计算机视觉应用与实践实验报告

目录

计算机视觉应用与实践实验报告 1
一、实验目的 1
二、 数据集与评估准则 1
2.1 Set5 数据集 1
2.2 评估准则 2
三、 SRCNN 图像超分辨率算法 3
3.1 SRCNN 模型基本原理 3
3.2 关键代码实现 4
3.3 实验结果 7
四、 SRGAN 图像超分辨率算法 8
4.1 SRGAN 模型基本原理 8
4.2 关键代码实现 11
4.3 实验结果 12
五、 对比分析 13
5.1 训练过程 13
5.2 图片生成质量 14
六、 实验总结与分析 14

一、实验目的

- 实现 SRCNN 图像超分辨率算法，在 Set5 数据集上测试，得到超分辨率图像，并进行分析。
- 实现 SRGAN 图像超分辨率算法，在 Set5 数据集上测试，得到超分辨率图像，并进行分析。
- 对比这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。

二、数据集与评估准则

2.1 Set5 数据集

Set5 数据集是用于图像超分辨率算法评估的常用基准测试数据集之一，由 Peter 等人于 2012 年发布。它包含了 5 张标准测试图像，分别是 baby，bird，butterfly，head 和 woman。这些图像分别涵盖了人脸、动物、自然风景等场景，具有不同的结构、纹理和细节，是测试算法性能和比较不同算法效果的常用数据集。

本次实验使用 Set5 数据集作为测试数据集，这些图像均为 RGB 彩色图像，

分辨率为 512*512 像素。使用之前要对该数据集做预处理。首先，使用 Bicubic 插值进行下采样，再分别使用 SRCNN 和 SRGAN 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。

下图 2.1 是 Set5 数据集中的五张图片展示。

图 2.1 Set5 数据集展示

2.2 评估准则

对于重构后的图像质量评价标准可以分为主观评价和客观评价。主观评价标准注重人眼视觉感受需要配备专业和非专业的人员来测试，主要是平均意见得分（Mean Opinion Score，MOS）。客观评价标准是由结构相似性（Structural Similarity Index Measure，SSIM）和峰值信噪比（Peak Signal to Noise Ratio，PSNR）两种指标组成。

平均意见得分（Mean Opinion Score，MOS）是一种被广泛使用的主观评价方法，旨在通过询问受试者的主观感受，从而评估图片质量。在 MOS 评分标准中，受试者会被要求对所评估的图片进行主观感受的评估，评估过程通常基于以下五个级别的分数：5（优秀）、4（良好）、3（一般）、2（差）、1（非常差）。每个受试者会对同一组图片样本进行多次评估，取平均值得到 MOS 分数。MOS 评分标准通常使用一个 0 到 5 的分数范围，其中 0 表示无法评价，5 表示最佳质量。MOS 分数越高，表示图片质量越好。

峰值信躁比(Peak Signal to Noise Ratio，PSNR)表示重构后的图像与真实图像之间像素间的差值，是图像质量评价方法之一，单位是分贝（dB），分贝值越大表示重构后的图像越接近真实图像。PSNR 的定义公式如下所示。

$$\mathrm{MSE} = \frac{1}{M \times N}\sum_{i=1}^{M}\sum_{j=1}^{N}(x_{ij} - y_{ij})^2$$

$$\mathrm{PSNR} = log(\frac{L^2}{MSE})$$

其中 MSE 指的是原始图像与重构图像的均方误差(Mean Square Error, MSE)，M×N 表示图像的尺寸，x_{ij}和y_{ij}分别表示重构图像和真实图像对应像素点的像素值，i、j 分别代表图像的行和列，L 是图像像素值的最大值。

结构相似指数(Structural Similarity Index Measure，SSIM）是一种衡量两幅图像相似度的图像评价指标。通过建模计算图像的亮度、对比度和结构。结构相似性的范围为[0,1]，值越大表示两幅图像结构相似程度越高，当 SSIM 的值等于 1 时表示两张图像完全一样。SSIM 的定义如下述公式所示。

$$SSIM(x,y)=\frac{(2u_xu_y+C_1)(2\sigma_{xy}+C_2)}{(u_x^2+u_y^2+C_1)(\sigma_x^2+\sigma_y^2+C_2)}$$

其中，u_x表示原始高分辨率图像的灰度平均值，u_y表示重构图像的灰度平均值，σ_x表示原始高分辨率图像的方差，σ_y表示重构图像的方差，σ_{xy}表示原始图像和重构图像的协方差，C_1和C_2使用下述两个公式表示，通常参数k_1=0.01，k_2=0.03，L=255。

$$C_1=k_1\times L$$

$$C_2=k_2\times L$$

三、SRCNN 图像超分辨率算法

3.1 SRCNN 模型基本原理

SRCNN（Super-Resolution Convolutional Neural Network）模型是香港中文大学 Dong 等人首次将卷积神经网络应用于单张图像超分辨率重构上。该模型框架如图 3.1 所示。

