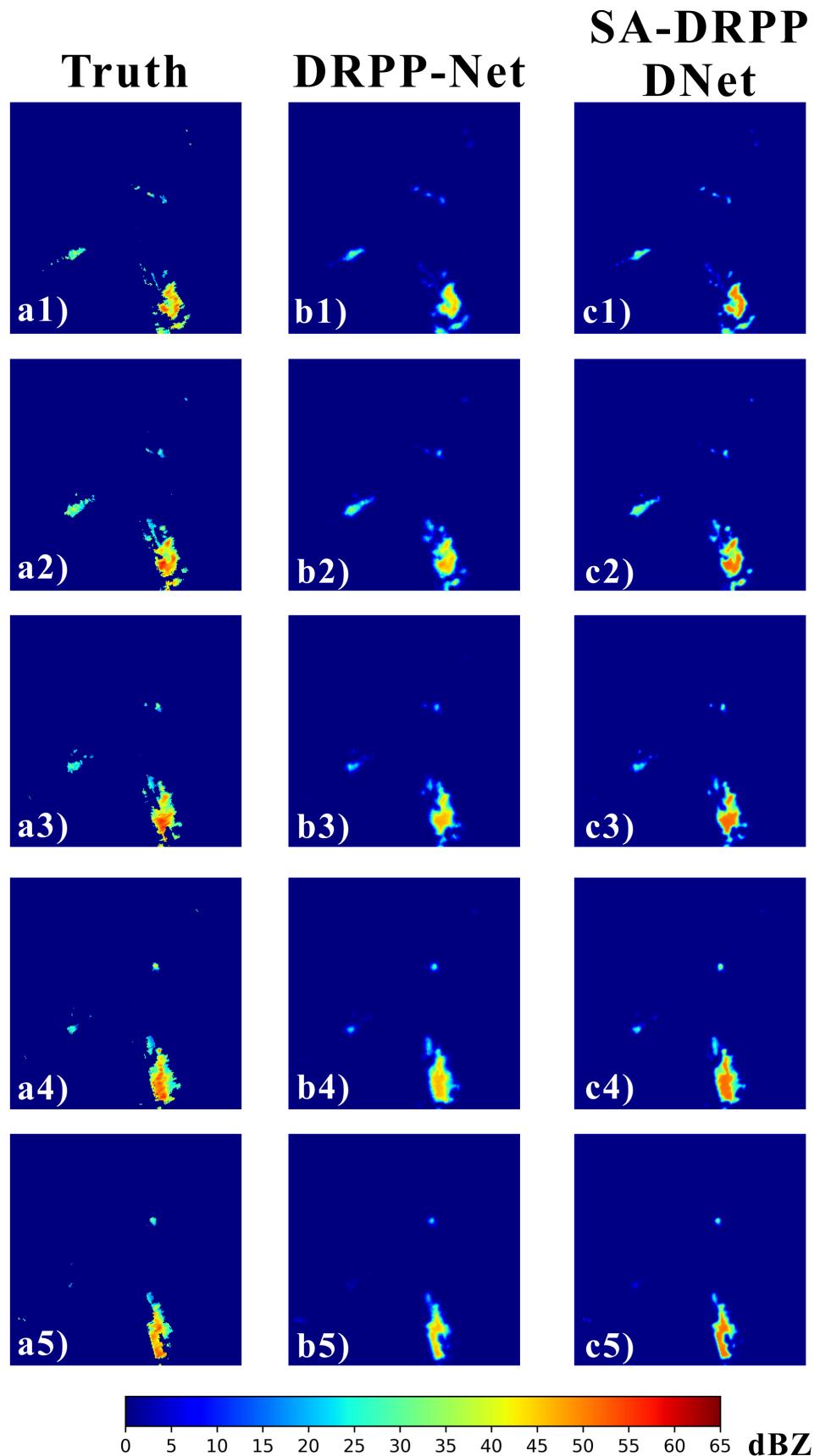


图 5-12 第 229 个降水事件的水平反射率因子 $Z_h$ 的真值(a1-a5)、DRPP-Net 预测值(b1-b5)和 SA-DRPP-DNet 预测值(c1-c5)，分别对应于 6 分钟、18 分钟、30 分钟、42 分钟和 54 分钟

表 5-6 第 229 个降水事件一小时的 DRPP-Net 气象评估指数得分与 SA-DRPP-DNet 气象评估指数得分的差值，已经做了逆指标一致化处理，正值代表有提升，负值代表无提升

预测时间	FAR	MAR	CSI	ETS	POD	RMSE
6	-0.028	0.124	0.109	0.109	0.124	0.030
12	-0.008	0.119	0.109	0.109	0.119	0.044
18	-0.032	0.067	0.152	0.152	0.067	0.097
24	-0.017	0.084	0.071	0.072	0.084	0.109
30	-0.041	0.128	0.104	0.104	0.128	0.076
36	-0.027	0.064	0.048	0.048	0.064	0.059
42	-0.024	0.089	0.071	0.071	0.089	0.043
48	-0.004	0.113	0.103	0.103	0.113	0.050
54	-0.030	0.110	0.093	0.093	0.110	0.020
60	-0.031	0.111	0.093	0.093	0.111	0.064

## 6 问题三的分析与模型建立

### 6.1 问题三描述与分析

根据题目的描述，问题三要求根据题目提供的双偏振雷达观测数据和降水格点数据，输入 $3\text{ km}$ 等高面的 $Z_h$ 和 $Z_{DR}$ 数据，定量输出相应格点的 $K_{DP\_Rain}$ 数据。

问题三属于回归问题，传统的降水量预测模型大多依赖经验公式的拟合和回归，对于短临强降水，无法适应其变化快、形态多变的特征，因此引入深度学习算法，对于雷达大数据，剔除弱降水或无降水的降水事件，构建 $1\text{ km}$ 等高面的强降水双偏振雷达观测数据集，对于降水格点数据集，进行数据空洞弥补。随机划分为训练样本和测试样本，输入深度学习 PP-Net 模型，得到降水格点的定量预测值，进行模型精度评定。

问题三的技术流程如图 6-1 所示：

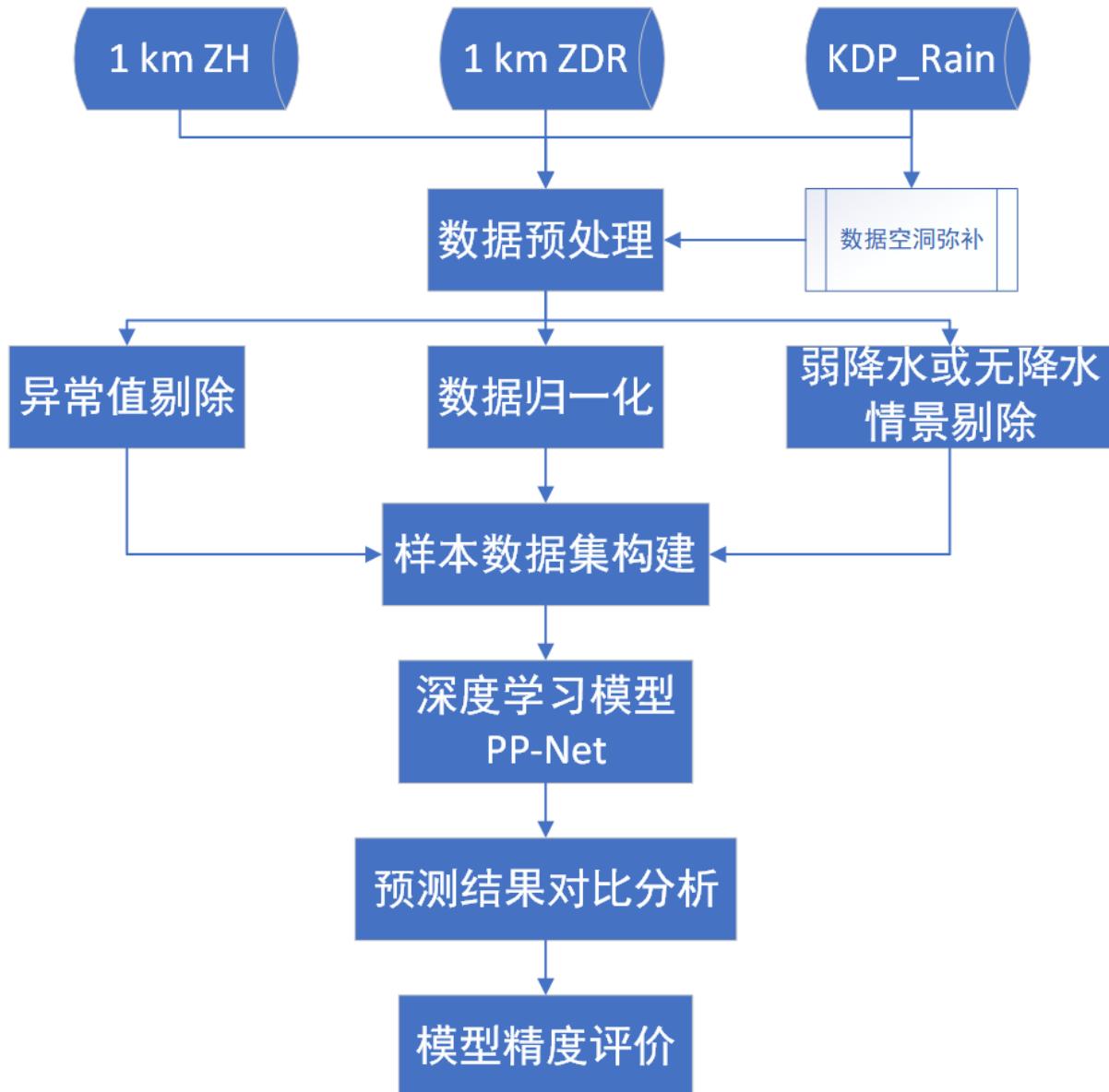


图 6-1 问题三技术流程图

### 6.2 降水格点数据空洞弥补

应用降水格点数据，首先对降水格点数据进行预处理，发现部分降水格点数据存在空洞，即水平反射率因子 $>15\text{ dBZ}$ ，且差分反射率 $>1\text{ dB}$ 的区域，理论上属于降水区域，但降

水格点数据中值为 0，同时也存在水平反射率因子 $<15 \text{ dBZ}$ ，且差分反射率 $<1 \text{ dB}$ 的区域，即无降水区域，降水格点数据存在值不为 0 的情况。如图 6-2 所示。

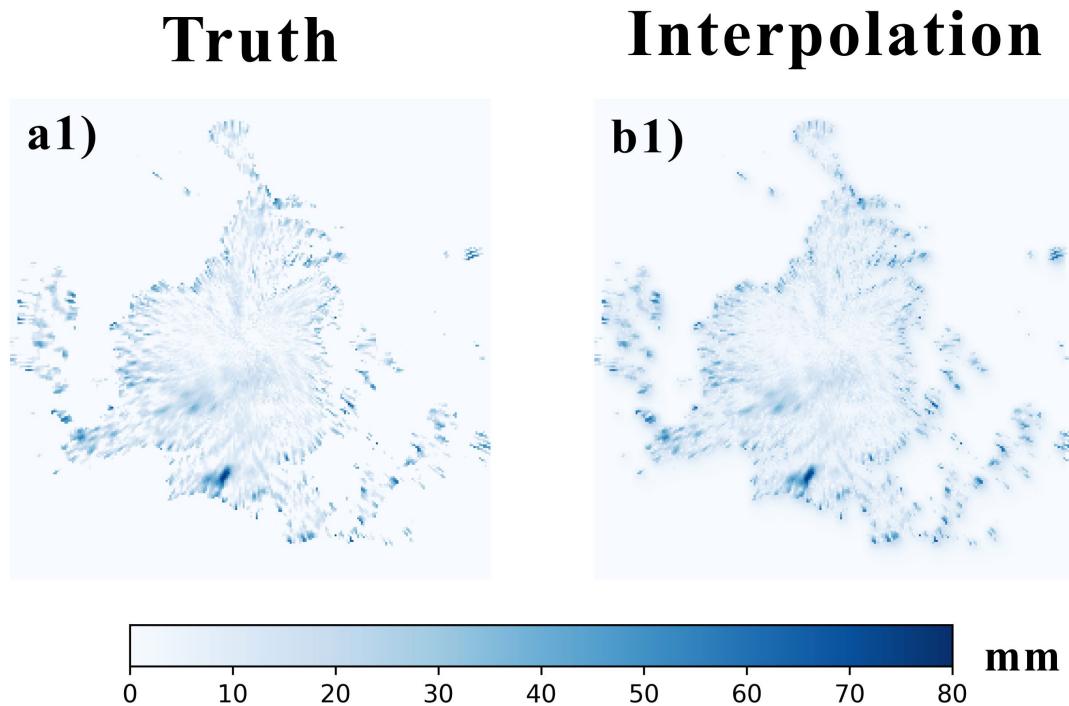


图 6-2 数据问题示例

对于前者，首先我们对无降水区域（水平反射率因子 $<15 \text{ dBZ}$ ，且差分反射率 $<1 \text{ dB}$ 的区域）进行提取，获取无降水区域的掩膜信息，然后对降水格点数据进行掩膜，即未掩膜区域为降水区域，对其中降水格点数据为 0 值的，进行最邻近插值，得到较为合理的降水格点数据。

