计算机视觉与应用实践实验报告（三）

目录

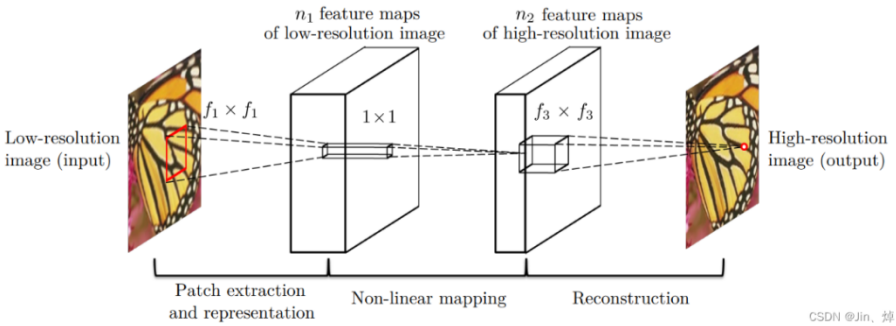
一、实验目的

- 1．实现 SRCNN 或其他一种基于逐像素损失的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.实现 SRGAN 或其他一种基于 GAN 的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。
- 3.对比两种类型的图像超分辨率方法在训练过程和生成图像质量上的不同，写一篇对比分析报告。
- 测试方式：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。

二、实验原理

2.1 SRCNN

SR，即 super resolution，即超分辨率。CNN 相对来说比较著名，就是卷积神经网络了。从名字可以看出，SRCNN 是首个应用于超分辨领域的卷积神经网络，如下图所示。



所谓超分辨率，就是把低分辨率(LR, Low Resolution)图片放大为高分辨率(HR, High Resolution)的过程。由于是开山之作，SRCNN 相对比较简单，总共分三步：

1、输入 LR 图像 X，经双三次(bicubic)插值，被放大成目标尺寸（如放大至 2 倍、3 倍、4 倍），得到 Y ，即低分辨率图像(Low-resolution image)

2、通过三层卷积网络拟合非线性映射

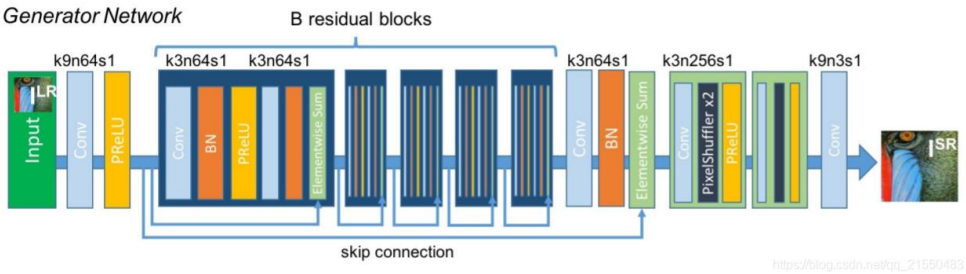
3、输出 HR 图像结果 F (Y)

