图像超分辨率实验分析报告

目录

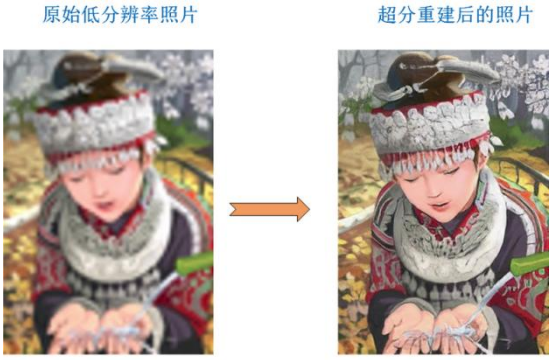
一、实验目的

实现 SRCNN 和 SRGAN 在 Set5 数据集上的测试和对比分析。

二、实验原理

1、图像超分辨率重建

图像超分辨率重建技术指的是将给定的低分辨率图像通过特定的算法恢复成相应的高分辨率图像，是利用数字图像处理、计算机视觉等领域的相关知识，借由特定的算法和处理流程，从给定的低分辨率图像中重建出高分辨率图像的过程。旨在克服或补偿由于图像采集系统或采集环境本身的限制，导致的成像图像模糊、质量低下、感兴趣区域不显著等问题。



按照时间和效果进行分类，可以将超分辨率重建算法分为传统算法和深度学习算法两类。

1.1 传统超分辨率重建算法

传统的超分辨率重建算法主要依靠基本的数字图像处理技术进行重建，常见有如下几类：

（1） 基于插值的超分辨率重建

将图像上每个像素看做图像平面上的一个点，对超分辨率图像的估计可以看做利用已知的像素信息为平面上未知的像素信息进行拟合，由一个预定义的变换函数或者插值核完成。基于插值的方法计算简单、易于理解，但是也存在一些明显的缺陷。

首先，假设像素灰度值的变化是一个连续的、平滑的过程，但实际上这种假设并不完全成立；其次，在重建过程中，仅根据一个事先定义的转换函数来计算超分辨率图像，不考虑图像的降质退化模型，往往会导致复原出的图像出现模糊、锯齿等现象。

常见的基于插值的方法包括最近邻插值法、双线性插值法和双立方插值法等。

（2） 基于退化模型的超分辨率重建

从图像的降质退化模型出发，假定高分辨率图像经过适当的运动变换、模糊及噪声得到低分辨率图像。这种方法通过提取低分辨率图像中的关键信息，并结合对未知的超分辨率图像的先验知识来约束超分辨率图像的生成。

常见的方法包括迭代反投影法、凸集投影法和最大后验概率法等。

（3） 基于学习的超分辨率重建

利用大量的训练数据，从中学习低分辨率图像和高分辨率图像之间某种对应关系，然后根据学习到的映射关系来预测低分辨率图像所对应的高分辨率图像，从而实现图像的超分辨率重建过程。常见的基于学习的方法包括流形学习、稀疏编码方法。

1.2 基于深度学习的超分辨率重建算法

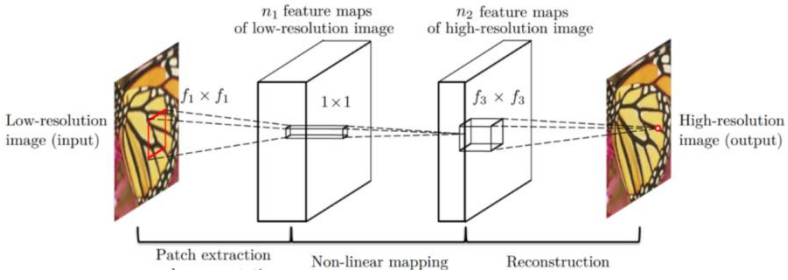
2014 年，Dong 等首次将深度学习应用到图像超分辨率重建领域，使用一个三层的卷积神经网络学习低分辨率图像与高分辨率图像之间的映射关系，网络模型名为 SRCNN。

SRCNN 采用插值，先将低分辨率图像放大，再通过模型复原。Shi 等人认为这种预先采用近邻插值的方式影响性能，如果从源头出发，应该从样本中去学习如何进行放大，基于此原理提出 ESPCN 算法。该算法在将低分辨率图像送入神经网络之前，无需对给定的低分辨率图像进行一个上采样过程，而是引入一个亚像素卷积层间接实现图像的放大过程。极大降低 SRCNN 的计算量，提高重建效率。

