


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		国防科技大学
参赛队号		**23910020015**
队员姓名	**1.**	廖增华
	2.	彭鐀
	3.	王腾云

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：　强对流降水临近预报

摘　要：

利用双偏振雷达的粒子微物理参数进行强对流降水短临预报，可以更好地帮助我们预测强对流短临降水，对于改进强对流降水短临预报有重要的意义。本文基于题目所给的数据，利用深度学习技术，优化了强对流降水短临预报方法，解决了当前模型存在的几种问题，并评估了双偏振雷达资料在强对流降水临近预报中的贡献。

针对问题一，给定一小时内的雷达观测量，预测后续一小时的水平反射率因子预报。首先对数据进行预处理，剔除无效数据并将数据标准化方便后续处理。为了提取雷达观测量的特征，采用基于 **U-Net 的卷积神经网络**模型架构，并引入了**残差连接**机制，将一小时内的三个雷达观测量数据一起输入神经网络，利用卷积神经网络提取雷达观测量的区域特征，并在不同观测量和不同帧之间进行融合。之后定义损失函数为均方误差，使用 Adam 优化算法对模型参数优化。为了定量评估模型效果，定义了 **MSE、MRE、RBIAS** 和 **CSI** 四个评价指标，结果见表 5.1，最终模型在这四个指标上取得了较好的得分。为了直观地评估模型效果，将模型预测的结果与实际观测数据进行可视化对比，见图 5.10 和 5.11，模型预测结果与实际较为相似。

针对问题二，缓解预报的模糊效应，使雷达回波细节更充分更真实。本文分析认为问题一中模型存在的“回归到平均”问题主要是因为模型对于细节特征的提取和预测能力不足，设计了三个改进措施，分别是在模型的第一层前和最后一层后添加额外能够提取和生成更细粒度特征的**残差卷积块**；引入**注意力机制**，优化特征提取过程，使模型能够更关注数据中的细节特征；利用**集成学习 Bagging 思想**，同时训练三个模型，通过对数据进行不同的处理使不同模型重点学习数据的不同方面特征，并将最终结果聚合以提升整体效果。之后与问题一相同对模型效果进行评估，在定量的评价指标上改进的模型比问题一中有较大的提升，且输出数据的分布平滑的问题有明显改善。在可视化对比图 6.13 中，模型的预测结果更加接近实际结果，细节更加充分真实。

针对问题三，利用提供的Z_H、Z_{DR}和降水量数据，进行定量降水估计。首先，我们通过统计图分析了Z-R关系，发现Z_H是影响降水量预测的主要因素，并且不同高度对降水量预测的影响趋势基本一致，结果见图 7.1、7.2。然后，为了解决当前模型存在的“模糊效应”问题，我们改进了**扩散模型**来适应降水量预测任务。接着，为了提高模型的预测精度和泛化能力，我们提出了**组合扩散策略**，通过集成多个预测的输出，来增强模型的预测能力。随后，我们进行了对比实验和消融案例分析。最后我们进行了预测影响因素探究。对比实验

结果表明，扩散模型能够很好地适应强对流降水短临预报任务，超过了传统的 U-Net、WF-UNet 模型，结果见表 7.1。消融案例结果表明，我们设计的组合扩散策略，能够有效地提高模型精度以及泛化能力，能够很好地解决“模糊效应”，结果见图 7.7。预测影响因素探究结果表明，传统的 U-Net 模型对中大雨的预测效果不佳，而组合扩散模型对小、中、大雨都有较好的预测结果，结果见表 7.2、图 7.8。

针对问题四，评估双偏振雷达资料在强对流降水临近预报中的贡献，优化数据融合策略，更好地应对突发性和局地性强的强对流天气。首先，我们对比了传统雷达数据和双偏振雷达训练的模型，结果表明，各个模型通过双偏振雷达数据训练的结果都好于利用传统雷达数据进行训练，结果见表 8.1、图 8.1、图 8.2。然后，我们通过适应**黑箱模型**的**沙普利值**来计算了不同高度下Z_H、Z_{DR}的特征贡献度，发现Z_H模型预测的主要影响因素，但是Z_{DR}也对模型预测具有不可或缺的贡献，结果见图 8.3。接着我们设计了**模糊检验**对比实验，结果表明利用传统雷达数据训练的模型更容易导致“模糊效应”，而利用双偏振雷达数据训练的模型能够缓解这个问题，结果见表 8.2。最后，我们设计了**数据级、特征级、决策级数据融合策略**，通过不同粒度的数据融合使模型更好地理解数据学习数据，消融实验结果表明，我们的数据融合策略能够较好地提高模型的精度，更加适应突发性和局地性强的强对流天气，结果见表 8.3。

关键词： U-Net 模型；残差卷积；注意力机制；集成学习；组合扩散模型；特征贡献度；模糊检验；数据融合

目录

1 问题背景与问题重述

1.1 问题背景

强降水导致的暴雨洪涝等自然灾害对农业、水资源、经济、民生等方面造成重大影响，获得准确的降水信息对于保障民生、规划水资源的利用、稳定农作物产量至关重要。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。而降水粒子的信息仅通过反射率因子获取并不能得到很好的效果。双偏振雷达同时拥有两个通道，包含水平和垂直方向，可以发射和接收相互正交的极化电磁波，从而获得两通道的强度差和相位差，可以更多地反映粒子的微物理信息。

深度学习旨在研究如何从数据中自动地提取多层特征表示，其核心思想是通过数据驱动的方式，采用一系列的非线性变换从原始数据中提取由低层到高层、由具体到抽象、由一般到特定语义的特征。深度学习依赖于大数据，而气象数据正好符合其特点，因此很适合有大量雷达观测数据积累的短临预报领域。

为了更好地应用双偏振雷达改进强对流降水短临预报，本题要求对双偏振雷达用于强对流降水短临预报进行研究分析，根据相关数据资料建立预测模型，解决目前强对流降水短临预报存在的一些问题，并探究双偏振雷达资料在强对流降水临近预报中的贡献。

1.2 问题重述

结合以上研究背景，利用NJU-CPOL双偏振雷达数据和降水格点数据，解决以下问题：

问题一：

利用题目所给的数据，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。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：

在问题一的基础上，设计数学模型以缓解预报的“回归到平均”问题（模糊效应），使预报出的雷达回波细节更充分、更真实。

问题三：

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。

问题四：

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。

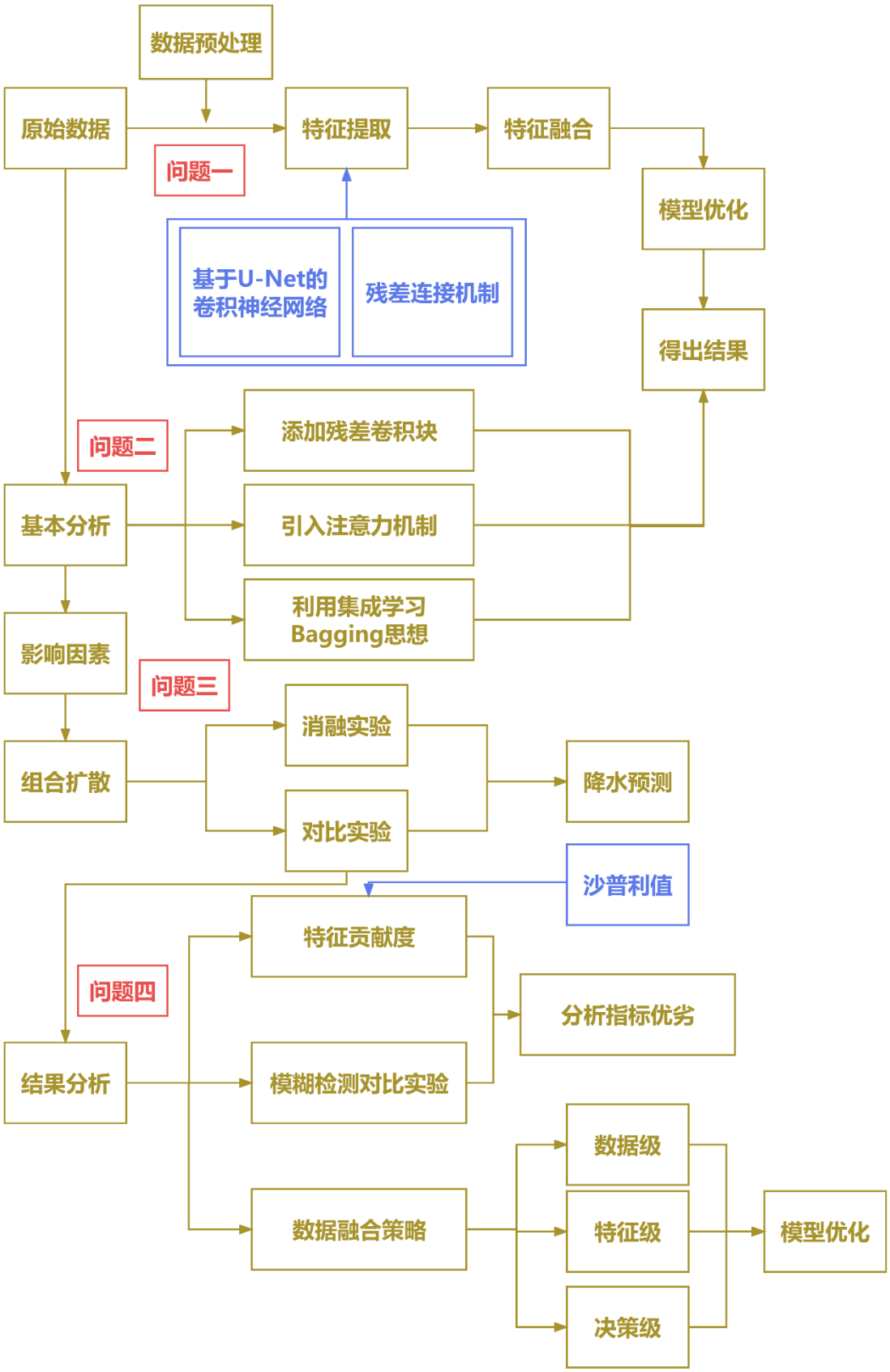
2 模型假设

1．允许部分数据存在着较大偏差，是符合实际情况的。
2．每一帧降水量矩阵变化符合马尔可夫过程或者非马尔可夫过程。
3．所有降雨都是自然情况下的降雨，不包括人工降雨等特殊情况。
4．降雨量与雷达反射因子存在相关性。

3 符号说明

符号	含义
Z_H	水平反射率因子，即水平方向的回波强度
Z_{DR}	差分反射率，即水平和垂直方向回波强度的差异
K_{DP}	比差分相移，即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
R	降雨量
W	模型参数权重
$\sigma(\cdot)$	激活函数
$*$	卷积操作
α	Adam 优化算法的初始步长
θ	模型的全部可学习参数
x	模型输入
y_{true}	真实标签
y_{pred}	模型预测的输出
t	扩散模型的时间步
x_t	扩散模型第时间步t的输入
α_t	扩散模型噪声率

4 论文框架图


数据预处理
原始数据
特征提取
特征融合
问题一
模型优化
基于U-Net的卷积神经网络
残差连接机制
得出结果
问题二
添加残差卷积块
基本分析
引入注意力机制
利用集成学习Bagging思想
影响因素
问题三
消融实验
组合扩散
降水预测
对比实验
沙普利值
特征贡献度
问题四
分析指标优劣
结果分析
模糊检测对比实验
数据级
数据融合策略
特征级
模型优化
决策级

5 问题一

5.1 问题分析

问题一的任务为根据给定的一小时内（10 帧）的三个雷达观测量，预测后续一小时（10 帧）里其中一个雷达观测量，即水平反射率因子的值。显然提取给定一小时内的观测数据的特征十分重要。本章考虑以下三个方面的特征提取：1）由于每一帧的每个雷达观测量数据均为二维矩阵形式，且通过观察发现其中的数据具有区域连续性，因此考虑使用卷积神经网络（CNN）来提取每个雷达观测量每一帧的特征。2）问题一需要根据三个雷达观测量预测其中一个观测量，因此需要考虑如何将三个雷达观测量的特征进行融合的问题。为此，本章设计了在分别对三个雷达观测量特征进行提取的基础上将三个雷达观测量融合的 U-Net 模型。3）问题一输入为一小时（10 帧）的雷达观测量，因此需要对帧与帧之间的关系进行建模。而卷积神经网络能够处理多通道的图像数据，本章将输入数据的不同帧当作卷积神经网络输入的不同通道，以此实现各帧数据的融合处理。

5.2 数据分析

（1）偏振变量特征分析

为了分析这强对流降水过程雷达偏振参量特点，我们对不同高度下的Z_{DR}-Z_H和K_{DP}-Z_H做了变化频率图，横坐标都为Z_H，第一列纵坐标为Z_{DR}，第二列纵坐标为K_{DP}，第一、二、三行分别为 1km、3km、7km 的数据。同时计算了平均值与标准差。

如图 5.1 所示，我们发现从均值来看Z_{DR}和K_{DP}与Z_H呈微线性关系，Z_{DR}和K_{DP}随Z_H的增加而缓慢增加，但是分布较为散乱，异常值也较多，因此这对Z_H的预测提出了更大的挑战。

（2）垂直结构演变特征分析

我们结合双偏振雷达各参量的垂直结构衍化结果，进一步分析降水过程中对流微物理结构特征。如图 5.2 所示，其中横坐标是时间帧，纵轴分为三块（1km、3km、7km），色块分别表示了当前帧下Z_H、Z_{DR}、K_{DP}矩阵的均值。我们的目的是探究双偏振雷达三个变量在垂直结构上的衍化是否有一致性的趋势，同一时间帧内不同垂直高度是否对变量值有影响。

由图 5.2 可知，三个高度下色块颜色的衍化趋势随时间的变化是一致的，双偏振雷达三个变量在垂直结构上的衍化具有一致性的趋势。对于同一时间帧内不同垂直高度对变量值的影响，我们发现同一时间内低层粒子占主导地位，随着高度的增加，三个变量的值有不同程度地增大，表明低层粒子数目较多，降水效率高。

5.3 数据预处理

在进行模型的构建与训练之前，需要对数据进行预处理，去除其中的无效值，提高数据质量，加快模型训练速度。

首先对数据进行观察，发现其中存在大量全为 0 的帧数据，而这类数据对于特征提取和预测并无帮助。因此，对于包含有全 0 雷达观测数据的样本，将该帧以及同一降水过程中同一等高面的相同时间的其他帧从数据集中剔除。之后，对数据进行标准化处理，采用 min-max 标准化方法将数据映射到[0,1]区间范围内：

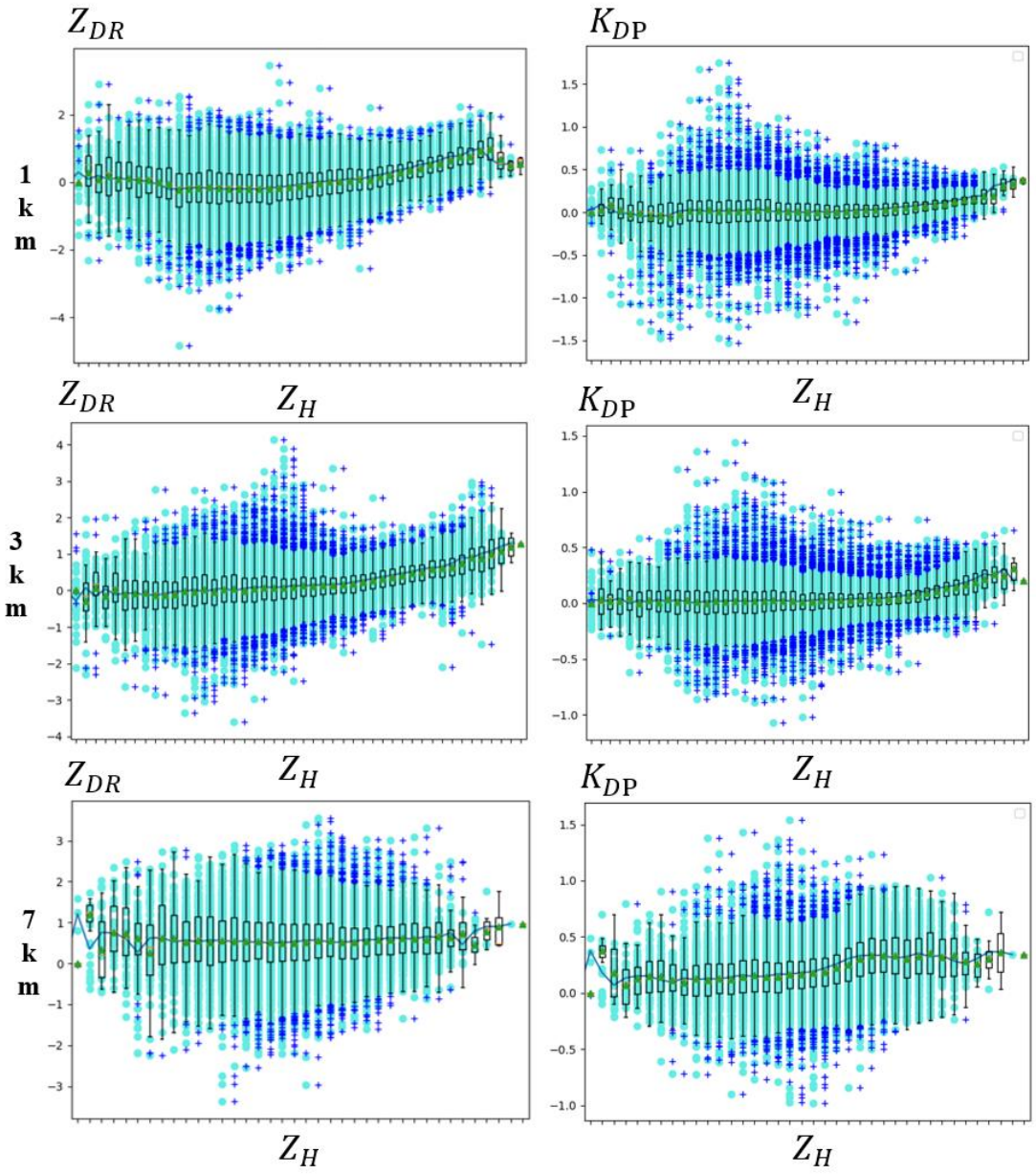


图 5.1 偏振变量特征分析

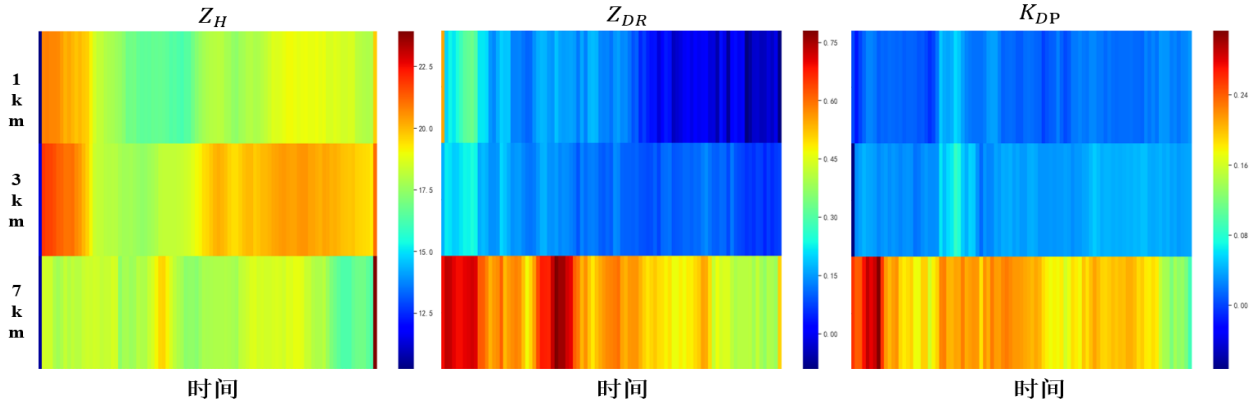


图 5.2 垂直结构演变特征

$$x = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{5.1}$$

按帧进行标准化处理，x_{max}，x_{min}分别为某帧数据中的最大值和最小值。

为充分利用数据以及契合问题任务要求，在模型训练过程中将每连续十帧的三个雷达观测数据作为一个训练样本，后续十帧的水平反射率因子作为训练的标签。

5.4 U-Net 模型

U-Net[1]是比较早的使用全卷积网络进行语义分割的算法之一，其主要由一系列的卷积层和池化层构成。其结构可以分为 Encoder 和 Decoder 两部分，Encoder 部分使用一般的卷积层将输入进行下采样，将“宽而浅”的输入数据转换为“窄而深”的潜在特征，而后通过 Decoder 部分的反卷积层进行上采样，将特征重新转换为类似样本的形式。U-Net 还使用了跳跃连接（skip-connection）的方式将 Encoder 部分的特征直接复制到 Decoder 的过程中，减少由于多次采样带来的原始样本信息的丢失。

本章使用的 U-Net 模型结构如图 5.4 所示，总体过程为，将三个雷达观测量的十帧数据分别进行卷积下采样，由 10 维逐渐增加到 512 维，最后得到三个高维特征图，然后将三个特征合并进行上采样，过程与下采样相反，特征逐渐由高维降至低维，并在上采样过程中使用跳跃连接与下采样的中间结果合并，最终生成十帧的预测结果。受 ResNet 启发，其中使用了带残差连接的卷积块，如图 5.5 所示，具体地，其包含三个卷积层，在上采样时第一个卷积为反卷积层，而后两个卷积层不改变输入的大小。在第一个卷积层之后和最后的输出之间进行了残差连接，将第一个卷积层的输出和最后的输出进行相加。

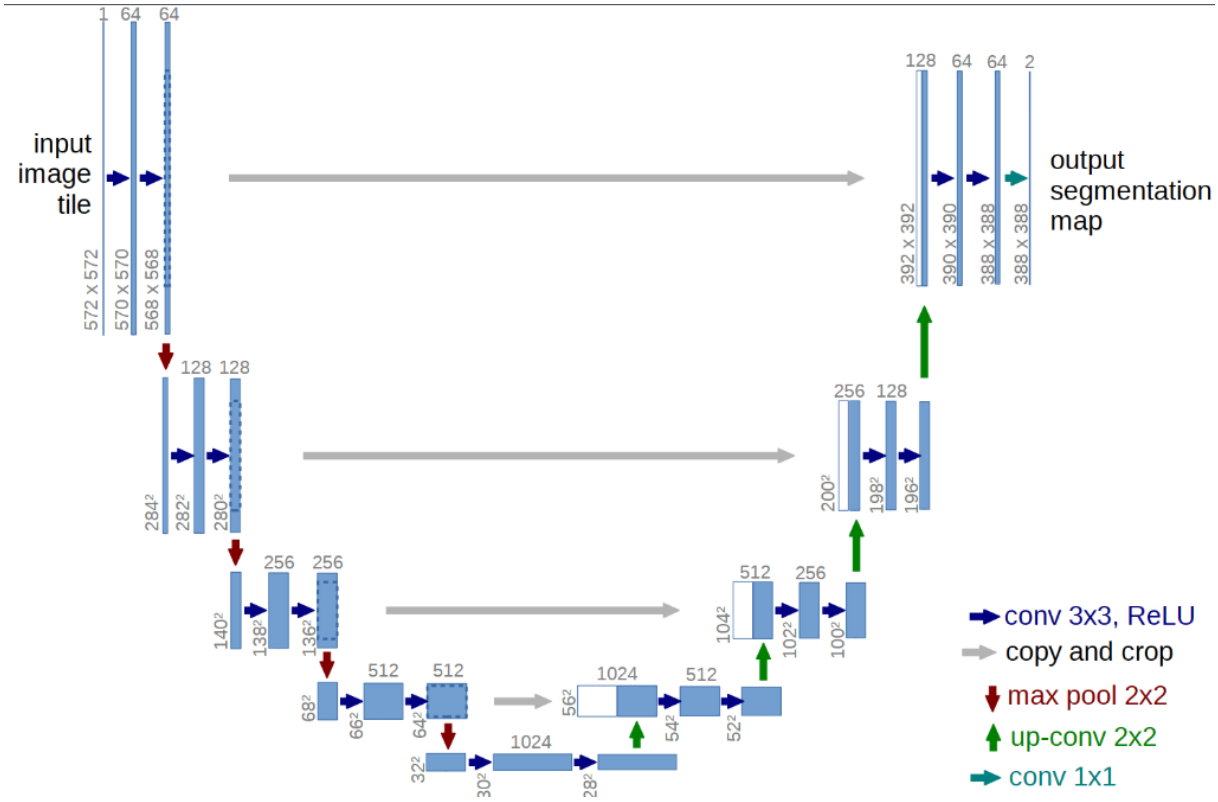


图 5.3 U-Net 模型结构图

本章使用的 U-Net 模型结构如图 5.4 所示，总体过程为，将三个雷达观测量的十帧数

据分别进行卷积下采样，由 10 维逐渐增加到 512 维，最后得到三个高维特征图，然后将三个特征合并进行上采样，过程与下采样相反，特征逐渐由高维降至低维，并在上采样过程中使用跳跃连接与下采样的中间结果合并，最终生成十帧的预测结果。受 ResNet[2]启发，其中使用了带残差连接的卷积块，如图 5.5 所示，具体地，其包含三个卷积层，在上采样时第一个卷积为反卷积层，而后两个卷积层不改变输入的大小。在第一个卷积层之后和最后的输出之间进行了残差连接，将第一个卷积层的输出和最后的输出进行相加。残差连接的引入，使得网络不会因为层数太多难以训练，消除了网络退化现象。残差卷积块的计算过程可表示为如下：

$$x_m = \sigma(W_1 * x_{in} + b_1) \tag{5.2}$$

$$x_{out} = W_3 * \sigma(W_2 * x_m + b_2) + b_3 + x_m \tag{5.3}$$

其中W，b分别为卷积核权重和偏置，是网络中的可学习参数，$*$表示卷积操作，$\sigma()$是 ReLU 激活函数，其表达式为$\sigma(x) = \max(0, x)$。