对于后者，为了更好的降水量定量估计结果，首先对降水格点数据进行非 0 值提取，获取降水区域的掩膜信息，对水平反射率因子和差分反射率数据进行掩膜，即未掩膜区域为无降水区域，对其中水平反射率因子和差分反射率不为 0 的值，设为 0 值。

### 6.3 传统降水定量预测方法

传统基于统计分析的方法，主要使用概率论和统计学的方法，对降水量进行预测，主要是选择相关的气象要素和降水量，拟合回归，求解出相应的回归方程，即当前情境下的降水量经验公式，称为 Z-R 关系，即反射率因子 Z 与降水强度 R 的关系公式<sup>[15]</sup>。

常见的 Z-R 关系包括传统默认关系(Default Relationship)、罗斯菲尔德热带关系(Rosenfeld tropical Realtionship)和马歇尔·帕尔默关系(Marshall-Palmer Realtionship)，其公式如下：

表 6-1 传统 Z-R 关系及数学公式

关系	公式	
传统默认 Z-R 关系	$Z = 300R^{1.4}$	(6-1)
罗斯菲尔德热带 Z-R 关系	$Z = 250R^{1.2}$	(6-2)
马歇尔·帕尔默 Z-R 关系	$Z = 200R^{1.6}$	(6-3)

随着双偏振雷达观测数据的引入，专家学者利用获取到的水平反射率因子 $Z_h$ 、差分反射率 $Z_{DR}$ 和比差分相移 $K_{DP}$ ，进行组合，获取到四类双偏振雷达观测数据和降水量的关系公式：

$$R(Z_h) = 0.017Z_h^{0.714} \quad (6-4)$$

$$R(Z_h, Z_{DR}) = 6.7 * 10^{-3}Z_h^{0.927} * 10^{-0.343Z_{DR}} \quad (6-5)$$

$$R(K_{DP}) = 40.5K_{DP}^{0.85} \quad (6-6)$$

$$R(K_{DP}, Z_{DR}) = 90.8K_{DP}^{0.85} * 10^{-0.169Z_{DR}} \quad (6-7)$$

对于输入水平反射率因子 $Z_h$ 和差分反射率 $Z_{DR}$ 来预测降水量，主要有两种 Z-R 关系，即：

$$R(Z_h, Z_{DR}) = aZ_h^b 10^{cZ_{DR}} \quad (6-8)$$

$$R(Z_h, Z_{DR}) = aZ_h^b Z_{DR}^c \quad (6-9)$$

#### 6.4 swin transformer 模块结构与原理

swin transformer 使用了类似卷积神经网络中的层次化构建方法，比如特征图尺寸中有对图像下采样 4 倍的，8 倍的以及 16 倍的，这样的结构有助于在此基础上进行特征图的提取。

在 swin transformer 中使用了 Windows Multi-Head Self-Attention(W-MSA)的模块，比如在 4 倍下采样和 8 倍下采样中，将特征图划分成了多个不相交的区域，并且 Multi-Head Self-Attention 只在每个窗口内进行，这样做的目的是能够减少计算量的，尤其是在浅层特征图很大的时候。这样做虽然减少了计算量，但也会隔绝不同窗口之间的信息传递，因此添加了 Shifted Windows Multi-Head Self-Attention(SW-MSA)模块，让信息在相邻的窗口中进行传递。

其数学公式如下所示：

$$\hat{Z}^l = W - MSA\left(LN(Z^{l-1})\right) + Z^{l-1} \quad (6-10)$$

$$Z^l = MLP\left(LN(\hat{Z}^l)\right) + \hat{Z}^l \quad (6-11)$$

$$\hat{Z}^{l+1} = W - MSA\left(LN(Z^l)\right) + Z^{l+1} \quad (6-12)$$

$$Z^{l+1} = MLP\left(LN(\hat{Z}^{l+1})\right) + \hat{Z}^{l+1} \quad (6-13)$$

其中 $\hat{Z}^l$ 是 W-MSA 模块的输出， $Z^l$ 是 MLP(Multilayer Perceptron，多层感知机)模块的输出， $l$ 代表第 $l$ 个 swin transformer 模块。

其结构框架如图 6-3 所示：

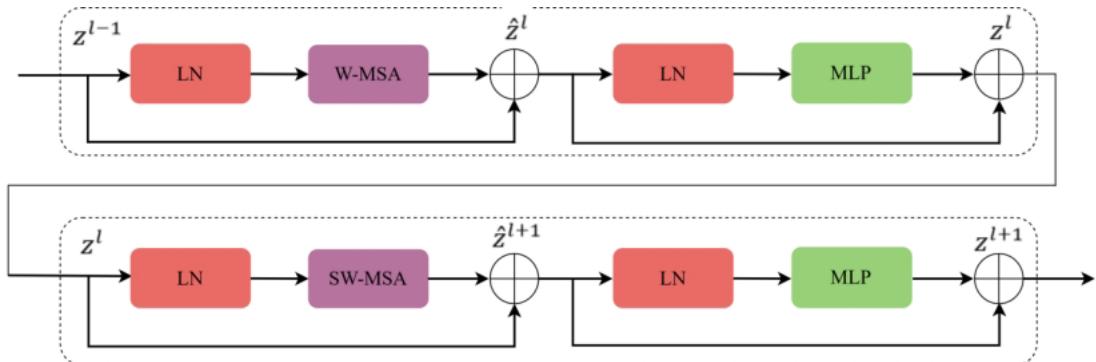


图 6-3 两个相连的 swin transformer 模块结构图

#### 6.5 ASPP 模块结构与原理

ASPP(Atrous Spatial Pyramid Pooling，空洞空间金字塔池化)对于给定的输入以不同采样率的空洞卷积并行采样，将得到的结果连接到一起，扩大通道数，然后再通过卷积将通道数降低到预期的数值。

其结构框架如图 6-4 所示：

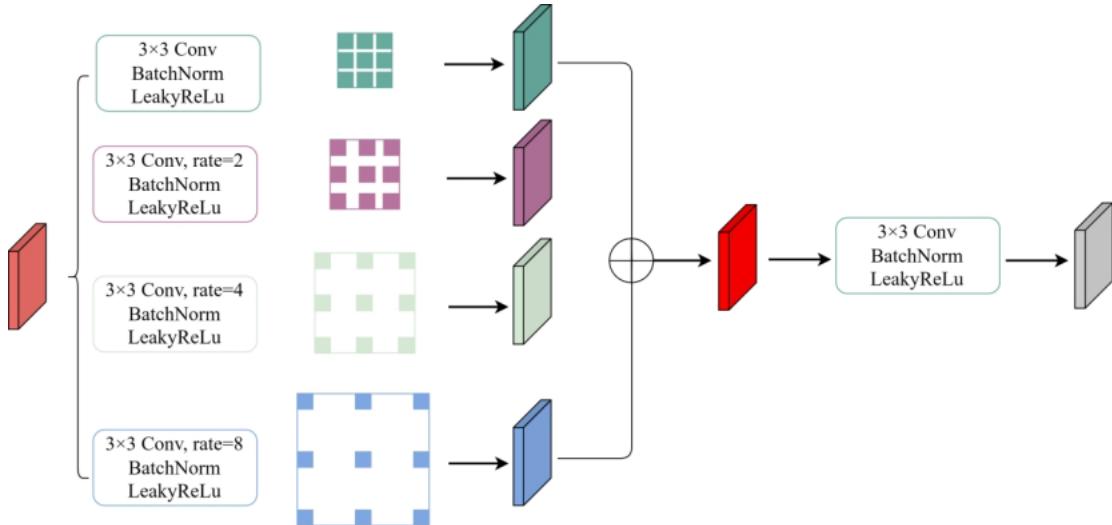


图 6-4 ASPP 模块结构示意图

## 6.6 深度学习降水量定量预测模型 PP-Net

为了精确定量的估计降水量，本文构建了以融合 swin transformer 模块和 ASPP 模块的 U-Net<sup>[16]</sup>为基本深度学习框架的降水量预测模型，即 PP-Net(Precipitation Prediction Net)。