SRCNN 和 ESPCN 均使用 MSE 作为目标函数来训练模型。2017 年，Christian Ledig 等人从照片感知角度出发，通过对抗网络进行超分重建。由于 MSE 损失函数会导致重建的图像过于平滑，缺乏感官上的照片真实感。他们改用生成对抗网络进行重建，定义新的感知目标函数，算法命名为 SRGAN，由一个生成器和一个判别器组成。生成器负责合成高分辨率图像，判别器用于判断给定的图像来自生成器还是真实样本。通过博弈的对抗过程，使得生成器能够将给定的低分辨率图像重建为高分辨率图像。SRGAN 中同时提出 SRResNet 对比算法，采用 MSE 损失函数，但 SRResNet 采用足够深的残差卷积网络模型，效果更好。

2、SRCNN

2.1 网络模型



图像特征提取层：通过 CNN 将图像 Y 的特征提取出来存到向量中。用一层的 CNN 以及 ReLU 去将图像 Y 变成一堆堆向量，即 feature map。——$F_1(Y) = \max(0,W_1Y+B_1)$

非线性映射层：提取到的特征进一步非线性映射——$F_2(Y) = \max(0,W_2F_1(X) +B_2)$

重建层：结合前面得到的补丁来产生最终的高分辨率图像——$F(Y) = W_3F_2(Y) +B_3$

2.2 实验步骤

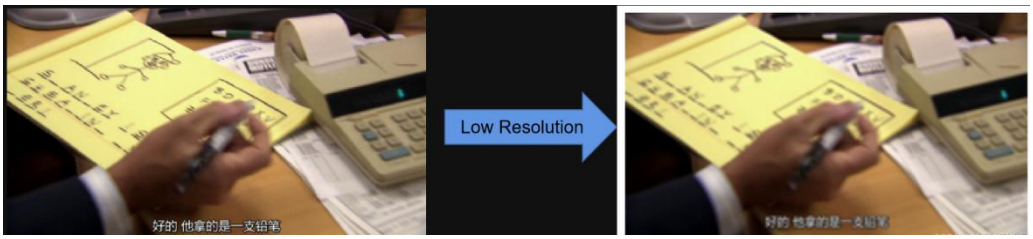
（1）输入高分辨率图像 X（Ground-Truth），经双三次(bicubic)插值，放大成目标尺寸（如 2、3、4 倍），得到 Y，即低分辨率图像 LR(Low-resolution image)；

（2）通过三层卷积网络拟合非线性映射；

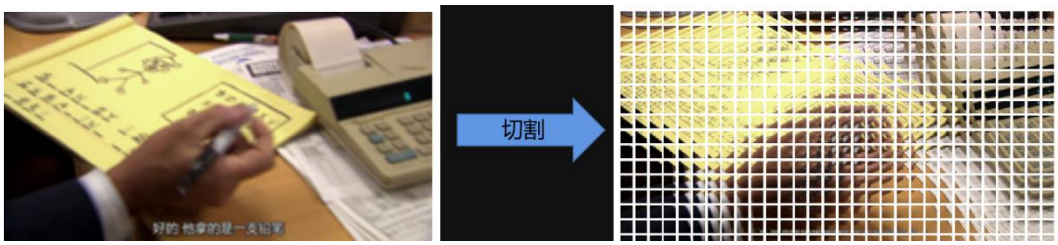
（3）输出 HR 图像结果 F (Y) 。通过优化 F(Y)和 Ground-Truth 的 loss 学习函数 F (·)。

