练习 3 实验报告

姓名：陈浩　学号：123106222839　学院：计算机科学与工程学院

1　实验目标

实现一种图像超分辨率方法在 Set5 数据集上的测试，得到超分辨率图像，测量其与原始真实图像之间的 PSNR、SSIM 指标值。对所选择方法的细节进行介绍，并试着讨论该方法可能存在的优缺点，以及可能的改进方向。

测试方法：先将图像用 Bicubic 插值进行下采样，再使用超分辨率算法处理，将得到的超分辨率图像与真实的原始图像进行对比。

2　方法描述

近些年来，高效、轻量化的单图像超分辨率（SISR）取得了显著的成绩。然而，目前最先进的模型仍然面临着诸如高计算开销等问题。为了解决这些问题，本报告中使用了大核蒸馏网络（LKDN），此方法简化了模型结构，并引入了更高效注意力模块，以降低计算成本的同时提高性能；此方法还从其它任务引入新的优化器，提高了训练速度和性能。

2.1 网络架构

本文使用的大核蒸馏网络（LKDN）遵循了与 BSRN 相同的框架设计，如图 1 所示。它包括四个模块：浅层特征提取块、多层叠特征蒸馏块、多层特征融合块和图像重建块。

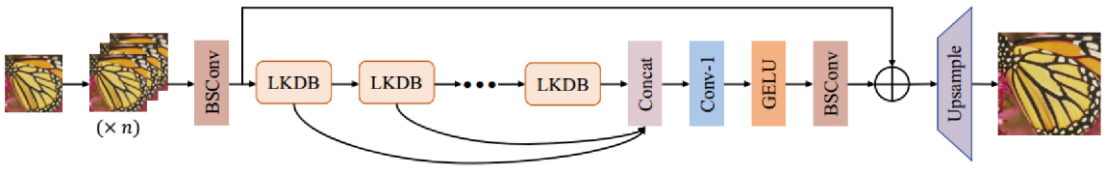

图 1 LKDN 网络架构

与传统的超分辨率模型相比，LKDN 在预处理阶段复制输入图像，然后将复制图像进行通道方向合并。给定输入的低分率图像 I_{LR}，这个复制流程可以表达为：

$$I_{LR}^{n} = \mathrm{Concat}(I_{LR}^{n}) \tag{1}$$

式（1）中的 $\mathrm{Concat}(\cdot)$ 表示沿通道维度的拼接操作，n 为复制输入图像 $\mathrm{I_{LR}}$ 的次数。然后通过一个 3×3 的 BSConv 进行初始特征提取，从输入 LR 图像中生成浅层特征可表示为：

$$F_0 = h_{ext}(I_{LR}^{n}) \tag{2}$$

式（2）中的 $h_{ext}(\cdot)$ 为浅层特征提取模块，$\mathrm{F_0}$ 表示生成的浅层特征。而 BSConv 的结构如图 2 所示，它由一个 1×1 卷积和一个深度卷积组成。

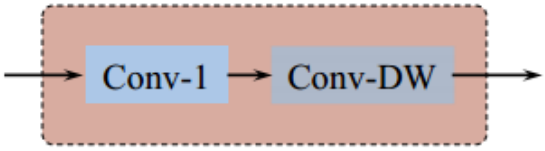

图 2 BSConv 结构

LKDN 的下一模块是通过堆叠的大核蒸馏块（LKDBs）来提取深层特征，可以表述为：

$$F_k = H_{LKDN}^m(\dots H_{LKDN}^1(F_0)\dots),\ 1 \le k \le m \tag{3}$$

式(3)中的 $H_{LKDN}^k(\cdot)$ 表示第 k 个 LKDB，m 表示使用的 LKDBs 数量，F_k 表示第 m 个 LKDB 的输出深层特征。

经过 LKDBs 的逐步特征提炼后，所有中间特征进行通道合并，再通过 1×1 卷积层和 GELU 函数进行融合和激活，随后采用 3×3 的 BSConv 层平滑融合特征。多层特征融合流程可表述为:

$$F_{fusion} = H_{fusion}(\mathrm{Concat}(F_1,\dots,F_k)) \tag{4}$$

式（4）中的 $\mathrm{H_{fusion}}(\cdot)$ 为特征融合模块， $\mathrm{F_{fusion}}$ 表示融合后的特征。

最后在模型中使用跳跃连接来增强残差学习，通过图像重建得到 SR 图像可表示为:

$$I_{SR} = H_{rec}(F_{fusion} + F_0) \tag{5}$$

式（5）中的 $\mathrm{H_{rec}}(\cdot)$ 表示图像重建模块，$\mathrm{I_{SR}}$ 表示模型输出，此处的重建流程只包含一个 3×3 卷积和 pixel-shuffle 操作。