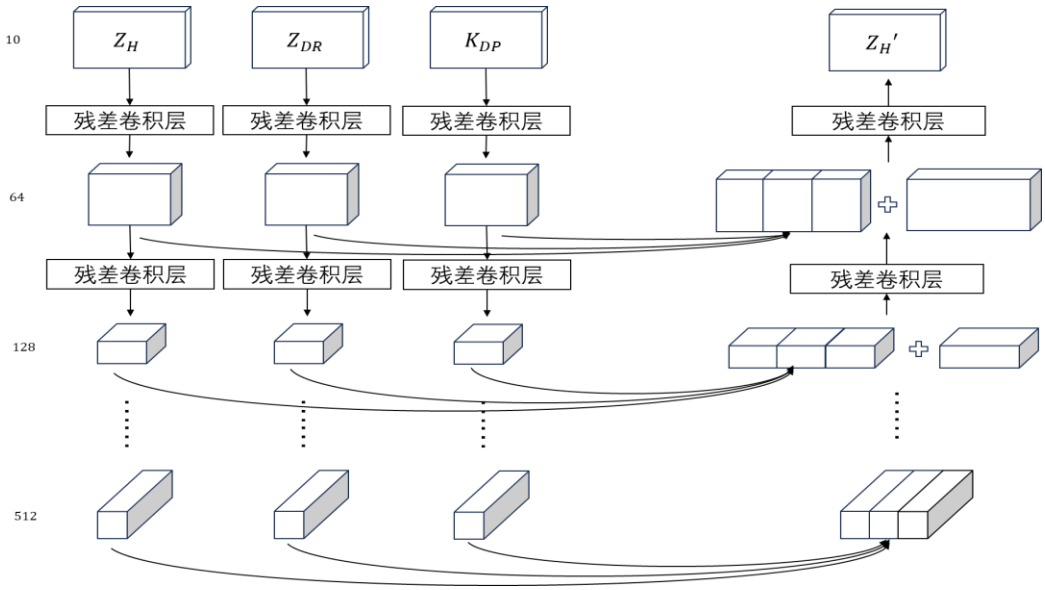


图 5.4 本章使用的 U-Ne

模型中残差卷积块的输出通道数，从下采样输入开始，依次为 64、128、192、256、384、512。这样的结构能够使得初始输入的十帧数据先被分散为多个维度，然后再融合恢复十帧大小，以此达到隐式地利用十帧序列数据的效果。在上采样时，将三个雷达观测量提取的特征进行拼接，然后通过反卷积融合成十帧，这样就同时达到了融合不同雷达观测量的效果。

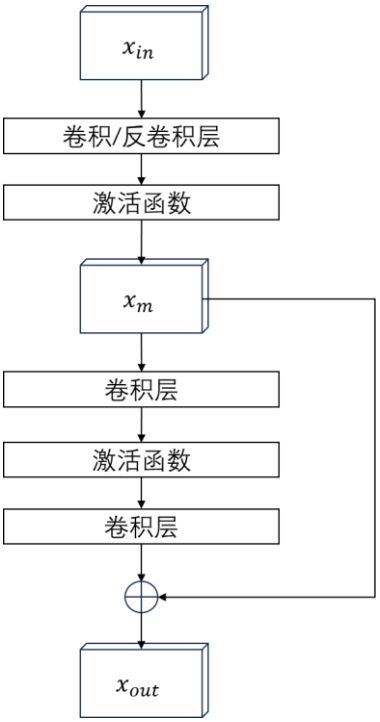


图 5.5 残差卷积块结构图

5.5 模型训练

在训练模型之前，使用 Xavier[3]初始化方法对模型的参数进行初始化，保持每一层输出的方差与输入的方差一致，避免出现训练时出现梯度爆炸或梯度消失的问题。具体地，该初始化方法将参数初始化为从一个正态分布$N(0,\frac{2}{n_{out}+n_{in}})$中随机采样得到的数，其中$n_{in}$，$n_{out}$分别为该网络层输入和输出的维度。

模型的训练过程中，使用均方误差作为损失函数，其公式为：

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(y_{pred_i} - y_{true_i}\right)^2 \tag{5.4}$$

其中y_{truth}为真实数据，即后续一小时中实际的水平反射率因子，而y_{pred}则为模型预测的数据。在训练过程中模型将不断优化参数权重使得该损失尽可能小，也即模型预测结果与实际结果尽可能接近。训练时使用 Adam[4]优化方法，该方法能够在优化过程中自动调整步长，且超参数不需要或仅需较少的微调。

算法：Adam 优化算法

输入：步长α，梯度矩估计的指数衰减率$\beta_1,\beta_2 \in [0,1)$，初始参数$\theta_0$，损失函数$f(\theta)$

$m_0 \leftarrow 0$（初始化梯度一阶矩估计）

$v_0 \leftarrow 0$（初始化梯度二阶矩估计）

$t \leftarrow 0$（初始化时间步）

While θ_t未达到收敛 **do**

$t \leftarrow t+1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$（计算损失函数对参数的梯度）
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t$（更新有偏一阶矩估计）
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2$（更新有偏二阶矩估计）
$\widehat{m}_t \leftarrow m_t/(1-\beta_1^t)$（计算纠正偏差的一阶矩估计）
$\hat{v}_t \leftarrow v_t/(1-\beta_2^t)$（计算纠正偏差的二阶矩估计）
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t/(\sqrt{\hat{v}_t} + \epsilon)$（更新参数，$\epsilon$为一充分小的正常数）

End while
输出： θ_t

模型训练的其他细节为，训练初始步长$\alpha = 10^{-4}$，训练迭代轮数为 100 轮，不同等高面的雷达观测数据分别进行训练。

5.6 结果与分析

除了作为损失函数的均方误差 MSE，本节还使用以下三个指标评估训练的结果：

$$MRE = \frac{\frac{1}{n}\sum_{i=1}^{n}\left(\left|y_{true_i} - y_{pred_i}\right|\right)}{\sum_{i=1}^{n} y_{true_i}} \tag{5.5}$$

$$RBIAS = \frac{\sum_{i}^{n} |y_{true_i} - y_{pred_i}|}{\sum_{i=1}^{n} y_{true_i}} \tag{5.6}$$

$$CSI = \frac{TP}{TP+FN+FP} \tag{5.7}$$

分别为平均相对误差、相对偏差和临界成功指数。其中临界成功指数的计算方式为，设定一个固定阈值，将预测概率大于该阈值的数据点记为阳性，否则记为阴性。TP 表示实际和预测结果均为阳性的数据点数量，FN 表示实际结果为阳性而预测结果为阴性的数据点数量，FP 表示实际结果为阴性而预测结果为阳性的数据点数量。这四个指标中，MSE 反映预测结果与实际结果的绝对差距，MRE 和 RBIAS 则反映预测结果与实际结果的相对差距，而 CSI 反映预测结果的准确程度。本节选取 0.35 作为 CSI 的阈值，随机选择一个小时的雷达观测量数据，预测后续一个小时的Z_H，并与实际的Z_H对比计算以上四个指标，所得到的结果指标如下表所示。

表 5.1 预测结果比较

MSE	MRE	RBIAS	CSI
9.0280	3.9792e-7	0.2608	0.6603

其中 MSE、MRE、RBIAS 数值越小则拟合效果越好，而 CSI 数值越接近 1 则准确程度越高。可以看到模型在前三个指标上能够取得较小的数值，而 CSI 达到了 0.6，说明模型

初步达到了可观的预测效果。此外，还对这一个小时的十帧数据分别进行了指标的计算，并画出如下指标变化图。可以看到 MSE、MRE、RBIAS 都随着时间升高，而 CSI 则呈下降趋势，这说明随着预测时间的变长模型预测的效果也在下降，越靠后的预测越不准确，这也符合常识和经验规律。

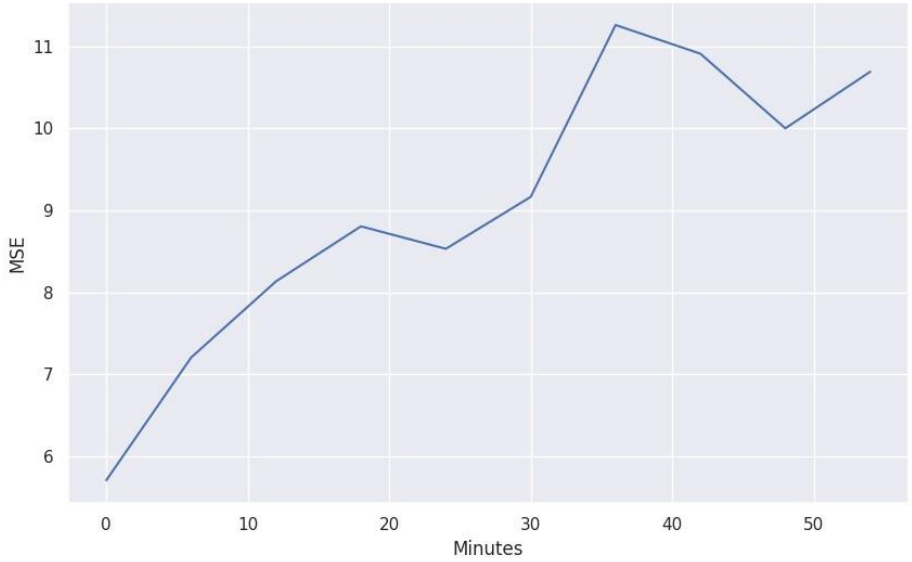


图 5.6 MSE 随时间变化

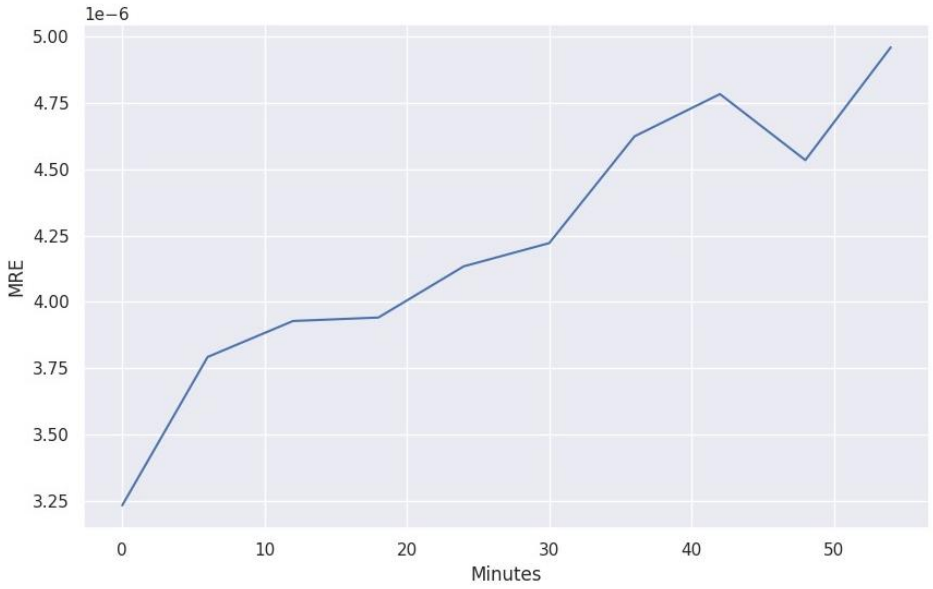


图 5.7 MRE 随时间变化

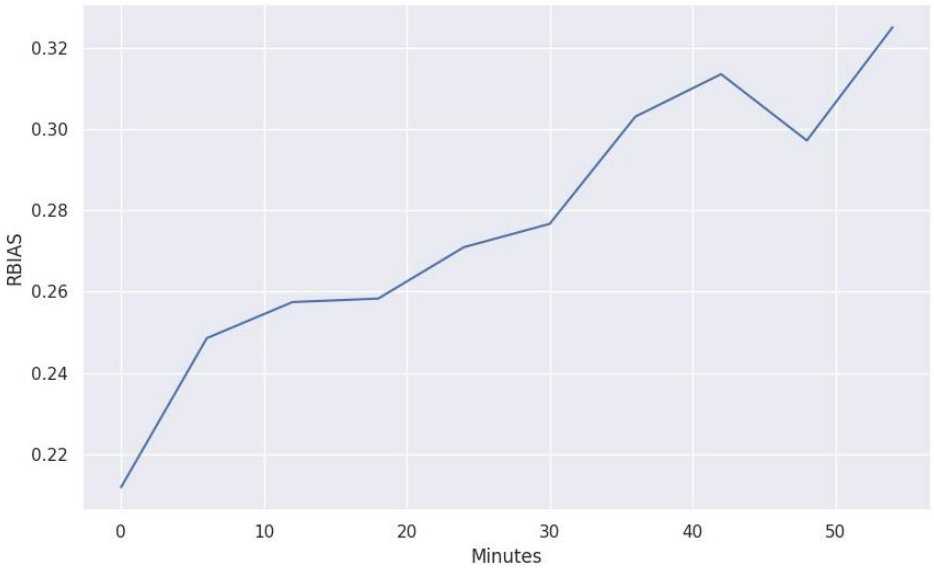


图 5.8 RBIAS 随时间变化

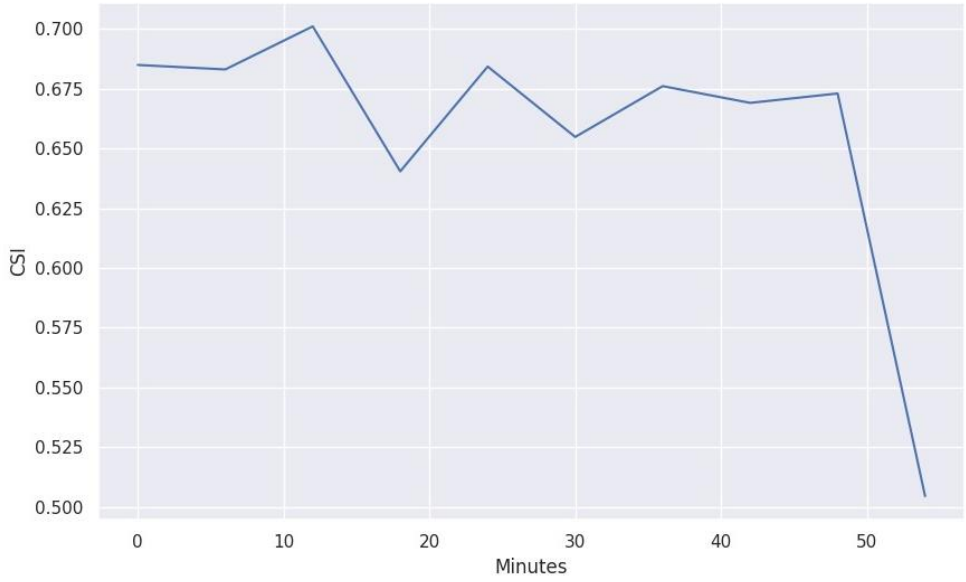


图 5.9 CSI 随时间变化

为了更直观体现模型预测效果，我们随机抽选了 1.0km 等高面的一个小时的雷达观测数据并进行预测，然后将预测的后一小时的数据与实际数据可视化进行对比，如图所示为一小时中的其中五帧，第一行为预测结果，第二行为对应的实际数据，颜色亮暗代表格点数值的大小。可以看到模型预测的结果与实际数据已经非常相近，仅在细节处有所不同。

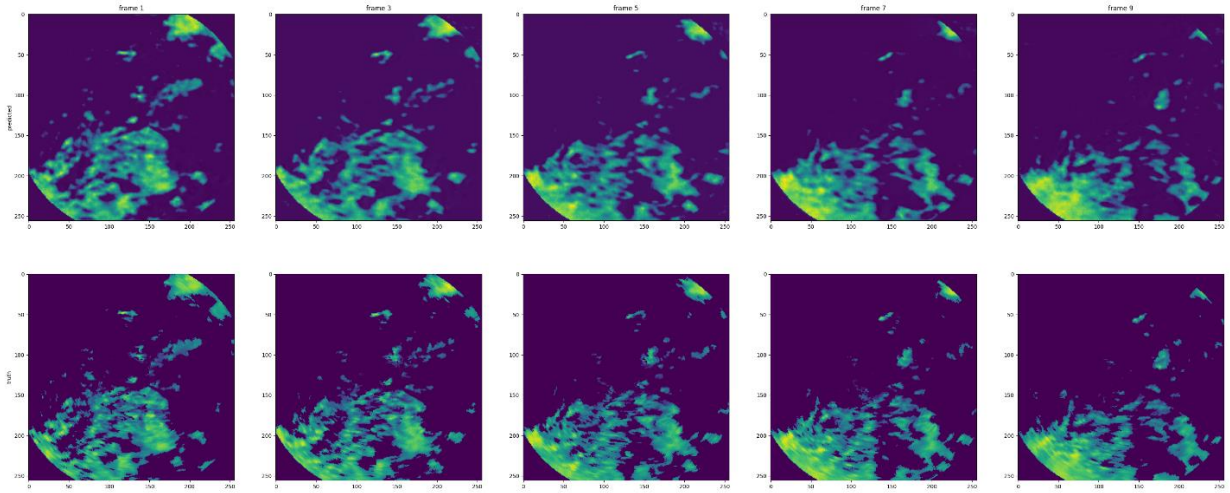


图 5.10 不同帧预测与实际结果可视化对比图

下图则为不同等高面预测的结果与实际结果对比，随机选择了其中一帧进行可视化。可以看到在不同等高面上的预测虽有一些细节丢失，但总体图像的轮廓和分布仍已十分接近。

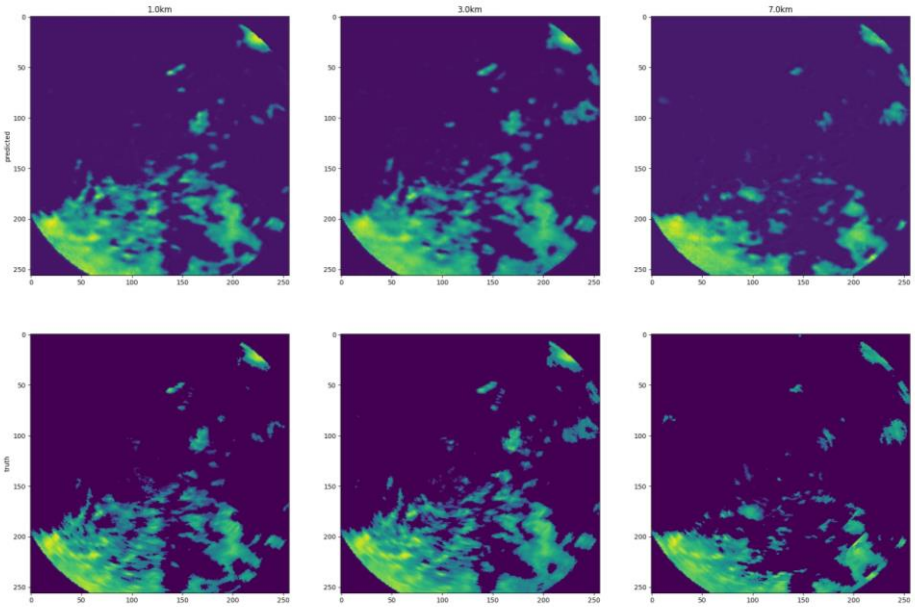


图 5.11 不同等高面预测与实际结果可视化对比图

6 问题二

6.1 问题分析

在问题一的结果对比图 5.10 中可以看到，相比实际结果，预测的结果可视化后边缘更模糊，并且在一些变化较为细小的位置拟合得不够精确。而在图 5.11 中，还能观察到预测的数据可视化后与实际数据存在色差，主要体现在预测数据的背景色比实际数据要略亮一点。这也就是说明，模型的输出存在较为平滑、总体更趋近于平均的问题。如图 6.1 所示，对预测结果的各格点数值与实际结果的直方图进行比较，为了便于观察这里去除了占大多数的作为背景的低值数据。从两者的直方图分布可以看出，预测结果较实际结果更为平均，总体方差更小。

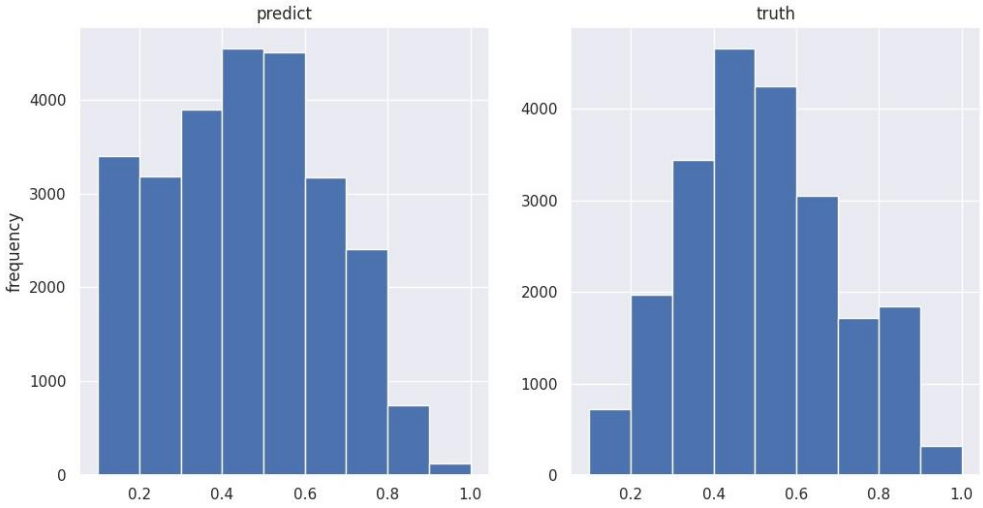


图 6.1 问题一预测结果与实际结果格点数值直方图

我们认为造成这一问题的主要原因是模型对于细节的把控能力仍存在不足。为增强模型生成细节，提高精度，本章在问题一的模型基础上做出三个改进：1）添加更多网络层数，增强模型拟合能力；2）引入注意力机制，使模型能够对细节有更好的控制；3）采用集成学习的方法，用多个网络分别提取输入的不同特征，聚合增强最终效果。

6.2 改进方法

6.2.1 增加网络结构

深度学习中，神经网络的深度即网络层数决定了模型能力的上限。更深的网络往往能够拥有更好的拟合能力、更强的学习能力和更优秀的性能。针对第一问存在的“回归到平均”问题，我们通过增加网络的层数来增强模型对细节的识别和建模能力。具体地，我们在第一问使用模型上添加两个残差卷积层，分别添加在输入第一层之前和输出最后一层之后，如图所示：

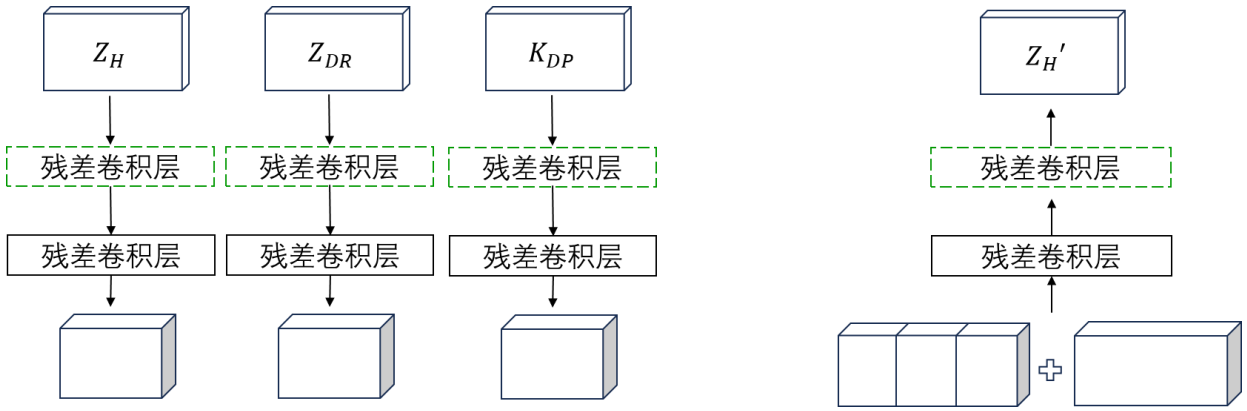


图 6.2 网络结构修改

与原有的残差卷积层不同，新添加的残差卷积层使用更小尺寸的卷积核与更小的步长。由此，新的模型能够在更小尺度上对输入数据进行特征提取，并且也能够生成更精细的数据分布。得益于残差卷积层中的残差连接，在增加了网络深度后该模型也仍能较好地收敛。