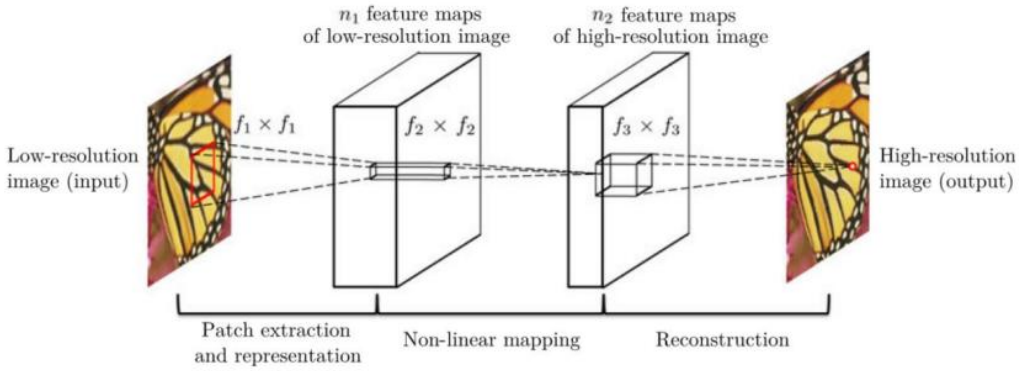

图 3.1 SRCNN 模型结构

SRCNN 是端到端的超分算法，所以在实际应用中不需要任何人工干预或者多阶段的计算。SRCNN 网络包含三个模块：Patch extraction and representation（图像块提取）、Non-linear mapping（非线性映射）、Reconstruction（高分辨率重构）。这三个模块对应三个卷积操作，接下来分别进行分析：

(1) 图像块提取。首先利用双三次插值的方法将图像放大到目标尺寸。此时，放大到目标尺寸的图像称为低分辨率图像，即上图中的输入（ input），假设输入图像为 Y，X 为原始图像，则第一层卷积神经网络可以用下述公式表示。

$$F_1(Y)=max(0,W_1*Y+B_1)$$

其中W_1和 B_1分别代表滤波器和偏置，*表示卷积操作，W_1的大小为$c\times f_1\times f_1\times n_1$，$c$ 是输入图像中的通道数，f_1是滤波器的空间大小，n_1是滤波器的数量。W_1在图像上应用n_1个卷积，每个卷积的内核大小为$c\times f_1\times f_1$。输出由n_1个特征映射组成。B_1代表n_1个偏置。卷积层之后使用 ReLU 激活函数来激活。最终训练之后调整的参数为f_1= 9, n_1= 64。

（2）非线性映射。卷积神经网络对于第一层生成的n_1个特征图映射为n_2个特征图，通过非线性映射的方法可以将特征向量从低分辨率空间映射到高分辨率空间上。计算方法如下述公式所示。

$$F_2(Y) = max(0, W_2 * F_1(Y) + B_2)$$

其中W_2和B_2分别代表滤波器和偏置，*代表卷积操作，W_2代表 n_2个$f_1 \times f_1 \times n_1$过滤器。$B_2$代表 n_2个偏置。f_2 =1， n_1 =32。

（3）高分辨率重构。利用前面得到的特征图来生成最后的高分辨图像，计算过程如下述公式所示。

$$F(Y) = W_3 * F_2(Y) + B_3$$

其中W_3表示$n_2 \times f_3 \times f_3$个过滤器，$B_3$表示 c 个偏置，经过实验和设计得到c=3,f_3=5。

对于 SRCNN 的训练过程，学习端到端的映射函数 F 需要评估参数$\Theta = \{W_1, W_2, W_3, B_1, B_2, B_3\}$，重构函数$F(\Theta, Y)$与高分辨率图像 X 之间的损失，是的损失函数达到最小值，给出一组高分辨率图像$\{X_i\}$和对应的低分辨率图像$\{Y_i\}$，使用均方误差 MSE 作为损失函数，得到损失如下述公式所示。