2.2 大核蒸馏块

LKDN 使用了一个更高效的大内核蒸馏块(LKDB)，LKDB 的完整结构如图 3 所示。它包括四个部分：特征提取、特征融合、特征增强和特征转换。

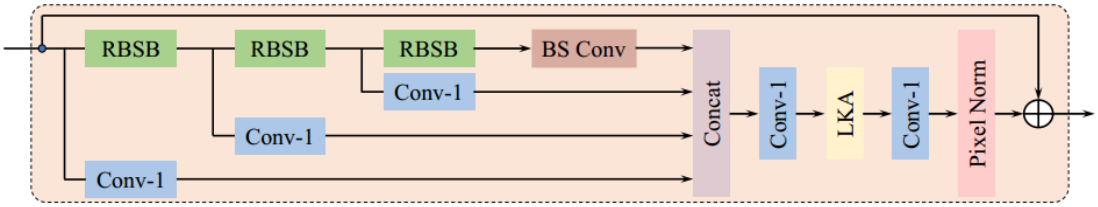

图 3 LKDB 结构

在第一阶段特征提取中，对于给定输入 F_{in} ，特征蒸馏操作可描述为：

$$\begin{aligned} F_{d_1}, F_{r_1} &= D_1(F_{in}), R_1(F_{in}), \\ F_{d_2}, F_{r_2} &= D_2(F_{r_1}), R_2(F_{r_1}), \\ F_{d_3}, F_{r_3} &= D_3(F_{r_2}), R_3(F_{r_2}), \\ F_{d_4} &= D_4(F_{r_3}) \end{aligned} \tag{6}$$

式（6）中 D_i 、R_i 分别表示第 i 个蒸馏层和第 i 个精炼层，而 F_{d_i} 、$\mathrm{F_{r_i}}$ 分别表示第 i 个蒸馏特征和第 i 个精炼特征。而 RBSB 的结构如图 4 所示，RBSB 中有两个串联的并行结构，第一个是 identity connection 与 1×1 的卷积并行，第二个是 identity connection 与 1×1 的深度卷积和 3×3 的深度卷积并行，最后使用 GELU 函数进行激活。

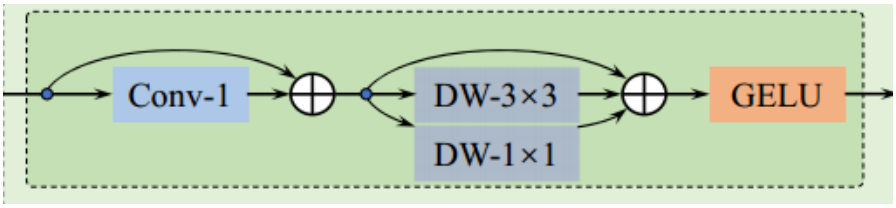

图 4 RBSB 结构

在特征融合阶段，将特征提取阶段的所有蒸馏特征拼接在一起，然后通过一个 1×1 的卷积层完成特征融合，而特征融合阶段可表示为：

$$F_{fusion} = H_{fusion}(\text{Concat}(F_{d_1}, F_{d_2}, F_{d_3}, F_{d_4})) \tag{7}$$

式（7）中的 $H_{fusion}(\cdot)$ 表示 1×1 的卷积， $F_{fusion}(\cdot)$ 表示融合后的特征。

在特征增强阶段，LKDB 引入了一种高效的大核注意（LKA）块，如图 5 所示。LKA 使用分解策略，将一个大的 17×17 卷积分解为一个 1×1 的点向卷积 Conv-1、一个 5×5 的深度卷积和一个核大小为 5、膨胀度为 3 的深度膨胀卷积，降低了模型复杂度，提高性能。其特征增强阶段可用式（8）来表示。