其框架结构如图 6-5 所示

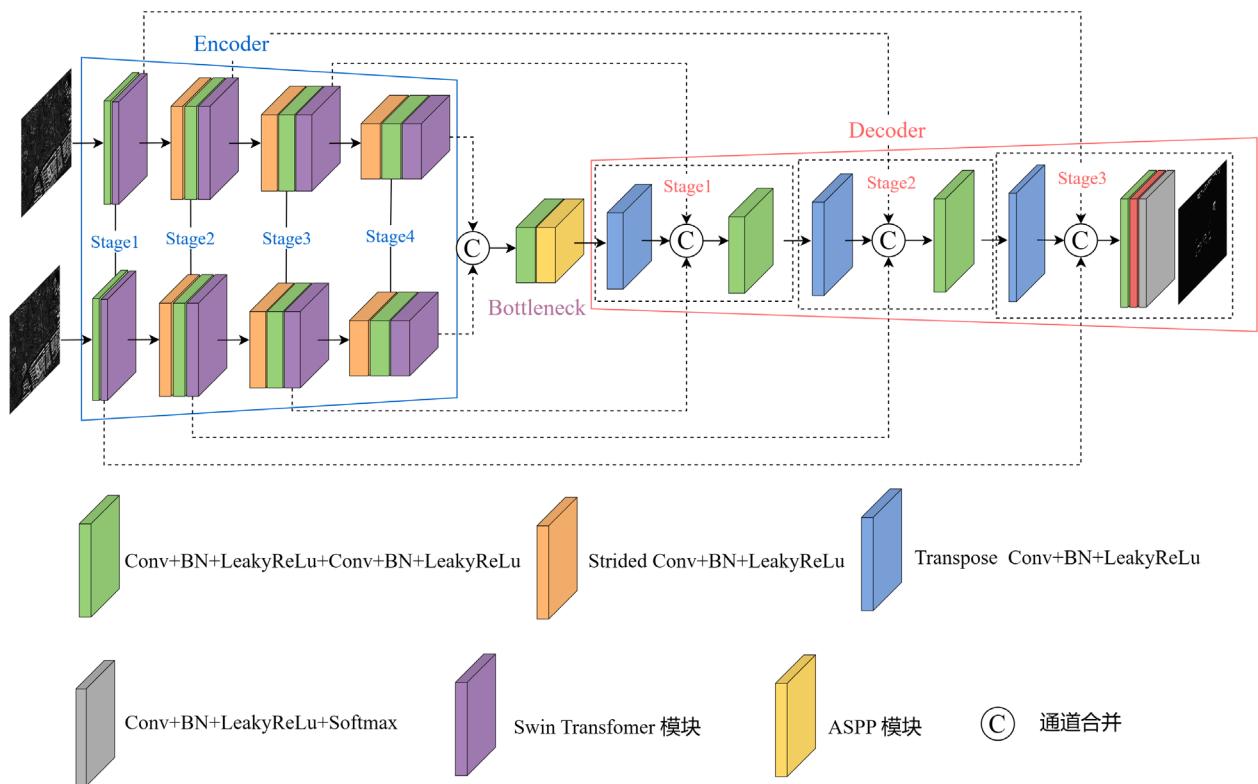


图 6-5 PP-Net 结构框架图

## 6.7 PP-Net 降水量预测结果与分析

选用第 63 个降水事件作为代表性事件。其预测结果如图 6-6 所示

# Truth                  Prediction

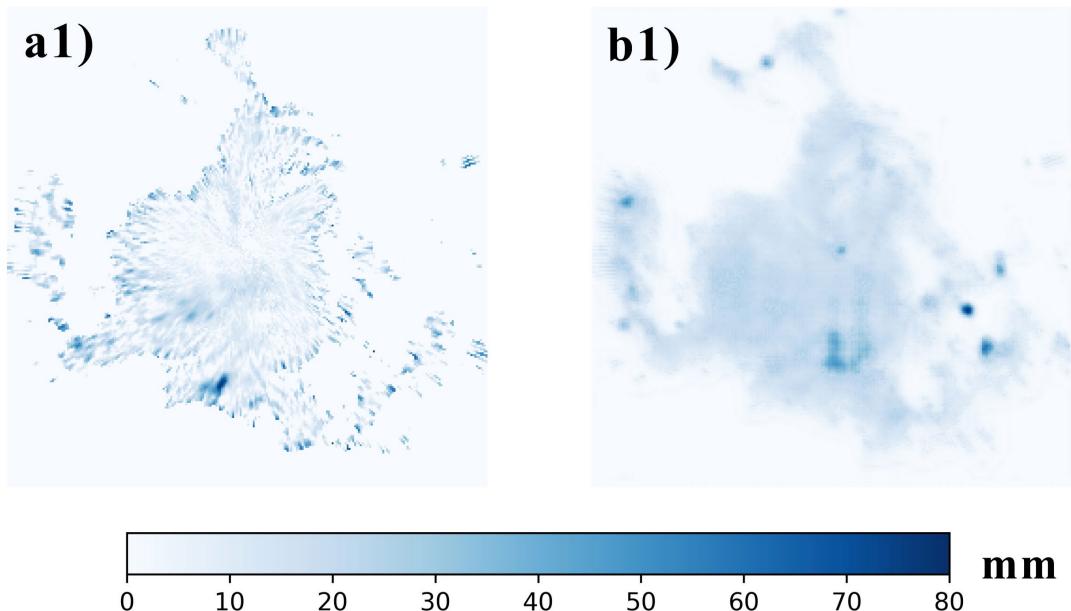


图 6-6 第 63 个降水事件的降水量  $K_{DP\_Rain}$  真值(a1)和预测结果(b1)

从图 6-6，我们可以观察出，PP-Net 预测结果数值上接近，预测范围一致，但内部存在模糊效应，后续需要进一步改善。

## 6.8 模型精度评定

表 6-2 第 63 个降水事件的气象预测评估指数表(↑代表数值越大越好，↓代表数值越小越好)，以 5 mm<sup>[17]</sup>为阈值

降水事件	FAR ↓	MAR ↓	CSI ↑	ETS ↑	POD ↑	RMSE ↓
63	0.387	0.197	0.533	0.427	0.803	8.275

从表 6-2，我们可以看出，PP-Net 模型 CSI 得分超过 0.5，ETS 得分超过 0.4，RMSE 控制在 10 以内，可以得出结论，PP-Net 模型的降水量  $K_{DP\_Rain}$  定量预测精度较好。

## 7 问题四的分析与模型建立

### 7.1 问题四的描述与分析

根据题目的描述，问题四要求对题目提供的双偏振雷达观测数据在强对流降水临近预报中的贡献，其次优化数据融合策略，提高预测精度。

问题四属于数据贡献度评定和模型优化问题，传统的天气雷达数据仅包含水平反射率因子 $Z_h$ ，无法获取粒子的形态、大小、浓度等属性，对于强对流降水的预测，精度不够稳定，无法达到要求。

针对问题二提出的优化模型，进行输入数据的多种组合，对比分析不同组合类型的预测精度；

针对问题三提出的模型，不输入差分反射率 $Z_{DR}$ 数据，仅输入水平反射率因子 $Z_h$ 数据，对比预测结果前后的精度；

问题三中仅输入了1 km等高面的双偏振雷达观测数据，3 km和7 km等高面的双偏振雷达观测数据没有使用，对三种等高面的数据进行加权融合，形成多等高面的多模态样本数据集；

问题四的技术流程如图 7-1 所示：

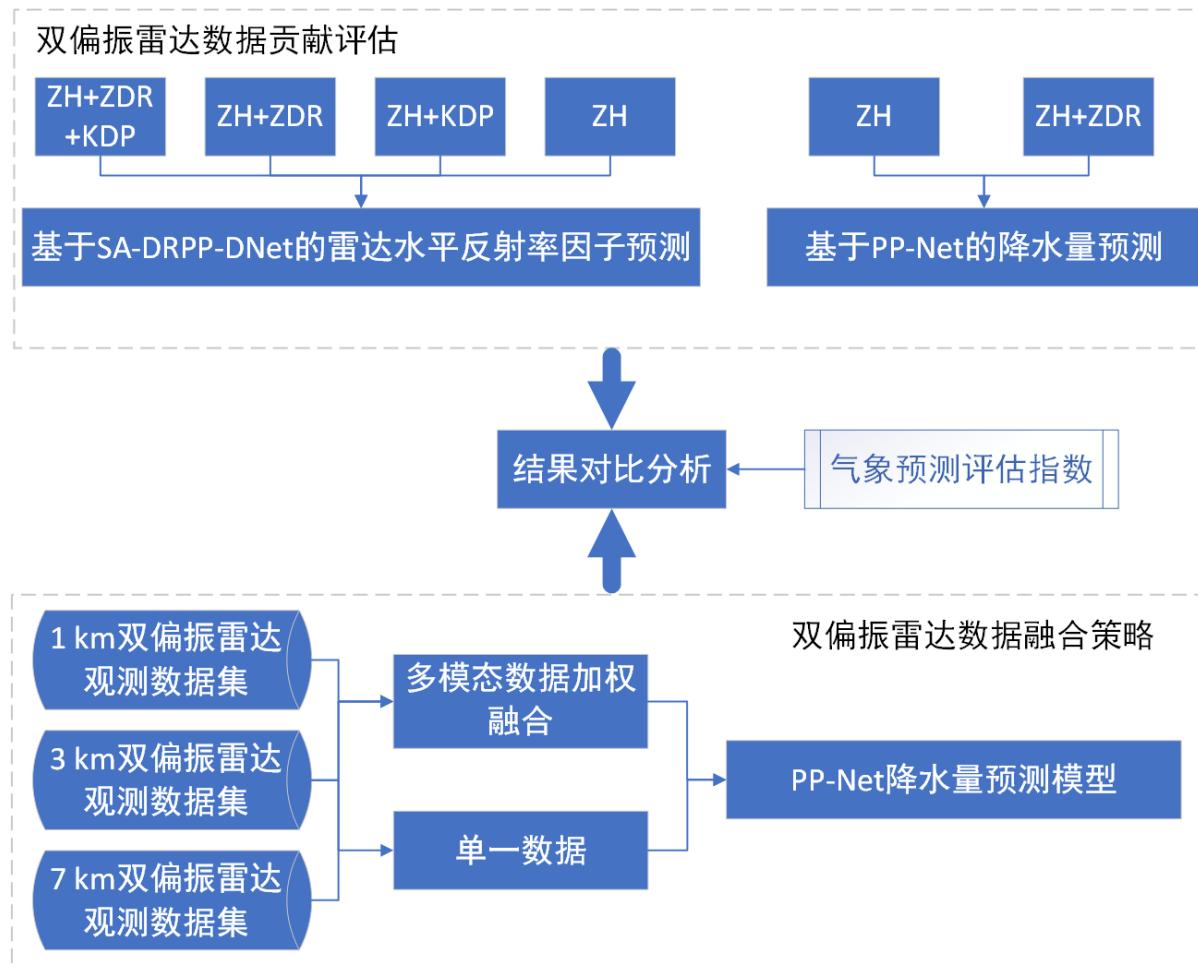


图 7-1 问题四技术流程图

### 7.2 双偏振雷达观测数据贡献度评定实验设计

为了评定双偏振雷达观测数据的贡献度，对于问题二提出的优化模型，设计了四种数据组合方案，对比分析预测结果的精度。组合方案如表 7-1 所示：

表 7-1 双偏振雷达观测数据预测水平反射率因子  $Z_h$  方案

方案	数据组合
方案一	$Z_h$
方案二	$Z_h + Z_{DR}$
方案三	$Z_h + K_{DP}$
方案四	$Z_h + Z_{DR} + K_{DP}$

对于问题三提出的降水量  $K_{DP\_Rain}$  定量估计模型，设计了两种数据组合方案，对比如分析预测结果的精度。组合方案如表 7-2 所示：

表 7-2 双偏振雷达观测数据预测降水量  $K_{DP\_Rain}$  方案

方案	数据组合
方案一	$Z_h$
方案二	$Z_h + Z_{DR}$

### 7.3 双偏振雷达观测数据贡献度评定实验结果与分析

从表 7-3、7-4、7-5、7-6、7-7 和 7-8，可以看出，整体上方案四的预测精度最高，方案一的预测精度最差，因此，双偏振雷达观测数据对于降水预测具有高贡献度。

表 7-3 双偏振雷达观测数据预测水平反射率因子  $Z_h$  CSI 得分表

预测时刻	方案一	方案二	方案三	方案四
6	0.256	0.596	0.442	<b>0.718</b>
18	0.394	0.675	0.564	<b>0.729</b>
30	0.375	0.630	0.532	<b>0.673</b>
42	0.255	<b>0.653</b>	0.522	0.625
54	0.257	<b>0.634</b>	0.475	<b>0634</b>

表 7-4 双偏振雷达观测数据预测水平反射率因子  $Z_h$  ETS 得分表

预测时刻	方案一	方案二	方案三	方案四
6	0.237	0.596	0.417	<b>0.695</b>
18	0.377	0.659	0.546	<b>0.714</b>
30	0.363	0.618	0.519	<b>0.660</b>
42	0.247	<b>0.644</b>	0.511	0.612
54	0.250	<b>0.625</b>	0.465	0.624

表 7-5 双偏振雷达观测数据预测水平反射率因子  $Z_h$  POD 得分表

预测时刻	方案一	方案二	方案三	方案四
6	0.256	0.599	0.443	<b>0.776</b>
18	0.394	0.682	0.567	<b>0.779</b>
30	0.376	0.638	0.536	<b>0.734</b>
42	0.256	0.666	0.530	<b>0.731</b>
54	0.257	0.647	0.483	<b>0.685</b>

表 7-6 双偏振雷达观测数据预测水平反射率因子  $Z_h$  FAR 得分表

预测时刻	方案一	方案二	方案三	方案四
6	<b>0.001</b>	0.008	0.005	0.093
18	<b>0.002</b>	0.014	0.010	0.079
30	<b>0.007</b>	0.020	0.015	0.109
42	<b>0.012</b>	0.028	0.026	0.188
54	<b>0.008</b>	0.032	0.034	0.151

表 7-7 双偏振雷达观测数据预测水平反射率因子 $Z_h$ MAR 得分表

预测时刻	方案一	方案二	方案三	方案四
6	0.744	0.401	0.557	<b>0.224</b>
18	0.606	0.318	0.433	<b>0.221</b>
30	0.624	0.362	0.464	<b>0.266</b>
42	0.744	0.334	0.470	<b>0.269</b>
54	0.743	0.353	0.517	<b>0.282</b>

表 7-8 双偏振雷达观测数据预测水平反射率因子 $Z_h$ RMSE 得分表

预测时刻	方案一	方案二	方案三	方案四
6	3.670	3.168	3.341	<b>3.164</b>
18	3.616	3.125	3.253	<b>3.078</b>
30	3.562	<b>3.193</b>	3.376	3.258
42	3.322	<b>2.863</b>	3.080	3.254
54	2.971	<b>2.765</b>	2.957	2.850