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

其中，n 为训练样本数，损失的最小化使用随即梯度下降法和标准的 BP 算法进行反向传播。使用 MSE 作为损失函数有利于得到较高的 PSNR 值。

3.2 关键代码实现

本节介绍具体的关键代码实现。

下采样：

```python
image_dir = "./dataset/Set5/"
# 获取文件夹中图片名

image_name = os.listdir(image_dir)
image = []
image_resize = []

save_dir = "./dataset/Set5_downscale_4/"
for i in range(len(image_name)):
    # image_name.append(image_path[i].split('/')[2][5:-4])
    image_path = image_dir + image_name[i]
    image.append(cv2.cvtColor(cv2.imread(image_path), cv2.COLOR_BGR2RGB))
    # 进行Bicubic插值下采样，缩小3倍
    image_resize.append(cv2.resize(image[i], None, fx=1/4, fy=1/4, interpolation=cv2.INTER_CUBIC))
    # cv2.imwrite(save_dir+image_name[i], cv2.cvtColor(image_resize[i], cv2.COLOR_RGB2BGR))

f, ax = plt.subplots(2, 5, figsize=(20, 10))
for i in range(5):
    ax[0, i % 5].imshow(image[i])
    ax[0, i % 5].set_title(image_name[i], y=-0.2)
    ax[1, i % 5].imshow(image_resize[i])
    ax[1, i % 5].set_title('Bicubiced  ' + image_name[i], y=-0.2)
plt.show()
```

图 3.2 Bicubic 下采样

首先对原始数据集使用 Bicubic 插值进行下采样，将图片缩小三倍。

搭建模型：

```python
import torch.nn as nn

# f1 = 9, f2 = 5, f3 = 5, n1 = 64, n2 = 32
# 输入是y通道 in_channel = out_channel = 1
class SRCNN(nn.Module):
    def __init__(self):
        super(SRCNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=64, kernel_size=9, padding=4)
        self.conv2 = nn.Conv2d(in_channels=64, out_channels=32, kernel_size=5, padding=2)
        self.conv3 = nn.Conv2d(in_channels=32, out_channels=1, kernel_size=5, padding=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
```

图 3.3 SRCNN 模型搭建

从图 3.3 可以看出，SRCNN 模型较为简单，主要包括三个卷积层，输入通道和最终的输出通道都是 1，激活函数采用 ReLU 函数。

测试模型：

```
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
model_weights_path = './checkpoints/srcnn_x4.pth'
scale_factor = 4

model = SRCNN().to(device)

# Load the SRCNN weights
checkpoint = torch.load(model_weights_path, map_location=lambda storage, loc: storage)
model.load_state_dict(checkpoint)
model.eval()

lr_image_dir = '../dataset/Set5_downscale_4'
sr_image_dir = '../dataset/SR_SRCNN'
sr_bic_dir = '../dataset/SR_bic'
image_names = os.listdir(lr_image_dir)

for image_name in image_names:
    lr_image_path = os.path.join(lr_image_dir, image_name)
    sr_image_path = os.path.join(sr_image_dir, image_name)
    sr_bic_path = os.path.join(sr_bic_dir, image_name)

    image = pil_image.open(lr_image_path).convert('RGB')

    # get bicubiced images
    image = image.resize((image.width * scale_factor, image.height * scale_factor),
                         resample=pil_image.BICUBIC)
    image.save(sr_bic_path)
    image = np.array(image).astype(np.float32)

    # 对y通道做处理
    ycbcr = convert_rgb_to_ycbcr(image)
    y = ycbcr[..., 0]
    y /= 255.
    y = torch.from_numpy(y).to(device)
    y = y.unsqueeze(0).unsqueeze(0)
    with torch.no_grad():
        preds = model(y).clamp(0.0, 1.0)

    preds = preds.mul(255.0).cpu().numpy().squeeze(0).squeeze(0)
    output = np.array([preds, ycbcr[..., 1], ycbcr[..., 2]]).transpose([1, 2, 0])
    output = np.clip(convert_ycbcr_to_rgb(output), 0.0, 255.0).astype(np.uint8)
    output = pil_image.fromarray(output)
```

图 3.4 测试 SRCNN

本次实验使用的是预训练好的模型，将与训练的参数加载进来，直接对 SRCNN 在 Set5 数据集上进行测试。将测试出的图片信息保存在 dataset 下的 SR_SRCNN 中。

评估模型：

```
device = torch.device("cuda", 0)
# Initialize the sharpness evaluation function
psnr = PSNR(0, only_test_y_channel=True).to(device, non_blocking=True)
ssim = SSIM(0, only_only_test_y_channel=True).to(device, non_blocking=True)
# Initialize metrics
psnr_metrics = 0.0
ssim_metrics = 0.0

gt_image_dir = '../dataset/Set5/'
sr_image_dir = '../dataset/SR_SRCNN'

image_names = os.listdir(gt_image_dir)

for image_name in image_names:
    gt_image_path = os.path.join(gt_image_dir, image_name)
    sr_image_path = os.path.join(sr_image_dir, image_name)
    gt_tensor = preprocess_one_image(gt_image_path, device)
    sr_tensor = preprocess_one_image(sr_image_path, device)

    # cal metrics
    p = psnr(sr_tensor, gt_tensor).item()
    s = ssim(sr_tensor, gt_tensor).item()
    psnr_metrics += p
    ssim_metrics += s

avg_psnr = psnr_metrics / len(image_names)
avg_ssim = ssim_metrics / len(image_names)
print(f"PSNR: {avg_psnr:4.2f} [dB]\n"f"SSIM: {avg_ssim:4.4f} [u]")
```