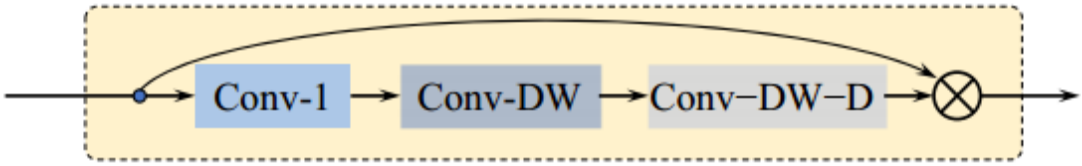

图 5 LKA 结构

$$F_{enhance} = H_{LKA}(F_{fusion}) \tag{8}$$

在特征转换阶段，为了提高模型的性能，LKDB 使用了 1 × 1 卷积，同时加入了像素归一化模块，以确保模型训练稳定。特征转换阶段如公式（9），其中 $H_{trans}(\cdot)$ 表示 1×1 卷积，F_{trans} 表示转换后的特征， $\text{Norm}_{\text{pixel}}$ 表示像素归一化操作。最后，LKDB 采用长跳跃连接来增强模型的残差学习能力，流程如公式（10）。

$$F_{trans} = \text{Norm}_{\text{pixel}}(H_{trans}(F_{enhance})) \tag{9}$$

$$F_{out} = F_{trans} + F_{in} \tag{10}$$

2.3 网络改进

2.3.1 去除冗余复制

LKDN 在预处理阶段复制输入图像 n 次，然后将复制图像进行通道方向合并，意图可能是想要缩小两模块间的通道鸿沟，但是本质上无法提供更多的细节特征，因为复制图像与原输入图像是完全一样的，而且复制操作增加了输入特征维度，增加了网络的参数，加大了训练的开销。

为此，我将 LKDN 中的复制操作去除。根据实验 4.6 分析，不执行复制操作的 LKDN 比执行四次复制操作的 LKDN 的网络性能稍优，而执行复制操作的 LKDN 的网络参数更多，所以验证了去除复制操作的合理性。

2.3.2 重参数化

过多的跳跃连接操作会增加内存访问成本和推理时间，然而图 4 中 RBSB 结构中有多

个跳跃连接，因此有优化的可能。为了解决这个问题，如图 6 受 RepVGG 启发，它在训练阶段保持了多跳跃连接的多分支结构，以保证复杂结构的强大学习潜能，但在推理时将网络重构成简单的串行结构，以提高推理速度。

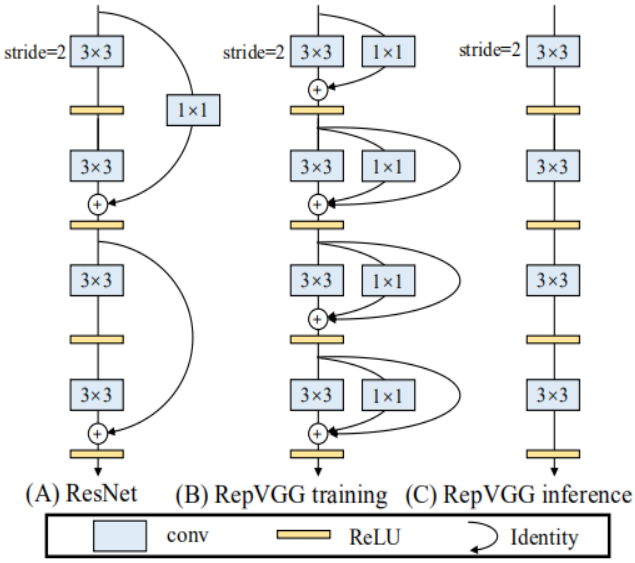

图 6 RepVGG 重参数化

使用重参数化技术将第一个 identity connection 与 1×1 卷积并行结构在推理阶段转为一个 1×1 卷积、将第二个 identity connection、1×1 的深度卷积和 3×3 的深度卷积并行结果在推理阶段转化为一个 3×3 的深度卷积，如图 7。由此，在推理阶段 RBSB 中的跳跃连接被去除，整体为串行结构，达到了降低内存访问成本和提高推理速度的目的。

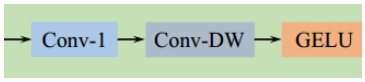

图 7 RBSB 重参数化结果

重参数化是基于卷积运算是线性运算且具有可加性，如图 8 重参数化原理，identity connection 可视为一个 1×1 卷积，而 1×1 卷积可视为一个 3×3 卷积，所以 identity connection 和 1×1 卷积都可以视为 3×3 卷积，再利用卷积运算的可加性，将三个 3×3 卷积权重合并为一个 3×3 卷积，到此完成了重参数化过程。

