图像超分辨率实验报告

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。

**2.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。数据集下载网址:https://github.com/spnova12/get\_set5\_psnr

1. **研究背景**

单幅图像超分辨率(SISR)的目的是将相应的低分辨率(LR)图像降级后重建成高分辨率(HR)图像。随着深度学习的快速发展，它已经取得了实质性的进步，并在研究界获得了更多的关注。开创性的工作SRCNN提出通过卷积神经网络(CNN)学习LR输入到HR输入的映射，并且优于传统方法。随后，许多基于cnn的工作探索了更有效的架构。除了CNN架构外，还提出了一种基于transformer的架构，并实现了最先进(SOTA)的性能。

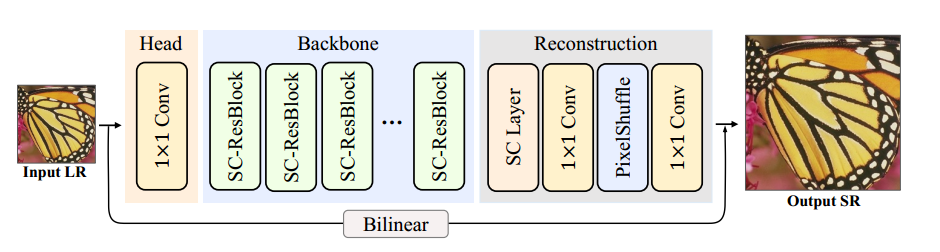
然而，上述模型在非常深或复杂的网络架构下提高了SISR性能，导致参数量和计算成本负担沉重。这使得在资源受限的环境(如移动设备或边缘设备)中部署它们变得困难。因此，对高效和轻量级SR模型的需求很高。已经提出了许多工作来减少参数或浮点运算(FLOPs)的数量，以实现用于SISR的轻量级神经网络。

本文通过空间位移扩展1 × 1卷积。值得注意的是，空间移位操作是非参数化的，不需要额外的flop，这对于高度优化的实际应用是有利的。通过沿着通道维度将输入特征映射划分为不同的组，然后对每一组具有不同空间方向的组进行空间移位操作。它确保生成的特征图中的每个像素沿着通道维度围绕特征进行组装，弥合了表示能力与3 × 3卷积的差距。我们将这种扩展的1 × 1卷积与通过空间移位操作进行的局部特征聚合称为Shift-Conv层(或简单地称为SC层)。与普通的3 × 3卷积相比，SC层在保持相当性能的同时显著减少了参数的数量。

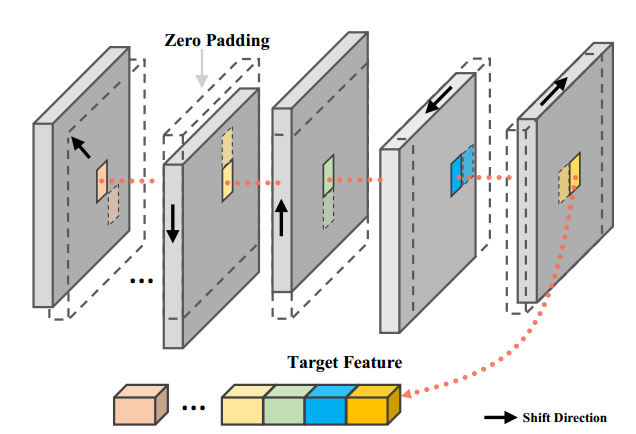
1. **实验内容**

4.1整体流程

给定一个低分辨率图像，首先浅层特征提取器将其映射到指定的隐层特征空间，得到特征图。接着，浅层特征图通过深层特征提取器，提取深层特征图。最后，使用高分辨图像重建模块对深层特征进行上采样，获得最终的超分辨率结果。学习的目标函数是最小化超分辨率结果与目标高分辨图像的误差。如下图所示，整体的流程是通过许多基本SC-Resblock叠加在所提出的SCNet的主干网上，然后向上缩放层以重建高分辨率(HR)结果。



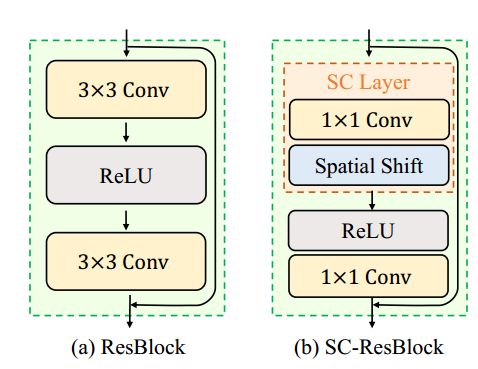
4.2 移位卷积模块



移位卷积包含逐点卷积和空间移位操作。通过空间移位操作沿着通道方向实现近邻特征对齐，从而实现局部特征聚合。具体的空间移位操作实现如图上所示。对输入特征图首先进行通道拆分，均匀划分为N组，N这里表示移位的近邻特征数量。为了与3×3卷积保持一致，我们默认采取八组。接着对不同的组分别沿着不同的方向移动指定步长。如图所示，沿着不同方向移动后实现了对应位置处近邻特征的聚合。在这里，为了跟3×3卷积保持一致，采用8个方向和步长为1作为默认设置。值得注意的是，移位卷积相较于3×3卷积，不仅实现了局部特征聚合，通过控制移位的特征点位选取，还可以进一步扩展到长距特征关系提取。