图 3.4 评估 SRCNN

根据评估指标 PSNR 和 SSIM 得到评估结果，分别进行输出。

3.3 实验结果

Bicubic **下采样：**下图 3.5 展示了原始图像和使用 Bicubic 插值进行下采样之后的图片。这一步是预处理，以便后续对下采样之后的图片作为输入进行处理。

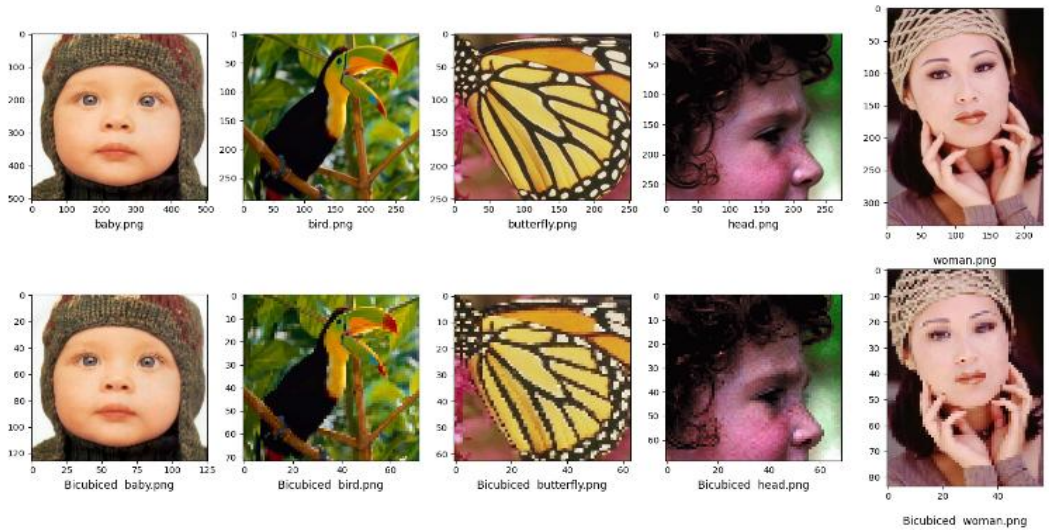

图 3.5 原图和下采样图

评估结果：如下图 3.5 所示，SRCNN 模型在 Set5 数据集上测试的最终 PSNR 平均为 30.13dB，SSIM 平均为 0.8598u。

```
PSNR: 30.13 [dB]
SSIM: 0.8595 [u]
```