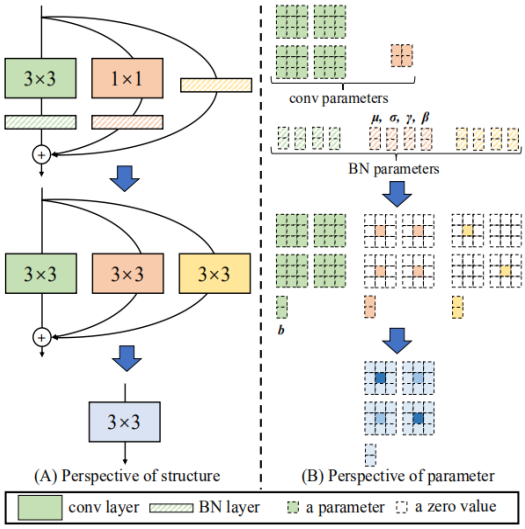

图 8 重参数化原理

3 代码描述

3.1 函数功能

项目中的主要模块包含文件夹 basicsr、datasets、experiment、options、pth 和 result。其中 basicsr 文件夹下的 archs 文件是网络架构定义，其中 LKDN 和 LKDB 的网络定义分别在 lkdn_arch.py 文件和 lkdn_blocks.py 文件；basicsr 文件夹下的 train.py 和 test.py 分别是训练和测试代码。

datasets 文件夹下是测试 Set5 数据集以及下采样文件，尽管训练集要包含 Set5、Set14、BSD100、Urban100、Manga109、DIV2K 和 DF2K，但是实验只要求在 Set5 数据集测试，所以 datasets 文件夹下仅包含 Set5 数据集。下采样文件实现了对 Set5 中的图片用 Bicubic 插值进行四倍下采样。

experiment 文件夹下的 pretrained_models 是预训练好的权重文件，而 self_training 是自己训练好的权重文件。每个自训练文件下又包含 models 文件夹和 log 日志文件，其中 models 文件夹下保存了权重文件，log 日志文记录了训练时每 100 次迭代的信息，如 epoch、学习率、损失函数值等。

options 文件夹下有 train 和 test 文件夹，分别对应 LKDN 的训练和测试参数配置文件，且都以 yml 文件形式存储。

pth 文件夹是重参数化功能模块，其下的 reparm.py 文件考虑了跳跃连接的不同情况，分别实现了将 identity connection 与 1×1 卷积重参数化为 1×1 卷积、将 identity connection、1×1 深度卷积与 3×3 深度卷积重参数化为 3×3 深度卷积。

result 文件夹保存了每次测试生成的超分辨率图像结果，以及每次测试的日志文件。

3.2 网络改进

3.2.1 去除冗余复制

要将 LKDN 中的复制操作去除，我将 basicsr/archs/lkdn_arch.py 中 num_in=4 改为 num_in=1，并且重构了 options/train/LKDN 文件夹下 yml 文件中的 network_g 部分，yml 文件中的 network_g 部分如图 9 所示。由此，网络预处理阶段仅复制一次图像，即为原图像，不再进行冗余复制操作。

```yaml
# network structures
network_g:
  type: LKDN
  num_in_ch: 3
  num_out_ch: 3
  num_feat: 56
  num_atten: 56
  num_block: 8
  upscale: 4
#  num_in: 4
  num_in: 1
  conv: BSConvU
  upsampler: pixelshuffledirect
```

图 9 LKDN 网络部分输入参数

3.2.2 重参数化

重参数化功能由 pth/reparm.py 文件实现，我考虑了跳跃连接的不同情况，分别实现了将 identity connection 与 1×1 卷积重参数化为 1×1 卷积、将 identity connection、1×1 深度卷积与 3×3 深度卷积重参数化为 3×3 深度卷积。

对于将 identity connection 与 1×1 卷积重参数化为 1×1 卷积，由图 8 可知，需要将 identity connection 等价为一个 1×1 卷积，具体是初始化一个同尺寸的 1×1 卷积，将 1×1 核值设为 1，这样图像经过该卷积运算后输出结果即为原图像，为此实现了将 identity connection 等价为一个 1×1 卷积，然后再利用卷积运算的可加性，将等价的 1×1 卷积与并联的 1×1 卷积权重加和，即完成了第一步的重参数化，即完成了从图 4 到图 7 的第一部分转化，过程如图 10 所示。第一行筛选了点向卷积，第三行初始化了一个同尺寸的 1×1 卷积，第四、五行实现了将该 1×1 卷积核设为 1，完成了将 identity connection 等价为一个 1×1 卷积，最后一行利用卷积运算的可加性将两权重直接相加。

```
# conv1 + identity -->  conv1
if blk_name[i].endswith('r.pw.weight'):
    weight = dict[blk_name[i]]
    weight_idt = torch.zeros(42, 42, 1, 1)
    for j in range(42):
        weight_idt[j, j, 0, 0] = 1.0
    rep_weight = weight + weight_idt
```