2.1 SRGAN

网络结构

下面介绍 SRGAN 的网络结构，和其他对抗生成网络一样，SRGAN 有生成网络和辨别网

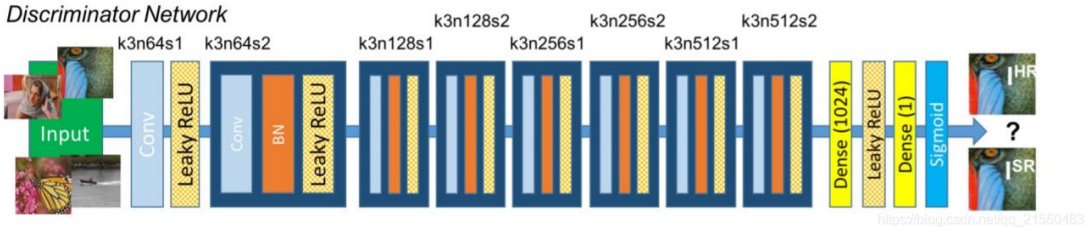
络，我们先看生成网络：



这是摘自原论文的网络结构图， 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）。

下面来看辨别网络：



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。

三、实验步骤

在 Scale=4 的条件下分别对 SRCNN 和 SRGAN 进行训练，其中 SRGAN 在 VOC2012 数据集上对生成器和鉴别器进行训练，SRCNN 在 91-image 上进行训练，取 f1=9, f2=5, f3=5, n1=64, n2=32, n3=1。最后 SRCNN 和 SRGAN 同在 Set5 上进行测试，计算 PSNR 并输出 bicubic 后的图像以及超分辨结果。

四、关键程序代码

1、SRCNN 模型代码

```
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 模型代码

```
class Generator(nn.Module):
    def __init__(self, scale_factor):
        upsample_block_num = int(math.log(scale_factor, 2))

        super(Generator, self).__init__()
        self.block1 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=9, padding=4),
            nn.PReLU()
        )
        self.block2 = ResidualBlock(64)
        self.block3 = ResidualBlock(64)
        self.block4 = ResidualBlock(64)
        self.block5 = ResidualBlock(64)
        self.block6 = ResidualBlock(64)
        self.block7 = nn.Sequential(
            nn.Conv2d(64, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64)
        )
        block8 = [UpsampleBLock(64, 2) for _ in range(upsample_block_num)]
        block8.append(nn.Conv2d(64, 3, kernel_size=9, padding=4))
        self.block8 = nn.Sequential(*block8)

    def forward(self, x):
        block1 = self.block1(x)
        block2 = self.block2(block1)
        block3 = self.block3(block2)
        block4 = self.block4(block3)
        block5 = self.block5(block4)
        block6 = self.block6(block5)
        block7 = self.block7(block6)
        block8 = self.block8(block1 + block7)

        return (torch.tanh(block8) + 1) / 2


class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.net = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, padding=1),
            nn.LeakyReLU(0.2),

            nn.Conv2d(64, 64, kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(0.2),
```

```
            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2),

            nn.Conv2d(128, 128, kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2),

            nn.Conv2d(128, 256, kernel_size=3, padding=1),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2),

            nn.Conv2d(256, 256, kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2),

            nn.Conv2d(256, 512, kernel_size=3, padding=1),
            nn.BatchNorm2d(512),
            nn.LeakyReLU(0.2),

            nn.Conv2d(512, 512, kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(512),
            nn.LeakyReLU(0.2),

            nn.AdaptiveAvgPool2d(1),
            nn.Conv2d(512, 1024, kernel_size=1),
            nn.LeakyReLU(0.2),
            nn.Conv2d(1024, 1, kernel_size=1)
        )

    def forward(self, x):
        batch_size = x.size(0)
        return torch.sigmoid(self.net(x).view(batch_size))

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

    def forward(self, x):
        residual = self.conv1(x)
```

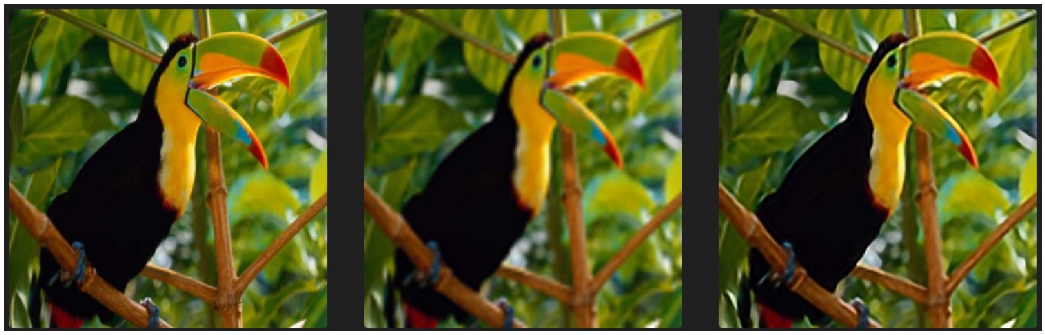
```
        residual = self.bn1(residual)
        residual = self.prelu(residual)
        residual = self.conv2(residual)
        residual = self.bn2(residual)