图 3.5 评估结果

效果展示：如下图 3.6 所示，SRCNN 在 Set5 数据集上展示了不错的效果。

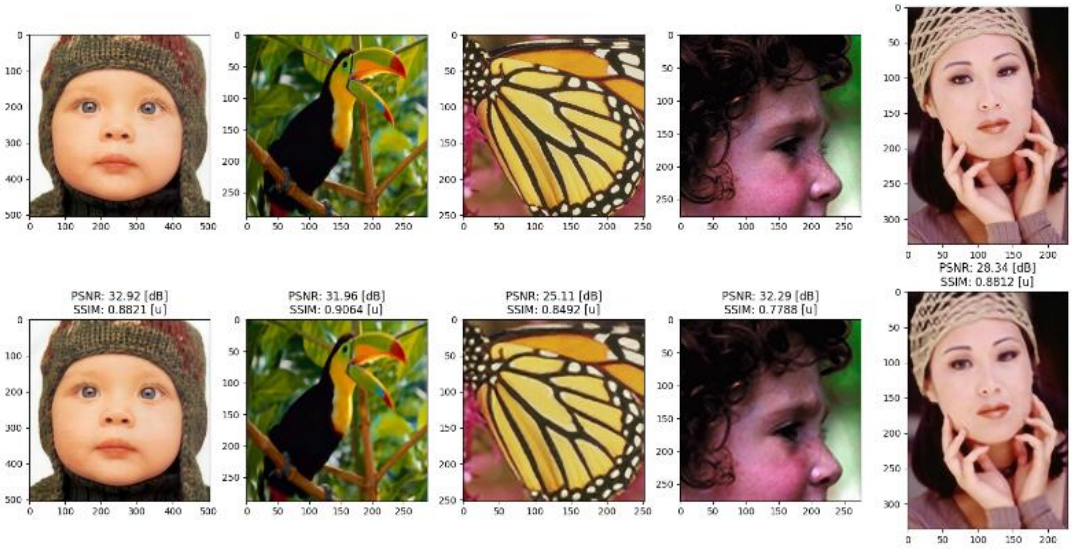

图 3.6 原图和 SRCNN 效果图

四、SRGAN 图像超分辨率算法

4.1 SRGAN 模型基本原理

SRGAN 是一种基于 GAN 模型来对图像进行超分辨率重建的算法模型，由

生成网络(generator)和判别网络(discriminator)构成。通过输入低分辨率的像素图来输出其对应的超分辨率像素图像，将原始分辨率像素图像和生成的超分辨率像素图像依次输入判别网络，当判别网络 D 无法区分二者时，最终达到平衡状态。SRGAN 就实现了重构超分辨率像素图像。

训练 SRGAN 模型的过程如下：首先将高分辨率图像(I^{HR})进行降采样得到其对应的低分辨率图像(I^{LR})，生成网络将低分辨率图像经由一个卷积层的处理，获得该像素图像的特征及其参数；其次调用激活函数，再通过卷积层获得该像素图像的主要特征参数；然后调用残差模型；再调用批量归一化(BN)层，残差模块重新导入激活函数；最后经过两次池化层操作，使用亚像素来提高分辨率，获得对应的超分辨率图像(I^{SR})。

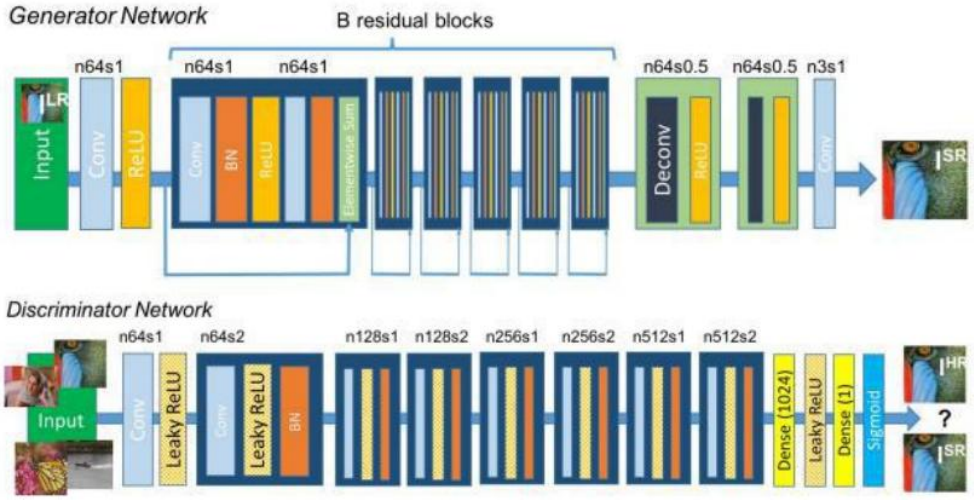

图 4.1 SRGAN 模型结构

如上图 4.1 所示，SRGAN 模型结构包括两个部分，生成网络和判别网络。

4.1.1 生成网络

生成器使用 SRResNet。其输入为一张低分辨率图像(I^{LR})，将卷积层和 prelu 函数作为激活函数的激活层来提取浅层的特征参数。将提取到的浅层特征参数传入残差块来提取深层特征参数。最后经由一个上采样层，生成较高分辨率的像素图像。如图 4.1 上半部分所示，k 表示卷积核尺寸、n 表示输出通道数、s 为卷积步长。

残差块：残差块(如图 4.2 所示)由 Conv2D 卷积层、Batch-Norm2D 函数、prelu 函数以及跳过连接(skip connection)组成，其优点是可以避免梯度消失，较为稳定地通过加深层数来提高模型。

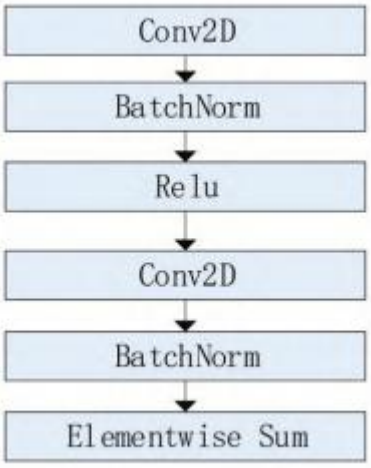

图 4.2 残差块

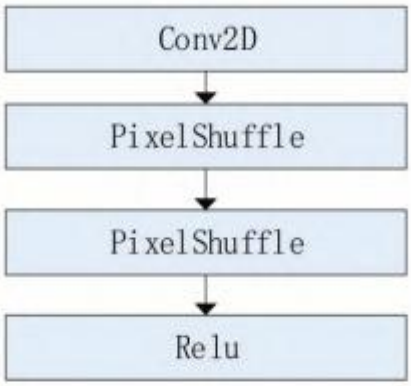

图 4.3 上采样块

上采样块：上采样层有两个上采样块。上采样块包括 Conv2D、upscale_factor 为 2 的 pixelshuffle 和 prelu 函数。其作用是放大原图像，获得更高的分辨率。如图 4.3 所示。