6.2.2 引入注意力机制

注意力机制是一种模仿人类视觉和认知系统的方法，它允许神经网络在处理输入数据时集中注意力于相关的部分。通过引入注意力机制，神经网络能够自动地学习并选择性地关注输入中的重要信息，提高模型的性能和泛化能力。具体来说，注意力机制需要输入三个变量k,q和v，用k和q计算出v的权重，输出加权后的v，从而实现对v中重要部分的注意力。在神经网络中，这三个量通常是可学习的，能够随着网络的训练进行优化，进而自动地调整权重。

具体地，我们在第一问模型的上采样之前，对合并的三个雷达观测量特征进行自注意力加权，计算过程如图所示。

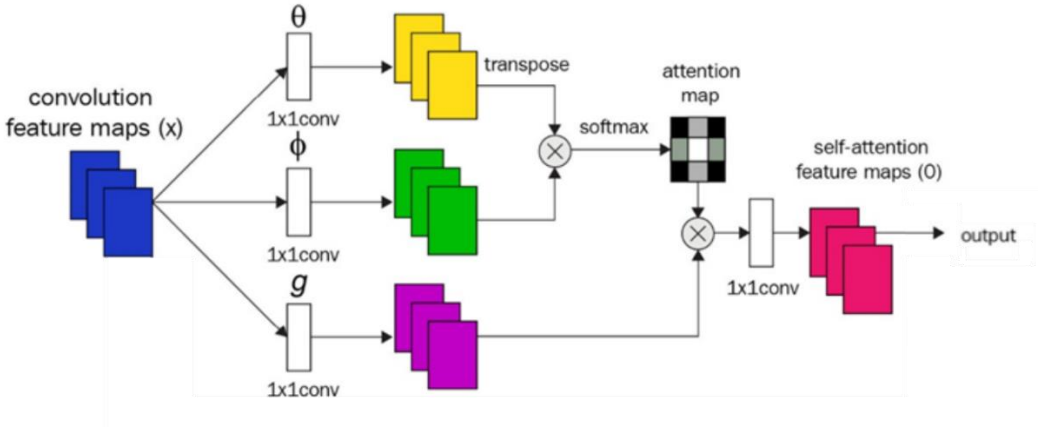


图 6.3 模型中的自注意力过程

首先用三个大小为1×1的卷积核分别生成三个分量，其相当于对输入特征分别乘以三个权重，得到用于注意力的K,Q和V。接着将计算K,Q的乘积并用 softmax 归一化得到注意力系数图，再与V相乘，得到带有自注意力的特征图。最后再经过一个1×1的卷积核得到最终输出。以上过程以公式描述为：

$$K = W_1 * X \tag{6.1}$$

$$Q = W_2 * X \tag{6.2}$$

$$V = W_3 * X \tag{6.3}$$

$$A = \text{softmax}(KQ^T) \tag{6.4}$$

$$X_{out} = W_4 * (VA^T) \tag{6.5}$$

其中X为输入特征，即合并的三个雷达观测特征，W为卷积核权重。经过这种自注意力加权，模型能够更加注重特征的细节部分，提升模型预测效果。

6.2.3 集成学习

集成学习是机器学习中常用的增强模型学习效果的方法，其主要思想是使用多个模型加强学习效果，即使每个模型并不能达到最佳效果，但它们各有所长，将结果结合则能得到比单个模型更好的性能。集成学习可以分成 Bagging 和 Boosting 两种，前者同时并行训练多个相互独立的模型，通过聚合这些模型的结果得到比单个模型更好的性能；后者则先训练一个基本模型，然后选出数据中不能被基本模型很好拟合或分类的样本，训练第二个增强模型专用于学习这些基本模型不能覆盖的样本。我们使用 Bagging 集成学习的思想，同时训练多个 U-Net 模型增强学习效果。

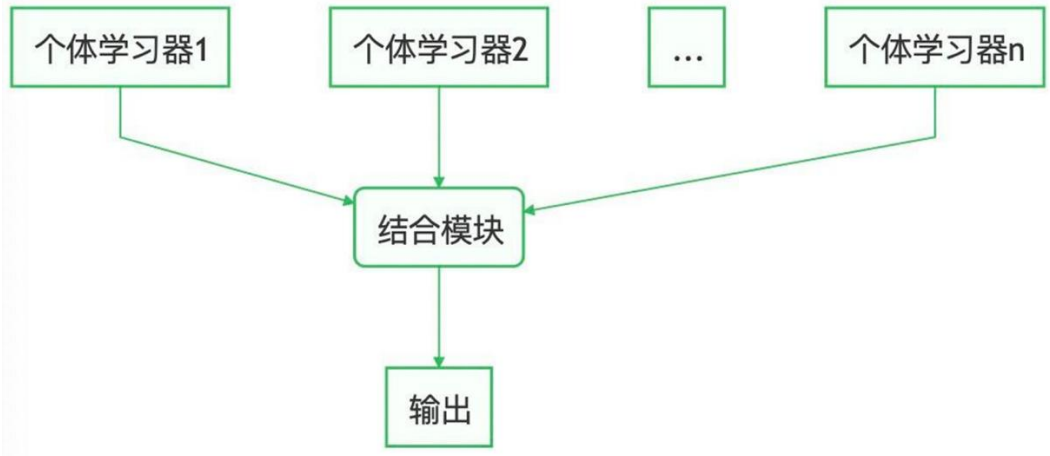


图 6.4 集成学习示例

具体地，我们同时训练三个 U-Net 来对数据进行更加精细的学习。为了学习数据的不同方面的特征，对三个模型的输入采取不同的处理：第一个模型不做处理，直接输入原本的数据。第二个模型的输入为经过边缘提取的雷达观测量数据，边缘的定义为，数据突变较大的连续区域。因此，如果将每个格点的数据与其相邻的格点做差值计算，那么差值较大的格点处就有较大可能是图形的边缘。具体提取方法为，使用一个特定权重的卷积核对数据进行卷积操作，在图像处理领域，该卷积核被称为算子。我们使用 Sobel 算子来提取边缘，Sobel 算子定义如下：

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix},\ S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{6.6}$$

这两个算子分别用于提取沿着横竖两个方向上的边缘格点，而提取后的值为两个算子卷积结果的平均值。边缘提取的效果如图所示，仅有边缘部分的格点有较大的数值，其他格点均接近 0。

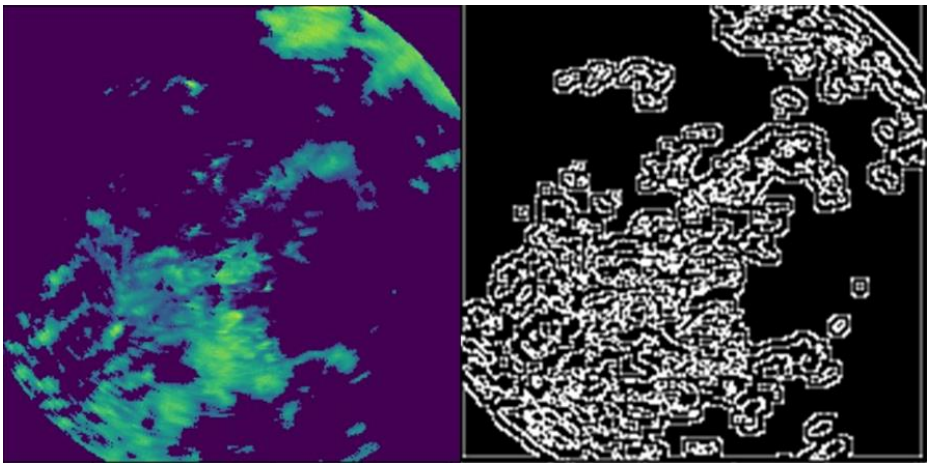


图 6.5 边缘提取示例图

第三个模型的输入为经过增强处理的雷达观测数据，其中的数值都经过一个线性拉伸，使得数据的方差变得更大，并减弱了部分模糊的边缘。具体地，对于每个格点数据 x，变换后的数据为

$$y = \begin{cases} \frac{g_a}{f_a} x,\ 0 < x \le f_a \\ \frac{g_b - g_a}{f_b - f_a} x + g_a,\ f_a < x \le f_b \\ \frac{1 - g_b}{(1 - f_b)} x + g_b,\ f_b < x \le 1 \end{cases} \tag{6.7}$$

其函数图像如图所示，能够将数据之间的差异放大，突出较大值的数据点减弱较小值的数据点。经过拉伸变换后的图像如图所示。

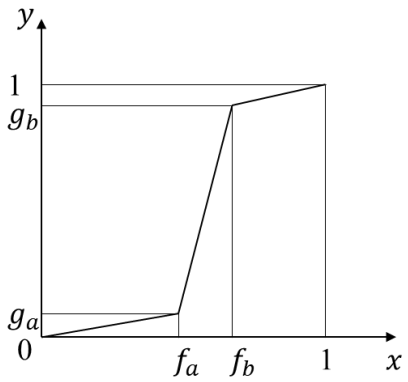


图 6.6 线性拉伸函数

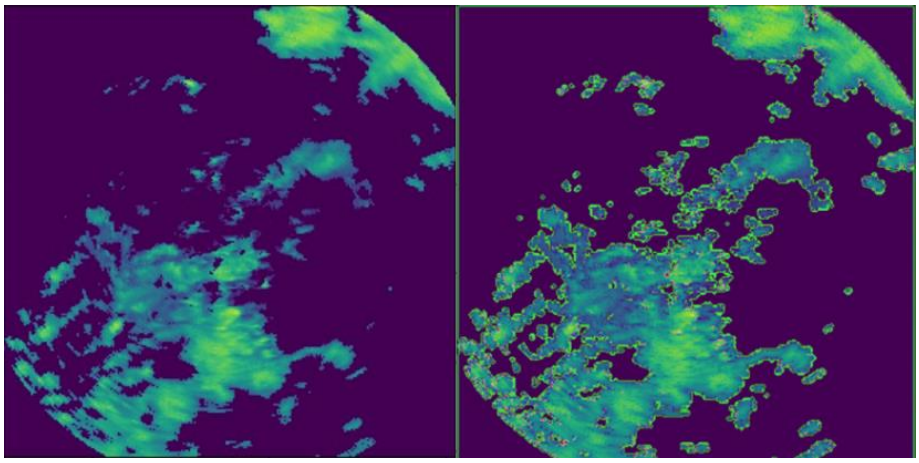


图 6.7 拉伸变换示例图

三个模型分别对经过不同处理的雷达观测数据进行特征学习，而最终的结果由可学习的参数加权得到：

$$Z_H^f = w_1 Z_H^1 + w_2 Z_H^2 + w_3 Z_H^3 \tag{6.8}$$

$$w_1 + w_2 + w_3 = 1 \tag{6.9}$$

6.3 求解结果与分析

为便于比较说明，本章使用和第一问相同的设置，即相同的求解算法、损失函数和评价指标。各评价指标结果如表所示。可以看到相比第一问的结果，第二问的评价指标明显更优，MSE、MRE 和 RBIAS 相比第一问均下降约 50%，而 CSI 则提升了 0.2 达到 0.87。

表 6.1 评价指标结果

	MSE	MRE	RBIAS	CSI
第一问	9.0280	3.9792e-7	0.2608	0.6603
第二问	4.7222	2.0732e-7	0.1359	0.8741

同样绘制四个指标在一小时内的变化趋势如图所示，仍然是时间越靠后的预测误差越大，但是可以注意到相比第一问的变化趋势，改进后模型的预测评价指标在后续一小时内的前几帧上误差增加得更慢了，也即在短时间内模型预测的准确率是有所改善的。

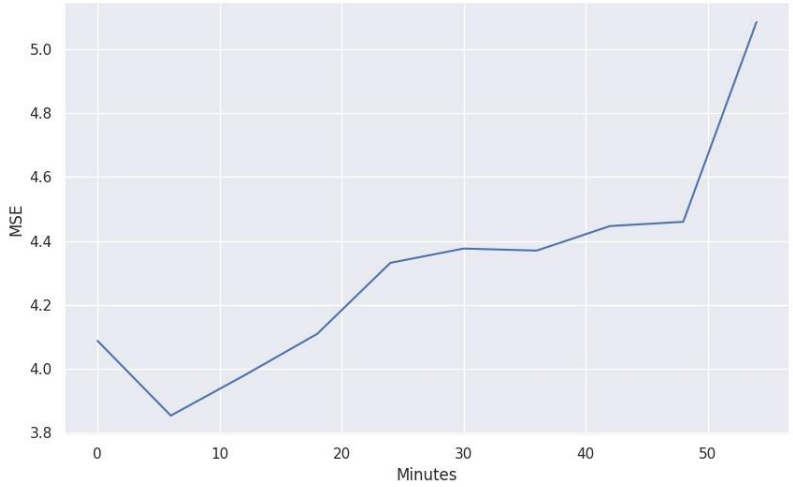


图 6.8 MSE 变化曲线

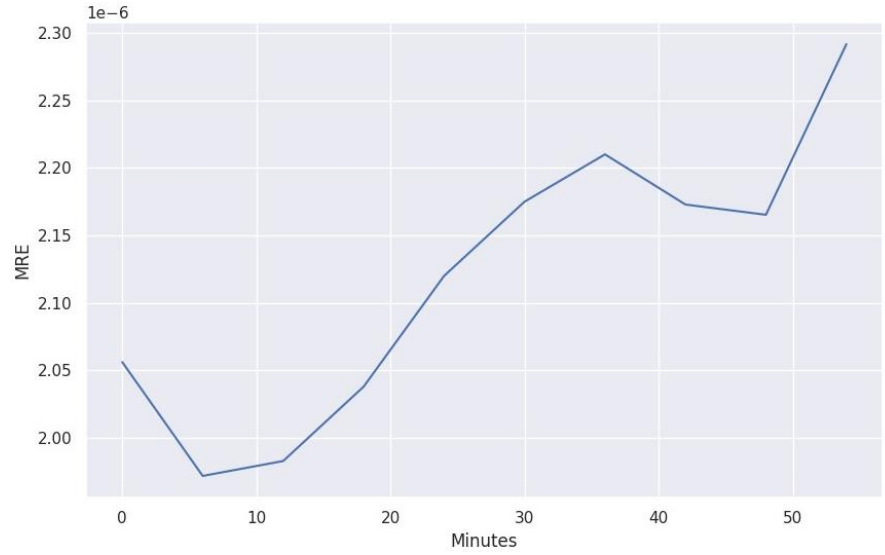


图 6.9 MRE 变化曲线

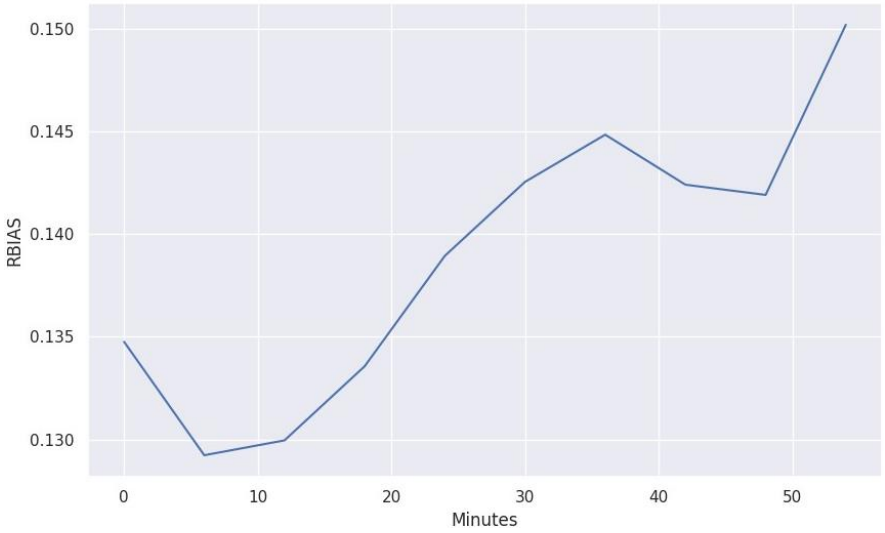


图 6.10 RBIAS 变化曲线

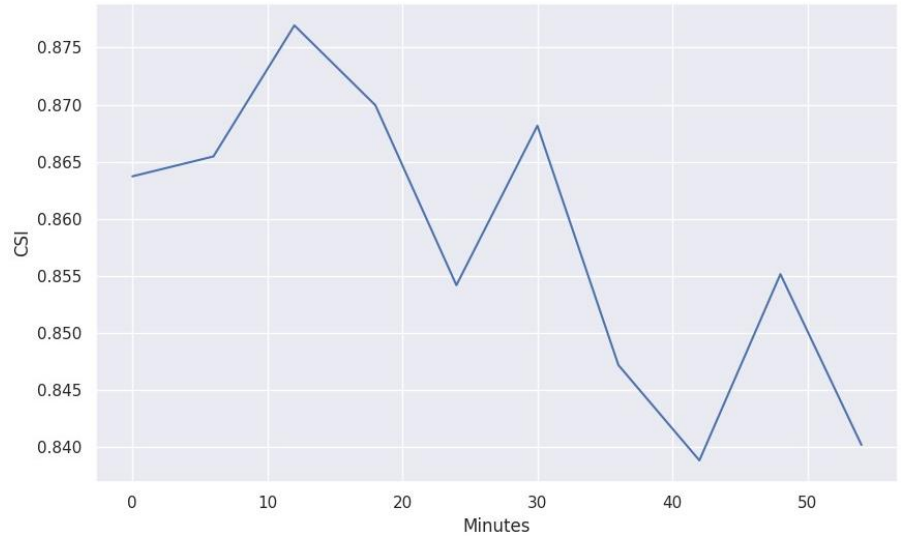


图 6.11 CSI 变化曲线

如图所示，改进后的模型预测数据的直方图分布相比之前图 6.1 更接近实际结果，并且回归到平均的问题有明显改善。

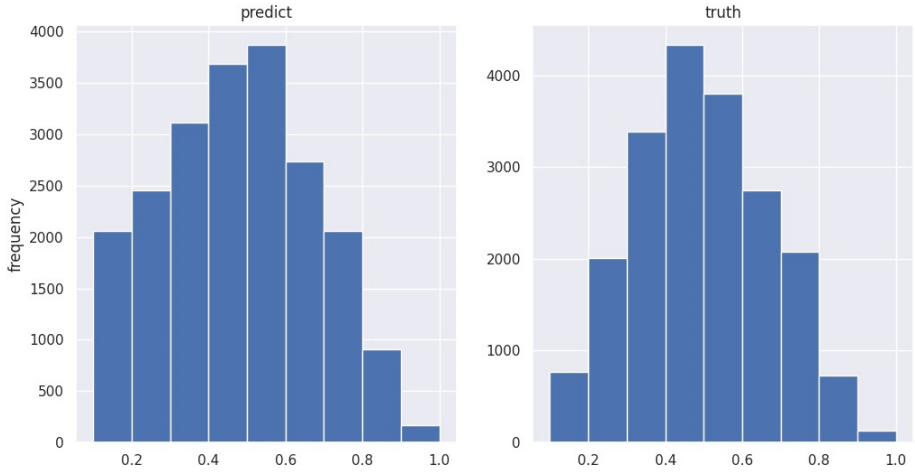


图 6.12 预测与实际结果直方图

预测结果与实际结果可视化比较如图 6.13 所示，同样是随机选取一小时的雷达观测数据预测下一小时的Z_H值，选取了其中五帧进行可视化展示。从图中可以直观得看出预测结果已经和实际结果相差无几，并且能很好的还原实际结果中边缘和零散的数据信息，也没有出现整体色差的情况。

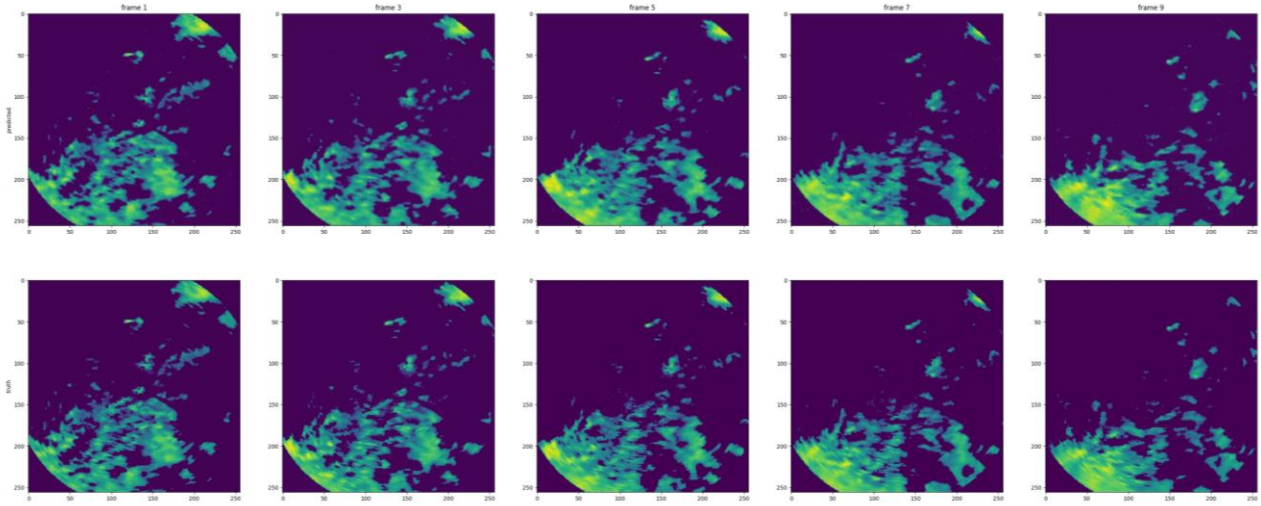


图 6.13 预测结果与实际结果可视化比较

7 问题三

7.1 问题分析

问题三要求利用题目提供的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R，并且要求算法不可使用K_{DP}变量。本问题的难点在于目前的大多数主流模型都以最小化预测对数似然，均方误差(MSE)等指标来训练模型。但是基于对数似然的优化通常会导致预测的平均、模糊性的问题，并且随着预测步数的增加，预测值变得更宽泛，天气预测的不确定性越来越大。为此，采用扩散模型来解决降水量预测的问题，用马尔可夫过程的概率分布来预测降水量。具体来说，整合Z_H和Z_{DR}作为扩散模型学习过程的输入，通过扩散模型逐步去噪的方式，输出降水量R的预测。结果表明，在典型的均方误差指标上，扩散模型优于传统的 WF-UNet 等模型。我们的主要步骤如下：

（1）整理Z_H、Z_{DR}和降水量R数据，分析Z_H、Z_{DR}和R之间的关系，总结降水规律，进行数据预处理，标准化等操作。

（2）介绍扩散模型的主要架构以及公式推导、输入、输出等，证明其用于降水量预测的可行性。

（3）对原始扩散模型进行改进使其更加适应降水量预测任务，首先利用扩散模型生成一组可能的降水量矩阵，然后通过后处理网络合并成最终预测。

（4）对比了扩散模型可传统 WF-UNet 模型在均方误差下的性能。

7.2 Z-R 关系分析

我们通过数据分析探究了不同高度下的Z_H、Z_{DR}对降水量R的影响，重点分析了：1）Z_H和Z_{DR}的值对降水量的影响，什么情况下降水量更高。2）不同高度的Z_H和Z_{DR}对降水量的影响趋势是否一致。3）不同高度的Z_H是否呈线性关系，不同高度的Z_{DR}是否呈线性关系。