        return x + residual
class UpsampleBLock(nn.Module):
    def __init__(self, in_channels, up_scale):
        super(UpsampleBLock, self).__init__()
        self.conv = nn.Conv2d(in_channels, in_channels * up_scale ** 2, kernel_size=3,
padding=1)
        self.pixel_shuffle = nn.PixelShuffle(up_scale)
        self.prelu = nn.PReLU()

    def forward(self, x):
        x = self.conv(x)
        x = self.pixel_shuffle(x)
        x = self.prelu(x)
        return x
```

五、实验结果

1、SRCNN 在 Set5 上的结果（从左到右依次是原图、bicubic 后的图，SRCNN 结果图）

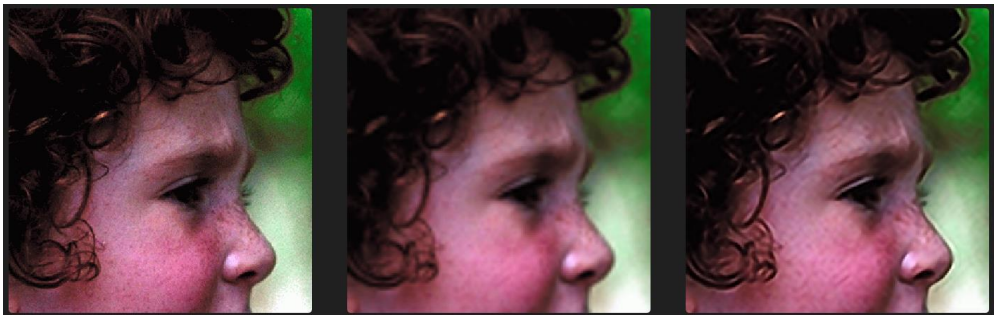
（1）baby PSNR:31.63

（2）bird PSNR:29.08

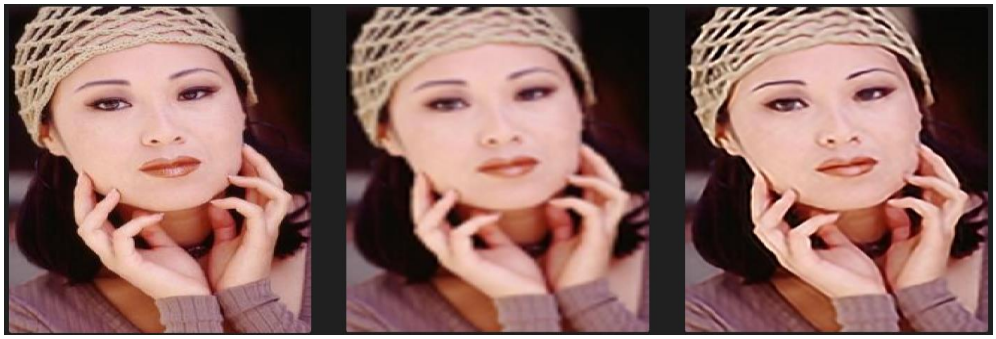


（3）butterfly PSNR:21.82

（4）head PSNR:33.69



（5）women PSNR:25.30

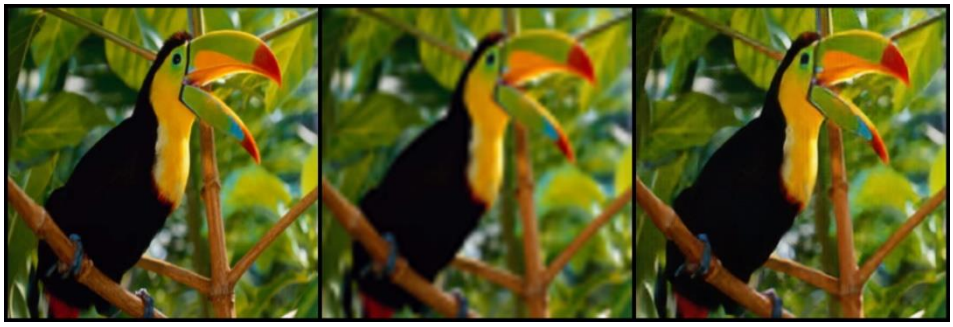


SRCNN（Super-Resolution CNN）首次提出将深度学习技术应用于 SR， SRCNN 结构很简单，但是效果很好。计算效率高，速度快。对于一个 LR 图像，首先使用 bicubic 插值到所需的尺寸得到图像 Y，然后学习一个映射 F，使得 F(Y)尽可能和 ground-truth 接近，整个训练过程非常简单。从上图（1）-（5）我们也可以明显看出，SRCNN 的效果虽然达不到原图的清晰度和细节程度，看上去边界有些锐利，和周围像素过渡不够自然，但在图像超分辨率领域已经是开创性的结果了。

2、 SRGAN 在 Set5 上的结果（从左到右依次是原图、bicubic 后的图，SRCNN 结果图）

（1）baby PSNR:31.00

（2）bird PSNR:29.84