表 7-9 双偏振雷达观测数据预测降水量 $K_{DP\_Rain}$ 气象预测评估指数得分表(↑代表数值越大越好, ↓代表数值越小越好)

方案	CSI↑	ETS↑	POD↑	FAR↓	MAR↓	RMSE↓
方案一	0.350	0.269	0.551	0.511	0.449	6.712
方案二	<b>0.420</b>	<b>0.333</b>	<b>0.717</b>	<b>0.497</b>	<b>0.283</b>	<b>6.277</b>

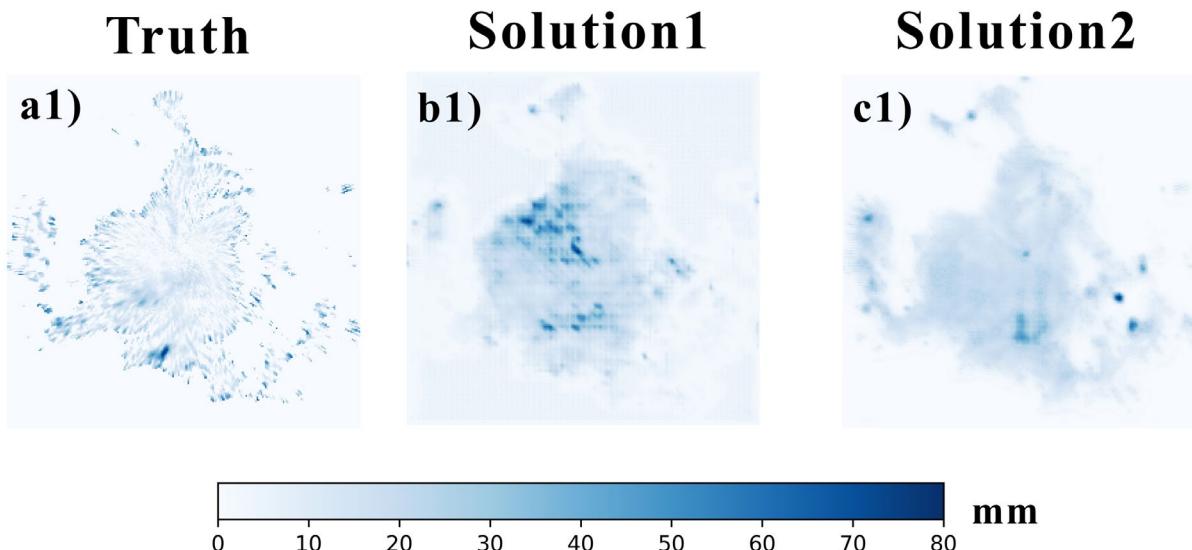


图 7-2 双偏振雷达观测数据预测降水量 $K_{DP\_Rain}$ 中, 第 63 个降水事件的降水量真值(a1)、方案一的预测值(b1)和方案二的预测值(c1)

从表 7-9 和图 7-2, 我们可以看出方案二的预测结果更接近于真值, 预测精度更高, 双偏振雷达观测数据对降水量定量估计有高贡献度。

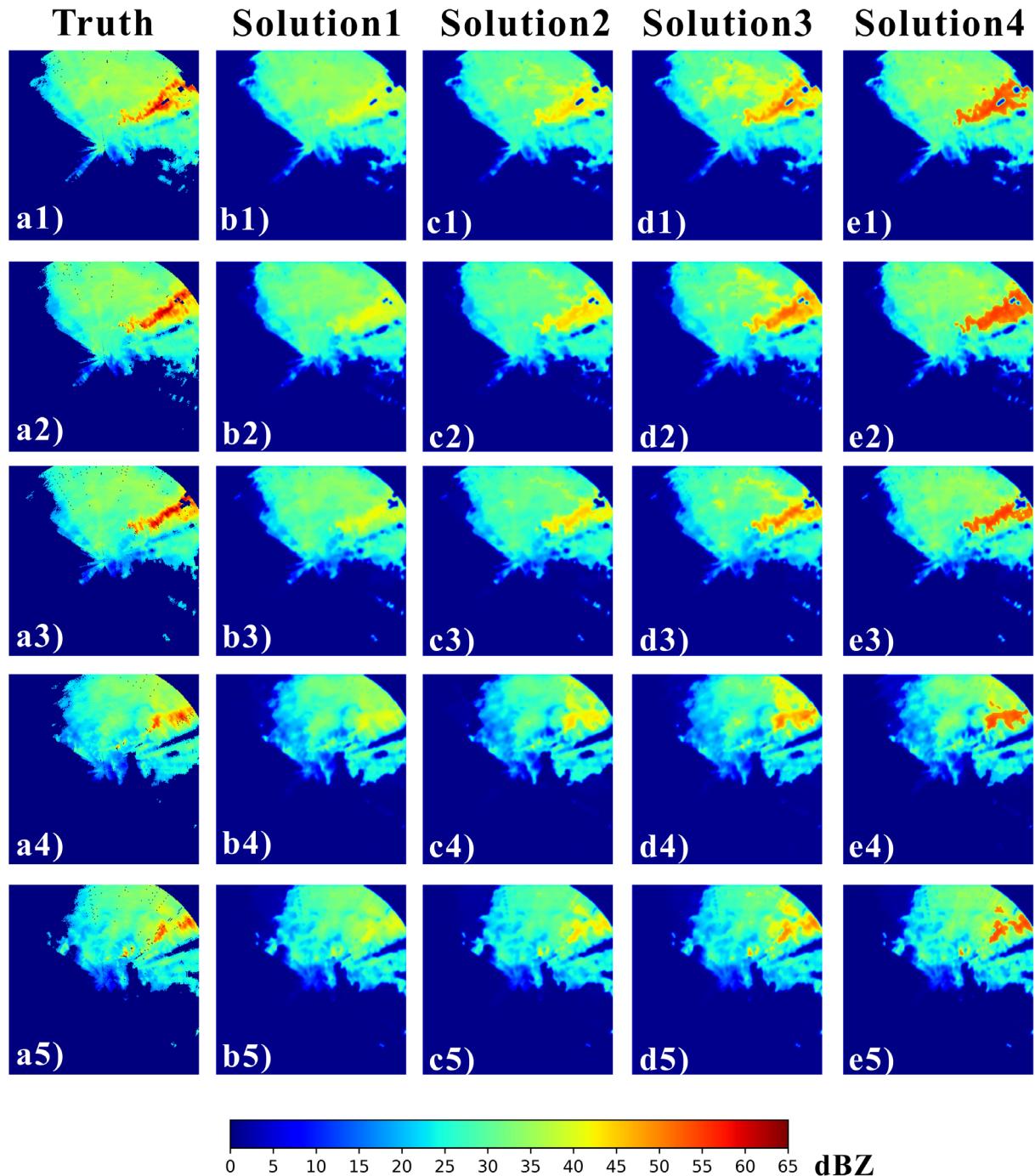


图 7-3 双偏振雷达观测数据预测水平反射率因子 $Z_h$ 中，第 178 个降水事件的一小时真值(a1-a5)、方案一预测结果(b1-b5)、方案二预测结果(c1-c5)、方案三预测结果(d1-d5)和方案四预测结果(e1-e5)，分别对应于 6 分钟、18 分钟、30 分钟、42 分钟和 54 分钟

对于双偏振雷达观测数据预测 $Z_h$ ，其结果如图 7-3 所示。我们可以看出方案四的预测结果更接近于真值，模糊效应也最小。

#### 7.4 多模态降水预测模型的实验设计

为了进一步提升降水量定量估计的准确度，更好地应对突发性和局地性强的强对流天气，由于问题三仅输入了 1 km 等高面的双偏振雷达观测数据，并未输入 3 km 和 7 km 等高面的双偏振雷达观测数据，现将三种等高面的数据进行加权融合，形成多模态的双偏振

雷达观测数据集。

组合方案如表 7-10 所示：

表 7-10 多模态降水量 $K_{DP\_Rain}$ 定量估计模型实验方案

方案	组合
方案一	单一等高面数据+深度学习 PP-Net 模型
方案二	多模态等高面数据+深度学习 PP-Net 模型

### 7.5 多模态降水量 $K_{DP\_Rain}$ 定量估计模型的实验结果与分析

表 7-11 多模态降水量 $K_{DP\_Rain}$ 定量估计模型气象预测评估得分表(↑ 代表数值越大越好,  
↓ 代表数值越小越好)

数据	CSI ↑	ETS ↑	POD ↑	FAR ↓	MAR ↓	RMSE ↓
1 km	0.533	<b>0.427</b>	0.803	0.387	0.197	8.275
3 km	0.305	0.223	0.378	0.386	0.622	9.570
7 km	0.249	0.182	0.288	<b>0.355</b>	0.712	9.247
1 km+3 km+7 km	<b>0.537</b>	0.420	<b>0.916</b>	0.435	<b>0.084</b>	<b>7.886</b>

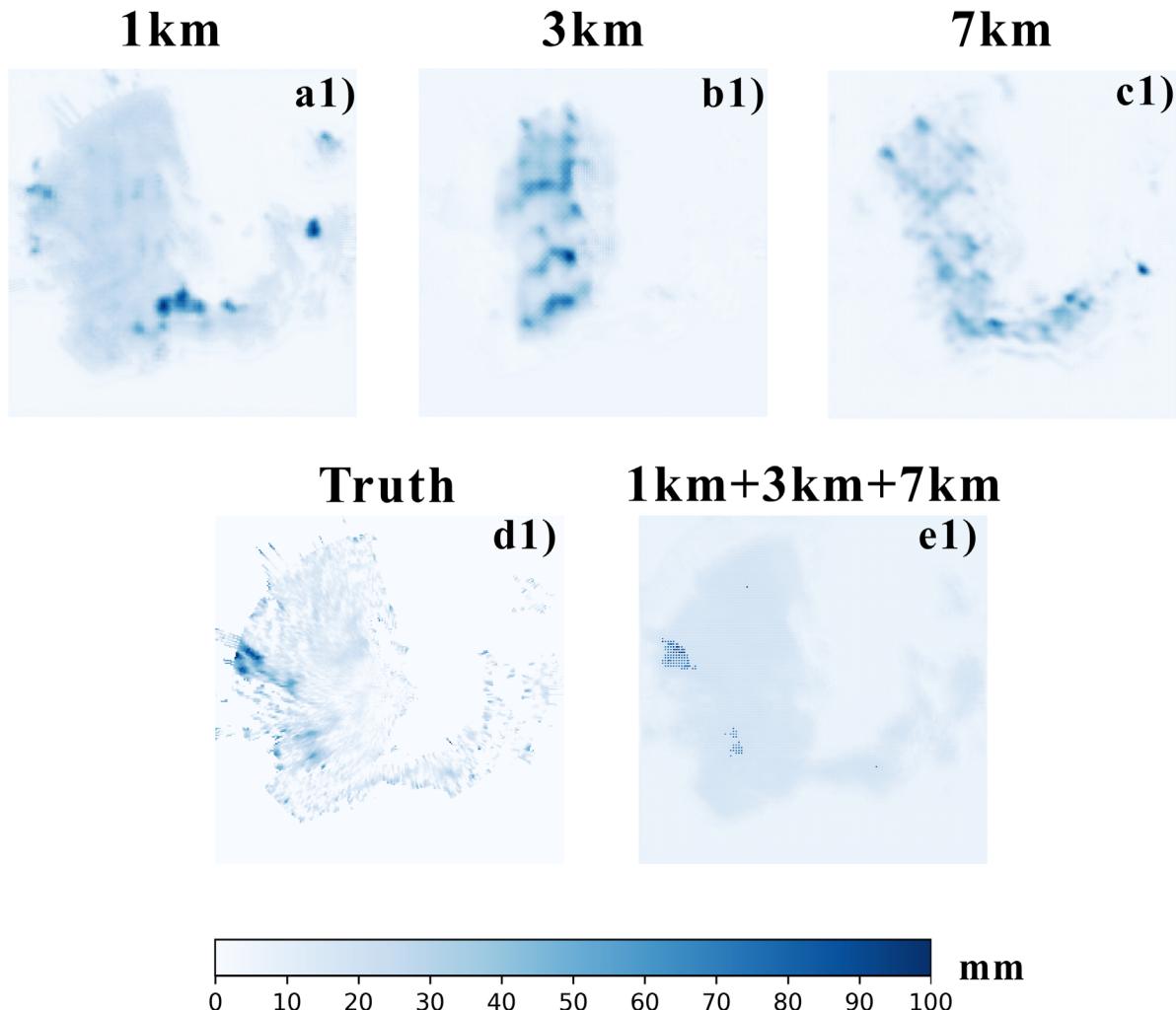


图 7-4 双偏振雷达观测数据预测降水量 $K_{DP\_Rain}$ 中，第 63 个降水事件的降水量真值(d1)、方案一的 1 km 预测值(a1)、3 km 预测值(b1)、7 km 预测值(c1)和方案二的预测值(e1)