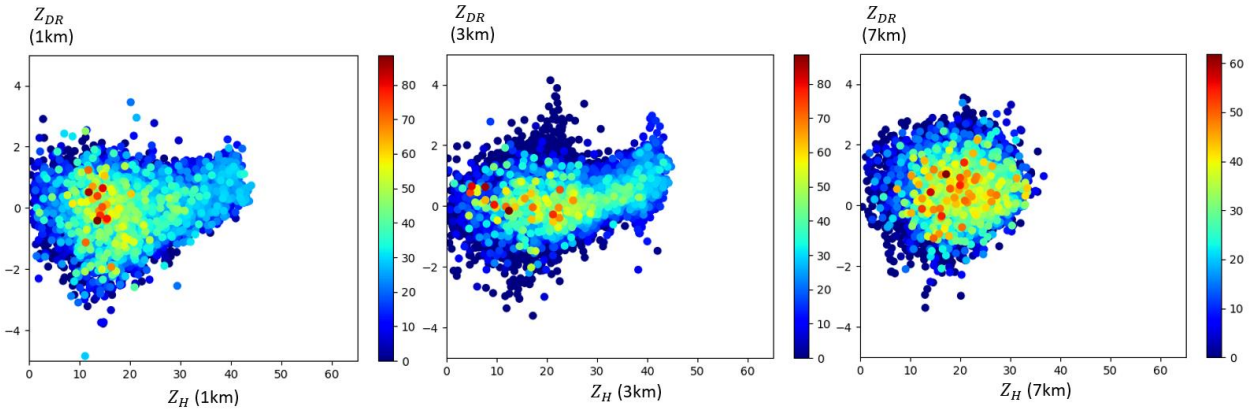


图 7.1 某一帧的Z_H-Z_{DR}-R散点图（色阶为降水量）

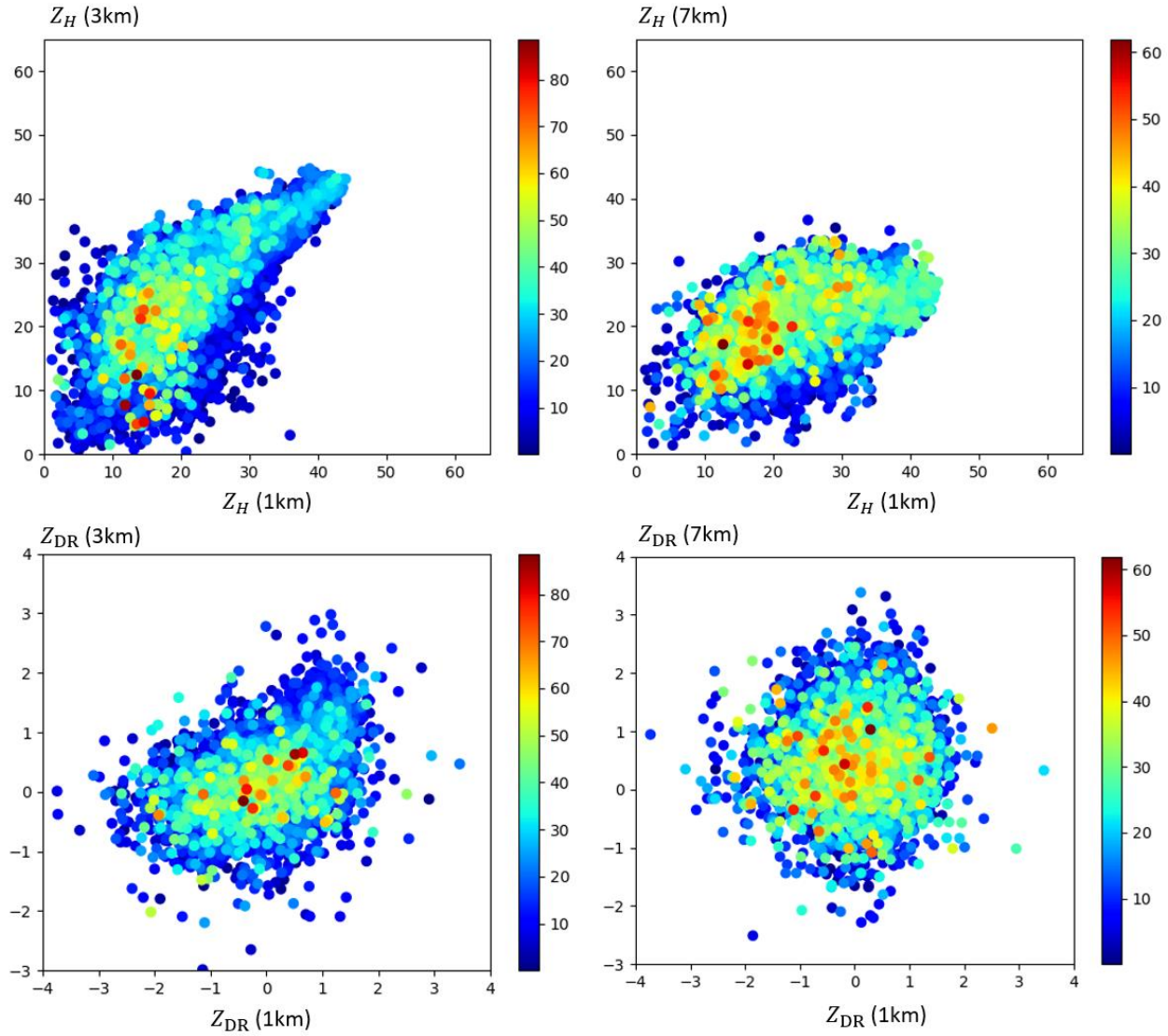


图 7.2 某一帧不同高度下（3km、7km）的Z_H-R和Z_{DR}-R散点图（色阶为降水量）

根据图 7.1 和图 7.2 我们可以得出三点结论：

1）$\boldsymbol{Z_H}$是降雨量的主要影响因素，当高度为 1km，$\boldsymbol{Z_H}$介于 10 到 20 之间，$\boldsymbol{Z_{DR}}$介于-1 到 1 时，降雨量较大。当高度为 3km，$\boldsymbol{Z_H}$介于 10 到 30 之间，$\boldsymbol{Z_{DR}}$介于-1 到 1 时，降雨量较大。当高度为 7km，$\boldsymbol{Z_H}$介于 10 到 30 之间，$\boldsymbol{Z_{DR}}$介于-1.5 到 1.5 时，降雨量较大。

2）不同高度的Z_H和Z_{DR}对降水量的影响趋势一致。

3）不同高度的Z_H基本呈线性关系，高度越高，Z_H越大。不同高度之间的Z_{DR}不呈线性关系。

根据我们的分析，Z_H、Z_{DR}对降水量R具有相关性，不同的高度对降水量R也有影响，因此，我们可以扩散模型来进行降水量R的分布预测。

7.3 扩散模型

扩散模型[5]是一类概率生成模型，在对复杂的高维数据分布进行建模时特别有效。在机器学习的背景下，扩散模型利用扩散过程的原理来模拟数据的生成。模型不是直接从固定分布中采样数据点，而是迭代地将简单的初始分布转换为所需的复杂数据分布。主要思

想是执行一系列扩散步骤，其中每个步骤都会更新数据的概率分布。这是通过向当前数据样本添加高斯噪声并迭代细化来实现的。

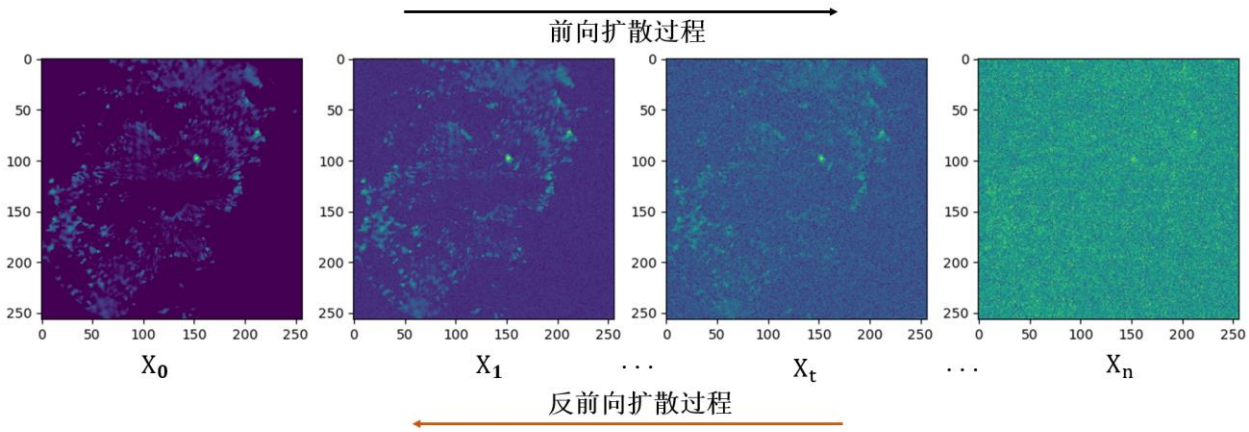


图 7.3 某一帧的降水量预测扩散模型过程示意

图 7.3 展示了某一帧的降水量预测扩散模型示意，X_0是随机选取的某一帧的降水量矩阵，首先进行前向扩散过程，通过添加随机高斯噪声，逐步将X_0加噪至不存在任何分布的X_n，同时训练一个 U-Net 网络来学习每一个时间步t所添加的噪声。然后进行反向扩散过程，将Z_H和Z_{DR}输入至 U-Net 网络来提示它预测每个时间步的噪声。逐渐将一个无分布矩阵去噪，最终得到预测的$\widehat{X_0}$。

7.3.1 训练过程

从数学角度来看，考虑生成数据的分布$q(x_0)$，生成模型旨在找到参数向量θ，使得由神经网络参数化的分布$p_\theta(x_0)$近似$q(x_0)$。

去噪扩散概率模假设生成分布$p_\theta(x_0)$具有以下形式：

$$p_\theta(x_0))\int p_\theta(x_{0:T})dx_{1:T} \tag{7.1}$$

给定时间范围范围$T > 0$。其中：

$$p_\theta(x_{0:T}) = p_\theta(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t) \tag{7.2}$$

训练传统上基于负对数似然的变分下界：

$$\begin{aligned} & -\log\, p_\theta(x_0) \\ \leq\ & -\log\, p_\theta(x_0) + D_{\mathrm{KL}}(q(x_{1:T} \mid x_0) \parallel p_\theta(x_{1:T} \mid x_0)) \\ =\ & -\log\, p_\theta(x_0) + \mathbb{E}_q[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})/p_\theta(x_0)}] \\ =\ & -\log\, p_\theta(x_0) + \mathbb{E}_q[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})} + \log\, p_\theta(x_0)] \\ =\ & \mathbb{E}_q[\log\, q(x_{1:T} \mid x_0) - p_\theta(x_{0:T})] = \mathcal{L}(\theta) \end{aligned} \tag{7.3}$$

扩散模型采用固定（不可训练）的推理过程$q(x_{1:T} \mid x_0)$。此外，潜在变量的特点是相对较

高的维度，通常与可见空间的维度相同。

在问题三中使用的去噪扩散隐式模型的特定情况下，我们考虑了非马尔可夫扩散过程：

$$q_\sigma(x_{1:T} \mid x_0) = q_\sigma(x_T \mid x_0) \prod_{t=2}^{T} q_\sigma(x_{t-1} \mid x_t, x_0) \tag{7.4}$$

接下来，需要定义一个可训练的生成过程$p_\theta(x_{0:T})$，其中$p_\theta(x_{t-1} \mid x_t)$利用$q_\sigma(x_{t-1} \mid x_t, x_0)$的结构。这个想法是给定一个有噪声的观测值$x_t$，开始对$x_0$进行预测，然后用它获得$x_{t-1}$。在实践中，我们训练神经网络$\epsilon_\theta^{(t)}(x_t, \alpha_t)$将给定的$x_t$和噪声率$\alpha_t$映射到添加到$x_0$以构造$x_t$的噪声$\epsilon_t$的估计。因此，$p_\theta(x_{t-1} \mid x_t)$变为$\delta_{f_\theta^{(t)}}$，其中：

$$f_\theta^{(t)}(x_t, \alpha_t) = \frac{x_t - \sqrt{1-\alpha_t}\epsilon_\theta(x_t, \alpha_t)}{\sqrt{\alpha_t}} \tag{7.5}$$

将$f_\theta^{(t)}(x_t, \alpha_t)$作为$x_0$在时间步长$t$处的近似值，得到$x_{t-1}$，如下所示:

$$x_{t-1} = \sqrt{\alpha_{t-1}} \cdot f_\theta^{(t)}(x_t, \alpha_t) + \sqrt{1 - \alpha_{t-1} - \sigma_t^2} \cdot \epsilon_\theta(x_t, \alpha_t) \tag{7.6}$$

对于损失函数，将L_θ表示为以下项的和:

$$L_\theta = L_T + L_{t-1} + \cdots + L_0 \tag{7.7}$$

其中：

$$\begin{aligned} &L_T = D_{\mathrm{KL}}(q(x_T \mid x_0) \parallel p_\theta(x_T)) \\ &L_t = D_{\mathrm{KL}}(q(x_t \mid x_{t+1}, x_0) \parallel p_\theta(x_t \mid x_{t+1})) \\ &L_0 = -\log p_\theta(x_0 \mid x_1) \\ &\text{for } 1 \le t \le T-1 \end{aligned} \tag{7.8}$$

训练和采样的伪代码在下面的算法中给出:

算法 6.1 训练

1: repeat
2: $x_0 \sim q(x_0)$
3: t ~Uniform $(1, .., \mathrm{T})$
4: $\epsilon \sim \mathcal{N}(0; I)$
5: $x_t = \sqrt{\alpha_t} x_b + \sqrt{1-\alpha_t}\epsilon$
6: Backpropagate on $\parallel \epsilon - \epsilon_\theta(x_t, \alpha_t) \parallel^2$
7: **until** converged

算法 6.2 采样

1: $x_T \sim \mathcal{N}(0, I)$
2: for $t = T, \ldots, 1$ **do**
3: $\epsilon = \epsilon_\theta(x_a, x_t, \alpha_t)$
4: $\tilde{x}_0 = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\alpha_t}}\epsilon)$
5: $x_{t-1} = \sqrt{\alpha_{t-1}}\tilde{x}_0 + \sqrt{1-\alpha_{t-1}}\epsilon$
6: end for

7.3.2 生成过程

生成过程通常需要一种方法来控制如何创建样本来影响最终的输出。这个过程通常被称为条件扩散。目的将图像或文本嵌入到扩散过程中，允许引导生成。在问题三中，我们将每一帧的Z_H和Z_{DR}嵌入到扩散过程中，用来引导生成本帧的降水量R。

为了将扩散模型p_θ转化为条件扩散模型，我们可以在每个扩散步骤引入条件信息y，如下所示:

$$p_\theta(x_{0:T} \mid y) = p_\theta(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t, y) \tag{7.9}$$

通常，相同的神经网络可以用于两个模型：在训练期间，y被随机设置为 0，将模型暴露给有条件和无条件的设置。在时间步长t处的估计噪声$\hat{\epsilon}_\theta(x_t \mid t, y)$是有条件预测和无条件预测的加权组合:

$$\hat{\epsilon}_\theta(x_t, t, y) = \epsilon_\theta(x_t, t, y) + s \cdot \epsilon_\theta(x_t, t) \tag{7.10}$$

7.3.3 降雨量预测可行性分析

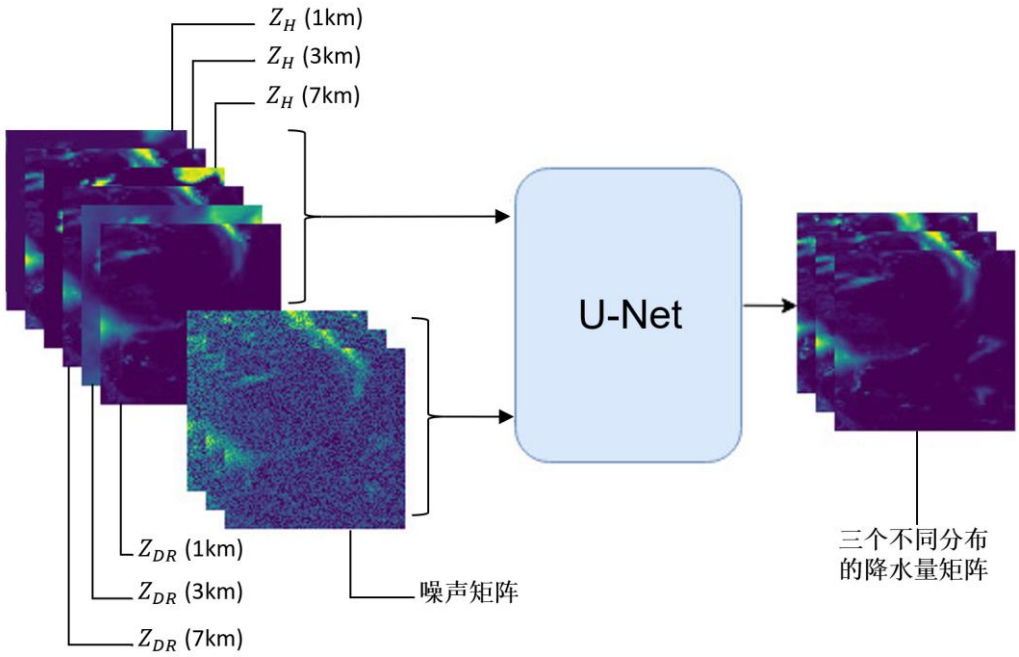


图 7.4 扩散模型降水量预测结构图

我们已经证明了扩散模型对于马尔可夫过程与非马尔可夫过程都是有效的，因此，扩散模型自然地契合降水量预测任务，在预测阶段，输入是不同高度的Z_H、Z_{DR}矩阵以及当前步t的噪声矩阵，通过训练好的 U-Net 网络，预测$t+1$步的噪声，经过n步最终得到降水量预测矩阵。值得注意的是，原始的扩散模型只能预测一个降水量矩阵，我们特别为降水量预测任务设计了组合式扩散结构，让模型生成三个输出，然后通过 U-Net 架构将这些输出合成为最终的预测。这种组合式扩散结构提供了全面的降水预测，提高了模型对降水量

数据的概率分布建模能力和 U-Net 的特征提取强度。

7.4 组合式扩散模型

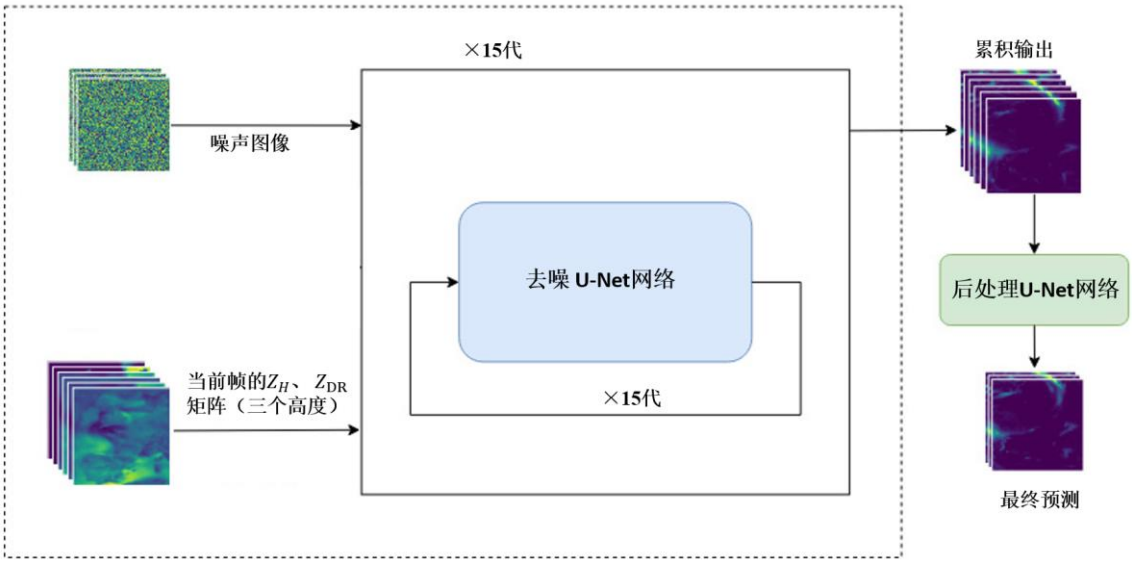


图 7.5 组合式扩散模型预测结构

7.4.1 去噪网络

我们选择的去噪网络模型是 U-Net 网络。U-Net 网络包括一个层的下采样序列和一个层的上采样序列，同时在相同大小的层之间合并跳过连接。通常，U-Net 的配置是通过定义下采样块的数量和每个块的通道数量来决定的。上采样结构遵循对称模式，空间维度依赖于矩阵维度，在降水量预测中，矩阵维度是 256×256。因此，U-Net 的整个结构可以简洁地编码在一个列表中。在我们的实验中，我们选择的 U-Net 尺寸为[64,128,256,384]。为了提高 U-Net 对噪声方差的敏感性，将α_t作为输入，然后使用一种特殊的正弦变换将其嵌入到一组频率中。使用 Lambda 层实现正弦嵌入。

7.4.2 模型调节

必须对模型进行调节，以引导扩散到由已知的过去帧降水量预测临近降水量。实际上，模型$\epsilon_\theta(x_t, t, y)$以噪声图像$x_t = \{r_1, r_2, r_3\}$作为输入，其中$r$表示降水量预测。条件信息$y = \{Z_H^{1km}, Z_{DR}^{1km}, Z_H^{3km}, Z_{DR}^{3km}, Z_H^{7km}, Z_{DR}^{7km}\}$包含当前的$Z_H$和$Z_{DR}$信息。

我们的实现直接向 U-Net 提供条件信息y，具体来说，输入的数据中的每个时间切片都被类似地处理为 RGB 图像中的颜色通道。例如，当批量大小为 16 时，输入到去噪网络的数据的形状为[16,256,256,9]，最后一个维度包含了条件作用信息（6 维）和噪声图像（3 维）。输出将只包含去噪的 3 帧，因此有[16,256,256,3]的形状。

7.4.3 组合扩散策略

我们的组合扩散策略利用模型的扩散能力来整合气象模式的固有概率分布，然后用于合成可能的降水量预测。通过扩散过程，模型依据不同的生成特性，尽管共享相同的条件信息，但产生了一组高度多样化的输出。我们选择神经模型来执行集成预测，使用 U-Net 架构将生成的结果合并成更可能的预测。我们将这种方法与平均集成的方法相对比，结果

表明选择神经模型来执行集成预测能够输出更准确的预测，对解决“回归到平均问题”也有很大帮助，

7.5 问题求解

7.5.1 数据预处理

同问题一，我们对数据进行预处理，剔除无效数据，删除全零帧，并做标准化处理。

7.5.2 评价指标

所有的实验都是使用 pytorch 框架中实现的。测试数据为 data_dir_000 至 data_dir_010，训练数据为 data_dir_011 至 data_dir_257。我们通过四个指标对模型的好坏进行评价，分别是均方误差（MSE）、平均相对误差（MRE）、相对偏差（RBIAS）和临界成功指数（CSI）[6]。

7.5.3 预测结果

表 7.1 预测结果比较

模型	MSE	MRE	RBIAS	CSI
U-Net	41.616	0.324	0.052	0.910
WF-UNet	35.937	0.308	0.027	0.913
扩散模型	33.346	0.289	0.022	0.893
组合扩散模型 (平均集成)	22.168	0.256	**0.011**	0.917
组合扩散模型 (U-Net 集成)	**18.934**	**0.253**	0.016	**0.932**

在表 7.1 中，我们比较了不同模型的实验结果。包括 U-Net 模型、WF-UNet 模型、扩散模型、平均集成策略的组合扩散模型、U-Net 集成策略的组合扩散模型。我们发现 U-Net 集成策略的组合扩散模型在总体上好于其他模型。

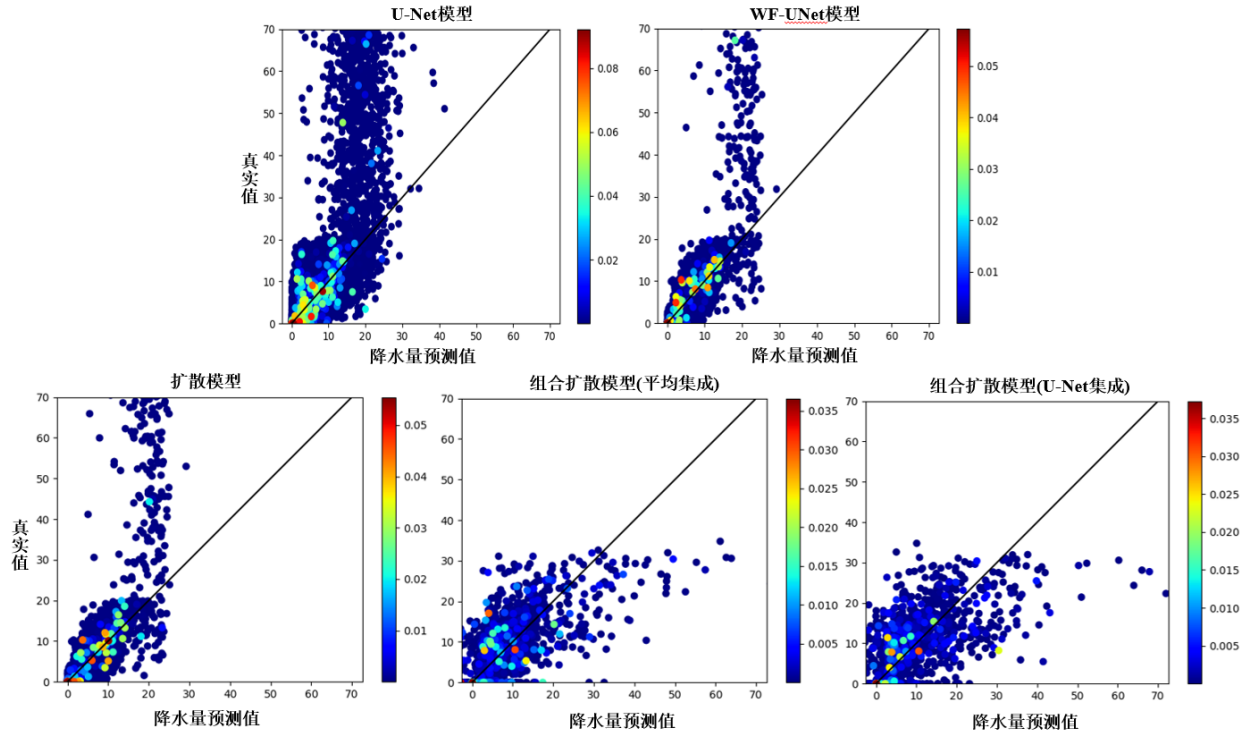


图 7.6 降水量预测真实值-预测值散点图（色阶为解释方差）

在图 7.6 中我们比较了不同模型的预测值和真实值之间的散点图，并计算了每个散点的解释方差，来说明不同模型的预测趋势。

7.5.4 消融案例对比

我们的设计思路是从普通 U-Net 模型到 WF-UNet 模型，再尝试了扩散模型，最后改进得到组合扩散模型，一步一步尝试、优化、改进的，因此我们对比了过程中各个模型的输出结果，红色圈代表预测不佳的区域。