2.3 训练过程

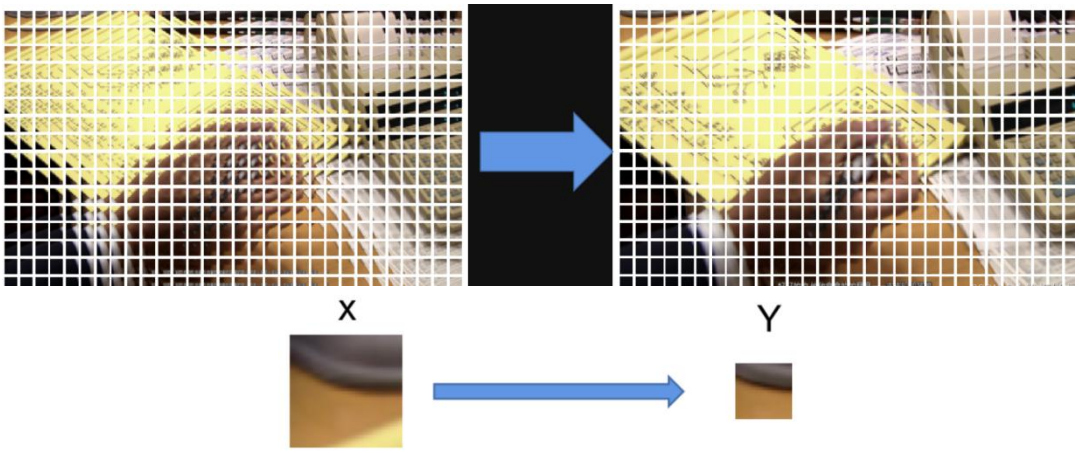
（1）降低分辨率



（2）切割图片，补丁之间有重复



（3）训练模型，学习低分辨率 → 高分辨率的映射关系



（4）SRCNN 训练过程

● 构建训练集，含有低分辨率和高分辨图像，图像需要将其从 RGB 图像转为 YCBCR 图像，并对图像进行分割为小块进行存储，高分辨率图像为未下采样前的图像，低分辨率图像为下采样，上采样后的图像。

● 构建 SRCNN 模型，即三层卷积模型，设置 MES 为损失函数，因为 MES 与评价图像

客观指标 PSNR 计算相似，即最大化 PSNR。设置其余常见的神经网络参数（学习率，Batch_size，num-epochs 等）。

● 训练模型 SRCNN，即学习低分辨率图像到高分辨率图像的映射关系。根据不同参数的不同 PSRN 值，保留最大 PSNR 值对应的模型参数。

2.4 一些参数

（1） 损失函数：MES（均方误差），因为 MSE 的格式和图像失真评价指标 PSNR 很像。公式如下，其中 F(Y; θ)是得到的超分辨率图像，X 是原高分辨率图像；

$$L(\theta)=\frac{1}{n}\sum_{i=1}^{n}\|F(Y_i;\theta)-X_i\|^2$$

（2）激活函数：Relu

（3）PSNR：峰值信噪比，一种评价图像的客观标准，具有局限性，一般是用于最大值信号和背景噪音之间的一个工程项目。

MSE 与 PSNR 公式对比：

$$MSE=\frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j)-K(i,j)]^2$$

$$PSNR=10\cdot log_{10}(\frac{MAX_I^2}{MSE})$$

（4）SSIM（另外一种衡量结果的参数）

$SSIM$ 公式基于样本 x 和 y 之间的三个比较衡量：亮度 (luminance)、对比度 (contrast) 和结构 (structure)。

$$l(x,y)=\frac{2\mu_x\mu_y+c_1}{\mu_x^2+\mu_y^2+c_1}\quad c(x,y)=\frac{2\sigma_x\sigma_y+c_2}{\sigma_x^2+\sigma_y^2+c_2}\quad s(x,y)=\frac{\sigma_{xy}+c_3}{\sigma_x\sigma_y+c_3}$$

一般取 $c_3=c_2/2$。

- μ_x 为 x 的均值
- μ_y 为 y 的均值
- σ_x^2 为 x 的方差
- σ_y^2 为 y 的方差
- σ_{xy} 为 x 和 y 的协方差
- $c_1=(k_1L)^2, c_2=(k_2L)^2$ 为两个常数，避免除零
- L 为像素值的范围，2^B-1
- $k_1=0.01, k_2=0.03$ 为默认值

那么

$$SSIM(x,y)=[l(x,y)^\alpha\cdot c(x,y)^\beta\cdot s(x,y)^\gamma]$$

将 α,β,γ 设为 1，可以得到

$$SSIM(x,y)=\frac{(2\mu_x\mu_y+c_1)(2\sigma_{xy}+c_2)}{(\mu_x^2+\mu_y^2+c_1)(\sigma_x^2+\sigma_y^2+c_2)}$$