4.1.2 判别网络

判别器用于估计输入到判别器的是真实的图像的概率。判别器含有 8 个卷积层，从第 2 个卷积层开始，每个卷积层后面加一个 Batch-Norm 层来归一化中间层。如图 4.1 下半部分所示，k 表示卷积核尺寸、n 表示输出通道数、s 为卷积步长。

4.1.3 损失函数

SRGAN 生成的网络损失函数为感知损失，由两部分组成：内容损失和对抗损失。

$$l^{SR} = l_x^{SR} + 10^{-3} l_{Gen}^{SR}$$

其中，l_x^{SR}是内容损失，l_{Gen}^{SR}是对抗损失。将两部分按上式进行加权求和即可得到感知损失l^{SR}。对于对抗损失的计算由下式给出：

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

其中，D_{θ_D}表示原始像素图象的分辨率被成功提升的概率（实现超分辨率），$G_{\theta_D}(I^{LR})$表示生成的高分辨率像素图像。

对于内容损失函数的计算有两个：MSEloss，表示像素层面的误差；VGGloss 表示激活函数的损失。MSEloss 计算方法如下：

$$l_{MSE}^{SR} = \frac{1}{r^2 WH} \sum_{x=1}^{rW} \sum_{y=1}^{rH} (I_{x,y}^{HR} - G_{\theta_D}(I^{LR})_{x,y})^2$$

VGGloss 计算方法如下：

$$l_{VGG/i.j}^{SR} = \frac{1}{W_{i,j}H_{i,j}}\sum_{x=1}^{W_{i,j}}\sum_{y=1}^{H_{i,j}}(\Phi_{i,j}(I^{HR}) - \Phi_{i,j}(G_{\theta_D}(I^{LR})_{x,y}))^2$$

其中$\Phi_{i,j}$表示第 i 个池化层前的第 j 个卷积并经过激活层后的特征图。

4.2 关键代码实现

下采样：下采样是预处理过程，代码和 SRCNN 一样，目的都是获得下采样图片作为模型输入。

搭建模型：SRGAN 模型包括生成网络和判别网络两部分，代码过长，报告中不方便展示，具体的在代码中都有详细过程。

测试模型：

```
device = torch.device("cuda", 0)
g_model_weights_path = './checkpoints/SRGAN_x4-ImageNet-8c4a7569.pth.tar'
g_model = srresnet_x4(in_channels=3, out_channels=3,channels=64,num_rcb=16)
g_model = g_model.to(device)
# Load the super-resolution bsrgan_model weights
checkpoint = torch.load(g_model_weights_path, map_location=lambda storage, loc: storage)
g_model.load_state_dict(checkpoint["state_dict"])
g_model.eval()

lr_image_dir = '../dataset/Set5_downscale_4/'
sr_image_dir = '../dataset/SR_SRGAN'

image_names = os.listdir(lr_image_dir)

for image_name in image_names:
    lr_image_path = os.path.join(lr_image_dir, image_name)
    sr_image_path = os.path.join(sr_image_dir, image_name)
    lr_tensor = preprocess_one_image(lr_image_path, device)
    with torch.no_grad():
        sr_tensor = g_model(lr_tensor)
    # save image
    sr_image = tensor_to_image(sr_tensor, False, False)
    sr_image = cv2.cvtColor(sr_image, cv2.COLOR_RGB2BGR)
    cv2.imwrite(sr_image_path, sr_image)
```

图 4.4 测试 SRGAN

评估模型：

```python
device = torch.device("cuda", 0)
# Initialize the sharpness evaluation function
psnr = PSNR(0, only_test_y_channel=True).to(device, non_blocking=True)
ssim = SSIM(0, only_only_test_y_channel=True).to(device, non_blocking=True)
# Initialize metrics
psnr_metrics = 0.0
ssim_metrics = 0.0

gt_image_dir = '../dataset/Set5/'
sr_image_dir = '../dataset/SR_SRGAN'

image_names = os.listdir(gt_image_dir)

for image_name in image_names:
    gt_image_path = os.path.join(gt_image_dir, image_name)
    sr_image_path = os.path.join(sr_image_dir, image_name)
    gt_tensor = preprocess_one_image(gt_image_path, device)
    sr_tensor = preprocess_one_image(sr_image_path, device)

    # cal metrics
    p = psnr(sr_tensor, gt_tensor).item()
    s = ssim(sr_tensor, gt_tensor).item()
    psnr_metrics += p
    ssim_metrics += s

avg_psnr = psnr_metrics / len(image_names)
avg_ssim = ssim_metrics / len(image_names)
print(f"PSNR: {avg_psnr:4.2f} [dB]\n"f"SSIM: {avg_ssim:4.4f} [u]")
```