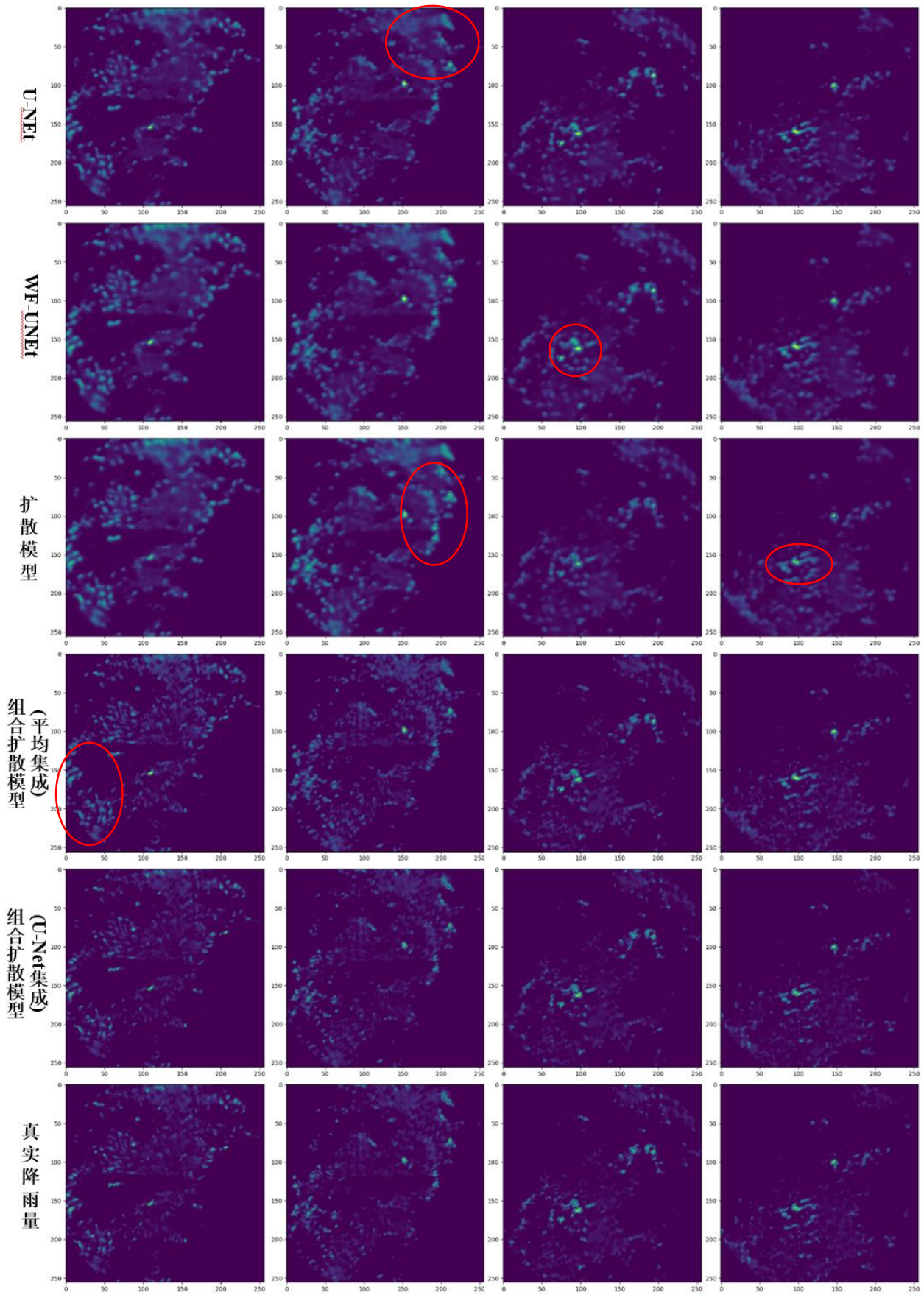


图 7.7 消融案例对比结果

7.6 影响因素探究

我们探究了在不同降雨量下各个模型的均方误差，具体来说，我们设置了两个阈值，分别为 30、50，降水量小于 30 为小雨，大于 30 小于 50 为中雨，大于 50 为大雨。我们分真实降雨量为小雨、中雨、大雨的情况下统计均方误差的变化。

表 7.2 不同雨量 MSE 比较

模型	小雨 (R<30)	中雨 (30≤R<50)	大雨 (R≥50)	总体
U-Net	35.189	42.370	50.090	41.616
WF-UNet	32.793	38.607	46.192	35.937
扩散模型	34.307	33.284	33.790	33.346
组合扩散模型 (平均集成)	22.050	22.286	23.002	22.168
组合扩散模型 (U-Net 集成)	**18.833**	**19.067**	**19.071**	**18.934**

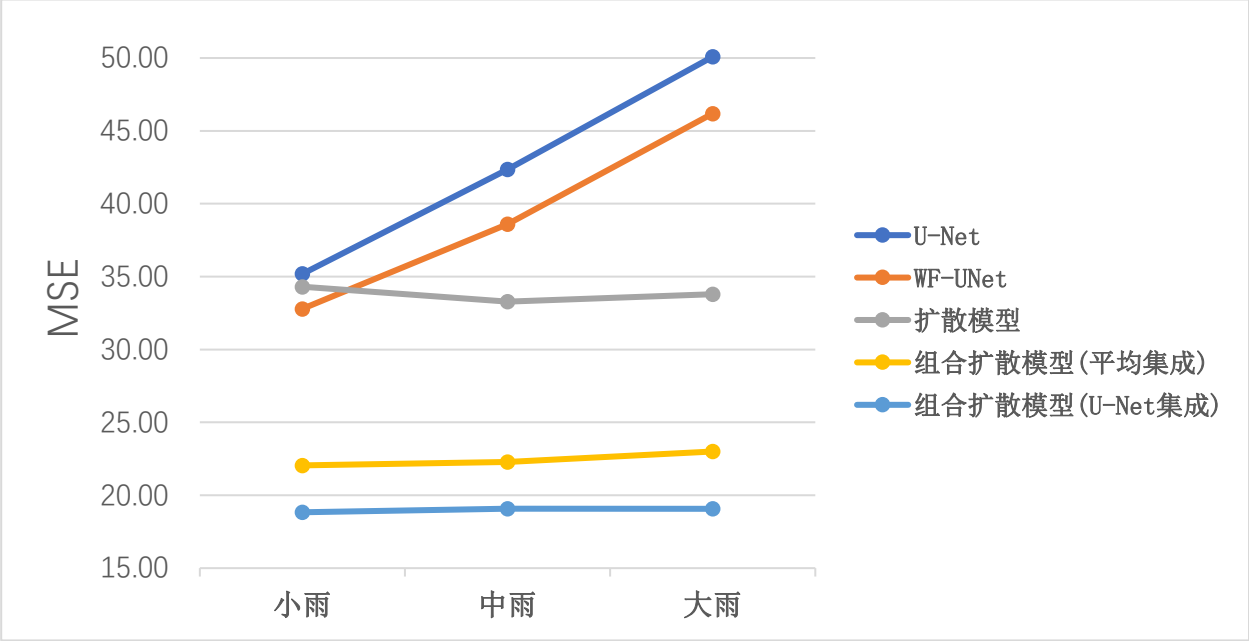


图 7.8 不同雨量 MSE 比较

7.7 结果分析

（1）从表 7.1 可以看出，普通的扩散模性能与 WF-UNet 模型相当，但是组合扩散模型的性能在三个指标上达到了最佳，说明组合扩散策略对临近降水量预测具有一定的作用，提高了模型预测精度，原因是通过先多样化输出后集成的方式，模型大大增加了预测的可信度与容错率，不会因为神经网络中某个神经元的错误而直接局部区域预测错误。

（2）从表 7.1 可以看出，U-Net 集成的组合扩散模型在性能上好于平均集成，说明 U-Net 集成策略在整合输出时进行了更明智的决策过程。

（3）从图 7.6 可以看出，U-Net 模型、WF-UNet 模型和扩散模型的预测值偏低，对降水量较小时预测效果比较好，对降水量较大时，预测偏差较大。而组合扩散模型则可以很好地解决这个问题，原因是组合扩散模型集成了多样化的预测，泛化性更强，更接近真实的预测值。

（4）从图 7.7 可以看出，所有模型都能较为准确的预测降水量矩阵的大概轮廓，但是不同的模型分别存在一些问题。U-Net 模型和 WF-UNet 模型出现了较严重的“回归到平均”问题，导致预测模糊，例如红圈圈起来的位置。扩散模型较好地解决了 U-Net 模型和 WF-UNet 模型“模糊”问题，原因是扩散模型是预测其分布而不是预测单个概率，但是可以看见扩散模型的结果比真实结果整体颜色偏亮，出现了整体预测降水量过大的问题。组合扩散模型的效果最接近真实结果，但是平均集成的组合扩散模型还会出现局部地区预测“较为模糊”的问题，原因是通过平均集成会减弱输出矩阵的局部特征。而 U-Net 集成的组合扩散模型效果最好，极为接近真实的降雨量结果，说明 U-Net 集成的组合扩散模型在临近降水量预测任务中是非常有效的。

（5）从表 7.2 和图 7.8 可以看出，U-Net 模型和 WF-UNet 在预测中大雨时效果低于预测小雨时的效果，原因是 U-Net 网络模型的限制导致预测量较大时均方差较大。而扩散模型解决了这个问题，在预测不同大小的雨均方差基本保持不变。这对于临近降水预测时一个非常有进步的一点。

8 问题四

8.1 问题分析

问题四可以分为两个小问，第一小问需要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，对比传统的方法，主要考虑加入双偏振雷达特有的垂直方向上的电磁波反射数据后，对强对流降水临近预报的贡献。第二问需要优化数据融合策略，以便更好地应对突发性和局地性强的强对流天气。

对比传统与双偏振雷达，各种分析好处，

数据融合，模型融合，贝叶斯等

第二问，那篇文章，分析原因，以及解决方法

8.2 双偏振雷达 VS 传统雷达

传统雷达只具有水平反射率因子信息Z_H，而双偏振雷达还额外具有垂直方向上的电磁波反射情况Z_{DR}和K_{DP}。因此具备更多的可用信息用来强对流天气降水预测。为了评估双偏振雷达资料在强对流降水临近预报中的贡献。我们只用Z_H作为传统雷达的数据，用Z_H和Z_{DR}作为双偏振雷达数据。并修改了 U-Net 模型、WF-UNet 模型、扩散模型、平均集成的组合扩散模型、U-Net 集成的组合扩散模型的超参数，使之适应传统雷达数据。最后对比了各个模型在传统数据和双偏振雷达数据上的预测性能差异[7]。

8.2.1 对比结果

同问题一，我们首先对数据进行预处理，剔除无效数据，删除全零帧，并做标准化处理。所有的实验都是使用 pytorch 框架中实现的。测试数据为 data_dir_000 至 data_dir_010，训练数据为 data_dir_011 至 data_dir_257。我们通过四个指标对模型的好坏进行评价，分别是均方误差（MSE）、平均相对误差（MRE）、相对偏差（RBIAS）和临界成功指数（CSI）。

表 8.1 对比结果比较

模型	**MSE**		**MRE**		**RBIAS**		**CSI**	
数据	原始雷达	双偏振雷达	原始雷达	双偏振雷达	原始雷达	双偏振雷达	原始雷达	双偏振雷达
U-Net	78.913	41.616	0.577	0.324	0.104	0.052	0.813	0.910
WF-UNet	66.347	35.937	0.493	0.308	0.082	0.027	0.869	0.913
扩散模型	69.724	33.346	0.509	0.289	**0.067**	0.022	0.851	0.893
组合扩散模型(平均集成)	43.369	22.168	**0.433**	0.256	0.083	**0.011**	0.876	0.917
组合扩散模型(U-Net 集成)	**42.918**	**18.934**	0.442	**0.253**	0.077	0.016	**0.907**	**0.932**

在表 8.1 中，我们比较了不同模型在不同雷达数据下的实验结果。包括 U-Net 模型、WF-UNet 模型、扩散模型、平均集成策略的扩散模型、U-Net 集成策略的组合扩散模型。数据包括传统雷达数据和双偏振雷达数据。

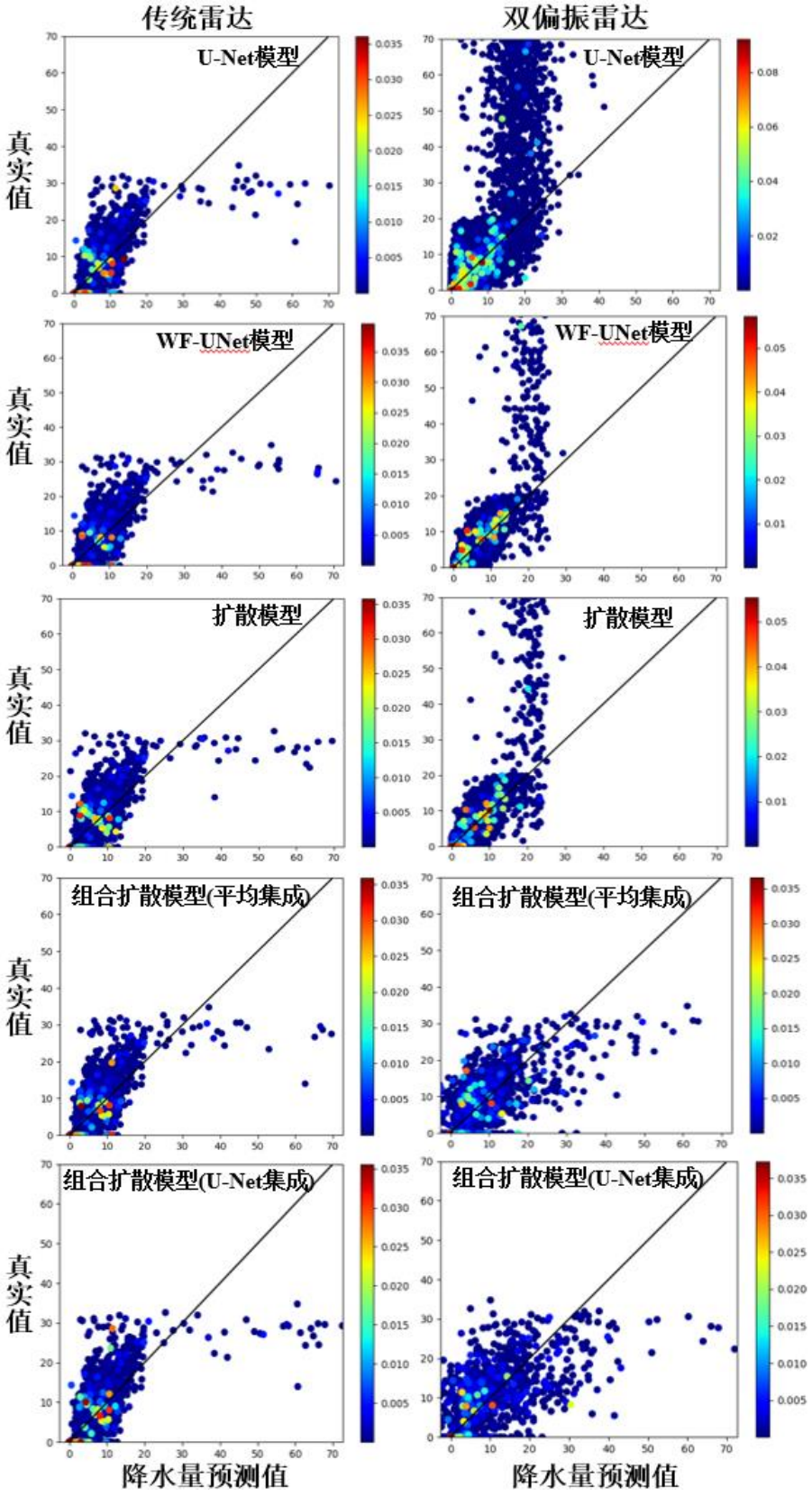


图 8.1 不同雷达数据降水量预测真实值-预测值散点图（色阶为解释方差）

在图 8.1 中我们比较了不同模型在两种数据下的预测值和真实值之间的散点图，并计算了每个散点的解释方差，来说明不同模型的预测趋势。

8.2.2 案例对比

我们对比了各个模型在不同训练数据下的输出结果，红色圈代表预测不佳的区域。

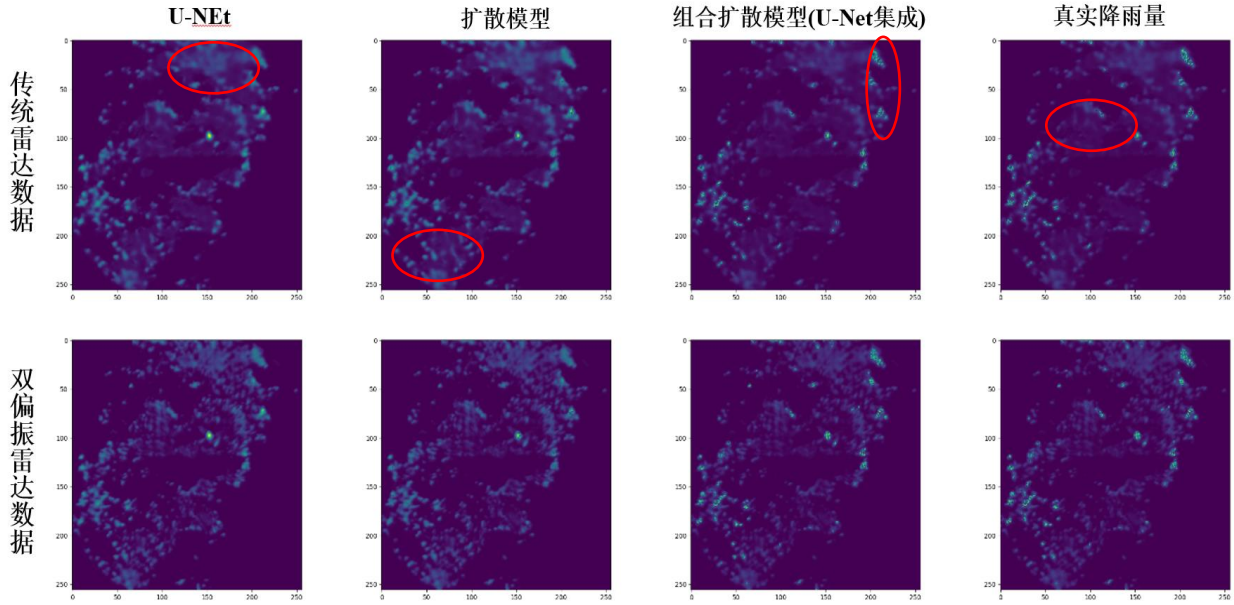


图 8.2 不同数据案例对比结果

8.2.3 对比结果分析

（1）从表 8.1 可以看出，无论是哪种模型，在利用传统雷达数据时，性能都比双偏振雷达数据差，说明双偏振雷达数据的Z_{DR}对模型预测降雨量具有帮助，也说明了双偏振雷达资料在强对流降水临近预报中的贡献。

（2）从图 8.1 可以看出，当利用传统雷达数据进行训练时，模型预测值偏高，并且较散，而当利用双偏振雷达数据进行训练时，模型预测值偏低，但是较有规律。说明双偏振雷达数据更好地捕捉了降水量的分布信息，因此模型预测更为准确。

（3）从图 8.2 可以看出，利用传统雷达数据进行强对流降雨量预测时，更容易出现“回归到平均”的模糊现象，而利用双偏振雷达数据预测的降雨量矩阵明显更清晰，说明Z_{DR}数据可以帮助模型学习降水量的衍化信息。

8.3 特征贡献度计算

神经网络中不同的输入维度对模型预测的影响是不同的，因此我们选择 U-Net 集成的组合扩散模型来计算每个特征的重要性是考量维度对模型贡献的重要依据。我们通过沙普利值法来计算 6 个输入维度（$\{Z_H^{1km}, Z_{DR}^{1km}, Z_H^{3km}, Z_{DR}^{3km}, Z_H^{7km}, Z_{DR}^{7km}\}$）对强对流降雨临近预测的重要性，进而突双偏振雷达资料在强对流降水临近预报中的贡献。

8.3.1 沙普利值计算

可解释性领域定义重要性的主要方法是用的是合作博弈论里面的沙普利值,具体定义是：

$$\phi_i(\boldsymbol{x}) = \sum_{S \subseteq N \setminus i} \frac{|S|!(|N|-|S|-1)!}{|N|!}\left(v(x_{S\cup\{i\}}) - v(x_S)\right) \tag{8.1}$$

其中N是所有特征集合，v是$\mathbb{R}^{|N|} \mapsto \mathbb{R}$的值函数，也可以理解为要计算特征贡献度的模型，$\phi_i(\boldsymbol{x})$即为$\boldsymbol{x}$中第$i$个特征的重要性，$x_S$指$\boldsymbol{x}$中只有在$S$中的特征存在，在实际解释机器学习模型中有许多不同的处理方法。

8.3.2 适应黑箱模型计算

由于 U-Net 集成的组合扩散模型用到了深度学习网络，因此属于黑箱模型，需要将沙普利值改造适配才能用于特诊贡献度的计算。模型分别三部分：1）沙普利采样：从数据集中抽样一个特征，然后将所有数据除去此特征。2）样本点沙普利值预测：我们对每个样本点计算了沙普利值，公式如(8.2)所示。与线性模型的加和方法类似，假设模型基准分（通常是所有样本的目标变量的均值）为y_{base}，第i个样本为x_i，第i个样本的第j个特征为$x_{i,j}$，当$f(x_i,j) > 0$时，说明该特征对目标值的预测起到了正向作用；反之，该特征与目标预测值有相反作用。因此沙普利值不仅给出特征影响力的大小，也反映出每一个样本中的特征的影响力的正负性。3）特征总体沙普利值预测：每个特征在整体样本上的沙普利值的绝对值取平均值来代表该特征的重要性，因此沙普利值均值越大，则特征越重要。

$$y_i = y_{\text{base}} + f(x_{i,1}) + f(x_{i,2}) + \cdots + f(x_{i,k}) \tag{8.2}$$

8.3.3 特征贡献度分析

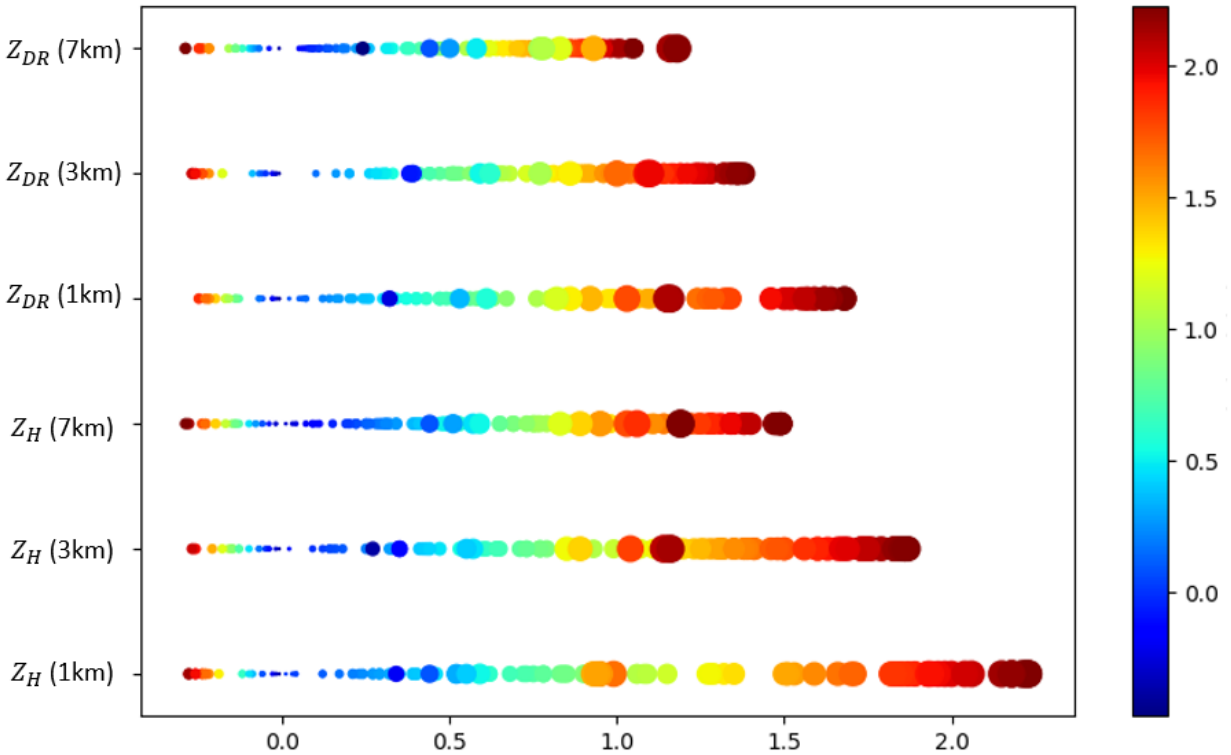


图 8.3 数据维度的特征贡献度结果（色阶表示特征值）

我们可以看到，Z_H的特征贡献度总体上略高于Z_{DR}的特征贡献度，但是Z_{DR}的特征贡献不可忽略。从垂直结构来看，高度越低，特征贡献率越高，说明低层的粒子对降雨的影响最大。