图 10 第一步重参数化

对于将 identity connection、1×1 深度卷积与 3×3 深度卷积重参数化为 3×3 深度卷积，由图 8 可知，同样的要将 identity connection 等价转化为一个 3×3 卷积、将 1×1 卷积价转化为一个 3×3 卷积，最后将等价转化的卷积与并行的 3×3 卷积进行权重加和与偏差加和。由于要保证输入输出特征图维度一致，所以要在卷积过程中进行 padding 操作，要将 1×1 卷积价转化为一个 3×3 卷积，只需将 1×1 核使用零填充扩充为 3×3 且原核值映射于 3×3 卷积核中心即可，同理，要将 identity connection 等价转化为一个 3×3 卷积，先初始化一个 3×3 卷积，除了卷积核中心值为 1，其它值全为 0。到此完成了第二步的重参数化，即完成了从图 4 到图 7 的第二部分转化，过程如图 11 所示。第一行初始化了一个同尺寸的 3×3 卷积，第二、三行实现了将该 3×3 卷积核中心值设为 1，完成了将 identity connection 等价为一个 3×3 卷积，第四行实现了将 1×1 卷积扩充为一个 3×3 卷积，并且原核值映射于 3×3 卷积核中心，最后两行利用卷积运算的可加性完成了三个卷积的权重和偏差相加。

```
weight_idt = torch.zeros(42, 1, 3, 3)
for j in range(42):
    weight_idt[j, 0, 1, 1] = 1.0
weight1x1 = F.pad(weight1x1, (1, 1, 1, 1), 'constant', 0.0)
rep_weight = weight + weight_idt + weight1x1
rep_bias = bias + bias1x1
```

图 11 第二步重参数化

4 实验与分析

4.1 实验说明

实验使用的训练数据集有 Set5、Set14、BSD100、Urban100、Manga109、DIV2K 和 DF2K，但由于测试时仅用 Set5 数据集，所以 datasets 文件夹仅有 Set5 数据集。实验使用平均峰值信噪比（PSNR）与结构相似度（SSIM）作为评价指标进行定量分析。

LKDN 模型由 8 个 LKDBs 组成，蒸馏结构通道数和注意力模块通道数设置为 56，并使用 BSB 进行训练以减少训练时间。每个 LR 输入的 batch size 和 patch size 分别设置为 64 和 48×48，并使用常见的 L1 损失函数和 Adan 优化器来训练模型，其中 $\beta_1 = 0.98$， $\beta_2 = 0.92$，

$\beta_3 = 0.99$，设置 10^6 次训练迭代，学习率设置为 5×10^{-3}。

还提出了 LKDN 的一个小版本 LKDN-S，LKDN-S 由 5 个 LKDBs 和 42 个通道组成。LKDN-S 的训练过程包括两个阶段：初始训练阶段和微调阶段。在初始训练阶段，batch size 为 128，HR patch size 为 256×256，使用 L1 损失函数训练 LKDN-S，学习率设为 5×10^{-3} 和 9.5×10^5 迭代；在微调阶段，将 HR 图像的 patch size 和 batch size 分别设置为 480×480 和 64，LKDN-S 使用 L2 损失函数进行微调，学习率为 2×10^{-5}，迭代次数为 5×10^4。两个阶段均将 EMA 设置为 0.999，使用 Adan 优化器，$\beta_1 = 0.98$，$\beta_2 = 0.92$，$\beta_3 = 0.99$。

4.2 运行说明

我的实验环境为 PyTorch 1.13 和 BasicSR 1.4.2，且使用 Nvidia GeForce RTX 3090 GPUs，在运行代码前要运行以下两行代码，安装好实验所需的包以及进行环境配置。

①pip install -r requirements.txt

②python setup.py develop

测试数据集 Set5 存放于 datasets 文件夹下，用 Bicubic 插值对 Set5 数据集进行四倍下采样的操作如下：

python datasets/downscale.py

如要测试网络，则运行以下代码即可，其中 yml 表示测试的配置文件，可以选择要测试的网络。其中 test_LKDN_x4.yml、test_LKDN-S_x4.yml、test_LKDN-S_del_rep_x4.yml 分别表示测试 LKDN、LKDN-S 和重参数化 LKDN-S 的配置文件。