3、SRGAN

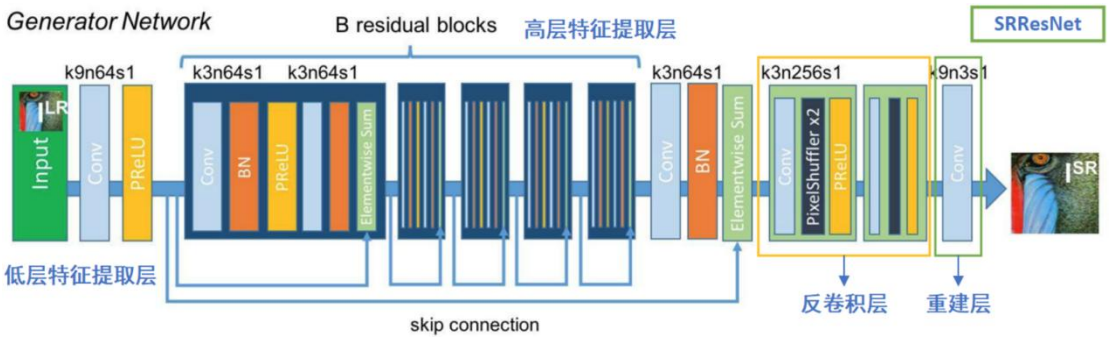
SRGAN 提供一种新的 Loss function——perceptual loss(感知损失)，之前的 SR 是由 MSE 损失函数教会网络如何实现 LR→HR，MSE 的不足：对图像的细节进行平滑，使得重建的图像虽然有很高的 PSNR，但失去了人肉眼感知的高分辨率感，即 Photo-Realistic。

SRGAN 实现在 up-scale factor 较大情况下，图像能有更多的重建细节，在 PSNR 并不高的情况下，产生让人肉眼感官更加舒适的 HR 图像。

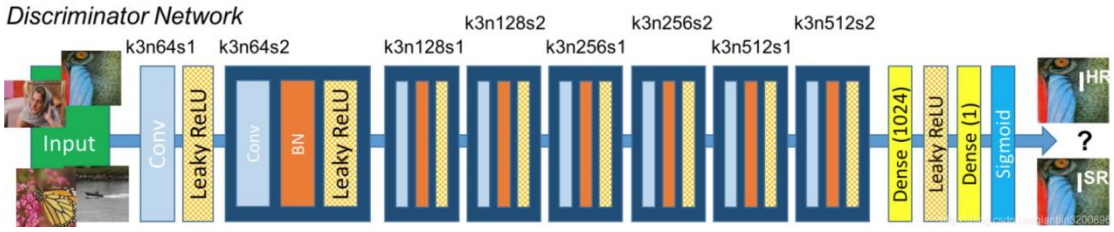
3.1 网络结构

k 代表卷积核尺寸，n 代表卷积输出通道数，s 代表步长，不同指向的箭头表示残差结构，Elementwise Sun 是残差中相加的操作。

生成网络——低分辨率图片（LR）输入网络后输出高分辨率图片（HR）。



判别网络——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。



3.2 损失函数

SRGAN 生成的网络损失函数为感知损失，由 content loss 和 adversarial loss 两部分组成。

$$l^{SR} = \underbrace{\underbrace{l_{\mathrm{X}}^{SR}}_{\text{content loss}} + \underbrace{10^{-3} l_{Gen}^{SR}}_{\text{adversarial loss}}}_{\text{perceptual loss (for VGG based content losses)}}$$

content loss 是生成的 HR 和真实的 HR 通过 VGG 网络前 16 层得到的特征之间的 MSE 损失——content loss = MSE（VGG(G(LR)），VGG(HR)）

adversarial loss 公式如下，G 表示辨别器判断生成图片为真实高分辨率图片的概率。

$$l_{Gen}^{SR} = \sum_{n=1}^{N} -\log D_{\theta_D}(G_{\theta_G}(I^{LR}))$$

正则项

$$l_{TV}^{SR} = \frac{1}{r^2 WH} \sum_{x=1}^{rW} \sum_{y=1}^{rH} ||\nabla G_{\theta_G}(I^{LR})_{x,y}||$$