（3）butterfly PSNR:24.96

（4）head PSNR:29.27

（5）women PSNR:27.74

SRGAN 把 GAN(Generative Adversarial Networks)，也就是生成式对抗神经网络应用到了图像高分辨率重建问题上。SRGAN 的网络结构由一个生成器(Generator Network)和一个判别器(Discriminator Network)组成。在 SRGAN 中，使用 SRResNet 作为生成网络，并将 MSE 作为优化目标，并提出了一种新的损失函数：感知损失。在使用 GAN 生成和判别的过程中，会不断地鼓励图像向真实图像所在的空间移动，这样产生的图像就会更加逼真。

从上面的实验结果我们可以看出，相较于 SRCNN 的结果，SRGAN 生成的图像边界过渡会更加自然，图像更加真实，但是从整体来看也达不到原图的清晰程度。

3、SRCNN 与 SRGAN 在 Set5 上的 PSNR 结果(scale=4)

	PSNR
SRCNN	**28.30**
SRGAN	**28.56**

从上表可以看出，SRCNN 和 SRGAN 的 PSNR 相差不大，但是他们生成的图片的效果差距并不一样，SRCNN 生成的图片边缘会更加锐利，而 SRGAN 的过渡会更加自然。

六、实验分析与总结

从 SRCNN 开始，超分辨领域在卷积神经网络下衍生了许许多多的经典方法。而作为开山鼻祖，SRCNN 直接学习低分辨率和高分辨率图像之间的端到端映射，除了优化之外，几乎不进行任何前、后处理。证明了深度学习在超分辨率的经典计算机视觉问题中是有用的，并且可以获得良好的质量和速度。

深度卷积神经网络在单个图像实现超分辨率在速度和精度上取得了突破，但是仍然存在一个核心问题：当在放大因子下的超分辨率时，如何恢复细小的纹理细节？基于这些问题：SRGAN 提出了一种将生成对抗网络用于图像 SR 的新模型，并且根据 GAN 网络结构提出了一种新的视觉损失(perceptual loss)。在之前的工作中，双三次插值、SRCNN、SRResNet 等网络的目标函数主要集中在最小化均方（MSE）重建误差，由此产生的结果具有较高的峰值信噪比（PSNR），但它们通常缺乏高频细节，在感知上不令人满意。MSE（和 PSNR）捕捉感知相关差异（如高纹理细节）的能力非常有限，因为它们是基于像素级图像差异定义的。有时较高的 PSNR 不一定反映出感知上更好的超分辨结果，PSNR 其实在 SR 领域作为一个评价指标来说是有一定局限性的，这也和我们的实验结果是相符的。