从表 7-11 和图 7-4，可以看出，多模态数据的引入，提高了 PP-Net 降水量定量估计模型的预测精度。

## 8 总结与评价

### 8.1 模型的结论与评价

#### 8.1.1 模型的优势

SA-DRPP-DNet 模型对水平反射率因子的未来一小时预测，精度高，不受预测时间影响，能很好地反映未来一小时的强对流降水的发展历程，通过气象预测评估指数，极大地保证了预测结果的合理性和正确性，有效缓解了预测结果的模糊效应。

PP-Net 模型在降水量定量估计上，取得了较好的精度，在数据输入方面，考虑了输入数据的广度，融合了多模态数据，进一步提高了模型精度。

#### 8.1.2 模型的劣势

预测结果的模糊效应虽然被缓解，但依然存在，一定程度上影响了最终的结果，后续将进一步改善模型。

并未考虑融合传统的 Z-R 关系，由于数据限制，并未添加气象相关的物理属性，后续将增加数据广度，增强模型的物理意义。

### 8.2 模型的推广与改进

#### 8.2.1 模型的推广

本文在研究短临强降水情景下，水平反射率因子未来预测与降水量定量估计中，并未使用传统的模型，而是融合深度学习算法，既考虑了数据本身的自我关联性和独立性，取得了较高的精度。因此，气象行业从业人员可以考虑使用本文建立的模型，进行短临强降水的预测。

#### 8.2.2 模型的改进

对于问题二优化后的模型，可以进一步提高硬件性能，对数据集进行多次训练，提高样本的利用率，也可以进一步增加可用的模块，提高对强降水的指标敏感度，提升预测准确性。

对于问题三，由于提供的是多个降水场景的降水格点数据，降水类型多样，可以进一步对降水事件进行分类，提高对各种降水类型的适应度，提高模型泛用性。

可以进一步融合传统的 Z-R 关系算法，提升模型的可解释性。

## 参考文献

- [1]张梦然.气候变化增加极端降水风险[N].科技日报,2023-07-05(004).
- [2]郭换换,王坤.基于双偏振雷达和降水现象仪的郑州“7·20”极端强降水微物理特征分析[J/OL].大气科学学报:1-12[2023-09-25].
- [3]许小峰.中国新一代多普勒天气雷达网的建设与技术应用[J].中国工程科学,2003(06):7-14.
- [4]潘佳文,彭婕,魏鸣等.副热带高压背景下极端短临强降水的双偏振相控阵雷达观测分析[J].气象学报,2022,80(05):748-764.
- [5]陈昊,汪章维,王晗等.基于CINRAD-SA双偏振雷达新型定量降水估测方法研究[J].气象科技,2022,50(05):611-622.
- [6]Kim D K, Suezawa T, Mega T, et al. Improving precipitation nowcasting using a three-dimensional convolutional neural network model from Multi Parameter Phased Array Weather Radar observations[J]. Atmospheric Research, 2021, 262: 105774.
- [7]Xiang Pan, Yinghui Lu, Kun Zhao, Hao Huang, & Mingjun Wang. (2021). Data for GRL - NJU-CPOL dataset [Data set]. Zenodo.
- [8]郭桐,柳东慧.C波段双偏振多普勒天气雷达原理及主要偏振参量应用分析[J].河南科技,2021,40(19):140-142.
- [9]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.
- [10]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.
- [11]Xiong T, He J, Wang H, et al. Contextual Sa-attention convolutional LSTM for precipitation nowcasting: A spatiotemporal sequence forecasting view[J]. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 2021, 14: 12479-12491.
- [12]Ma Z, Zhang H, Liu J. MM-RNN: A Multimodal RNN for Precipitation Nowcasting[J]. IEEE Transactions on Geoscience and Remote Sensing, 2023.
- [13]吴启树,韩美,刘铭等.基于评分最优化的模式降水预报订正算法对比[J].应用气象学报,2017,28(03):306-317.
- [14]张春杨.融合深度与集成学习的降水预报时空序列预测研究[D].南京航空航天大学,2021.
- [15]韩洁,李恩莉,冯富强.多普勒天气雷达在不同降水类型中Z-R关系及检验——以宝鸡市为例[J].绿色科技,2018(08):209-210.
- [16]黄俊豪.基于深度学习的短临降水预测方法研究[D].东南大学,2022.
- [17]何宇翔,张亚萍,刘术艳等.天气雷达定量估测降水量不同方法效果评估[J].南京气象学院学报,2004(06):743-752.

## 附录

代码：

DRPP-Net:

```
import torch.nn as nn
import torch
```

```
class ConvLSTMCell(nn.Module):
```

```
    def __init__(self, input_dim, hidden_dim, kernel_size, bias):
        """
```

```
        Initialize ConvLSTM cell.
```

```
        Parameters
```

```
        -----
```

```
        input_dim: int
```

```
            Number of channels of input tensor.
```

```
        hidden_dim: int
```

```
            Number of channels of hidden state.
```

```
        kernel_size: (int, int)
```

```
            Size of the convolutional kernel.
```

```
        bias: bool
```

```
            Whether or not to add the bias.
```

```
        """
```

```
        super(ConvLSTMCell, self).__init__()
```

```
        self.input_dim = input_dim
```

```
        self.hidden_dim = hidden_dim
```

```
        self.kernel_size = kernel_size
```

```
        self.padding = kernel_size[0] // 2, kernel_size[1] // 2
```

```
        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.padding,
                            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) # concatenate along channel axis
```

```

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

def init_hidden(self, batch_size, image_size):
    height, width = image_size
    return (torch.zeros(batch_size, self.hidden_dim, height, width,
device=self.conv.weight.device),
            torch.zeros(batch_size, self.hidden_dim, height, width,
device=self.conv.weight.device))

```

class ConvLSTM(nn.Module):

"""

Parameters:

- input\_dim: Number of channels in input
- hidden\_dim: Number of hidden channels
- kernel\_size: Size of kernel in convolutions
- num\_layers: Number of LSTM layers stacked on each other
- batch\_first: Whether or not dimension 0 is the batch or not
- bias: Bias or no bias in Convolution
- return\_all\_layers: Return the list of computations for all layers
- Note: Will do same padding.

Input:

A tensor of size B, T, C, H, W or T, B, C, H, W

Output:

- A tuple of two lists of length num\_layers (or length 1 if return\_all\_layers is False).
- 0 - layer\_output\_list is the list of lists of length T of each output
- 1 - last\_state\_list is the list of last states
  - each element of the list is a tuple (h, c) for hidden state and memory

Example:

```

>> x = torch.rand((32, 10, 64, 128, 128))
>> convlstm = ConvLSTM(64, 16, 3, 1, True, True, False)
>> _, last_states = convlstm(x)

```

```

    >> h = last_states[0][0] # 0 for layer index, 0 for h index
    """
def __init__(self, input_dim, hidden_dim, kernel_size, num_layers,
            batch_first=False, bias=True, return_all_layers=False):
    super(ConvLSTM, self).__init__()

    self._check_kernel_size_consistency(kernel_size)

    # Make sure that both `kernel_size` and `hidden_dim` are lists having len ==
    num_layers
    kernel_size = self._extend_for_multilayer(kernel_size, num_layers)
    hidden_dim = self._extend_for_multilayer(hidden_dim, num_layers)
    if not len(kernel_size) == len(hidden_dim) == num_layers:
        raise ValueError('Inconsistent list length.')

    self.input_dim = input_dim
    self.hidden_dim = hidden_dim
    self.kernel_size = kernel_size
    self.num_layers = num_layers
    self.batch_first = batch_first
    self.bias = bias
    self.return_all_layers = return_all_layers

    cell_list = []
    for i in range(0, self.num_layers):
        cur_input_dim = self.input_dim if i == 0 else self.hidden_dim[i - 1]

        cell_list.append(ConvLSTMCell(input_dim=cur_input_dim,
                                      hidden_dim=self.hidden_dim[i],
                                      kernel_size=self.kernel_size[i],
                                      bias=self.bias))

    self.cell_list = nn.ModuleList(cell_list)

    self.linear_conv = nn.Conv2d(in_channels=self.hidden_dim[-1], out_channels=1,
                               kernel_size=3, stride=1, padding=1)

    self.sigmoid = nn.Sigmoid()
def forward(self, input_tensor, hidden_state=None):
    """
Parameters
-----
input_tensor: todo

```

```

    5-D Tensor either of shape (t, b, c, h, w) or (b, t, c, h, w)
hidden_state: todo
    None. todo implement stateful
Returns
-----
last_state_list, layer_output
"""
if not self.batch_first:
    # (t, b, c, h, w) -> (b, t, c, h, w)
    input_tensor = input_tensor.permute(1, 0, 2, 3, 4)

b, _, _, h, w = input_tensor.size()

# Implement stateful ConvLSTM
if hidden_state is not None:
    raise NotImplementedError()
else:
    # Since the init is done in forward. Can send image size here
    hidden_state = self._init_hidden(batch_size=b,
                                    image_size=(h, w))

layer_output_list = []
last_state_list = []

seq_len = input_tensor.size(1)
cur_layer_input = input_tensor

for layer_idx in range(self.num_layers):

    h, c = hidden_state[layer_idx]
    output_inner = []
    for t in range(seq_len):
        h, c = self.cell_list[layer_idx](input_tensor=cur_layer_input[:, t, :, :, :],
                                         cur_state=[h, c])
        output_inner.append(h)

    layer_output = torch.stack(output_inner, dim=1)
    cur_layer_input = layer_output

    layer_output_list.append(layer_output)
    last_state_list.append([h, c])

if not self.return_all_layers:
    layer_output_list = layer_output_list[-1:]

```

```

        last_state_list = last_state_list[-1:]

    return self.sigmoid(self.linear_conv(last_state_list[0][0]))
    # return layer_output_list, last_state_list

def __init_hidden(self, batch_size, image_size):
    init_states = []
    for i in range(self.num_layers):
        init_states.append(self.cell_list[i].init_hidden(batch_size, image_size))
    return init_states

@staticmethod
def _check_kernel_size_consistency(kernel_size):
    if not (isinstance(kernel_size, tuple) or
            (isinstance(kernel_size, list) and all([isinstance(elem, tuple) for elem in
kernel_size]))):
        raise ValueError('`kernel_size` must be tuple or list of tuples')

@staticmethod
def _extend_for_multilayer(param, num_layers):
    if not isinstance(param, list):
        param = [param] * num_layers
    return param

if __name__ == '__main__':
    input = torch.randn(1, 12, 1, 256, 256)
    convlstm = ConvLSTM(1, 16, (3, 3), 2, True, True, False)
    res = convlstm(input)
    print(res.shape)

```

### SA-DRPP-DNet:

```

import torch.nn as nn
import torch

```

```

class ChannelAttention(nn.Module):
    def __init__(self, in_planes, ratio=8):
        super(ChannelAttention, self).__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.max_pool = nn.AdaptiveMaxPool2d(1)