经过特征贡献度分析，我们得出结论，Z_{DR}对模型的预测具有很大帮助作用，但是Z_H起到主导作用。低高度的数据比高高度的数据对模型预测的贡献大。

8.4 模糊检验对比

由于在 8.2 节中我们观察到利用传统雷达数据进行降水量预测时会出现模糊现象，因此，我们利用多事件列联表的模糊检验方法，来验证是否传统雷达数据确实会出现模糊现象。如果传统雷达数据在模糊检验方法中提升比双偏振雷达数据更多的话，则侧面说明了传统雷达数据进行降水量预测时会出现模糊现象。

8.4.1 多事件列联表模糊检验方法

在多事件列联表方法中，对于某一个事件，考虑了几个不同的因子作为预报准确与否的判别标准。如对于强度列联表，分别采用几个不同强度阈值作为预报准确的判别标准，再计算传统的评分方法；对于空间列联表，分别采用几个不同的空间搜索尺度（窗区），只要预报的事件在该搜索尺度内发生即判定预报准确。我们采用空间列联表的方法。

得到多事件列联表之后，可以针对不同的判别标准,计算命中率（POD）和虚警率（F）。也可以计算HK评分。值得注意的是，多事件列联表只对预报进行了尺度模糊处理，对观测并没有进行尺度模糊处理。

$$HK = POD - F \tag{8.3}$$

8.4.2 模糊检验结果分析

表 8.2 模糊检验结果

模型	**MSE**		**HK**	
数据	原始雷达	双偏振雷达	原始雷达	双偏振雷达
U-Net	78.913	41.616	0.654	0.736
WF-UNet	66.347	35.937	0.733	0.758
扩散模型	69.724	33.346	0.787	0.825
组合扩散模型 (平均集成)	43.369	22.168	**0.792**	0.857
组合扩散模型 (U-Net 集成)	**42.918**	**18.934**	0.784	**0.861**

可以看出，在计算模糊检验指标 HK 时，利用原始雷达数据预测的结果比精确检验指标 MSE 时更加接近双偏振雷达的预测结果，说明模糊检验对原始雷达数据的加成比双偏

振雷达数据更高。因此，说明了原始雷达数据进行强对流降水量预测会导致模糊问题。

8.5 数据融合策略

气象数据的例子微物理信息是大量的、多维的，且具有一定相关性的。然而，大量数据也给数据驱动的深度学习模型带来困难和挑战，传统的数据融合技术很难满足大数据时代的需求。因此设计可靠的数据融合策略，从大量数据中获取可靠、有价值和准确信息对强对流降水量预测具有重要意义。

8.5.1 数据融合架构

我们设计的数据融合架构包括三层，包括信号级数据融合、特征级数据融合、决策级特征融合。其中信号级特征融合的输入是数据信号，输出是数据信号或特征。特征级数据融合输入时特征，输出也是特征。决策级数据融合输入时信号或决策，输出是决策。具体架构如下图所示：

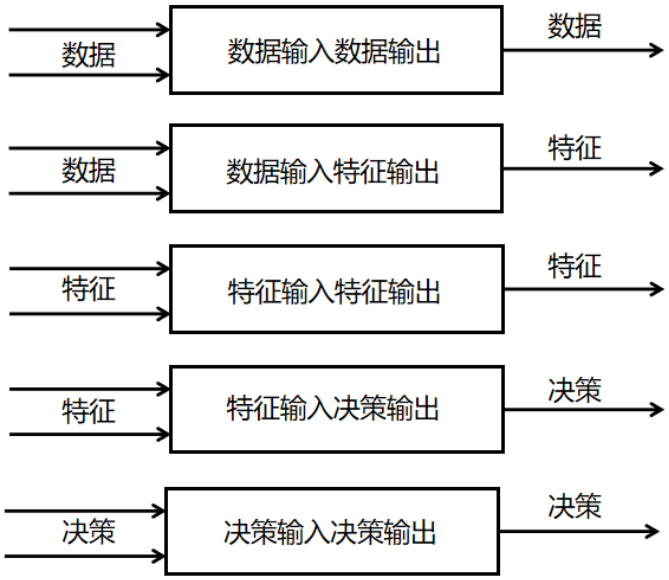


图 8.3 数据融合架构

8.5.2 信号级数据融合

我们首先在数据的层面进行融合，目的是结合一帧的所有微物理数据，给每一帧数据评估数据质量，再决定是否用此帧数据进行训练。这样可以剔除掉质量差的数据，例如全 0 数据，超过变量取值范围的错误数据等。极大程度上助力深度学习预测模型的训练。

具体来说，我们使用了支持向量机 SVM，支持向量机提供了合适的信号级融合函数。由于我们的例子微物理数据是非线性变化的，SVM 作为一种优秀的非线性模式识别工具，特别是在动态过程中，保证了错误数据检测的准确性和性能。与未融合的数据相比，分类精度和平均分类性能都有了提高。基于 SVM 的融合可以克服不完整数据的融合难题，分数归一化可进一步提升 SVM 的效率和鲁棒性。

我们将每一帧的$\{Z_H^{1km}, Z_{DR}^{1km}, Z_H^{3km}, Z_{DR}^{3km}, Z_H^{7km}, Z_{DR}^{7km}\}$输入至 SVM 中，SVM 输出一个标量得分，作为此帧数据的质量评分。训练集由 300 帧组成，由人工进行标注（标注标准为降雨格点越多，0 格点越少，格点间差异越大）。

8.5.3 特征级数据融合

在特征级数据融合中，数据输入既可以是数据，也可以特征；输出可以是细化的特征或决策。与信号级数据融合相比，此过程得到的信息更加精炼和全面。

在第二问中，我们为了缓解模糊效应问题，将原始粒子微物理数据、经过边缘提取的粒子微物理数据、经过锐化的粒子微物理数据输入至模型里，来克服模型偏置的问题。这个过程就属于特征级数据融合，融合了数据的边缘轮廓特征，以及梯度变化特征。将这些特征输入至后续的预测模型当中，帮助模型获取更多的信息，有助于最终的预测。具体方法见问题二多样特征提取。

8.5.4 决策级数据融合

决策级数据融合旨在进一步融合已生成的一些信息，得到某个任务的最终决策。因此决策级融合常常出现在最终决策做出之前。

在第二问和第三问中，我们都用到了决策级数据融合。在第二问中，我们利用了三个模型，输入不同的特征，得到了三个预测输出，最后根据 U-Net 集成这三个输出得到最终的预测。同样的在第三问中，我们利用扩散模型的不同分布预测得到三个不同的输出，最后根据 U-Net 集成这三个输出得到最终的预测。这种组合集成的方法可以提高模型的容错率，其中一个模型如果预测错误可以被其他两个模型纠正。因此，大大提高了模型的预测正确率。具体方法见问题二与问题三组合集成。

8.6 数据融合实验与分析

我们选扩散模型进行消融实验，探究三种融合策略对预测指标的影响。

表 8.3 数据融合对比试验

模型	MSE	MRE	RBIAS	CSI
扩散模型	33.346	0.289	0.022	0.893
+SVM (信号级数据融合)	+0.182	-0.070	+0.008	+0.014
+边缘、梯度特征 (特征级数据融合)	+4.694	+0.013	-0.007	+0.042
+U-Net 组合集成 (决策级数据融合)	+8.791	-0.083	-0.012	+0.054
+全架构数据融合	**+12.810**	**-0.104**	**-0.014**	**+0.060**

从表 8.3 可知，加入了数据融合策略后，无论是信号级融合、特征级融合还是决策级融合，都对模型的性能具有提升作用，全架构数据融合后，模型在各种指标上均有了不同幅度的提升，说明对于突发性和局地性强的强对流天气的数据多变性，我们的数据融合架构能够更好地应对。

9 模型评价

9.1 模型优点

（1）问题一中提出的基于残差卷积块的 U-Net 模型能够较好地捕获雷达观测数据的区域特征，并且能够将不同的雷达观测量、不同的帧进行融合，最终输出较为准确的水平反射率因子预测结果。

（2）问题二中提出的缓解“回归到平均”以及增强模型输出细节的措施，能够确实改善模型输出分布较平缓的问题，并且评价指标和预测的拟合度都相比改进前有明显提升，预测结果的细节更加充分真实。

（3）本文利用组合扩散模型进行降水量预测效果好于传统模型。在问题三中，我们设计了组合扩散策略，利用预测降水量变化的分布来预测降水量的值，保证了模型的泛化能力，提高了预测精度。

（4）本文提出了一种数据融合架构，即信号级、特征级、决策级数据融合，并设计了相应的方法进行实现，极大程度上保证了数据的融合性、可学性。

9.2 模型缺点

（1）问题二改进措施中的集成学习中，需要较多的人工设计，例如不同模型学习数据的哪些方面，如何提取数据的不同方面信息，因此泛用性不强。并且由于模型的增加而带来的整体性能提升不如训练成本的增加，使得该方法可拓展性受限。

（2）对于问题 3，由于扩散模型使预测其分布，具有随机性，因此我们在预测同一帧数据时，每次预测的值是不同的，不能保证结果的唯一性。

9.3 未来方向

（1）通过显式地建模不同帧的时序特征，设计能够提取和预测更长时间范围的模型，将可准确预测的时间拓展到一小时以上。

（2）在预测降水量问题中，扩散模型的去噪设计还可以改进，例如先把不同高度的数据融合，再把它输入去噪 U-Net 中来提示扩散模型逐步去噪。这样可以避免无法收敛的情况。

（3）在数据融合问题中，还存在许多更好的融合策略，如贝叶斯网络等。未来的工作可以设计实验，找到最适合强对流降水量临近预测的方法。进一步优化数据融合效果，进而对提高模型的预测能力。

参考文献

[1] 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

[2] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

[3] Glorot X, Bengio Y. Understanding the difficulty of training deep feedforward neural networks[C]//Proceedings of the thirteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings, 2010: 249-256.

[4] Kingma D , Ba J .Adam: A Method for Stochastic Optimization[J].Computer Science, 2014.DOI:10.48550/arXiv.1412.6980.

[5] Asperti A, Merizzi F, Paparella A, et al. Precipitation nowcasting with generative diffusion models[J]. arxiv preprint arxiv:2308.06733, 2023.

[6] 皇甫江,胡志群,郑佳锋,等.利用深度学习开展偏振雷达定量降水估测研究[J].气象学报, 2022(004):080.

[7] Zhang Y, Long M, Chen K, et al. Skilful nowcasting of extreme precipitation with NowcastNet[J]. Nature, 2023: 1-7.

附录 A 主要代码

A.1 一二问主要代码

（1）数据加载

```
import numpy as np

#导入 npy 文件路径位置
test = np.load('D:/储存/桌面/研究生建模/NJU_CPOL_update2308/dBZ/1.0km/data_dir_000/frame_022.npy')

print(test)
```

（2）FURENet

```
import torch
from torch import nn, Tensor

from ResidualConv2d import ResidualConv2d
from SEBlock import SEBlock

__all__ = ["FURENet"]


class FURENet(nn.Module):
    def __init__(self, in_channels: int, out_channels: int):
        super(FURENet, self).__init__()
        self.z_downSample_operate_1_to_2 = ResidualConv2d(in_channels=in_channels, out_channels=64)
        self.z_downSample_operate_2_to_3 = ResidualConv2d(in_channels=64, out_channels=128)
        self.z_downSample_operate_3_to_4 = ResidualConv2d(in_channels=128, out_channels=192)
        self.z_downSample_operate_4_to_5 = ResidualConv2d(in_channels=192, out_channels=256)
        self.z_downSample_operate_5_to_6 = ResidualConv2d(in_channels=256, out_channels=384)
        self.z_downSample_operate_6_to_7 = ResidualConv2d(in_channels=384, out_channels=512)

        self.zdr_downSample_operate_1_to_2 = ResidualConv2d(in_channels=in_channels, out_channels=64)
```

```
        self.zdr_downSample_operate_2_to_3 = ResidualConv2d(in_channels=64,
out_channels=128)
        self.zdr_downSample_operate_3_to_4 = ResidualConv2d(in_channels=128,
out_channels=192)
        self.zdr_downSample_operate_4_to_5 = ResidualConv2d(in_channels=192,
out_channels=256)
        self.zdr_downSample_operate_5_to_6 = ResidualConv2d(in_channels=256,
out_channels=384)
        self.zdr_downSample_operate_6_to_7 = ResidualConv2d(in_channels=384,
out_channels=512)

        self.kdp_downSample_operate_1_to_2 = ResidualConv2d(in_channels=in_channels,
out_channels=64)
        self.kdp_downSample_operate_2_to_3 = ResidualConv2d(in_channels=64,
out_channels=128)
        self.kdp_downSample_operate_3_to_4 = ResidualConv2d(in_channels=128,
out_channels=192)
        self.kdp_downSample_operate_4_to_5 = ResidualConv2d(in_channels=192,
out_channels=256)
        self.kdp_downSample_operate_5_to_6 = ResidualConv2d(in_channels=256,
out_channels=384)
        self.kdp_downSample_operate_6_to_7 = ResidualConv2d(in_channels=384,
out_channels=512)

        self.se_block = SEBlock(channels=512 * 3)

        self.upSample_7_to_6 = ResidualConv2d(in_channels=512 * 3, out_channels=384)
        self.upSample_6_to_5 = ResidualConv2d(in_channels=384 * 4, out_channels=256)
        self.upSample_5_to_4 = ResidualConv2d(in_channels=256 * 4, out_channels=192)
        self.upSample_4_to_3 = ResidualConv2d(in_channels=192 * 4, out_channels=128)
        self.upSample_3_to_2 = ResidualConv2d(in_channels=128 * 4, out_channels=64)
        self.upSample_2_to_1 = ResidualConv2d(in_channels=64 * 4,
out_channels=out_channels)
        for m in self.modules():
            if isinstance(m, nn.Conv2d) or isinstance(m, nn.ConvTranspose2d):
                if m.bias is not None:
                    m.bias.data.zero_()
                if m.bias is not None:
                    nn.init.xavier_uniform(m.weight)


    def forward(self, z: Tensor, zdr: Tensor, kdp: Tensor):
        z2 = self.z_downSample_operate_1_to_2(z)
        z3 = self.z_downSample_operate_2_to_3(z2)
```

```
        z4 = self.z_downSample_operate_3_to_4(z3)
        z5 = self.z_downSample_operate_4_to_5(z4)
        z6 = self.z_downSample_operate_5_to_6(z5)
        z7 = self.z_downSample_operate_6_to_7(z6)

        zdr2 = self.zdr_downSample_operate_1_to_2(zdr)
        zdr3 = self.zdr_downSample_operate_2_to_3(zdr2)
        zdr4 = self.zdr_downSample_operate_3_to_4(zdr3)
        zdr5 = self.zdr_downSample_operate_4_to_5(zdr4)
        zdr6 = self.zdr_downSample_operate_5_to_6(zdr5)
        zdr7 = self.zdr_downSample_operate_6_to_7(zdr6)

        kdp2 = self.kdp_downSample_operate_1_to_2(kdp)
        kdp3 = self.kdp_downSample_operate_2_to_3(kdp2)
        kdp4 = self.kdp_downSample_operate_3_to_4(kdp3)
        kdp5 = self.kdp_downSample_operate_4_to_5(kdp4)
        kdp6 = self.kdp_downSample_operate_5_to_6(kdp5)
        kdp7 = self.kdp_downSample_operate_6_to_7(kdp6)

        concat_7 = torch.cat([zdr7, kdp7, z7], dim=1)
        upSampleInput = self.se_block(concat_7)

        upSample6 = self.upSample_7_to_6(upSampleInput)
        concat_6 = torch.cat([zdr6, kdp6, z6, upSample6], dim=1)

        upSample5 = self.upSample_6_to_5(concat_6)
        concat_5 = torch.cat([zdr5, kdp5, z5, upSample5], dim=1)

        upSample4 = self.upSample_5_to_4(concat_5)
        concat_4 = torch.cat([zdr4, kdp4, z4, upSample4], dim=1)

        upSample3 = self.upSample_4_to_3(concat_4)
        concat_3 = torch.cat([zdr3, kdp3, z3, upSample3], dim=1)

        upSample2 = self.upSample_3_to_2(concat_3)
        concat_2 = torch.cat([zdr2, kdp2, z2, upSample2], dim=1)

        out = self.upSample_2_to_1(concat_2)

        return out


if __name__ == '__main__':
    z = torch.ones(3, 10, 256, 256)
```

```
    zdr = torch.ones(3, 10, 256, 256)
    kdp = torch.ones(3, 10, 256, 256)
    net = FURENet(10)
    r = net(z, zdr)
    print(r.shape)
```

（3）ResidualConv2d

```
from typing import Tuple

import torch
from torch import nn, Tensor

__all__ = ["ResidualConv2d"]


class ResidualConv2d(nn.Module):
    def __init__(self, in_channels: int, out_channels: int, kernel_size: Tuple[int] = (3, 3),
                 stride: Tuple[int] = (2, 2), padding: Tuple[int] = (1, 1)):
        super(ResidualConv2d, self).__init__()
        if in_channels <= out_channels:
            self.operate1 = nn.Sequential(
                nn.Conv2d(in_channels=in_channels, out_channels=out_channels,
                          kernel_size=kernel_size, stride=stride, padding=padding),
                nn.ReLU()
            )
        else:
            self.operate1 = nn.Sequential(
                nn.ConvTranspose2d(in_channels=in_channels, out_channels=out_channels,
                                   kernel_size=kernel_size, stride=stride,
padding=padding, output_padding=(1, 1)),
                nn.ReLU()
            )

        self.operate2 = nn.Sequential(
            nn.Conv2d(in_channels=out_channels, out_channels=out_channels,
kernel_size=(3, 3),
                      stride=(1, 1), padding=(1, 1)),
            nn.LeakyReLU(),
            nn.BatchNorm2d(out_channels),
            nn.Conv2d(in_channels=out_channels, out_channels=out_channels,
kernel_size=(3, 3),
                      stride=(1, 1), padding=(1, 1))
        )
```

```
    def forward(self, x1: Tensor):
        x2 = self.operate1(x1)
        x3 = self.operate2(x2)
        x3 = x2 + x3

        return x3


# if __name__ == '__main__':
#     a = torch.ones(3, 10, 256, 256)
#     net = ResidualConv2d(in_channels=10, out_channels=64)
#     r = net(a)
#     print(r.shape)
#
#     b = torch.ones(5, 512, 4, 4)
#     net2 = ResidualConv2d(in_channels=512, out_channels=256)
#     t = net2(b)
#     print(t.shape)
```

（4）SEBlock

```
from FURENet import FURENet
import os
import numpy as np
import random
import torch
from torch import nn, optim
from torch.utils.data import Dataset
from matplotlib import pyplot as plt
import seaborn as sns
sns.set()

class MyDataSet(Dataset):
    def __init__(self, data_path, height, data_ids) -> None:
        super(MyDataSet, self).__init__()

        frames = []
        for data_id in data_ids:
            pwd = os.path.join(data_path, 'dBZ', height, 'data_dir_' + data_id)
            files = os.listdir(pwd)
            files.sort()
            frames += [np.load(os.path.join(pwd, f)) for f in files]
        self.z_h = torch.tensor(np.stack(frames))
```

```
            frames = []
            for data_id in data_ids:
                pwd = os.path.join(data_path, 'ZDR', height, 'data_dir_' + data_id)
                files = os.listdir(pwd)
                files.sort()
                frames += [np.load(os.path.join(pwd, f)) for f in files]
            self.z_dr = torch.tensor(np.stack(frames))

            frames = []
            for data_id in data_ids:
                pwd = os.path.join(data_path, 'KDP', height, 'data_dir_' + data_id)
                files = os.listdir(pwd)
                files.sort()
                frames += [np.load(os.path.join(pwd, f)) for f in files]
            self.k_dp = torch.tensor(np.stack(frames))

        def __getitem__(self, index):
            return self.z_h[index: index + 10].unsqueeze(0), self.z_dr[index: index + 10].unsqueeze(0), self.k_dp[index: index + 10].unsqueeze(0), self.z_h[index + 10: index + 20].unsqueeze(0)

        def __len__(self):
            return self.z_h.size(0) - 20

    device = 'cuda'
    epoches = 60
    lr = 1e-4
    train_set = MyDataSet('./data/NJU_CPOL_update2308', '1.0km', [str(s).rjust(3, '0') for s in range(0, 3)])
    if os.path.exists('model.ckpt'):
        with open('model.ckpt', 'rb') as f:
            model = torch.load(f)
    else:
        model = FURENet(10, 10).to(device)

        criterion = nn.MSELoss()
        optimizer = optim.Adam(model.parameters(), lr=lr)

        losses = []
        for i in range(epoches):
            loss_train = 0.
            for j in range(len(train_set)):
                z_h, z_dr, k_dp, target = train_set[j]
```

```
                optimizer.zero_grad()
                out = model(z_h.to(device), z_dr.to(device), k_dp.to(device))
                loss = criterion(out, target.to(device))
                loss_train += loss.item()
                loss.backward()
                optimizer.step()
        loss_train /= len(train_set)
        print('epoch:', i, '\tloss:', loss_train)
    #torch.save(model, 'model.ckpt')

model.cpu()
randindex = random.randint(0, len(train_set) - 1)
z_h, z_dr, k_dp, target = train_set[randindex]
out = model(z_h, z_dr, k_dp).squeeze().detach()
target = target.squeeze().detach()

fig = plt.figure(figsize=(12, 6))
pred = out[-1].flatten()
pred = (pred-pred.min())/(pred.max()-pred.min())
true = target[-1].flatten()
true = (true-true.min())/(true.max()-true.min())
plt.subplot(121)
plt.hist(pred[pred>0.1].numpy(), bins=9)
plt.title('predict')
plt.ylabel('frequency')
plt.subplot(122)
plt.hist(true[true>0.1].numpy(), bins=9)
plt.title('truth')
fig.savefig('hist.png')

mse = ((out-target)**2).mean(-1).mean(-1)
mre = torch.abs(out-target).mean(-1).mean(-1)/target.sum(-1).sum(-1)
rbias = torch.abs(out-target).sum(-1).sum(-1)/target.sum(-1).sum(-1)
tp = ((out>0.35)&(target>0.35)).sum(-1).sum(-1)
fn = ((out<0.35)&(target>0.35)).sum(-1).sum(-1)
fp = ((out>0.35)&(target<0.35)).sum(-1).sum(-1)
csi = tp/(tp+fn+fp)

fig = plt.figure(figsize=(10, 6))
plt.plot(torch.arange(10)*6, mse.numpy())
plt.xlabel('Minutes')
plt.ylabel('MSE')
fig.savefig('mse.png')
fig = plt.figure(figsize=(10, 6))
```

```
plt.plot(torch.arange(10)*6, mre.numpy())
plt.xlabel('Minutes')
plt.ylabel('MRE')
fig.savefig('mre.png')
fig = plt.figure(figsize=(10, 6))
plt.plot(torch.arange(10)*6, rbias.numpy())
plt.xlabel('Minutes')
plt.ylabel('RBIAS')
fig.savefig('rbias.png')
fig = plt.figure(figsize=(10, 6))
plt.plot(torch.arange(10)*6, csi.numpy())
plt.xlabel('Minutes')
plt.ylabel('CSI')
fig.savefig('csi.png')

fig = plt.figure(figsize=(40, 16))
ax = plt.subplot(2, 5, 1)
ax.set_ylabel('predicted')
ax = plt.subplot(2, 5, 6)
ax.set_ylabel('truth')
for i in range(5):
    ax = plt.subplot(2, 5, i + 1)
    ax.set_title('frame %d'%(i*2 + 1))
    plt.imshow(out[i*2].numpy())
    plt.subplot(2, 5, i + 6)
    plt.imshow(target[i*2].numpy())
fig.savefig('figure.png')
```