三．实验测试

1、SRCNN

（1）设置参数（训练好的权重，图片，放大倍数）

```
# 设置权重参数目录，处理图像目录，放大倍数
parser = argparse.ArgumentParser()
parser.add_argument('--weights-file', default='outputs/x3/srcnn_x4.pth', type=str)
parser.add_argument('--image-file', default='data/Set5/woman.png', type=str)
parser.add_argument('--scale', type=int, default=4)
args = parser.parse_args()
```

weights-file 是权重路径，实验测试了三种权重文件内 srcnn_x2.pth、srcnn_x3.pth、srcnn_x4.pth，分别对应 scale 为 2、3、4，即插值操作所对应的尺度。

（2）创建 SRCNN 模型，给模型赋值最优参数

```
#  Benchmark模式会提升计算速度
cudnn.benchmark = True
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

model = SRCNN().to(device)   # 新建一个模型

state_dict = model.state_dict()  # 通过 model.state_dict()得到模型有哪些 parameters and persistent
# torch.load('tensors.pth', map_location=lambda storage, loc: storage)  使用函数将所有张量加载到CPU
for n, p in torch.load(args.weights_file, map_location=lambda storage, loc: storage).items():
    if n in state_dict.keys():
        state_dict[n].copy_(p)
    else:
        raise KeyError(n)

model.eval()   # 切换为测试模式 ，取消dropout

image = pil_image.open(args.image_file).convert('RGB')   # 将图片转为RGB类型
```

（3）对图像进行插值得到低分辨率图像

```
# 经过一个插值操作，首先将原始图片重设尺寸，使之可以被放大倍数scale整除
# 得到低分辨率图像Lr，即三次插值后的图像，同时保存输出
image_width = (image.width // args.scale) * args.scale
image_height = (image.height // args.scale) * args.scale
image = image.resize((image_width, image_height), resample=pil_image.BICUBIC)
image = image.resize((image.width // args.scale, image.height // args.scale), resample=pil_image.BICUBIC)
image = image.resize((image.width * args.scale, image.height * args.scale), resample=pil_image.BICUBIC)
image.save(args.image file.replace('.', ' bicubic x{}.'.format(args.scale)))
```

（4）对 Lr 低分辨率图像的 y 颜色空间进行训练

```
# 将图像转化为数组类型，同时图像转为ycbcr类型
image = np.array(image).astype(np.float32)
ycbcr = convert_rgb_to_ycbcr(image)
# 得到 ycbcr中的 y 通道
y = ycbcr[..., 0]
y /= 255.  # 归一化处理
y = torch.from_numpy(y).to(device)  # 把数组转换成张量，且二者共享内存，对张量进行修改比如重新赋值，那么原始数组也会相应发生改变
y = y.unsqueeze(0).unsqueeze(0)  # 增加两个维度
# 令reqires_grad自动设为False，关闭自动求导
# clamp将inputs归一化为0到1区间
with torch.no_grad():
    preds = model(y).clamp(0.0, 1.0)
```

（5）计算 PSNR 值并输出

```
psnr = calc_psnr(y, preds)   # 计算y通道的psnr值
print('PSNR: {:.2f}'.format(psnr))  # 格式化输出PSNR值
```

（6）将转换为图像并进行输出

```
# 1.mul函数类似矩阵.*，即每个元素×255
# 2. *.cpu（）.numpy（）  将数据的处理设备从其他设备（如gpu拿到cpu上），不会改变变量类型，转换后仍然是Tensor变量，同时将Tensor转化
# 3. *.squeeze(0).squeeze(0)数据的维度进行压缩
preds = preds.mul(255.0).cpu().numpy().squeeze(0).squeeze(0)  # 得到的是经过模型处理，取值在[0,255]的y通道图像

# 将img的数据格式由（channels,imagesize,imagesize）转化为（imagesize,imagesize,channels），进行格式的转换后方可进行显示。
output = np.array([preds, ycbcr[..., 1], ycbcr[..., 2]]).transpose([1, 2, 0])

# 将图像格式从ycbcr转为rgb，限制取值范围[0,255]，同时矩阵元素类型为uint8类型
output = np.clip(convert_ycbcr_to_rgb(output), 0.0, 255.0).astype(np.uint8)
output = pil_image.fromarray(output)   # array转换成image，即将矩阵转为图像
output.save(args.image_file.replace('.', '_srcnn_x{}.'.format(args.scale)))  # 对图像进行保存
```