python basicsr/test.py -opt options/test/LKDN/test_LKDN_x4.yml

python basicsr/test.py -opt options/test/LKDN/test_LKDN-S_x4.yml

python basicsr/test.py -opt options/test/LKDN/test_LKDN-S_del_rep_x4.yml

如要进行重参数化，可以在设置好要重参数化的权重文件和保存路径后，依次执行：

①python pth/del_params_ema.py

②python pth/reparm.py

4.3 实验流程

①下载 Set5，用 Bicubic 插值进行四倍下采样。下载好数据集并设置好 Set5 数据集的文件路径和下采样图片输出路径后，遍历输入文件夹以匹配 png 文件，使用插值方法为 Bicubic 的 cv2.resize 函数对各图片进行四倍下采样，具体是设置函数中的 fx=1/4、fy=1/4 以及 interpolation=cv2.INTER_CUBIC。

②加载训练好的模型，输入下采样四倍图片，输出高分辨率预测图片。

③计算预测图片与原始真实图像之间的 PSNR、SSIM 指标值。

4.4 重参数化

我在 Nvidia GeForce RTX 3090 GPU 训练好 LKDN-S，并在推理阶段使用重参数化技术将 RBSB 转化为图 7 所示，得到了重参数化网络 LKDN-S-Rep，并在测试数据集 Set5 上比较了网络 LKDN-S 和 LKDN-S-Rep 的性能，结果如表 1 所示。

表 1 重参数化前后 LKDN 在 Set5 上的 PSNR/SSIM

方法	参数（M）	baby	bird	butterfly	head	woman	Average
LKDN-S-paper	0.172	33.78/0.8939	34.76/0.9428	28.41/0.9227	32.87/0.7954	30.69/0.9140	32.10/0.8938
LKDN-S-mine	0.172	**33.80/0.8943**	**34.77/0.9432**	**28.58/0.9231**	**32.91/0.7963**	**30.82/0.9150**	**32.18/0.8944**
LKDN-S-Rep-paper	**0.129**	33.78/0.8939	34.76/0.9428	28.41/0.9227	32.87/0.7954	30.69/0.9140	32.10/0.8938
LKDN-S-Rep-mine	**0.129**	**33.80/0.8943**	**34.77/0.9432**	**28.58/0.9231**	**32.91/0.7963**	**30.82/0.9150**	**32.18/0.8944**

从结果分析，我自己训练的 LKDN-S 可以达到报告中的性能，并且性能稍优，而且重参数化技术的运用可以在推理阶段保持模型性能的同时进一步降低模型的复杂度。

4.5 优化器

以往的 SR 模型优化主要依赖于基于 Adam 优化器。然而，Adan 优化器最近在各种视觉任务上取得了最先进的结果，因此报告研究了 Adan 优化器对 SR 任务的影响。我使用 Adam 和 Adan 优化器分别训练了 LKDN，并将之用于测试集上进行性能测试，结果如表 2 所示。由于我使用的是 GeForce RTX 3090，而原论文实验使用的是 GeForce RTX 3080，所以训练时间会更快，但两者性能相当。对比两个优化器的结果可知，使用 Adan 优化器可以产生训练加速，并且在多数基准数据集上都有性能提升，同时由图 12 可知使用 Adan 优化器的收敛速度更快。

表 2 应用 Adam 和 Adan 优化器在基准数据集上的 PSNR/SSIM

方法	训练(h)	Set	Set14	B100	Urban100	Manga109
LKDN_Adam-paper	50	**32.41**/0.8975	28.77/0.7854	27.69/0.7399	26.36/0.7949	30.93/0.9132
LKDN_Adam-mine	35	32.38/0.8972	28.79/0.7855	27.69/0.7400	26.37/0.7948	30.93/0.9134
LKDN_Adan-paper	45	32.39/0.**8979**	28.79/0.7859	27.69/0.7402	**26.42/0.7965**	30.97/0.9140
LKDN_Adan-mine	**33.5**	32.37/0.8972	**28.80/0.7860**	**27.70/0.7404**	26.41/0.7964	**30.98/0.9141**