```

```

self.fc1 = nn.Conv2d(in_planes, in_planes // ratio, 1, bias=False)
self.relu1 = nn.ReLU()
self.fc2 = nn.Conv2d(in_planes // ratio, in_planes, 1, bias=False)
self.sigmoid = nn.Sigmoid()

def forward(self, x):
    avg_out = self.fc2(self.relu1(self.fc1(self.avg_pool(x))))
    max_out = self.fc2(self.relu1(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__()

        assert kernel_size in (3, 7), 'kernel size must be 3 or 7'
        padding = 3 if kernel_size == 7 else 1

        self.conv1 = nn.Conv2d(2, 1, kernel_size, padding=padding, bias=False) # 7,3
3,1
        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)
        x = torch.cat([avg_out, max_out], dim=1)
        x = self.conv1(x)
        return self.sigmoid(x)

class CBAM(nn.Module):
    def __init__(self, in_planes, ratio=8, kernel_size=3):
        super(CBAM, self).__init__()
        self.ca = ChannelAttention(in_planes, ratio)
        self.sa = SpatialAttention(kernel_size)

    def forward(self, x):
        out = x * self.ca(x)
        result = out * self.sa(out)
        return result

```

```

class ConvLSTMCell(nn.Module):

    def __init__(self, input_dim, hidden_dim, kernel_size, bias):
        """
        Initialize ConvLSTM cell.

        Parameters
        -----
        input_dim: int
            Number of channels of input tensor.
        hidden_dim: int
            Number of channels of hidden state.
        kernel_size: (int, int)
            Size of the convolutional kernel.
        bias: bool
            Whether or not to add the bias.

        """
        super(ConvLSTMCell, self).__init__()

        self.input_dim = input_dim
        self.hidden_dim = hidden_dim

        self.kernel_size = kernel_size
        self.padding = kernel_size[0] // 2, kernel_size[1] // 2
        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.padding,
                            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)  # concatenate along channel axis

        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

def init_hidden(self, batch_size, image_size):
    height, width = image_size
    return (torch.zeros(batch_size, self.hidden_dim, height, width,
device=self.conv.weight.device),
            torch.zeros(batch_size, self.hidden_dim, height, width,
device=self.conv.weight.device))

```

```
class ConvLSTM(nn.Module):
```

```
"""
```

Parameters:

- input\_dim: Number of channels in input
- hidden\_dim: Number of hidden channels
- kernel\_size: Size of kernel in convolutions
- num\_layers: Number of LSTM layers stacked on each other
- batch\_first: Whether or not dimension 0 is the batch or not
- bias: Bias or no bias in Convolution
- return\_all\_layers: Return the list of computations for all layers

Note: Will do same padding.

Input:

A tensor of size B, T, C, H, W or T, B, C, H, W

Output:

A tuple of two lists of length num\_layers (or length 1 if return\_all\_layers is False).  
 0 - layer\_output\_list is the list of lists of length T of each output  
 1 - last\_state\_list is the list of last states  
 each element of the list is a tuple (h, c) for hidden state and memory

Example:

```

>> x = torch.rand((32, 10, 64, 128, 128))
>> convlstm = ConvLSTM(64, 16, 3, 1, True, True, False)
>> _, last_states = convlstm(x)
>> h = last_states[0][0] # 0 for layer index, 0 for h index
"""

```

```

def __init__(self, input_dim, hidden_dim, kernel_size, num_layers,
            batch_first=False, bias=True, return_all_layers=False):
    super(ConvLSTM, self).__init__()

    self._check_kernel_size_consistency(kernel_size)

```

```

# Make sure that both 'kernel_size' and 'hidden_dim' are lists having len ==
num_layers
    kernel_size = self._extend_for_multilayer(kernel_size, num_layers)
    hidden_dim = self._extend_for_multilayer(hidden_dim, num_layers)
    if not len(kernel_size) == len(hidden_dim) == num_layers:
        raise ValueError('Inconsistent list length.')

    self.input_dim = input_dim
    self.hidden_dim = hidden_dim
    self.kernel_size = kernel_size
    self.num_layers = num_layers
    self.batch_first = batch_first
    self.bias = bias
    self.return_all_layers = return_all_layers

    cell_list = []
    for i in range(0, self.num_layers):
        cur_input_dim = self.input_dim if i == 0 else self.hidden_dim[i - 1]

        cell_list.append(ConvLSTMCell(input_dim=cur_input_dim,
                                      hidden_dim=self.hidden_dim[i],
                                      kernel_size=self.kernel_size[i],
                                      bias=self.bias))

    self.cell_list = nn.ModuleList(cell_list)
    self.CBAM = CBAM(self.hidden_dim[-1])
    self.linear_conv = nn.Conv2d(in_channels=self.hidden_dim[-1], out_channels=1,
                               kernel_size=3, stride=1, padding=1)

    self.sigmoid = nn.Sigmoid()

def forward(self, input_tensor, hidden_state=None):
    """
    Parameters
    -----
    input_tensor: todo
        5-D Tensor either of shape (t, b, c, h, w) or (b, t, c, h, w)
    hidden_state: todo
        None. todo implement stateful
    Returns
    -----
    last_state_list, layer_output
    """
    if not self.batch_first:

```

```

# (t, b, c, h, w) -> (b, t, c, h, w)
input_tensor = input_tensor.permute(1, 0, 2, 3, 4)

b, _, _, h, w = input_tensor.size()

# Implement stateful ConvLSTM
if hidden_state is not None:
    raise NotImplementedError()
else:
    # Since the init is done in forward. Can send image size here
    hidden_state = self._init_hidden(batch_size=b,
                                    image_size=(h, w))

layer_output_list = []
last_state_list = []

seq_len = input_tensor.size(1)
cur_layer_input = input_tensor

for layer_idx in range(self.num_layers):

    h, c = hidden_state[layer_idx]
    output_inner = []
    for t in range(seq_len):
        h, c = self.cell_list[layer_idx](input_tensor=cur_layer_input[:, t, :, :, :],
                                         cur_state=[h, c])
        h = self.CBAM(h)
        c = self.CBAM(c)
        output_inner.append(h)

    layer_output = torch.stack(output_inner, dim=1)
    cur_layer_input = layer_output

    layer_output_list.append(layer_output)
    last_state_list.append([h, c])

if not self.return_all_layers:
    layer_output_list = layer_output_list[-1:]
    last_state_list = last_state_list[-1:]

return self.sigmoid(self.linear_conv(last_state_list[0][0]))
# return layer_output_list, last_state_list

def __init__(self, batch_size, image_size):

```

```

        init_states = []
        for i in range(self.num_layers):
            init_states.append(self.cell_list[i].init_hidden(batch_size, image_size))
        return init_states

    @staticmethod
    def _check_kernel_size_consistency(kernel_size):
        if not (isinstance(kernel_size, tuple) or
                (isinstance(kernel_size, list) and all([isinstance(elem, tuple) for elem in
kernel_size]))):
            raise ValueError('`kernel_size` must be tuple or list of tuples')

    @staticmethod
    def _extend_for_multilayer(param, num_layers):
        if not isinstance(param, list):
            param = [param] * num_layers
        return param

if __name__ == '__main__':
    input = torch.randn(1, 12, 1, 256, 256)
    convlstm = ConvLSTM(1, 16, (3, 3), 2, True, True, False)
    res = convlstm(input)
    print(res.shape)

```

PP-Net:

```

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.utils.checkpoint as checkpoint
import numpy as np
from typing import Optional
from einops import rearrange, repeat
from einops.layers.torch import Rearrange

class PatchPartition(nn.Module):
    def __init__(self, patch_height, patch_width):
        super(PatchPartition, self).__init__()
        self.partition = Rearrange('b c (h p1) (w p2) -> b h w (p1 p2 c)', p1=patch_height,
p2=patch_width)

```

```

def forward(self, img):
    """将图片的维度做变化:b c w h --> b h/4 w/4 16C"""
    x = self.partition(img)
    _, H, W, _ = x.shape
    return x, H, W

class LinearEmbedding(nn.Module):
    def __init__(self, patch_dim, embed_dim=96, norm_layer=None):
        super(LinearEmbedding, self).__init__()
        self.embed_dim = embed_dim
        self.embedding = Rearrange('b h w c -> b (h w) c')
        self.Linear = nn.Linear(patch_dim, embed_dim)

    def forward(self, img):
        x = self.embedding(img)
        x = self.Linear(x)
        return x

class ToPatchEmbed(nn.Module):
    def __init__(self, patch_height, patch_width):
        super(ToPatchEmbed, self).__init__()
        patch_dim = int(patch_height * patch_width * 6)  # 3 输入通道数
        self.patch_partition = PatchPartition(patch_height, patch_width)
        self.linear_embedding = LinearEmbedding(patch_dim, embed_dim=96)

    def forward(self, img):
        x, H, W = self.patch_partition(img)
        x = self.linear_embedding(x)

        return x, H, W

class WindowAttention(nn.Module):
    r""" Window based multi-head self attention (W-MSA) module with relative position bias.
    It supports both of shifted and non-shifted window.
    """

```

Args:

- dim (int): Number of input channels.
- window\_size (tuple[int]): The height and width of the window.
- num\_heads (int): Number of attention heads.
- qkv\_bias (bool, optional): If True, add a learnable bias to query, key, value. Default:

True

```
attn_drop (float, optional): Dropout ratio of attention weight. Default: 0.0
proj_drop (float, optional): Dropout ratio of output. Default: 0.0
"""
def __init__(self, dim, window_size, num_heads, qkv_bias=True, attn_drop=0.,
proj_drop=0.):
    super().__init__()
    self.dim = dim
    self.window_size = window_size # [Mh, Mw]
    self.num_heads = num_heads
    head_dim = dim // num_heads
    self.scale = head_dim ** -0.5

    # define a parameter table of relative position bias
    self.relative_position_bias_table = nn.Parameter(
        torch.zeros((2 * window_size[0] - 1) * (2 * window_size[1] - 1), num_heads)) # [2*Mh-1 * 2*Mw-1, nH]

    # get pair-wise relative position index for each token inside the window
    coords_h = torch.arange(self.window_size[0])
    coords_w = torch.arange(self.window_size[1])
    coords = torch.stack(torch.meshgrid([coords_h, coords_w])) # [2, Mh, Mw]
    coords_flatten = torch.flatten(coords, 1) # [2, Mh*Mw]
    # [2, Mh*Mw, 1] - [2, 1, Mh*Mw]
    relative_coords = coords_flatten[:, :, None] - coords_flatten[:, None, :] # [2, Mh*Mw, Mh*Mw]
    relative_coords = relative_coords.permute(1, 2, 0).contiguous() # [Mh*Mw, Mh*Mw,
2]
    relative_coords[:, :, 0] += self.window_size[0] - 1 # shift to start from 0
    relative_coords[:, :, 1] += self.window_size[1] - 1
    relative_coords[:, :, 0] *= 2 * self.window_size[1] - 1
    relative_position_index = relative_coords.sum(-1) # [Mh*Mw, Mh*Mw]
    self.register_buffer("relative_position_index", relative_position_index)

    self.qkv = nn.Linear(dim, dim * 3, bias=qkv_bias)
    self.attn_drop = nn.Dropout(attn_drop)
    self.proj = nn.Linear(dim, dim)
    self.proj_drop = nn.Dropout(proj_drop)

    nn.init.trunc_normal_(self.relative_position_bias_table, std=.02)
    self.softmax = nn.Softmax(dim=-1)
```

```

def forward(self, x, mask: Optional[torch.Tensor] = None):
    """
    Args:
        x: input features with shape of (num_windows*B, Mh*Mw, C)
        mask: (0/-inf) mask with shape of (num_windows, Wh*Ww, Wh*Ww) or None
    """
    # [batch_size*num_windows, Mh*Mw, total_embed_dim]
    B_, N, C = x.shape
    # qkv(): -> [batch_size*num_windows, Mh*Mw, 3 * total_embed_dim]
    #   reshape: -> [batch_size*num_windows, Mh*Mw, 3, num_heads,
    embed_dim_per_head]
    #   permute: -> [3, batch_size*num_windows, num_heads, Mh*Mw,
    embed_dim_per_head]
    qkv = self.qkv(x).reshape(B_, N, 3, self.num_heads, C // self.num_heads).permute(2, 0,
3, 1, 4)
    # [batch_size*num_windows, num_heads, Mh*Mw, embed_dim_per_head]
    q, k, v = qkv.unbind(0)  # make torchscript happy (cannot use tensor as tuple)

    # transpose: -> [batch_size*num_windows, num_heads, embed_dim_per_head,
    Mh*Mw]
    # @: multiply -> [batch_size*num_windows, num_heads, Mh*Mw, Mh*Mw]
    q = q * self.scale
    attn = (q @ k.transpose(-2, -1))

    # relative_position_bias_table.view: [Mh*Mw*Mh*Mw,nH] -> [Mh*Mw,Mh*Mw,nH]
    relative_position_bias
    self.relative_position_bias_table[self.relative_position_index.view(-1)].view(
        self.window_size[0] * self.window_size[1], self.window_size[0] *
    self.window_size[1], -1)
    relative_position_bias = relative_position_bias.permute(2, 0, 1).contiguous()  # [nH,
    Mh*Mw, Mh*Mw]
    attn = attn + relative_position_bias.unsqueeze(0)

    if mask is not None:
        # mask: [nW, Mh*Mw, Mh*Mw]
        nW = mask.shape[0]  # num_windows
        # attn.view: [batch_size, num_windows, num_heads, Mh*Mw, Mh*Mw]
        # mask.unsqueeze: [1, nW, 1, Mh*Mw, Mh*Mw]
        attn = attn.view(B_ // nW, nW, self.num_heads, N, N) +
mask.unsqueeze(1).unsqueeze(0)
        attn = attn.view(-1, self.num_heads, N, N)
        attn = self.softmax(attn)
    else:

