计算机视觉实践-练习 3

122106222784 贺梦瑶

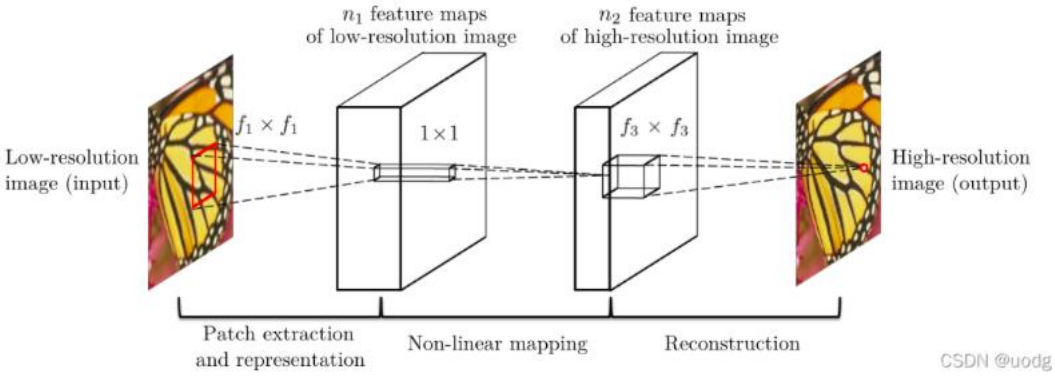
1、实验目标

实现 SRCNN 与 SRGAN 这两类图像超分辨率算法在 Set5 数据集上的测试，得到超分辨率图像，并进行分析。

2、模型介绍

(1) SRCNN

SRCNN 的结构仅用了三个卷积层，结构如下：



使用双三次插值方法对低分辨率图像进行缩小和放大，得到处理后的低分辨率图像（预处理）。

输入：处理后的低分辨率图像 Y Y Y

卷积层 1：kernelSize 为 9 × 9 9\times9 9×9

卷积层 2：kernelSize 为 1 × 1 1\times1 1×1

卷积层 3：kernelSize 为 5 × 5 5\times5 5×5

输出：高分辨率图像

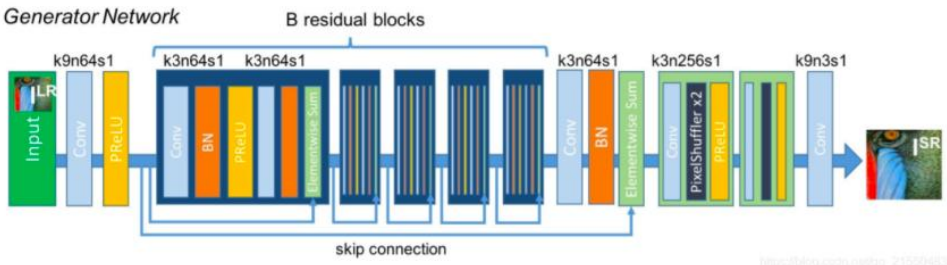
卷积层 1 特征块提取和表示：此操作从低分辨率图像 Y Y Y 中提取（重叠）特征块，并将每个特征块表示为一个高维向量。这些向量包括一组特征图，其数量等于向量的维数。

卷积层 2 非线性映射：该操作将每个高维向量非线性映射到另一个高维向量。每个映射向量在概念上都是高分辨率特征块的表示。这些向量同样包括另一组特征图。

卷积层 3 重建：该操作聚合上述高分辨率 patch-wise (介于像素级别和图像级别的区域) 表示，生成最终的高分辨率图像。

(2) SRGAN

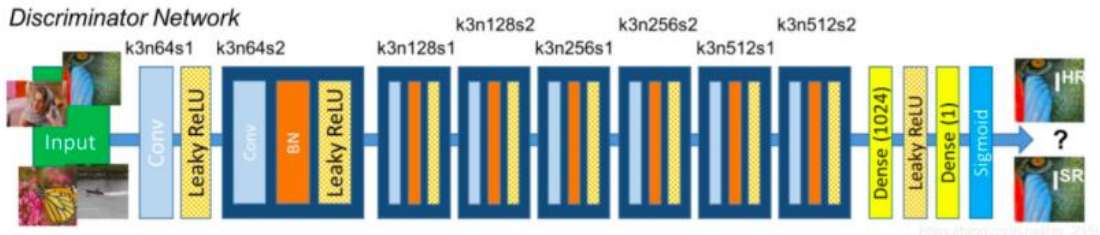
和其他对抗生成网络一样，SRGAN 有生成网络和辨别网络，生成网络：



其中 k 代表卷积核的尺寸，n 代表卷积输出的通道数，s 代表步长，不同指向的箭头表示残差结构，Elementwise Sun 就是残差中相加的操作。相同颜色表示相同的操作，低分辨率图片（LR）输入网络后输出高分辨率图片（HR）。跳跃连接有两个地方：1）在 block 内部

有 skip-connection；2）多个 block 也由 skip-connection 进行连接。

辨别网络：



辨别网络没有残差结构，其中的符号表示的意思和上面解释的一样，辨别网络输入一张图片，判断这张图片是原始高分辨率的图片还是生成网络输出的高分辨率图片。判别网络含有 8 个卷积层，从第 2 个卷积层开始，每一个卷积层后面加一个 BN 层来归一化中间层 feature map 的分布。判别器采用 stride=2 来降低分辨率。

SRGAN 由于要解决 4x 退化的问题，从低清到高清图像有更复杂的映射。所以它采用更深的网络来拟合这种映射。但是更深的网络有一个问题：不容易训练。

针对该问题，以上采用了两个方法缓解：1）添加 BN 层（Batch Normalization）；2）建立跳跃链接（skip-connection）+ residual block.