2、SRGAN

（1）设置参数

```
# 模型参数
large_kernel_size = 9    # 第一层卷积和最后一层卷积的核大小
small_kernel_size = 3    # 中间层卷积的核大小
n_channels = 64          # 中间层通道数
n_blocks = 16            # 残差模块数量
scaling_factor = 4       # 放大比例
ngpu = 1                 # GP数量
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

（2）创建 SRCNN 模型，赋予权重

```
# 预训练模型
srgan_checkpoint = "./results/checkpoint_srgan.pth"
# srresnet_checkpoint = "./results/checkpoint_srresnet.pth"

# 加载模型SRResNet 或 SRGAN
checkpoint = torch.load(srgan_checkpoint)
generator = Generator(large_kernel_size=large_kernel_size,
                      small_kernel_size=small_kernel_size,
                      n_channels=n_channels,
                      n_blocks=n_blocks,
                      scaling_factor=scaling_factor)
generator = generator.to(device)
generator.load_state_dict(checkpoint['generator'])
```

（3）加载测试数据集

```
# 定制化数据加载器
test_dataset = SRDataset(data_folder,
                         split='test',
                         crop_size=0,
                         scaling_factor=4,
                         lr_img_type='imagenet-norm',
                         hr_img_type='[-1, 1]',
                         test_data_name=test_data_name)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=1, shuffle=False, num_workers=1,
                                          pin_memory=True)
```

（4）对测试集的每张数据进行模型推理

```
# 逐批样本进行推理计算
for i, (lr_imgs, hr_imgs) in enumerate(test_loader):

    # 低分辨率图像和高分辨率图像数据移至默认设备
    lr_imgs = lr_imgs.to(device)  # (batch_size (1), 3, w / 4, h / 4), imagenet-normed
    hr_imgs = hr_imgs.to(device)  # (batch_size (1), 3, w, h), in [-1, 1]

    # 前向传播.
    sr_imgs = model(lr_imgs)  # (1, 3, w, h), in [-1, 1]
    sr_imgs1 = convert_image(sr_imgs.squeeze(0).cpu().detach(), source='[-1, 1]', target='pil')
    sr_imgs1.save('./results/set5/test_ori{}.jpg'.format(i))
    hr_imgs1 = convert_image(hr_imgs.squeeze(0).cpu().detach(), source='[-1, 1]', target='pil')
    hr_imgs1.save('./results/set5/test_inf{}.jpg'.format(i))
```

（5）计算 PSNR 和 SSIM

```
# 计算 PSNR 和 SSIM
sr_imgs_y = convert_image(sr_imgs, source='[-1, 1]', target='y-channel').squeeze(0)  # (w,h)in y-channel
hr_imgs_y = convert_image(hr_imgs, source='[-1, 1]', target='y-channel').squeeze(0)  # (w,h)in y-channel
psnr = peak_signal_noise_ratio(hr_imgs_y.cpu().numpy(), sr_imgs_y.cpu().numpy(), data_range=255.)
ssim = structural_similarity(hr_imgs_y.cpu().numpy(), sr_imgs_y.cpu().numpy(), data_range=255.)
PSNRs.update(psnr, lr_imgs.size(0))
SSIMs.update(ssim, lr_imgs.size(0))