```

```

attn = self.softmax(attn)

attn = self.attn_drop(attn)

    # @: multiply -> [batch_size*num_windows, num_heads, Mh*Mw,
embed_dim_per_head]
    # transpose: -> [batch_size*num_windows, Mh*Mw, num_heads,
embed_dim_per_head]
    # reshape: -> [batch_size*num_windows, Mh*Mw, total_embed_dim]
    x = (attn @ v).transpose(1, 2).reshape(B_, N, C)
    x = self.proj(x)
    x = self.proj_drop(x)
    return x

```

def window\_reverse(windows, window\_size: int, H: int, W: int):

"""

将一个个 window 还原成一个 feature map

Args:

- windows: (num\_windows\*B, window\_size, window\_size, C)
- window\_size (int): Window size(M)
- H (int): Height of image
- W (int): Width of image

Returns:

x: (B, H, W, C)

"""

```

B = int(windows.shape[0] / (H * W / window_size / window_size))
# view: [B*num_windows, Mh, Mw, C] -> [B, H//Mh, W//Mw, Mh, Mw, C]
x = windows.view(B, H // window_size, W // window_size, window_size, window_size, -1)
# permute: [B, H//Mh, W//Mw, Mh, Mw, C] -> [B, H//Mh, Mh, W//Mw, Mw, C]
# view: [B, H//Mh, Mh, W//Mw, Mw, C] -> [B, H, W, C]
x = x.permute(0, 1, 3, 2, 4, 5).contiguous().view(B, H, W, -1)
return x

```

def drop\_path\_f(x, drop\_prob: float = 0., training: bool = False):

"""Drop paths (Stochastic Depth) per sample (when applied in main path of residual blocks).

This is the same as the DropConnect impl I created for EfficientNet, etc networks, however, the original name is misleading as 'Drop Connect' is a different form of dropout in a separate paper...

See discussion: <https://github.com/tensorflow/tpu/issues/494#issuecomment-532968956> ...

I've opted for

changing the layer and argument names to 'drop path' rather than mix DropConnect as a layer name and use  
'survival rate' as the argument.

"""

if drop\_prob == 0. or not training:

    return x

keep\_prob = 1 - drop\_prob

shape = (x.shape[0],) + (1,) \* (x.ndim - 1) # work with diff dim tensors, not just 2D

ConvNets

random\_tensor = keep\_prob + torch.rand(shape, dtype=x.dtype, device=x.device)

random\_tensor.floor\_() # binarize

output = x.div(keep\_prob) \* random\_tensor

return output

class DropPath(nn.Module):

    """Drop paths (Stochastic Depth) per sample (when applied in main path of residual blocks).

"""

def \_\_init\_\_(self, drop\_prob=None):

    super(DropPath, self).\_\_init\_\_()

    self.drop\_prob = drop\_prob

def forward(self, x):

    return drop\_path\_f(x, self.drop\_prob, self.training)

class Mlp(nn.Module):

    """ MLP as used in Vision Transformer, MLP-Mixer and related networks

"""

def \_\_init\_\_(self, in\_features, hidden\_features=None, out\_features=None, act\_layer=nn.GELU, drop=0.):

    super().\_\_init\_\_()

    out\_features = out\_features or in\_features

    hidden\_features = hidden\_features or in\_features

    self.fc1 = nn.Linear(in\_features, hidden\_features)

    self.act = act\_layer()

    self.drop1 = nn.Dropout(drop)

    self.fc2 = nn.Linear(hidden\_features, out\_features)

```

    self.drop2 = nn.Dropout(drop)

def forward(self, x):
    x = self.fc1(x)
    x = self.act(x)
    x = self.drop1(x)
    x = self.fc2(x)
    x = self.drop2(x)
    return x

```

def window\_partition(x, window\_size: int):

"""

将 feature map 按照 window\_size 划分成一个个没有重叠的 window

Args:

x: (B, H, W, C)  
 window\_size (int): window size(M)

Returns:

windows: (num\_windows\*B, window\_size, window\_size, C)

"""

B, H, W, C = x.shape

x = x.view(B, H // window\_size, window\_size, W // window\_size, window\_size, C)

# permute: [B, H//Mh, Mh, W//Mw, Mw, C] -> [B, H//Mh, W//Mh, Mw, Mw, C]

# view: [B, H//Mh, W//Mw, Mh, Mw, C] -> [B\*num\_windows, Mh, Mw, C]

windows = x.permute(0, 1, 3, 2, 4, 5).contiguous().view(-1, window\_size, window\_size, C)

return windows

class SwinTransformerBlock(nn.Module):

r""" Swin Transformer Block.

Args:

dim (int): Number of input channels.

num\_heads (int): Number of attention heads.

window\_size (int): Window size.

shift\_size (int): Shift size for SW-MSA.

mlp\_ratio (float): Ratio of mlp hidden dim to embedding dim.

qkv\_bias (bool, optional): If True, add a learnable bias to query, key, value. Default:

True

drop (float, optional): Dropout rate. Default: 0.0

attn\_drop (float, optional): Attention dropout rate. Default: 0.0

drop\_path (float, optional): Stochastic depth rate. Default: 0.0

act\_layer (nn.Module, optional): Activation layer. Default: nn.GELU

```

    norm_layer (nn.Module, optional): Normalization layer. Default: nn.LayerNorm
"""

def __init__(self, dim, num_heads, window_size=7, shift_size=0,
             mlp_ratio=4., qkv_bias=True, drop=0., attn_drop=0., drop_path=0.,
             act_layer=nn.GELU, norm_layer=nn.LayerNorm):
    super().__init__()
    self.dim = dim
    self.num_heads = num_heads
    self.window_size = window_size
    self.shift_size = shift_size
    self.mlp_ratio = mlp_ratio
    assert 0 <= self.shift_size < self.window_size, "shift_size must in 0-window_size"

    self.norm1 = norm_layer(dim)
    self.attn = WindowAttention(
        dim, window_size=(self.window_size, self.window_size),
        num_heads=num_heads, qkv_bias=qkv_bias,
        attn_drop=attn_drop, proj_drop=drop)

    self.drop_path = DropPath(drop_path) if drop_path > 0. else nn.Identity()
    self.norm2 = norm_layer(dim)
    mlp_hidden_dim = int(dim * mlp_ratio)
    self.mlp = Mlp(in_features=dim, hidden_features=mlp_hidden_dim,
                  act_layer=act_layer, drop=drop)

def forward(self, x, attn_mask):
    H, W = self.H, self.W
    B, L, C = x.shape
    assert L == H * W, "input feature has wrong size"

    shortcut = x
    x = self.norm1(x)
    x = x.view(B, H, W, C)

    # pad feature maps to multiples of window size
    # 把 feature map 给 pad 到 window size 的整数倍
    pad_l = pad_t = 0
    pad_r = (self.window_size - W % self.window_size) % self.window_size
    pad_b = (self.window_size - H % self.window_size) % self.window_size
    x = F.pad(x, (0, 0, pad_l, pad_r, pad_t, pad_b)) # 前两个最后一维扩充 0 行 0 列,
    维度以此往前扩充
    _, Hp, Wp, _ = x.shape

```

```

# cyclic shift
if self.shift_size > 0:
    shifted_x = torch.roll(x, shifts=(-self.shift_size, -self.shift_size), dims=(1, 2))
else:
    shifted_x = x
    attn_mask = None

# partition windows
x_windows = window_partition(shifted_x, self.window_size)  # [nW*B, Mh, Mw, C]
x_windows = x_windows.view(-1, self.window_size * self.window_size, C)  #
[nW*B, Mh*Mw, C]

# W-MSA/SW-MSA
attn_windows = self.attn(x_windows, mask=attn_mask)  # [nW*B, Mh*Mw, C]

# merge windows
attn_windows = attn_windows.view(-1, self.window_size, self.window_size, C)  #
[nW*B, Mh, Mw, C]
shifted_x = window_reverse(attn_windows, self.window_size, Hp, Wp)  # [B, H', W',
C]

# reverse cyclic shift
if self.shift_size > 0:
    x = torch.roll(shifted_x, shifts=(self.shift_size, self.shift_size), dims=(1, 2))
else:
    x = shifted_x

if pad_r > 0 or pad_b > 0:
    # 把前面 pad 的数据移除掉
    x = x[:, :H, :W, :].contiguous()

x = x.view(B, H * W, C)

# FFN
x = shortcut + self.drop_path(x)
x = x + self.drop_path(self.mlp(self.norm2(x)))

return x

```

```
class BasicLayer(nn.Module):
```

```
    """

```

A basic Swin Transformer layer for one stage.

Args:

- dim (int): Number of input channels.
- depth (int): Number of blocks.
- num\_heads (int): Number of attention heads.
- window\_size (int): Local window size.
- mlp\_ratio (float): Ratio of mlp hidden dim to embedding dim.
- qkv\_bias (bool, optional): If True, add a learnable bias to query, key, value. Default: True

True

- drop (float, optional): Dropout rate. Default: 0.0
- attn\_drop (float, optional): Attention dropout rate. Default: 0.0
- drop\_path (float | tuple[float], optional): Stochastic depth rate. Default: 0.0
- norm\_layer (nn.Module, optional): Normalization layer. Default: nn.LayerNorm
- downsample (nn.Module | None, optional): Downsample layer at the end of the layer.