A.2 三四问主要代码

（1）扩散模型主函数

```
import argparse, os, sys, datetime, glob, importlib, csv
import numpy as np
import time
import torch
import torchvision
import pytorch_lightning as pl

from packaging import version
from omegaconf import OmegaConf
from torch.utils.data import random_split, DataLoader, Dataset, Subset
from functools import partial
from PIL import Image
```

```
from pytorch_lightning import seed_everything
from pytorch_lightning.trainer import Trainer
from pytorch_lightning.callbacks import ModelCheckpoint, Callback, LearningRateMonitor
from pytorch_lightning.utilities.distributed import rank_zero_only
from pytorch_lightning.utilities import rank_zero_info

from ldm.data.base import Txt2ImgIterableBaseDataset
from ldm.util import instantiate_from_config


def get_parser(**parser_kwargs):
    def str2bool(v):
        if isinstance(v, bool):
            return v
        if v.lower() in ("yes", "true", "t", "y", "1"):
            return True
        elif v.lower() in ("no", "false", "f", "n", "0"):
            return False
        else:
            raise argparse.ArgumentTypeError("Boolean value expected.")

    parser = argparse.ArgumentParser(**parser_kwargs)
    parser.add_argument(
        "-n",
        "--name",
        type=str,
        const=True,
        default="",
        nargs="?",
        help="postfix for logdir",
    )
    parser.add_argument(
        "-r",
        "--resume",
        type=str,
        const=True,
        default="",
        nargs="?",
        help="resume from logdir or checkpoint in logdir",
    )
    parser.add_argument(
        "-b",
        "--base",
```

```
            nargs="*",
            metavar="base_config.yaml",
            help="paths to base configs. Loaded from left-to-right. "
                 "Parameters can be overwritten or added with command-line options of the form
`--key value`.",
            default=list(),
        )
        parser.add_argument(
            "-t",
            "--train",
            type=str2bool,
            const=True,
            default=False,
            nargs="?",
            help="train",
        )
        parser.add_argument(
            "--no-test",
            type=str2bool,
            const=True,
            default=False,
            nargs="?",
            help="disable test",
        )
        parser.add_argument(
            "-p",
            "--project",
            help="name of new or path to existing project"
        )
        parser.add_argument(
            "-d",
            "--debug",
            type=str2bool,
            nargs="?",
            const=True,
            default=False,
            help="enable post-mortem debugging",
        )
        parser.add_argument(
            "-s",
            "--seed",
            type=int,
            default=23,
            help="seed for seed_everything",
```

```
    )
    parser.add_argument(
        "-f",
        "--postfix",
        type=str,
        default="",
        help="post-postfix for default name",
    )
    parser.add_argument(
        "-l",
        "--logdir",
        type=str,
        default="logs",
        help="directory for logging dat shit",
    )
    parser.add_argument(
        "--scale_lr",
        type=str2bool,
        nargs="?",
        const=True,
        default=True,
        help="scale base-lr by ngpu * batch_size * n_accumulate",
    )
    return parser


def nondefault_trainer_args(opt):
    parser = argparse.ArgumentParser()
    parser = Trainer.add_argparse_args(parser)
    args = parser.parse_args([])
    return sorted(k for k in vars(args) if getattr(opt, k) != getattr(args, k))


class WrappedDataset(Dataset):

    def __init__(self, dataset):
        self.data = dataset

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        return self.data[idx]
```

```
def worker_init_fn(_):
    worker_info = torch.utils.data.get_worker_info()

    dataset = worker_info.dataset
    worker_id = worker_info.id

    if isinstance(dataset, Txt2ImgIterableBaseDataset):
        split_size = dataset.num_records // worker_info.num_workers
        dataset.sample_ids = dataset.valid_ids[worker_id * split_size:(worker_id + 1) *
split_size]
        current_id = np.random.choice(len(np.random.get_state()[1]), 1)
        return np.random.seed(np.random.get_state()[1][current_id] + worker_id)
    else:
        return np.random.seed(np.random.get_state()[1][0] + worker_id)


class DataModuleFromConfig(pl.LightningDataModule):
    def __init__(self, batch_size, train=None, validation=None, test=None, predict=None,
                 wrap=False, num_workers=None, shuffle_test_loader=False,
use_worker_init_fn=False,
                 shuffle_val_dataloader=False):
        super().__init__()
        self.batch_size = batch_size
        self.dataset_configs = dict()
        self.num_workers = num_workers if num_workers is not None else batch_size * 2
        self.use_worker_init_fn = use_worker_init_fn
        if train is not None:
            self.dataset_configs["train"] = train
            self.train_dataloader = self._train_dataloader
        if validation is not None:
            self.dataset_configs["validation"] = validation
            self.val_dataloader = partial(self._val_dataloader, shuffle=shuffle_val_dataloader)
        if test is not None:
            self.dataset_configs["test"] = test
            self.test_dataloader = partial(self._test_dataloader, shuffle=shuffle_test_loader)
        if predict is not None:
            self.dataset_configs["predict"] = predict
            self.predict_dataloader = self._predict_dataloader
        self.wrap = wrap

    def prepare_data(self):
        for data_cfg in self.dataset_configs.values():
            instantiate_from_config(data_cfg)
```

```
    def setup(self, stage=None):
        self.datasets = dict(
            (k, instantiate_from_config(self.dataset_configs[k]))
            for k in self.dataset_configs)
        if self.wrap:
            for k in self.datasets:
                self.datasets[k] = WrappedDataset(self.datasets[k])

    def _train_dataloader(self):
        is_iterable_dataset = isinstance(self.datasets['train'], Txt2ImgIterableBaseDataset)
        if is_iterable_dataset or self.use_worker_init_fn:
            init_fn = worker_init_fn
        else:
            init_fn = None
        return DataLoader(self.datasets["train"], batch_size=self.batch_size,
                          num_workers=self.num_workers, shuffle=False if
is_iterable_dataset else True,
                          worker_init_fn=init_fn)

    def _val_dataloader(self, shuffle=False):
        if isinstance(self.datasets['validation'], Txt2ImgIterableBaseDataset) or
self.use_worker_init_fn:
            init_fn = worker_init_fn
        else:
            init_fn = None
        return DataLoader(self.datasets["validation"],
                          batch_size=self.batch_size,
                          num_workers=self.num_workers,
                          worker_init_fn=init_fn,
                          shuffle=shuffle)

    def _test_dataloader(self, shuffle=False):
        is_iterable_dataset = isinstance(self.datasets['train'], Txt2ImgIterableBaseDataset)
        if is_iterable_dataset or self.use_worker_init_fn:
            init_fn = worker_init_fn
        else:
            init_fn = None

        shuffle = shuffle and (not is_iterable_dataset)

        return DataLoader(self.datasets["test"], batch_size=self.batch_size,
                          num_workers=self.num_workers, worker_init_fn=init_fn,
shuffle=shuffle)
```

```
    def _predict_dataloader(self, shuffle=False):
        if isinstance(self.datasets['predict'], Txt2ImgIterableBaseDataset) or
self.use_worker_init_fn:
            init_fn = worker_init_fn
        else:
            init_fn = None
        return DataLoader(self.datasets["predict"], batch_size=self.batch_size,
                          num_workers=self.num_workers, worker_init_fn=init_fn)


class SetupCallback(Callback):
    def __init__(self, resume, now, logdir, ckptdir, cfgdir, config, lightning_config):
        super().__init__()
        self.resume = resume
        self.now = now
        self.logdir = logdir
        self.ckptdir = ckptdir
        self.cfgdir = cfgdir
        self.config = config
        self.lightning_config = lightning_config

    def on_keyboard_interrupt(self, trainer, pl_module):
        if trainer.global_rank == 0:
            print("Summoning checkpoint.")
            ckpt_path = os.path.join(self.ckptdir, "last.ckpt")
            trainer.save_checkpoint(ckpt_path)

    def on_pretrain_routine_start(self, trainer, pl_module):
        if trainer.global_rank == 0:
            os.makedirs(self.logdir, exist_ok=True)
            os.makedirs(self.ckptdir, exist_ok=True)
            os.makedirs(self.cfgdir, exist_ok=True)

            if "callbacks" in self.lightning_config:
                if 'metrics_over_trainsteps_checkpoint' in self.lightning_config['callbacks']:
                    os.makedirs(os.path.join(self.ckptdir, 'trainstep_checkpoints'),
exist_ok=True)
            print("Project config")
            print(OmegaConf.to_yaml(self.config))
            OmegaConf.save(self.config,
                           os.path.join(self.cfgdir, "{}-project.yaml".format(self.now)))

            print("Lightning config")
            print(OmegaConf.to_yaml(self.lightning_config))
```

```
            OmegaConf.save(OmegaConf.create({"lightning": self.lightning_config}),
                           os.path.join(self.cfgdir, "{}-lightning.yaml".format(self.now)))

        else:
            if not self.resume and os.path.exists(self.logdir):
                dst, name = os.path.split(self.logdir)
                dst = os.path.join(dst, "child_runs", name)
                os.makedirs(os.path.split(dst)[0], exist_ok=True)
                try:
                    os.rename(self.logdir, dst)
                except FileNotFoundError:
                    pass


class ImageLogger(Callback):
    def __init__(self, batch_frequency, max_images, clamp=True, increase_log_steps=True,
                 rescale=True, disabled=False, log_on_batch_idx=False, log_first_step=False,
                 log_images_kwargs=None):
        super().__init__()
        self.rescale = rescale
        self.batch_freq = batch_frequency
        self.max_images = max_images
        self.logger_log_images = {
            pl.loggers.TestTubeLogger: self._testtube,
        }
        self.log_steps = [2 ** n for n in range(int(np.log2(self.batch_freq)) + 1)]
        if not increase_log_steps:
            self.log_steps = [self.batch_freq]
        self.clamp = clamp
        self.disabled = disabled
        self.log_on_batch_idx = log_on_batch_idx
        self.log_images_kwargs = log_images_kwargs if log_images_kwargs else {}
        self.log_first_step = log_first_step

    @rank_zero_only
    def _testtube(self, pl_module, images, batch_idx, split):
        for k in images:
            grid = torchvision.utils.make_grid(images[k])
            grid = (grid + 1.0) / 2.0  # -1,1 -> 0,1; c,h,w

            tag = f"{split}/{k}"
            pl_module.logger.experiment.add_image(
                tag, grid,
```

```
                global_step=pl_module.global_step)

    @rank_zero_only
    def log_local(self, save_dir, split, images,
                  global_step, current_epoch, batch_idx):
        root = os.path.join(save_dir, "images", split)
        for k in images:
            grid = torchvision.utils.make_grid(images[k], nrow=4)
            if self.rescale:
                grid = (grid + 1.0) / 2.0    # -1,1 -> 0,1; c,h,w
            grid = grid.transpose(0, 1).transpose(1, 2).squeeze(-1)
            grid = grid.numpy()
            grid = (grid * 255).astype(np.uint8)
            filename = "{}_gs-{:06}_e-{:06}_b-{:06}.png".format(
                k,
                global_step,
                current_epoch,
                batch_idx)
            path = os.path.join(root, filename)
            os.makedirs(os.path.split(path)[0], exist_ok=True)
            Image.fromarray(grid).save(path)

    def log_img(self, pl_module, batch, batch_idx, split="train"):
        check_idx = batch_idx if self.log_on_batch_idx else pl_module.global_step
        if (self.check_frequency(check_idx) and    # batch_idx % self.batch_freq == 0
                hasattr(pl_module, "log_images") and
                callable(pl_module.log_images) and
                self.max_images > 0):
            logger = type(pl_module.logger)

            is_train = pl_module.training
            if is_train:
                pl_module.eval()

            with torch.no_grad():
                images = pl_module.log_images(batch, split=split, **self.log_images_kwargs)

            for k in images:
                N = min(images[k].shape[0], self.max_images)
                images[k] = images[k][:N]
                if isinstance(images[k], torch.Tensor):
                    images[k] = images[k].detach().cpu()
                    if self.clamp:
                        images[k] = torch.clamp(images[k], -1., 1.)
```

```
                self.log_local(pl_module.logger.save_dir, split, images,
                               pl_module.global_step, pl_module.current_epoch, batch_idx)

                logger_log_images = self.logger_log_images.get(logger, lambda *args, **kwargs:
None)
                logger_log_images(pl_module, images, pl_module.global_step, split)

                if is_train:
                    pl_module.train()

        def check_frequency(self, check_idx):
            if ((check_idx % self.batch_freq) == 0 or (check_idx in self.log_steps)) and (
                    check_idx > 0 or self.log_first_step):
                try:
                    self.log_steps.pop(0)
                except IndexError as e:
                    print(e)
                    pass
                return True
            return False

        def on_train_batch_end(self, trainer, pl_module, outputs, batch, batch_idx, dataloader_idx):
            if not self.disabled and (pl_module.global_step > 0 or self.log_first_step):
                self.log_img(pl_module, batch, batch_idx, split="train")

        def on_validation_batch_end(self, trainer, pl_module, outputs, batch, batch_idx,
dataloader_idx):
            if not self.disabled and pl_module.global_step > 0:
                self.log_img(pl_module, batch, batch_idx, split="val")
            if hasattr(pl_module, 'calibrate_grad_norm'):
                if (pl_module.calibrate_grad_norm and batch_idx % 25 == 0) and batch_idx > 0:
                    self.log_gradients(trainer, pl_module, batch_idx=batch_idx)


class CUDACallback(Callback):

        def on_train_epoch_start(self, trainer, pl_module):
            torch.cuda.reset_peak_memory_stats(trainer.root_gpu)
            torch.cuda.synchronize(trainer.root_gpu)
            self.start_time = time.time()

        def on_train_epoch_end(self, trainer, pl_module, outputs):
            torch.cuda.synchronize(trainer.root_gpu)
```

```
            max_memory = torch.cuda.max_memory_allocated(trainer.root_gpu) / 2 ** 20
            epoch_time = time.time() - self.start_time

            try:
                max_memory = trainer.training_type_plugin.reduce(max_memory)
                epoch_time = trainer.training_type_plugin.reduce(epoch_time)

                rank_zero_info(f"Average Epoch time: {epoch_time:.2f} seconds")
                rank_zero_info(f"Average Peak memory {max_memory:.2f}MiB")
            except AttributeError:
                pass


if __name__ == "__main__":

    now = datetime.datetime.now().strftime("%Y-%m-%dT%H-%M-%S")
    sys.path.append(os.getcwd())
    parser = get_parser()
    parser = Trainer.add_argparse_args(parser)
    opt, unknown = parser.parse_known_args()
    if opt.name and opt.resume:
        raise ValueError(
            "-n/--name and -r/--resume cannot be specified both."
            "If you want to resume training in a new log folder, "
            "use -n/--name in combination with --resume_from_checkpoint"
        )
    if opt.resume:
        if not os.path.exists(opt.resume):
            raise ValueError("Cannot find {}".format(opt.resume))
        if os.path.isfile(opt.resume):
            paths = opt.resume.split("/")
            logdir = "/".join(paths[:-2])
            ckpt = opt.resume
        else:
            assert os.path.isdir(opt.resume), opt.resume
            logdir = opt.resume.rstrip("/")
            ckpt = os.path.join(logdir, "checkpoints", "last.ckpt")

        opt.resume_from_checkpoint = ckpt
        base_configs = sorted(glob.glob(os.path.join(logdir, "configs/*.yaml")))
        opt.base = base_configs + opt.base
        _tmp = logdir.split("/")
        nowname = _tmp[-1]
    else:
```

```
        if opt.name:
            name = "_" + opt.name
        elif opt.base:
            cfg_fname = os.path.split(opt.base[0])[-1]
            cfg_name = os.path.splitext(cfg_fname)[0]
            name = "_" + cfg_name
        else:
            name = ""
        nowname = now + name + opt.postfix
        logdir = os.path.join(opt.logdir, nowname)

    ckptdir = os.path.join(logdir, "checkpoints")
    cfgdir = os.path.join(logdir, "configs")
    seed_everything(opt.seed)

    try:
        configs = [OmegaConf.load(cfg) for cfg in opt.base]
        cli = OmegaConf.from_dotlist(unknown)
        config = OmegaConf.merge(*configs, cli)
        lightning_config = config.pop("lightning", OmegaConf.create())
        trainer_config = lightning_config.get("trainer", OmegaConf.create())
        trainer_config["accelerator"] = "ddp"
        for k in nondefault_trainer_args(opt):
            trainer_config[k] = getattr(opt, k)
        if not "gpus" in trainer_config:
            del trainer_config["accelerator"]
            cpu = True
        else:
            gpuinfo = trainer_config["gpus"]
            print(f"Running on GPUs {gpuinfo}")
            cpu = False
        trainer_opt = argparse.Namespace(**trainer_config)
        lightning_config.trainer = trainer_config

        model = instantiate_from_config(config.model)
        trainer_kwargs = dict()
        default_logger_cfgs = {
            "wandb": {
                "target": "pytorch_lightning.loggers.WandbLogger",
                "params": {
                    "name": nowname,
                    "save_dir": logdir,
                    "offline": opt.debug,
                    "id": nowname,
```

```
            }
        },
        "testtube": {
            "target": "pytorch_lightning.loggers.TestTubeLogger",
            "params": {
                "name": "testtube",
                "save_dir": logdir,
            }
        },
}
default_logger_cfg = default_logger_cfgs["testtube"]
if "logger" in lightning_config:
    logger_cfg = lightning_config.logger
else:
    logger_cfg = OmegaConf.create()
logger_cfg = OmegaConf.merge(default_logger_cfg, logger_cfg)
trainer_kwargs["logger"] = instantiate_from_config(logger_cfg)

default_modelckpt_cfg = {
    "target": "pytorch_lightning.callbacks.ModelCheckpoint",
    "params": {
        "dirpath": ckptdir,
        "filename": "{epoch:06}",
        "verbose": True,
        "save_last": True,
    }
}
if hasattr(model, "monitor"):
    print(f"Monitoring {model.monitor} as checkpoint metric.")
    default_modelckpt_cfg["params"]["monitor"] = model.monitor
    default_modelckpt_cfg["params"]["save_top_k"] = 3

if "modelcheckpoint" in lightning_config:
    modelckpt_cfg = lightning_config.modelcheckpoint
else:
    modelckpt_cfg =  OmegaConf.create()
modelckpt_cfg = OmegaConf.merge(default_modelckpt_cfg, modelckpt_cfg)
print(f'Merged modelckpt-cfg: \n{modelckpt_cfg}')
if version.parse(pl.__version__) < version.parse('1.4.0'):
    trainer_kwargs["checkpoint_callback"] = instantiate_from_config(modelckpt_cfg)

default_callbacks_cfg = {
    "setup_callback": {
        "target": "main.SetupCallback",
```

```
                "params": {
                    "resume": opt.resume,
                    "now": now,
                    "logdir": logdir,
                    "ckptdir": ckptdir,
                    "cfgdir": cfgdir,
                    "config": config,
                    "lightning_config": lightning_config,
                }
            },
            "image_logger": {
                "target": "main.ImageLogger",
                "params": {
                    "batch_frequency": 750,
                    "max_images": 4,
                    "clamp": True
                }
            },
            "learning_rate_logger": {
                "target": "main.LearningRateMonitor",
                "params": {
                    "logging_interval": "step",
                    # "log_momentum": True
                }
            },
            "cuda_callback": {
                "target": "main.CUDACallback"
            },
        }
        if version.parse(pl.__version__) >= version.parse('1.4.0'):
            default_callbacks_cfg.update({'checkpoint_callback': modelckpt_cfg})

        if "callbacks" in lightning_config:
            callbacks_cfg = lightning_config.callbacks
        else:
            callbacks_cfg = OmegaConf.create()

        if 'metrics_over_trainsteps_checkpoint' in callbacks_cfg:
            print(
                'Caution: Saving checkpoints every n train steps without deleting. This might require some free space.')
            default_metrics_over_trainsteps_ckpt_dict = {
                'metrics_over_trainsteps_checkpoint':
                    {"target": 'pytorch_lightning.callbacks.ModelCheckpoint',
```

```
                    'params': {
                        "dirpath": os.path.join(ckptdir, 'trainstep_checkpoints'),
                        "filename": "{epoch:06}-{step:09}",
                        "verbose": True,
                        'save_top_k': -1,
                        'every_n_train_steps': 10000,
                        'save_weights_only': True
                    }
                    }
            }
            default_callbacks_cfg.update(default_metrics_over_trainsteps_ckpt_dict)

        callbacks_cfg = OmegaConf.merge(default_callbacks_cfg, callbacks_cfg)
        if 'ignore_keys_callback' in callbacks_cfg and hasattr(trainer_opt, 'resume_from_checkpoint'):
            callbacks_cfg.ignore_keys_callback.params['ckpt_path'] = trainer_opt.resume_from_checkpoint
        elif 'ignore_keys_callback' in callbacks_cfg:
            del callbacks_cfg['ignore_keys_callback']

        trainer_kwargs["callbacks"] = [instantiate_from_config(callbacks_cfg[k]) for k in callbacks_cfg]

        trainer = Trainer.from_argparse_args(trainer_opt, **trainer_kwargs)
        trainer.logdir = logdir

        data = instantiate_from_config(config.data)

        data.prepare_data()
        data.setup()
        print("#### Data #####")
        for k in data.datasets:
            print(f"{k}, {data.datasets[k].__class__.__name__}, {len(data.datasets[k])}")

        bs, base_lr = config.data.params.batch_size, config.model.base_learning_rate
        if not cpu:
            ngpu = len(lightning_config.trainer.gpus.strip(",").split(','))
        else:
            ngpu = 1
        if 'accumulate_grad_batches' in lightning_config.trainer:
            accumulate_grad_batches = lightning_config.trainer.accumulate_grad_batches
        else:
            accumulate_grad_batches = 1
        print(f"accumulate_grad_batches = {accumulate_grad_batches}")
```

```
        lightning_config.trainer.accumulate_grad_batches = accumulate_grad_batches
        if opt.scale_lr:
            model.learning_rate = accumulate_grad_batches * ngpu * bs * base_lr
            print(
                "Setting learning rate to {:.2e} = {} (accumulate_grad_batches) * {}
(num_gpus) * {} (batchsize) * {:.2e} (base_lr)".format(
                    model.learning_rate, accumulate_grad_batches, ngpu, bs, base_lr))
        else:
            model.learning_rate = base_lr
            print("++++ NOT USING LR SCALING ++++")
            print(f"Setting learning rate to {model.learning_rate:.2e}")

        def melk(*args, **kwargs):
            if trainer.global_rank == 0:
                print("Summoning checkpoint.")
                ckpt_path = os.path.join(ckptdir, "last.ckpt")
                trainer.save_checkpoint(ckpt_path)

        def divein(*args, **kwargs):
            if trainer.global_rank == 0:
                import pudb;
                pudb.set_trace()

        import signal
        signal.signal(signal.SIGUSR1, melk)
        signal.signal(signal.SIGUSR2, divein)

        if opt.train:
            try:
                trainer.fit(model, data)
            except Exception:
                melk()
                raise
        if not opt.no_test and not trainer.interrupted:
            trainer.test(model, data)
    except Exception:
        if opt.debug and trainer.global_rank == 0:
            try:
                import pudb as debugger
            except ImportError:
                import pdb as debugger
            debugger.post_mortem()
        raise
    finally:
```

```
        if opt.debug and not opt.resume and trainer.global_rank == 0:
            dst, name = os.path.split(logdir)
            dst = os.path.join(dst, "debug_runs", name)
            os.makedirs(os.path.split(dst)[0], exist_ok=True)
            os.rename(logdir, dst)
        if trainer.global_rank == 0:
            print(trainer.profiler.summary())
```