# 输出平均PSNR和SSIM
print('PSNR  {psnrs.avg:.3f}'.format(psnrs=PSNRs))
print('SSIM  {ssims.avg:.3f}'.format(ssims=SSIMs))
print('平均单张样本用时  {:.3f} 秒'.format((time.time()-start)/len(test_dataset)))
```

四、实验结果

1、数据集 Set5 由以下五张图像组成

2、SRCNN 测试结果

SRCNN 分别使用三种 scale 得到的权重进行 Set5 数据集的测试，每张图像得到的 PSNR 和数据集的平均值如下表所示。由此看出，scale=2 得到的 PSNR 值更高，scale 越小，原图损失的信息越少，最终重建效果越好。

	srcnn_x2	srcnn_x3	srcnn_x4
baby	41.20	37.89	35.50
bird	38.32	35.31	33.52
butterfly	29.55	27.53	25.93
head	41.27	39.70	38.87
woman	34.62	31.72	30.21
mean	36.992	34.43	32.806

以 baby 图像为例，下面是原图分别经过 scale=2,3,4 的三次 bicubic 三次插值结果，可以看出，scale 值越大，得到的图像越模糊。

baby

baby_bicubic_x2

baby_bicubic_x3

baby_bicubic_x4

下面是三种模型下的重建结果，scale 值越小，重建的结果质量越好，图像越清晰。

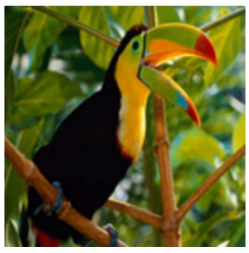
baby

baby_srcnn_x2

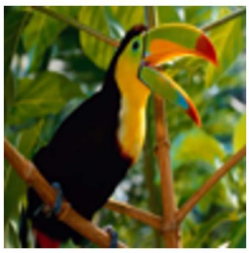
baby_srcnn_x3

baby_srcnn_x4

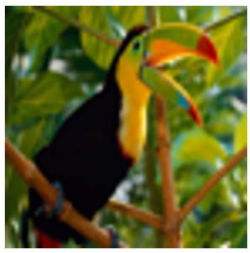
bird



bird_bicubic_x2

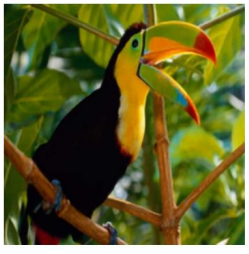


bird_bicubic_x3

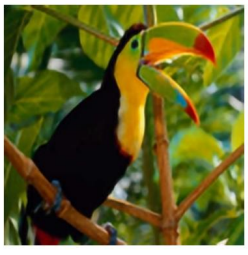


bird_bicubic_x4

bird



bird_srcnn_x2



bird_srcnn_x3

bird_srcnn_x4

butterfly

butterfly_bicubic_x2

butterfly_bicubic_x3

butterfly_bicubic_x4

butterfly

butterfly_srcnn_x2

butterfly_srcnn_x3

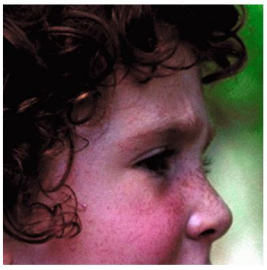
butterfly_srcnn_x4

head

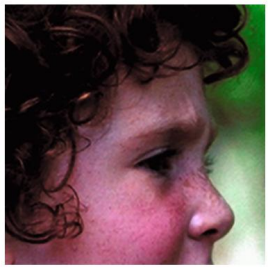
head_bicubic_x2

head_bicubic_x3

head_bicubic_x4



head



head_srcnn_x2

head_srcnn_x3

head_srcnn_x4




woman　　woman_bicubic_x2　　woman_bicubic_x3　　woman_bicubic_x4




woman　　woman_srcnn_x2　　woman_srcnn_x3　　woman_srcnn_x4

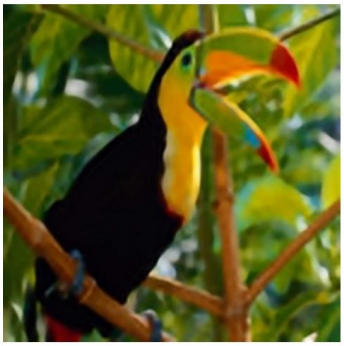
3、SRGAN 测试结果

SRGAN 的 PSNR 值不是很高，但其重建图像细节更好，不会过于平滑。以下是 scale=4 条件下，SRCNN 和 SRGAN 的实验对比结果，可以看出 SRGAN 的重建效果更好，细节更清晰。



baby　　baby_srgan_x4　　baby_srcnn_x4





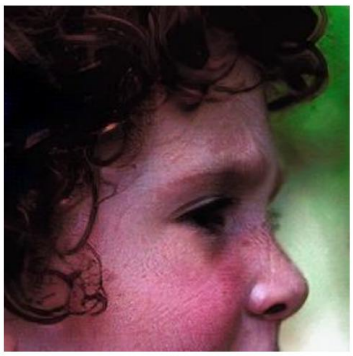
bird　　bird_srgan_x4　　bird_srcnn_x4

butterfly

butterfly_srgan_x4

butterfly_srcnn_x4

head



head_srgan_x4

head_srcnn_x4

woman

woman srgan x4

woman srcnn x4