Default: None

- use\_checkpoint (bool): Whether to use checkpointing to save memory. Default: False.

```
def __init__(self, dim, depth, num_heads, window_size,
             mlp_ratio=4., qkv_bias=True, drop=0., attn_drop=0.,
             drop_path=0.,           norm_layer=nn.LayerNorm,       downsample=None,
             use_checkpoint=False):
    super().__init__()
    self.dim = dim
    self.depth = depth
    self.window_size = window_size
    self.use_checkpoint = use_checkpoint
    self.shift_size = window_size // 2

    # build blocks
    self.blocks = nn.ModuleList([
        SwinTransformerBlock(
            dim=dim,
            num_heads=num_heads,
            window_size=window_size,
            shift_size=0 if (i % 2 == 0) else self.shift_size,
            mlp_ratio=mlp_ratio,
            qkv_bias=qkv_bias,
            drop=drop,
            attn_drop=attn_drop,
            drop_path=drop_path[i] if isinstance(drop_path, list) else drop_path,
            norm_layer=norm_layer)
        for i in range(depth)])

```

# patch merging layer

```

if downsample is not None:
    self.downsample = downsample(dim=dim, norm_layer=norm_layer)
else:
    self.downsample = None

def create_mask(self, x, H, W):
    # calculate attention mask for SW-MSA
    # 保证 Hp 和 Wp 是 window_size 的整数倍
    Hp = int(np.ceil(H / self.window_size)) * self.window_size
    Wp = int(np.ceil(W / self.window_size)) * self.window_size
    # 拥有和 feature map 一样的通道排列顺序，方便后续 window_partition
    img_mask = torch.zeros((1, Hp, Wp, 1), device=x.device) # [1, Hp, Wp, 1]
    h_slices = (slice(0, -self.window_size),
                slice(-self.window_size, -self.shift_size),
                slice(-self.shift_size, None))
    w_slices = (slice(0, -self.window_size),
                slice(-self.window_size, -self.shift_size),
                slice(-self.shift_size, None))
    cnt = 0
    for h in h_slices:
        for w in w_slices:
            img_mask[:, h, w, :] = cnt
            cnt += 1

    mask_windows = window_partition(img_mask, self.window_size) # [nW, Mh, Mw,
1]
    mask_windows = mask_windows.view(-1, self.window_size * self.window_size) # [nW, Mh*Mw]
    attn_mask = mask_windows.unsqueeze(1) - mask_windows.unsqueeze(2) # [nW, 1,
Mh*Mw] - [nW, Mh*Mw, 1]
    attn_mask = attn_mask.masked_fill(attn_mask != 0, float(-100.0)).masked_fill(attn_mask == 0, float(0.0))
    return attn_mask

def forward(self, x):
    B, C, H, W = x.shape
    x = x.reshape(B, H*W, C)
    attn_mask = self.create_mask(x, H, W) # [nW, Mh*Mw, Mh*Mw]
    for blk in self.blocks:
        blk.H, blk.W = H, W
        # 这里目前不知道什么作用
        if not torch.jit.is_scripting() and self.use_checkpoint:

```

```

        x = checkpoint.checkpoint(blk, x, attn_mask)
    else:
        x = blk(x, attn_mask)
    if self.downsample is not None:
        x = self.downsample(x, H, W)
        H, W = (H + 1) // 2, (W + 1) // 2

    return x.reshape(B, C, H, W)

class SkipConnection(nn.Module):
    """用于跳跃连接"""

    def __init__(self, in_size, out_size):
        """in_size 指的是 concat 操作過後的維度, out_size 还原成原来的维度"""
        super(SkipConnection, self).__init__()
        self.conv1 = nn.Conv2d(in_size, out_size, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(out_size, out_size, kernel_size=3, padding=1)
        self.relu = nn.ReLU(inplace=True)

    def forward(self, inputs1, inputs2, H, W):
        inputs1 = rearrange(inputs1, 'b (h w) d -> b d h w ', h=H, w=W)
        inputs2 = rearrange(inputs2, 'b (h w) d -> b d h w ', h=H, w=W)
        outputs = torch.cat([inputs1, inputs2], 1)
        outputs = self.conv1(outputs)
        outputs = self.relu(outputs)
        outputs = self.conv2(outputs)
        outputs = self.relu(outputs)
        outputs = rearrange(outputs, 'b d h w -> b (h w) d', h=H, w=W) # 维度还原

        return outputs, H, W

class LinearProjection(nn.Module):
    def __init__(self, in_size, class_num):
        super(LinearProjection, self).__init__()
        self.Linear = nn.Linear(in_size, class_num)
        self.norm = nn.LayerNorm(in_size)

    def forward(self, x, H, W):
        x = self.norm(x)
        x = self.Linear(x)
        x = rearrange(x, 'b (h w) d -> b d h w', h=H, w=W)
        return x

```

```

class ASPP(nn.Module):
    def __init__(self, in_dims, out_dims, rate=None):
        super(ASPP, self).__init__()

        if rate is None:
            rate = [1, 2, 4, 8]
        self.aspp_block1 = nn.Sequential(
            nn.Conv2d(
                in_dims, out_dims, 3, stride=1, padding=rate[0], dilation=rate[0]
            ),
            nn.LeakyReLU(inplace=True),
            nn.Dropout2d(0.2),
            nn.BatchNorm2d(out_dims),
        )
        self.aspp_block2 = nn.Sequential(
            nn.Conv2d(
                in_dims, out_dims, 3, stride=1, padding=rate[1], dilation=rate[1]
            ),
            nn.LeakyReLU(inplace=True),
            nn.Dropout2d(0.2),
            nn.BatchNorm2d(out_dims),
        )
        self.aspp_block3 = nn.Sequential(
            nn.Conv2d(
                in_dims, out_dims, 3, stride=1, padding=rate[2], dilation=rate[2]
            ),
            nn.LeakyReLU(inplace=True),
            nn.Dropout2d(0.2),
            nn.BatchNorm2d(out_dims),
        )
        self.aspp_block4 = nn.Sequential(
            nn.Conv2d(
                in_dims, out_dims, 3, stride=1, padding=rate[3], dilation=rate[3]
            ),
            nn.LeakyReLU(inplace=True),
            nn.Dropout2d(0.2),
            nn.BatchNorm2d(out_dims),
        )

        self.conv_block = nn.Sequential(
            nn.Conv2d(
                out_dims, out_dims, 3, stride=1, padding=1

```

```

),
nn.LeakyReLU(inplace=True),
nn.Dropout2d(0.2),
nn.BatchNorm2d(out_dims),
)
self._init_weights()

def forward(self, x):
    x1 = self.aspp_block1(x)
    x2 = self.aspp_block2(x)
    x3 = self.aspp_block3(x)
    x4 = self.aspp_block4(x)
    out = x1 + x2 + x3 + x4
    final_out = self.conv_block(out)
    return final_out

def _init_weights(self):
    for m in self.modules():
        if isinstance(m, nn.Conv2d):
            nn.init.kaiming_normal_(m.weight)
        elif isinstance(m, nn.BatchNorm2d):
            m.weight.data.fill_(1)
            m.bias.data.zero_()

class ResidualConv(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(ResidualConv, self).__init__()
        self.conv_block2 = nn.Sequential(
            nn.Conv2d(
                input_dim, output_dim, kernel_size=3, padding=1
            ),
            nn.BatchNorm2d(output_dim),
            nn.Dropout2d(0.2),
            nn.LeakyReLU(inplace=True),
            nn.Conv2d(output_dim, output_dim, kernel_size=3, padding=1),
            nn.BatchNorm2d(output_dim),
            nn.Dropout2d(0.2),
            nn.LeakyReLU(inplace=True),
        )
        self.conv_block1 = nn.Sequential(
            nn.Conv2d(
                input_dim, output_dim, kernel_size=3, padding=1
            ),

```

```

        nn.BatchNorm2d(output_dim),
        nn.Dropout2d(0.2),
        nn.LeakyReLU(inplace=True),
    )

def forward(self, x):
    return x + self.conv_block1(x) + self.conv_block2(x)

class Conv_Block(nn.Module):
    def __init__(self,in_channel,out_channel):
        super(Conv_Block, self).__init__()
        self.layer=nn.Sequential(
            nn.Conv2d(in_channel,out_channel,3,1,1,padding_mode='reflect',bias=False),
            nn.BatchNorm2d(out_channel),
            nn.Dropout2d(0.1),
            nn.ReLU(),
            nn.Conv2d(out_channel, out_channel, 3, 1, 1, padding_mode='reflect',
bias=False),
            nn.BatchNorm2d(out_channel),
            nn.Dropout2d(0.1),
            nn.ReLU()
        )
    def forward(self,x):
        return self.layer(x)

class DownSample(nn.Module):
    def __init__(self,channel):
        super(DownSample, self).__init__()
        self.layer=nn.Sequential(
            nn.Conv2d(channel,channel,3,2,1,padding_mode='reflect',bias=False),
            nn.BatchNorm2d(channel),
            nn.ReLU()
        )
    def forward(self,x):
        return self.layer(x)

class UpSample(nn.Module):
    def __init__(self,channel):
        super(UpSample, self).__init__()
        self.layer=nn.Conv2d(channel,channel//2,1,1)
    def forward(self,x,y,feature_map):
        up=F.interpolate(feature_map,scale_factor=2,mode='nearest')

```

```

        out=self.layer(up)
        return torch.cat((out, x, y),dim=1)

class Up(nn.Module):
    def __init__(self,channel):
        super(Up, self).__init__()
        self.layer=nn.Conv2d(channel,channel//2,1,1)
    def forward(self,x, y):
        out=self.layer(torch.cat((x, y), dim=1))
        return out

class Conv_UP2(nn.Module):
    def __init__(self,channel):
        super(Conv_UP2, self).__init__()
        self.layer=nn.ConvTranspose2d(channel, channel//4, 3, 4, 0, 1)
    def forward(self,x):
        out=self.layer(x)
        return out

class Conv_UP1(nn.Module):
    def __init__(self,channel):
        super(Conv_UP1, self).__init__()
        self.layer=nn.ConvTranspose2d(channel, channel//2, 3, 2, 1, 1)
    def forward(self,x):
        out=self.layer(x)
        return out

class Swin_Transfomer(nn.Module):
    def __init__(self,input_dim):
        super(Swin_Transfomer, self).__init__()
        self.swin = nn.Sequential(
            BasicLayer(dim=input_dim,
                      depth=2,
                      num_heads=3,
                      window_size=7,
                      mlp_ratio=4,
                      qkv_bias=True,
                      drop=0,
                      attn_drop=0,
                      drop_path=0,
                      norm_layer=nn.LayerNorm,
                      downsample=None,
                      use_checkpoint=False),

```

```

        nn.BatchNorm2d(input_dim),
        nn.Dropout2d(0.2),
        nn.LeakyReLU(inplace=True),
    )
    def forward(self, x):
        out = self.swin(x)
        return out

class PP_Net(nn.Module):
    def __init__(self, input_channel_x, input_channel_y ):
        super(Pseudo_UNet, self).__init__()

        self.c1_x = Conv_Block(input_channel_x,18)
        self.c1_y = Conv_Block(input_channel_y, 18)
        self.d1=DownSample(18)
        self.c2=Conv_Block(18,36)
        self.swin2 = Swin_Transfomer(36)
        self.d2=DownSample(36)
        self.c3=Conv_Block(36,72)
        self.swin3 = Swin_Transfomer(72)
        self.d3=DownSample(72)
        self.c4=Conv_Block(72,144)
        self.swin4 = Swin_Transfomer(144)
        self.up = Up(288)
        self.aspp = ASPP(144, 144)
        self.u1=UpSample(144)
        self.c6=Conv_Block(216,72)
        self.residual1 = ResidualConv(72, 72)
        self.u2 = UpSample(72)
        self.c7 = Conv_Block(108, 36)
        self.residual2 = ResidualConv(36, 36)
        self.u3 = UpSample(36)
        self.c8 = Conv_Block(54, 18)
        self.residual3 = ResidualConv(18, 18)
        self.output_layer=nn.Sequential(
            nn.Conv2d(18, 1, 3, 1, 1),
            nn.Sigmoid()
        )

    def forward(self,x,y):
        R1_x = self.c1_x(x)
        R2_x = self.swin2(self.c2(self.d1(R1_x)))
        R3_x = self.swin3(self.c3(self.d2(R2_x)))

```

```

R4_x = self.swin4(self.c4(self.d3(R3_x)))

R1_y = self.c1_y(y)
R2_y = self.swin2(self.c2(self.d1(R1_y)))
R3_y = self.swin3(self.c3(self.d2(R2_y)))
R4_y = self.swin4(self.c4(self.d3(R3_y)))

O1 = self.residual1(self.c6(self.u1(R3_x, R3_y, self.aspp(self.up(R4_x, R4_y)))))
O2 = self.residual2(self.c7(self.u2(R2_x, R2_y, O1)))
O3 = self.residual3(self.c8(self.u3(R1_x, R1_y, O2)))
return self.output_layer(O3)

```

```

if __name__ == '__main__':
    arr1 = torch.randn(1,4,256,256)
    arr2 = torch.randn(1, 4, 256, 256)
    net = Pseudo_UNet(4, 4)
    res = net(arr1, arr2)
    print(res.shape)

```