（2）扩散部分

```
import math
import torch
import torch.nn as nn
import numpy as np
from einops import rearrange

from ldm.util import instantiate_from_config
from ldm.modules.attention import LinearAttention


def get_timestep_embedding(timesteps, embedding_dim):
    assert len(timesteps.shape) == 1

    half_dim = embedding_dim // 2
    emb = math.log(10000) / (half_dim - 1)
    emb = torch.exp(torch.arange(half_dim, dtype=torch.float32) * -emb)
    emb = emb.to(device=timesteps.device)
    emb = timesteps.float()[:, None] * emb[None, :]
    emb = torch.cat([torch.sin(emb), torch.cos(emb)], dim=1)
    if embedding_dim % 2 == 1:
        emb = torch.nn.functional.pad(emb, (0,1,0,0))
    return emb


def nonlinearity(x):
    return x*torch.sigmoid(x)


def Normalize(in_channels, num_groups=32):
    return torch.nn.GroupNorm(num_groups=num_groups, num_channels=in_channels,
eps=1e-6, affine=True)
```

```
class Upsample(nn.Module):
    def __init__(self, in_channels, with_conv):
        super().__init__()
        self.with_conv = with_conv
        if self.with_conv:
            self.conv = torch.nn.Conv2d(in_channels,
                                        in_channels,
                                        kernel_size=3,
                                        stride=1,
                                        padding=1)

    def forward(self, x):
        x = torch.nn.functional.interpolate(x, scale_factor=2.0, mode="nearest")
        if self.with_conv:
            x = self.conv(x)
        return x


class Downsample(nn.Module):
    def __init__(self, in_channels, with_conv):
        super().__init__()
        self.with_conv = with_conv
        if self.with_conv:
            self.conv = torch.nn.Conv2d(in_channels,
                                        in_channels,
                                        kernel_size=3,
                                        stride=2,
                                        padding=0)

    def forward(self, x):
        if self.with_conv:
            pad = (0,1,0,1)
            x = torch.nn.functional.pad(x, pad, mode="constant", value=0)
            x = self.conv(x)
        else:
            x = torch.nn.functional.avg_pool2d(x, kernel_size=2, stride=2)
        return x


class ResnetBlock(nn.Module):
    def __init__(self, *, in_channels, out_channels=None, conv_shortcut=False,
                 dropout, temb_channels=512):
        super().__init__()
        self.in_channels = in_channels
```

```
        out_channels = in_channels if out_channels is None else out_channels
        self.out_channels = out_channels
        self.use_conv_shortcut = conv_shortcut

        self.norm1 = Normalize(in_channels)
        self.conv1 = torch.nn.Conv2d(in_channels,
                                     out_channels,
                                     kernel_size=3,
                                     stride=1,
                                     padding=1)
        if temb_channels > 0:
            self.temb_proj = torch.nn.Linear(temb_channels,
                                             out_channels)
        self.norm2 = Normalize(out_channels)
        self.dropout = torch.nn.Dropout(dropout)
        self.conv2 = torch.nn.Conv2d(out_channels,
                                     out_channels,
                                     kernel_size=3,
                                     stride=1,
                                     padding=1)
        if self.in_channels != self.out_channels:
            if self.use_conv_shortcut:
                self.conv_shortcut = torch.nn.Conv2d(in_channels,
                                                     out_channels,
                                                     kernel_size=3,
                                                     stride=1,
                                                     padding=1)
            else:
                self.nin_shortcut = torch.nn.Conv2d(in_channels,
                                                    out_channels,
                                                    kernel_size=1,
                                                    stride=1,
                                                    padding=0)

    def forward(self, x, temb):
        h = x
        h = self.norm1(h)
        h = nonlinearity(h)
        h = self.conv1(h)

        if temb is not None:
            h = h + self.temb_proj(nonlinearity(temb))[:,:,None,None]

        h = self.norm2(h)
```

```
        h = nonlinearity(h)
        h = self.dropout(h)
        h = self.conv2(h)

        if self.in_channels != self.out_channels:
            if self.use_conv_shortcut:
                x = self.conv_shortcut(x)
            else:
                x = self.nin_shortcut(x)

        return x+h


class LinAttnBlock(LinearAttention):
    def __init__(self, in_channels):
        super().__init__(dim=in_channels, heads=1, dim_head=in_channels)


class AttnBlock(nn.Module):
    def __init__(self, in_channels):
        super().__init__()
        self.in_channels = in_channels

        self.norm = Normalize(in_channels)
        self.q = torch.nn.Conv2d(in_channels,
                                 in_channels,
                                 kernel_size=1,
                                 stride=1,
                                 padding=0)
        self.k = torch.nn.Conv2d(in_channels,
                                 in_channels,
                                 kernel_size=1,
                                 stride=1,
                                 padding=0)
        self.v = torch.nn.Conv2d(in_channels,
                                 in_channels,
                                 kernel_size=1,
                                 stride=1,
                                 padding=0)
        self.proj_out = torch.nn.Conv2d(in_channels,
                                        in_channels,
                                        kernel_size=1,
                                        stride=1,
                                        padding=0)
```

```
    def forward(self, x):
        h_ = x
        h_ = self.norm(h_)
        q = self.q(h_)
        k = self.k(h_)
        v = self.v(h_)

        b,c,h,w = q.shape
        q = q.reshape(b,c,h*w)
        q = q.permute(0,2,1)      # b,hw,c
        k = k.reshape(b,c,h*w) # b,c,hw
        w_ = torch.bmm(q,k)         # b,hw,hw        w[b,i,j]=sum_c q[b,i,c]k[b,c,j]
        w_ = w_ * (int(c)**(-0.5))
        w_ = torch.nn.functional.softmax(w_, dim=2)

        v = v.reshape(b,c,h*w)
        w_ = w_.permute(0,2,1)      # b,hw,hw (first hw of k, second of q)
        h_ = torch.bmm(v,w_)         # b, c,hw (hw of q) h_[b,c,j] = sum_i v[b,c,i] w_[b,i,j]
        h_ = h_.reshape(b,c,h,w)

        h_ = self.proj_out(h_)

        return x+h_


def make_attn(in_channels, attn_type="vanilla"):
    assert attn_type in ["vanilla", "linear", "none"], f'attn_type {attn_type} unknown'
    print(f"making attention of type '{attn_type}' with {in_channels} in_channels")
    if attn_type == "vanilla":
        return AttnBlock(in_channels)
    elif attn_type == "none":
        return nn.Identity(in_channels)
    else:
        return LinAttnBlock(in_channels)


class Model(nn.Module):
    def __init__(self, *, ch, out_ch, ch_mult=(1,2,4,8), num_res_blocks,
                 attn_resolutions, dropout=0.0, resamp_with_conv=True, in_channels,
                 resolution, use_timestep=True, use_linear_attn=False, attn_type="vanilla"):
        super().__init__()
        if use_linear_attn: attn_type = "linear"
```

```
self.ch = ch
self.temb_ch = self.ch*4
self.num_resolutions = len(ch_mult)
self.num_res_blocks = num_res_blocks
self.resolution = resolution
self.in_channels = in_channels

self.use_timestep = use_timestep
if self.use_timestep:
    self.temb = nn.Module()
    self.temb.dense = nn.ModuleList([
        torch.nn.Linear(self.ch,
                        self.temb_ch),
        torch.nn.Linear(self.temb_ch,
                        self.temb_ch),
    ])

self.conv_in = torch.nn.Conv2d(in_channels,
                               self.ch,
                               kernel_size=3,
                               stride=1,
                               padding=1)

curr_res = resolution
in_ch_mult = (1,)+tuple(ch_mult)
self.down = nn.ModuleList()
for i_level in range(self.num_resolutions):
    block = nn.ModuleList()
    attn = nn.ModuleList()
    block_in = ch*in_ch_mult[i_level]
    block_out = ch*ch_mult[i_level]
    for i_block in range(self.num_res_blocks):
        block.append(ResnetBlock(in_channels=block_in,
                                 out_channels=block_out,
                                 temb_channels=self.temb_ch,
                                 dropout=dropout))
        block_in = block_out
        if curr_res in attn_resolutions:
            attn.append(make_attn(block_in, attn_type=attn_type))
    down = nn.Module()
    down.block = block
    down.attn = attn
    if i_level != self.num_resolutions-1:
        down.downsample = Downsample(block_in, resamp_with_conv)
```

```
            curr_res = curr_res // 2
        self.down.append(down)

    self.mid = nn.Module()
    self.mid.block_1 = ResnetBlock(in_channels=block_in,
                                   out_channels=block_in,
                                   temb_channels=self.temb_ch,
                                   dropout=dropout)
    self.mid.attn_1 = make_attn(block_in, attn_type=attn_type)
    self.mid.block_2 = ResnetBlock(in_channels=block_in,
                                   out_channels=block_in,
                                   temb_channels=self.temb_ch,
                                   dropout=dropout)

    self.up = nn.ModuleList()
    for i_level in reversed(range(self.num_resolutions)):
        block = nn.ModuleList()
        attn = nn.ModuleList()
        block_out = ch*ch_mult[i_level]
        skip_in = ch*ch_mult[i_level]
        for i_block in range(self.num_res_blocks+1):
            if i_block == self.num_res_blocks:
                skip_in = ch*in_ch_mult[i_level]
            block.append(ResnetBlock(in_channels=block_in+skip_in,
                                     out_channels=block_out,
                                     temb_channels=self.temb_ch,
                                     dropout=dropout))
            block_in = block_out
            if curr_res in attn_resolutions:
                attn.append(make_attn(block_in, attn_type=attn_type))
        up = nn.Module()
        up.block = block
        up.attn = attn
        if i_level != 0:
            up.upsample = Upsample(block_in, resamp_with_conv)
            curr_res = curr_res * 2
        self.up.insert(0, up)

    self.norm_out = Normalize(block_in)
    self.conv_out = torch.nn.Conv2d(block_in,
                                    out_ch,
                                    kernel_size=3,
                                    stride=1,
                                    padding=1)
```

```
def forward(self, x, t=None, context=None):
    if context is not None:
        x = torch.cat((x, context), dim=1)
    if self.use_timestep:
        assert t is not None
        temb = get_timestep_embedding(t, self.ch)
        temb = self.temb.dense[0](temb)
        temb = nonlinearity(temb)
        temb = self.temb.dense[1](temb)
    else:
        temb = None

    hs = [self.conv_in(x)]
    for i_level in range(self.num_resolutions):
        for i_block in range(self.num_res_blocks):
            h = self.down[i_level].block[i_block](hs[-1], temb)
            if len(self.down[i_level].attn) > 0:
                h = self.down[i_level].attn[i_block](h)
            hs.append(h)
        if i_level != self.num_resolutions-1:
            hs.append(self.down[i_level].downsample(hs[-1]))

    h = hs[-1]
    h = self.mid.block_1(h, temb)
    h = self.mid.attn_1(h)

    for i_level in reversed(range(self.num_resolutions)):
        for i_block in range(self.num_res_blocks+1):
            h = self.up[i_level].block[i_block](
                torch.cat([h, hs.pop()], dim=1), temb)
            if len(self.up[i_level].attn) > 0:
                h = self.up[i_level].attn[i_block](h)
        if i_level != 0:
            h = self.up[i_level].upsample(h)

    h = self.norm_out(h)
    h = nonlinearity(h)
    h = self.conv_out(h)
    return h

def get_last_layer(self):
    return self.conv_out.weight
```

```
class Encoder(nn.Module):
    def __init__(self, *, ch, out_ch, ch_mult=(1,2,4,8), num_res_blocks,
                 attn_resolutions, dropout=0.0, resamp_with_conv=True, in_channels,
                 resolution, z_channels, double_z=True, use_linear_attn=False,
attn_type="vanilla",
                 **ignore_kwargs):
        super().__init__()
        if use_linear_attn: attn_type = "linear"
        self.ch = ch
        self.temb_ch = 0
        self.num_resolutions = len(ch_mult)
        self.num_res_blocks = num_res_blocks
        self.resolution = resolution
        self.in_channels = in_channels

        self.conv_in = torch.nn.Conv2d(in_channels,
                                       self.ch,
                                       kernel_size=3,
                                       stride=1,
                                       padding=1)

        curr_res = resolution
        in_ch_mult = (1,)+tuple(ch_mult)
        self.in_ch_mult = in_ch_mult
        self.down = nn.ModuleList()
        for i_level in range(self.num_resolutions):
            block = nn.ModuleList()
            attn = nn.ModuleList()
            block_in = ch*in_ch_mult[i_level]
            block_out = ch*ch_mult[i_level]
            for i_block in range(self.num_res_blocks):
                block.append(ResnetBlock(in_channels=block_in,
                                         out_channels=block_out,
                                         temb_channels=self.temb_ch,
                                         dropout=dropout))
                block_in = block_out
                if curr_res in attn_resolutions:
                    attn.append(make_attn(block_in, attn_type=attn_type))
            down = nn.Module()
            down.block = block
            down.attn = attn
            if i_level != self.num_resolutions-1:
                down.downsample = Downsample(block_in, resamp_with_conv)
```

```
                curr_res = curr_res // 2
            self.down.append(down)

        self.mid = nn.Module()
        self.mid.block_1 = ResnetBlock(in_channels=block_in,
                                       out_channels=block_in,
                                       temb_channels=self.temb_ch,
                                       dropout=dropout)
        self.mid.attn_1 = make_attn(block_in, attn_type=attn_type)
        self.mid.block_2 = ResnetBlock(in_channels=block_in,
                                       out_channels=block_in,
                                       temb_channels=self.temb_ch,
                                       dropout=dropout)

        self.norm_out = Normalize(block_in)
        self.conv_out = torch.nn.Conv2d(block_in,
                                        2*z_channels if double_z else z_channels,
                                        kernel_size=3,
                                        stride=1,
                                        padding=1)

    def forward(self, x):
        temb = None

        hs = [self.conv_in(x)]
        for i_level in range(self.num_resolutions):
            for i_block in range(self.num_res_blocks):
                h = self.down[i_level].block[i_block](hs[-1], temb)
                if len(self.down[i_level].attn) > 0:
                    h = self.down[i_level].attn[i_block](h)
                hs.append(h)
            if i_level != self.num_resolutions-1:
                hs.append(self.down[i_level].downsample(hs[-1]))

        h = hs[-1]
        h = self.mid.block_1(h, temb)
        h = self.mid.attn_1(h)
        h = self.mid.block_2(h, temb)

        h = self.norm_out(h)
        h = nonlinearity(h)
        h = self.conv_out(h)
        return h
```

```
class Decoder(nn.Module):
    def __init__(self, *, ch, out_ch, ch_mult=(1,2,4,8), num_res_blocks,
                 attn_resolutions, dropout=0.0, resamp_with_conv=True, in_channels,
                 resolution, z_channels, give_pre_end=False, tanh_out=False,
use_linear_attn=False,
                 attn_type="vanilla", **ignorekwargs):
        super().__init__()
        if use_linear_attn: attn_type = "linear"
        self.ch = ch
        self.temb_ch = 0
        self.num_resolutions = len(ch_mult)
        self.num_res_blocks = num_res_blocks
        self.resolution = resolution
        self.in_channels = in_channels
        self.give_pre_end = give_pre_end
        self.tanh_out = tanh_out

        in_ch_mult = (1,)+tuple(ch_mult)
        block_in = ch*ch_mult[self.num_resolutions-1]
        curr_res = resolution // 2**(self.num_resolutions-1)
        self.z_shape = (1,z_channels,curr_res,curr_res)
        print("Working with z of shape {} = {} dimensions.".format(
            self.z_shape, np.prod(self.z_shape)))

        self.conv_in = torch.nn.Conv2d(z_channels,
                                       block_in,
                                       kernel_size=3,
                                       stride=1,
                                       padding=1)

        self.mid = nn.Module()
        self.mid.block_1 = ResnetBlock(in_channels=block_in,
                                       out_channels=block_in,
                                       temb_channels=self.temb_ch,
                                       dropout=dropout)
        self.mid.attn_1 = make_attn(block_in, attn_type=attn_type)
        self.mid.block_2 = ResnetBlock(in_channels=block_in,
                                       out_channels=block_in,
                                       temb_channels=self.temb_ch,
                                       dropout=dropout)

        self.up = nn.ModuleList()
        for i_level in reversed(range(self.num_resolutions)):
```

```
            block = nn.ModuleList()
            attn = nn.ModuleList()
            block_out = ch*ch_mult[i_level]
            for i_block in range(self.num_res_blocks+1):
                block.append(ResnetBlock(in_channels=block_in,
                                         out_channels=block_out,
                                         temb_channels=self.temb_ch,
                                         dropout=dropout))
                block_in = block_out
                if curr_res in attn_resolutions:
                    attn.append(make_attn(block_in, attn_type=attn_type))
            up = nn.Module()
            up.block = block
            up.attn = attn
            if i_level != 0:
                up.upsample = Upsample(block_in, resamp_with_conv)
                curr_res = curr_res * 2
            self.up.insert(0, up) # prepend to get consistent order

        self.norm_out = Normalize(block_in)
        self.conv_out = torch.nn.Conv2d(block_in,
                                        out_ch,
                                        kernel_size=3,
                                        stride=1,
                                        padding=1)

    def forward(self, z):
        self.last_z_shape = z.shape

        temb = None

        h = self.conv_in(z)

        h = self.mid.block_1(h, temb)
        h = self.mid.attn_1(h)
        h = self.mid.block_2(h, temb)

        for i_level in reversed(range(self.num_resolutions)):
            for i_block in range(self.num_res_blocks+1):
                h = self.up[i_level].block[i_block](h, temb)
                if len(self.up[i_level].attn) > 0:
                    h = self.up[i_level].attn[i_block](h)
            if i_level != 0:
                h = self.up[i_level].upsample(h)
```

```
        if self.give_pre_end:
            return h

        h = self.norm_out(h)
        h = nonlinearity(h)
        h = self.conv_out(h)
        if self.tanh_out:
            h = torch.tanh(h)
        return h


class SimpleDecoder(nn.Module):
    def __init__(self, in_channels, out_channels, *args, **kwargs):
        super().__init__()
        self.model = nn.ModuleList([nn.Conv2d(in_channels, in_channels, 1),
                                    ResnetBlock(in_channels=in_channels,
                                                out_channels=2 * in_channels,
                                                temb_channels=0, dropout=0.0),
                                    ResnetBlock(in_channels=2 * in_channels,
                                                out_channels=4 * in_channels,
                                                temb_channels=0, dropout=0.0),
                                    ResnetBlock(in_channels=4 * in_channels,
                                                out_channels=2 * in_channels,
                                                temb_channels=0, dropout=0.0),
                                    nn.Conv2d(2*in_channels, in_channels, 1),
                                    Upsample(in_channels, with_conv=True)])
        self.norm_out = Normalize(in_channels)
        self.conv_out = torch.nn.Conv2d(in_channels,
                                        out_channels,
                                        kernel_size=3,
                                        stride=1,
                                        padding=1)

    def forward(self, x):
        for i, layer in enumerate(self.model):
            if i in [1,2,3]:
                x = layer(x, None)
            else:
                x = layer(x)

        h = self.norm_out(x)
        h = nonlinearity(h)
        x = self.conv_out(h)
```

```
        return x


class UpsampleDecoder(nn.Module):
    def __init__(self, in_channels, out_channels, ch, num_res_blocks, resolution,
                 ch_mult=(2,2), dropout=0.0):
        super().__init__()
        # upsampling
        self.temb_ch = 0
        self.num_resolutions = len(ch_mult)
        self.num_res_blocks = num_res_blocks
        block_in = in_channels
        curr_res = resolution // 2 ** (self.num_resolutions - 1)
        self.res_blocks = nn.ModuleList()
        self.upsample_blocks = nn.ModuleList()
        for i_level in range(self.num_resolutions):
            res_block = []
            block_out = ch * ch_mult[i_level]
            for i_block in range(self.num_res_blocks + 1):
                res_block.append(ResnetBlock(in_channels=block_in,
                                             out_channels=block_out,
                                             temb_channels=self.temb_ch,
                                             dropout=dropout))
                block_in = block_out
            self.res_blocks.append(nn.ModuleList(res_block))
            if i_level != self.num_resolutions - 1:
                self.upsample_blocks.append(Upsample(block_in, True))
                curr_res = curr_res * 2

        self.norm_out = Normalize(block_in)
        self.conv_out = torch.nn.Conv2d(block_in,
                                        out_channels,
                                        kernel_size=3,
                                        stride=1,
                                        padding=1)

    def forward(self, x):
        h = x
        for k, i_level in enumerate(range(self.num_resolutions)):
            for i_block in range(self.num_res_blocks + 1):
                h = self.res_blocks[i_level][i_block](h, None)
            if i_level != self.num_resolutions - 1:
                h = self.upsample_blocks[k](h)
        h = self.norm_out(h)
```

```
        h = nonlinearity(h)
        h = self.conv_out(h)
        return h


class LatentRescaler(nn.Module):
    def __init__(self, factor, in_channels, mid_channels, out_channels, depth=2):
        super().__init__()
        self.factor = factor
        self.conv_in = nn.Conv2d(in_channels,
                                 mid_channels,
                                 kernel_size=3,
                                 stride=1,
                                 padding=1)
        self.res_block1 = nn.ModuleList([ResnetBlock(in_channels=mid_channels,
                                                     out_channels=mid_channels,
                                                     temb_channels=0,
                                                     dropout=0.0) for _ in
range(depth)])
        self.attn = AttnBlock(mid_channels)
        self.res_block2 = nn.ModuleList([ResnetBlock(in_channels=mid_channels,
                                                     out_channels=mid_channels,
                                                     temb_channels=0,
                                                     dropout=0.0) for _ in
range(depth)])

        self.conv_out = nn.Conv2d(mid_channels,
                                  out_channels,
                                  kernel_size=1,
                                  )

    def forward(self, x):
        x = self.conv_in(x)
        for block in self.res_block1:
            x = block(x, None)
        x = torch.nn.functional.interpolate(x, size=(int(round(x.shape[2]*self.factor)),
int(round(x.shape[3]*self.factor))))
        x = self.attn(x)
        for block in self.res_block2:
            x = block(x, None)
        x = self.conv_out(x)
        return x
```

```
class MergedRescaleEncoder(nn.Module):
    def __init__(self, in_channels, ch, resolution, out_ch, num_res_blocks,
                 attn_resolutions, dropout=0.0, resamp_with_conv=True,
                 ch_mult=(1,2,4,8), rescale_factor=1.0, rescale_module_depth=1):
        super().__init__()
        intermediate_chn = ch * ch_mult[-1]
        self.encoder = Encoder(in_channels=in_channels, num_res_blocks=num_res_blocks,
ch=ch, ch_mult=ch_mult,
                               z_channels=intermediate_chn, double_z=False,
resolution=resolution,
                               attn_resolutions=attn_resolutions, dropout=dropout,
resamp_with_conv=resamp_with_conv,
                               out_ch=None)
        self.rescaler = LatentRescaler(factor=rescale_factor, in_channels=intermediate_chn,
                                       mid_channels=intermediate_chn,
out_channels=out_ch, depth=rescale_module_depth)

    def forward(self, x):
        x = self.encoder(x)
        x = self.rescaler(x)
        return x


class MergedRescaleDecoder(nn.Module):
    def __init__(self, z_channels, out_ch, resolution, num_res_blocks, attn_resolutions, ch,
ch_mult=(1,2,4,8),
                 dropout=0.0, resamp_with_conv=True, rescale_factor=1.0,
rescale_module_depth=1):
        super().__init__()
        tmp_chn = z_channels*ch_mult[-1]
        self.decoder = Decoder(out_ch=out_ch, z_channels=tmp_chn,
attn_resolutions=attn_resolutions, dropout=dropout,
                               resamp_with_conv=resamp_with_conv,
in_channels=None, num_res_blocks=num_res_blocks,
                               ch_mult=ch_mult, resolution=resolution, ch=ch)
        self.rescaler = LatentRescaler(factor=rescale_factor, in_channels=z_channels,
mid_channels=tmp_chn,
                                       out_channels=tmp_chn,
depth=rescale_module_depth)

    def forward(self, x):
        x = self.rescaler(x)
        x = self.decoder(x)
        return x
```

```
class Upsampler(nn.Module):
    def __init__(self, in_size, out_size, in_channels, out_channels, ch_mult=2):
        super().__init__()
        assert out_size >= in_size
        num_blocks = int(np.log2(out_size//in_size))+1
        factor_up = 1.+ (out_size % in_size)
        print(f"Building {self.__class__.__name__} with in_size: {in_size} --> out_size {out_size} and factor {factor_up}")
        self.rescaler = LatentRescaler(factor=factor_up, in_channels=in_channels, mid_channels=2*in_channels,
                                       out_channels=in_channels)
        self.decoder = Decoder(out_ch=out_channels, resolution=out_size, z_channels=in_channels, num_res_blocks=2,
                               attn_resolutions=[], in_channels=None, ch=in_channels,
                               ch_mult=[ch_mult for _ in range(num_blocks)])

    def forward(self, x):
        x = self.rescaler(x)
        x = self.decoder(x)
        return x


class Resize(nn.Module):
    def __init__(self, in_channels=None, learned=False, mode="bilinear"):
        super().__init__()
        self.with_conv = learned
        self.mode = mode
        if self.with_conv:
            print(f"Note: {self.__class__.__name} uses learned downsampling and will ignore the fixed {mode} mode")
            raise NotImplementedError()
            assert in_channels is not None
            self.conv = torch.nn.Conv2d(in_channels,
                                        in_channels,
                                        kernel_size=4,
                                        stride=2,
                                        padding=1)

    def forward(self, x, scale_factor=1.0):
        if scale_factor==1.0:
            return x
        else:
```

```
            x = torch.nn.functional.interpolate(x, mode=self.mode, align_corners=False, scale_factor=scale_factor)
        return x

class FirstStagePostProcessor(nn.Module):

    def __init__(self, ch_mult:list, in_channels,
                       pretrained_model:nn.Module=None,
                       reshape=False,
                       n_channels=None,
                       dropout=0.,
                       pretrained_config=None):
        super().__init__()
        if pretrained_config is None:
            assert pretrained_model is not None, 'Either "pretrained_model" or "pretrained_config" must not be None'
            self.pretrained_model = pretrained_model
        else:
            assert pretrained_config is not None, 'Either "pretrained_model" or "pretrained_config" must not be None'
            self.instantiate_pretrained(pretrained_config)

        self.do_reshape = reshape

        if n_channels is None:
            n_channels = self.pretrained_model.encoder.ch

        self.proj_norm = Normalize(in_channels,num_groups=in_channels//2)
        self.proj = nn.Conv2d(in_channels,n_channels,kernel_size=3,
                              stride=1,padding=1)

        blocks = []
        downs = []
        ch_in = n_channels
        for m in ch_mult:

blocks.append(ResnetBlock(in_channels=ch_in,out_channels=m*n_channels,dropout=dropout))
            ch_in = m * n_channels
            downs.append(Downsample(ch_in, with_conv=False))

        self.model = nn.ModuleList(blocks)
        self.downsampler = nn.ModuleList(downs)
```

```
def instantiate_pretrained(self, config):
        model = instantiate_from_config(config)
        self.pretrained_model = model.eval()
        for param in self.pretrained_model.parameters():
                param.requires_grad = False


@torch.no_grad()
def encode_with_pretrained(self,x):
        c = self.pretrained_model.encode(x)
        if isinstance(c, DiagonalGaussianDistribution):
                c = c.mode()
        return    c

def forward(self,x):
        z_fs = self.encode_with_pretrained(x)
        z = self.proj_norm(z_fs)
        z = self.proj(z)
        z = nonlinearity(z)

        for submodel, downmodel in zip(self.model,self.downsampler):
                z = submodel(z,temb=None)
                z = downmodel(z)

        if self.do_reshape:
                z = rearrange(z,'b c h w -> b (h w) c')
        return z
```