4.3 移位残差单元

基于上述的移位卷积层，将标准残差结构中的3×3卷积全部替换成逐点卷积，其中嵌入空间移位操作。改进后的移位残差单元包含一个移位卷积和一个逐点卷积以及激活层，具体的实现细节见下图。基于移位残差单元，我们通过堆叠不同的移位残差块实现了不同规模的SCNet。



1. **代码编写**

使用mindspore编写推理代码，主要代码如下：

import mindspore

import mindspore.nn as nn

import mindspore.ops as ops

from mindspore import Tensor

from mindspore.common.initializer import Normal

from mindspore.ops import operations as P

# 定义 SCNet 的 MindSpore 版本

class SCNet(nn.Cell):

    """

    SCNet based on the Modified SRResNet for MindSpore.

    Args:

        num\_in\_ch (int): Channel number of inputs. Default: 3.

        num\_out\_ch (int): Channel number of outputs. Default: 3.

        num\_feat (int): Channel number of intermediate features. Default: 64.

        num\_block (int): Block number in the body network. Default: 16.

        upscale (int): Upsampling factor. Support x2, x3 and x4. Default: 4.

    """

    def \_\_init\_\_(self, num\_in\_ch=3, num\_out\_ch=3, num\_feat=64, num\_block=16, upscale=4):

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

        self.upscale = upscale

        self.conv\_first = nn.Conv2d(num\_in\_ch, num\_feat, 1, has\_bias=True)

        self.body = nn.CellList(

            [ResidualBlockShift(num\_feat) for \_ in range(num\_block)]

        )

        # upsampling

        if self.upscale in [2, 3]:

            self.upconv1 = UpShiftMLP(num\_feat, scale=self.upscale)

        elif self.upscale == 4:

            self.upconv1 = UpShiftMLP(num\_feat)

            self.upconv2 = UpShiftMLP(num\_feat)

        elif self.upscale == 8:

            self.upconv1 = UpShiftMLP(num\_feat)

            self.upconv2 = UpShiftMLP(num\_feat)

            self.upconv3 = UpShiftMLP(num\_feat)

        self.conv\_hr = nn.Conv2d(num\_feat, num\_feat, kernel\_size=1, has\_bias=True)

        self.conv\_last = nn.Conv2d(num\_feat, num\_out\_ch, kernel\_size=1, has\_bias=True)

        # activation function

        self.lrelu = nn.LeakyReLU(0.1)

        # initialization

        for layer in [self.conv\_first, self.conv\_hr, self.conv\_last]:

            layer.weight.set\_data(Normal(0.1)(layer.weight.shape))

    def construct(self, x):

        feat = self.lrelu(self.conv\_first(x))

        for block in self.body:

            feat = block(feat)

        if self.upscale == 4:

            feat = self.lrelu(self.upconv1(feat))

            feat = self.lrelu(self.upconv2(feat))

        elif self.upscale in [2, 3]:

            feat = self.lrelu(self.upconv1(feat))

        elif self.upscale == 8:

            feat = self.lrelu(self.upconv1(feat))

            feat = self.lrelu(self.upconv2(feat))

            feat = self.lrelu(self.upconv3(feat))

        out = self.conv\_last(self.lrelu(self.conv\_hr(feat)))

        base = nn.ResizeBilinear()(x, scale\_factor=self.upscale)

        out += base

        return out

import os

from mindspore import load\_checkpoint, load\_param\_into\_net

from PIL import Image

if \_\_name\_\_ == "\_\_main\_\_":

    # 初始化模型

    model = SCNet(upscale=4)

    # 加载预训练的模型权重

    param\_dict = load\_checkpoint("SCNet-T-D64B16.ckpt")

    load\_param\_into\_net(model, param\_dict)

    model.set\_train(False)

    # 设置输入输出文件夹路径

    input\_folder = "Set5/LR"

    output\_folder = "Set5/SR2"

    # 创建输出文件夹

    if not os.path.exists(output\_folder):

        os.makedirs(output\_folder)

    # 遍历输入文件夹中的所有PNG图片

    for image\_file in os.listdir(input\_folder):

        if image\_file.endswith(".png"):

            # 图像预处理

            image\_path = os.path.join(input\_folder, image\_file)

            image = Image.open(image\_path)

            # MindSpore没有transforms.Compose，需要手动进行转换

            image = image.resize((image.size[1] \* 4, image.size[0] \* 4))

            image = mindspore.Tensor(np.array(image).astype(np.float32) / 255.0)

            # 进行超分辨率处理

            output = model.predict(image)

            # 将输出转换为图像

            output\_image = Image.fromarray((output.asnumpy() \* 255).astype(np.uint8))

            output\_image\_path = os.path.join(output\_folder, image\_file)

            output\_image.save(output\_image\_path)

1. **实验结果与分析**

如下图所示。第一行为原分辨率图像，第二行是低分辨率图像，第三行为超分辨率算法处理后的图像。可以看出，虽然第三行的结果没有第一行的好，但是已经超过了第二行的结果。







经过测试得出原图像和处理过后的图像，指标结果如下：



在测试过程中，处理长宽近似的图像效果更好，但是处理细长的图片效果会变差，需要改进。