3、实现说明

(1) SRCNN

```
from torch import nn


class SRCNN(nn.Module):
    def __init__(self, num_channels=1):
        super(SRCNN, self).__init__()
        self.conv1 = nn.Conv2d(num_channels, 64, kernel_size=9, padding=9 // 2)
        self.conv2 = nn.Conv2d(64, 32, kernel_size=5, padding=5 // 2)
        self.conv3 = nn.Conv2d(32, num_channels, kernel_size=5, padding=5 // 2)
        self.relu = nn.ReLU(inplace=True)

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.relu(self.conv2(x))
        x = self.conv3(x)
        return x
```

(2) SRGAN

```
import numpy as np
import torch
import torch.nn as nn

def convolution_block(in_channels, out_channels, stride):
```

```
    return nn.Sequential(
      nn.Conv2d(in_channels, out_channels, 3, padding=1, stride=stride),
      nn.BatchNorm2d(out_channels),
      nn.LeakyReLU(0.2)
    )

def upsample_block(in_channels):
    return nn.Sequential(
      nn.Conv2d(in_channels, in_channels*4, 3, padding=1),
      nn.PixelShuffle(2),
      nn.PReLU(in_channels)
    )

class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
      super().__init__()
      self.conv1 = nn.Conv2d(in_channels, out_channels, 3, padding=1)
      self.bn1 = nn.BatchNorm2d(out_channels)
      self.relu = nn.PReLU(out_channels)
      self.conv2 = nn.Conv2d(out_channels, out_channels, 3, padding=1)
      self.bn2 = nn.BatchNorm2d(out_channels)

    def forward(self, x):
      out = self.conv1(x)
      out = self.bn1(out)
      out = self.relu(out)
      out = self.conv2(out)
      out = self.bn2(out)
      out = out + x
      return out

class Generator(nn.Module):
    def __init__(self, upscale_factor=4, num_blocks=16):
      super().__init__()
      num_upblocks = int(np.log2(upscale_factor))
      self.conv1 = nn.Conv2d(3, 64, 9, padding=4)
      self.relu = nn.PReLU(64)
```

```
  self.resblocks = nn.Sequential(*([ResidualBlock(64, 64)]* num_blocks))
  self.conv2 = nn.Conv2d(64, 64, 3, padding=1)
  self.bn = nn.BatchNorm2d(64)
  self.upblocks = nn.Sequential(*([upsample_block(64)]* num_upblocks))
  self.conv3 = nn.Conv2d(64, 3, 9, padding=4)

def forward(self, x):

  out = self.conv1(x)
  identity = self.relu(out)
  out = self.resblocks(identity)
  out = self.conv2(out)
  out = self.bn(out)
  out += identity
  out = self.upblocks(out)
  out = self.conv3(out)
  return torch.tanh(out)

class Discriminator(nn.Module):
 def __init__(self, crop_size = 128):
  super().__init__()

  num_ds = 4
  size_list = [64, 128, 128, 256, 256, 512, 512]
  stride_list = [1,2]*3
  self.conv1 = nn.Conv2d(3, 64, 3, padding=1)
  self.relu = nn.LeakyReLU(0.2)
  self.convblocks = nn.Sequential(convolution_block(64, 64, 2),
          *[convolution_block(in_ch, out_ch, stride)
          for in_ch, out_ch, stride in zip(size_list, size_list[1:], stride_list)])
  self.fc1 = nn.Linear(int(512*(crop_size/ 2**num_ds)**2), 1024)
  self.fc2 = nn.Linear(1024, 1)
  self.sig = nn.Sigmoid()

def forward(self, x):
```

```
        out = self.conv1(x)
        out = self.relu(out)
        out = self.convblocks(out)
        out = self.fc1(out.view(out.size(0),-1))
        out = self.relu(out)
        out = self.fc2(out)
        out = self.sig(out)

        return out
```

4、结果展示

(1) SRCNN

展示在 Set5 数据集中生成的图像：

每组图从左至右分别为原图，Bicubic 插值下采样后的图，经 SRCNN 处理后得到的图。

(3) SRGAN

展示在 Set5 数据集中生成的图像：

每组图从左至右分别为原图，Bicubic 插值下采样后的图，经 SRCNN 处理后得到的图。

总体上来说，SRGAN 最终生成的图像效果要优于 SRCNN，SRCNN 的效果易受到卷积核大小及数量的较大影响，每个样本的最终效果与卷积核有较大关系，而从实际上来说，对于每个样本进行参数调整并不实际，同时也受到训练数据集的影响。而生成对抗网络 (GAN) 是一种无监督学习的人工智能算法，SRGAN 用于从低分辨率图像创建高分辨率图像。它们通过让两个神经网络相互竞争来工作，可以帮助生成与原始高分辨率图像几乎无法区分的图像，并且它们还能够生成比放大图像更详细、噪点更少的图像，适用更普遍，不依赖于训练数据集。