图 4.5 评估 SRGAN

根据评估指标 PSNR 和 SSIM 得到评估结果，分别进行输出。

4.3 实验结果

评估结果：如下图 4.6 所示，SRGAN 模型在 Set5 数据集上测试的最终 PSNR 平均为 30.58dB，SSIM 平均为 0.8633u。

```
PSNR: 30.58 [dB]
SSIM: 0.8633 [u]
```

图 4.6 评估结果

效果展示：如下图 4.7 所示，SRCNN 在 Set5 数据集上展示了比 SRCNN 更好的效果。

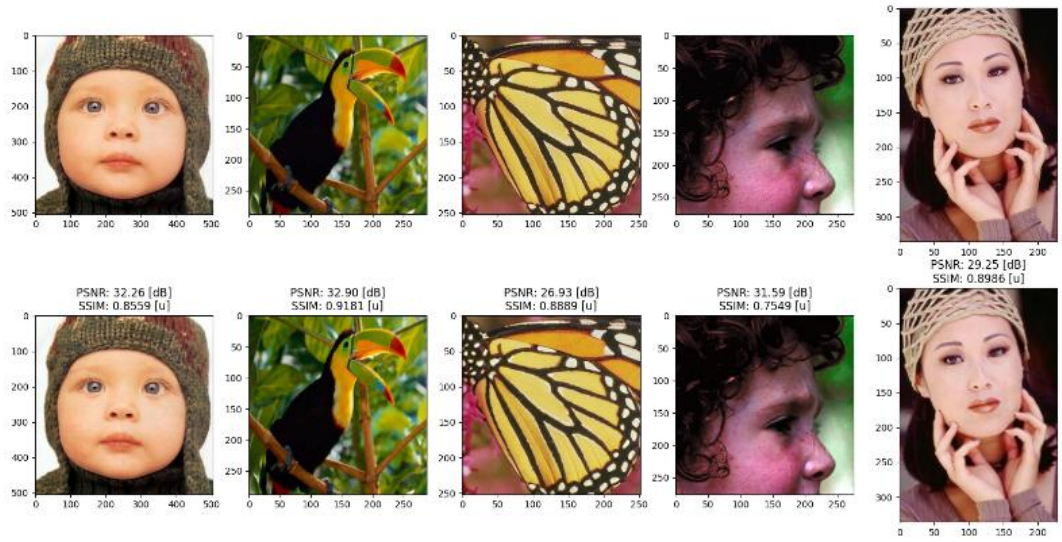

图 4.7 效果展示

五、对比分析

5.1 训练过程

SRCNN 和 SRGAN 在训练过程上存在较大的不同，本文主要分析了模型结构、目标函数和训练过程上的不同。

模型结构：SRCNN 采用的是三层卷积神经网络，其中第一层是卷积层，第二层是非线性映射层，第三层是反卷积层。SRCNN 通过最小化输入图像与目标图像之间的均方误差来训练网络参数。相比之下，SRGAN 采用的是生成对抗网络（GAN）模型，包含了生成器和鉴别器两个网络。生成器网络负责将低分辨率图像转换成高分辨率图像，鉴别器网络负责判断生成器网络输出的高分辨率图像是否与真实图像相似。SRGAN 通过博弈的方式来训练生成器和鉴别器网络，生成器网络的目标是欺骗鉴别器网络，鉴别器网络的目标是区分真实图像和生成图像。

目标函数：SRCNN 的目标函数是最小化输入图像与目标图像之间的均方误差。在训练过程中，SRCNN 将低分辨率图像输入到网络中，通过网络得到高分辨率图像，并计算与目标图像之间的均方误差。通过反向传播算法来更新网络参数。而 SRGAN 的目标函数是通过生成器网络和判别器网络之间的博弈来进行训练，生成器网络的目标是欺骗判别器网络，判别器网络的目标是区分真实图像和生成图像。在训练过程中，生成器网络通过最小化判别器网络判别生成图像为假的概率来进行参数优化，判别器网络通过最大化真实图像和生成图像之间的差异来进行参数优化。