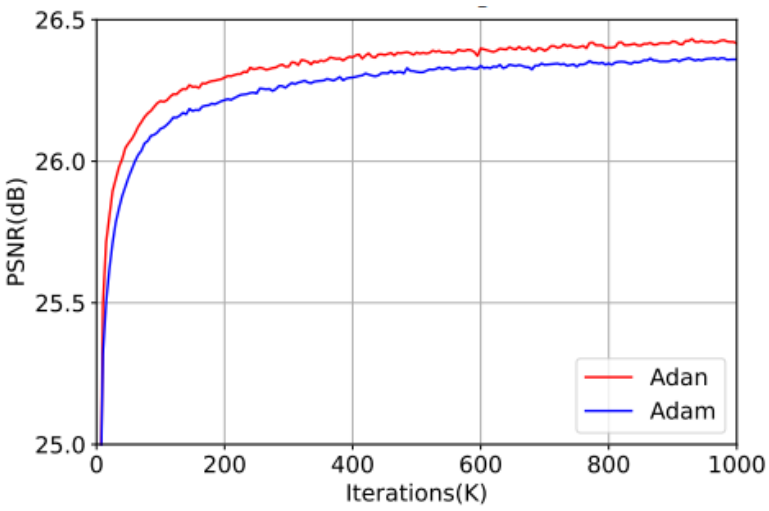

图 12 在 Urban100 上 Adan 和 Adam 优化器的收敛性比较

4.6 冗余复制

LKDN 在预处理阶段复制输入图像 n 次，然后将复制图像进行通道方向合并，为了验证复制操作的有效性，我进行了对比实验，使用 Adan 优化器分别训练了使用复制输入图像 4 次的网络 LKDN_IN4 和仅复制 1 次的网络 LKDN_IN1，并在测试集在比较了这两个网络的性能，结果如表 3 和表 4 所示。分析表 3 和表 4 中的结果可知，复制多次的操作并不能带来性能提升，甚至在 PSNR 指标下仅复制 1 次的网络 LKDN_IN1 在大多数测试集上的性能还优于 LKDN_IN4，并且 LKDN_IN1 参数比 LKDN_IN4 更少。

因此去除冗余的复制操作可以在保持模型性能的同时进一步降低模型参数，降低了训练的开销，更好地符合轻量化模型的设计。

表 3 复制 1 次和 4 次的 LKDN 在 Set5 上的 PSNR/SSIM

方法	参数（M）	baby	bird	butterfly	head	woman	Average
LKDN_IN1	**0.320**	33.86/0.8951	35.10/0.9471	**29.02/0.9299**	**32.98/0.7982**	**31.05/0.9181**	**32.40/0.8977**
LKDN_IN4	0.322	**33.89/0.8953**	**35.25/0.9474**	28.95/0.9294	32.96/0.7978	30.83/0.9163	32.37/0.8972

表 4 复制 1 次和 4 次的 LKDN 在基准数据集上的 PSNR/SSIM

方法	参数（M）	Set5	Set14	B100	Urban100	Manga109
LKDN_IN1	0.320	**32.40/0.8977**	**28.81**/0.7859	**27.70**/0.7403	26.38/0.7957	**30.99**/0.9140
LKDN_IN4	0.322	32.37/0.8972	28.80/**0.7860**	**27.70/0.7404**	**26.41/0.7964**	30.98/**0.9141**

4.7 阶段训练探究

LKDN-S 的训练过程包括两个阶段：初始训练阶段和微调阶段，在初始阶段使用的是 L1 损失函数，而在微调阶段使用的是 L2 损失函数。为了验证阶段损失函数选择的合理性，我进行了阶段损失函数训练探究。

在初始训练阶段，我分别使用 L1 损失函数和 L2 损失函数完成第一阶段的训练，并分析两中网络在验证集 DIV2K 上的性能表现，如图 13。可以看出在验证集上使用 L1 损失函数训练的网络性能要略高于使用 L2 损失函数的性能，这是因为 L2 损失函数较之于 L1 损失函数对离群点比较敏感、受其影响较大，因当差值大于 1 时会放大误差，这就会牺牲其他正常点数据的预测效果，最终降低整体的模型性能，所以在第一阶段使用 L1 损失函数是因为其较强的鲁棒性。同时，我在测试集上比较了第一阶段训练得到的两个网络的性能，结果如表 5 所示，在第一阶段使用 L1 损失函数训练的网络能够达到更优的性能。