训练过程：SRCNN 的训练过程比较简单，只需要单纯地对模型参数进行优化。在训练过程中，SRCNN 将低分辨率图像输入到网络中，通过网络得到高分辨率图像，并计算与目标图像之间的均方误差。通过反向传播算法来更新网络参数。而 SRGAN 的训练过程比较复杂，需要交替进行生成器和判别器网络的训练，同时需要使用一些技巧来提高训练效率和训练稳定性。SRGAN 采用的渐进式训练方法可以有效提高训练效率，采用批量归一化方法可以提高模型的稳定性。此

外，SRGAN 还采用了残差块和像素洛伦兹损失等技巧来进一步提高生成器网络的性能。

5.2 图片生成质量

SRCNN 和 SRGAN 在生成图像质量上的区别主要表现在两个方面。

首先，SRCNN 在生成高分辨率图像时可能会出现失真和伪影等问题，尤其是对于复杂纹理和细节信息的恢复表现较差。相比之下，SRGAN 通过引入判别器网络，可以更好地保留图像的细节信息，生成的高分辨率图像质量更高。其次，由于 SRGAN 采用的是生成对抗网络，其生成的图像更加真实自然，具有更高的视觉感知质量。而 SRCNN 生成的图像则显得较为平滑，缺乏真实感。

因此，虽然 SRCNN 和 SRGAN 都可以用于图像超分辨率，但在图像质量方面，SRGAN 具有更好的表现。这得益于 SRGAN 采用的生成对抗网络的训练方法以及引入的像素洛伦兹损失、残差块等技术，从而使得 SRGAN 生成的图像更加清晰、自然和真实。下图 5.1 是可视化图片 bird 的一组效果对比图。

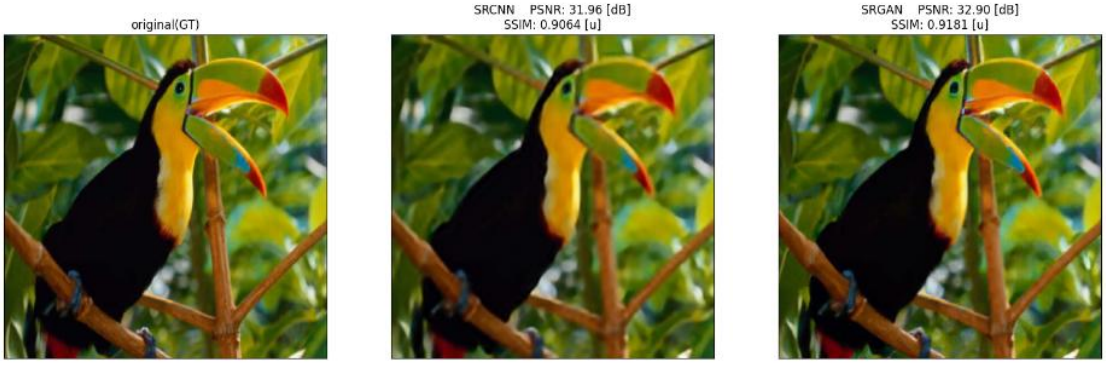

图 5.1STCNN 和 SRGAN 效果对比

六、实验总结与分析

本次实验旨在分别实现 SRCNN 和 SRGAN 算法，并在 Set5 数据集上进行测试，比较两种方法在训练过程和生成质量上的区别。实验过程中，首先搭建了两个网络模型，分别为 SRCNN 和 SRGAN，并使用 Pytorch 框架进行训练。在训练阶段，SRCNN 算法的收敛速度较快，但容易出现过拟合现象。而 SRGAN 算法则需要更长时间的训练，并且训练过程中需要更大的内存和计算资源，但其生成的图像质量更高，具有更好的视觉感知效果。

在测试阶段，使用 Set5 数据集进行测试，并使用平均峰值信噪比(PSNR）和结构相似度(SSIM）这两种指标对生成的高分辨率图像进行评价。实验结果表明，相对于 SRCNN, SRGAN 生成的图像质量更高，具有更高的 PSNR 和 SSIM 值。此外，还进行了图像可视化展示，并使用人眼观察评估了两种方法生成的图像质量。结果显示，SRGAN 生成的图像更加清晰自然，具有更高的真实感，相比之下 SRCNN 生成的图像则显得较为平滑，缺乏真实感。

综上所述，实验结果表明，SRGAN 算法相对于 SRCNN 具有更好的图像生成效果，生成的图像更加清晰自然，具有更高的视觉感知质量。虽然 SRGAN 算法的训练时间和资源需求更高，但其生成的图像质量与真实感更加接近。因此，在进行图像超分辨率任务时，选择合适的算法对于保证生成图像质量至关重要。