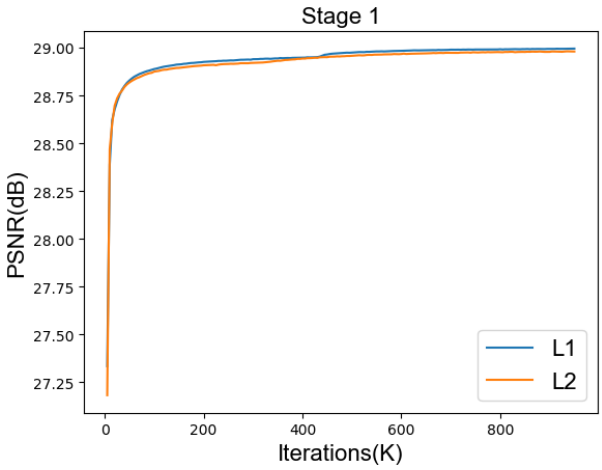

图 13 初始训练阶段在 DIV2K 使用不同损失函数的性能比较

表 5 第一阶段训练中不同损失函数的性能比较

第一阶段	训练（h）	Set5	Set14	B100	Urban100	Manga109
L1	119	**32.21/0.8949**	**28.64/0.7821**	**27.59/0.7367**	**26.06/0.7854**	30.50/**0.9082**
L2	129.5	32.09/0.8934	28.58/0.7812	27.58/0.7363	**26.06**/0.7843	30.47/0.9062

在微调阶段，我也分别使用 L1 损失函数和 L2 损失函数完成第一阶段的训练，并分析两中网络在验证集 DIV2K 上的性能表现，如图 14。可以看出在验证集上使用 L2 损失函数的网络性能更优且基本收敛，而使用 L1 损失函数的网络还未收敛，这是因为 L1 损失函数的梯度为常数，所以在较小的损失值时，得到的梯度也相对较大，可能造成模型震荡不利于收敛，所以在微调阶段使用 L2 损失函数有利于模型的收敛。同时，在测试集上比较了以上两个网络的性能，结果如表 6 所示，在微调阶段使用 L2 损失函数训练的网络在 PSNR 指标下能够达到更优的性能。

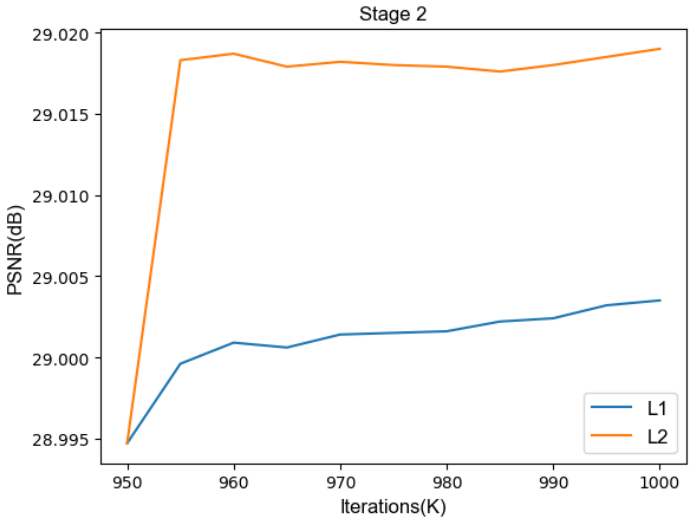

图 14 微调阶段在 DIV2K 使用不同损失函数的性能比较

表 6 第二阶段训练中不同损失函数的性能比较

第二阶段	训练（h）	Set5	Set14	B100	Urban100	Manga109
L1	11.5	**32.21/0.8950**	**28.66**/0.7824	**27.60**/0.7369	26.08/**0.7862**	30.52/**0.9085**
L2	11.5	32.18/0.8944	**28.66/0.7827**	**27.60/0.7371**	**26.09**/0.7857	**30.56**/0.9077

5 总结

大核蒸馏网络（LKDN）是一种有效的单图像超分辨率（SISR）解决方案。通过结合大内核设计、简化模型结构、引入更高效的注意力模块，LKDN 在性能和计算效率之间实现了的平衡。同时还引入了一种新的优化器，以加快 LKDN 的收敛速度和提高模型性能。

通过个人实验，①发现对 LKDN-S 使用重参数化技术可以在保持模型性能的同时，进一步降低模型复杂度。②使用 Adan 优化器较之于 Adam 优化器可以提高训练速度以及模型性能。③去除网络预处理阶段冗余的复制操作可以进一步减少模型参数，而保持模型精度。④对 LKDN-S 使用两阶段训练具有合理性，初始阶段使用 L1 损失函数具有鲁棒性、能避免离群点的干扰，而微调阶段使用 L2 损失函数有利于收敛。

通过实验与分析，发现模型也有以下缺点：①网络模块中存在多并联的蒸馏结构，降低了模型的推理速度；②网络结构较复杂，参数较多。针对以上缺点可能的改进方向有：①受 RLFN 的启发，去除分层蒸馏连接，增加卷积层的通道数，以实现模型性能和推理时间之间的良好平衡；②可以使用网络裁剪技术进一步降低模型参数。