南京理工大学

NANJING UNIVERSITY OF SCIENCE & TECHNOLOGY

“计算机视觉与应用实践”

课程作业

姓 名：余 涛 学 号：122106222833

专 业：计算机科学与工程

项目名称：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

时间：2023 年 4 月 20 日

1、实验目标

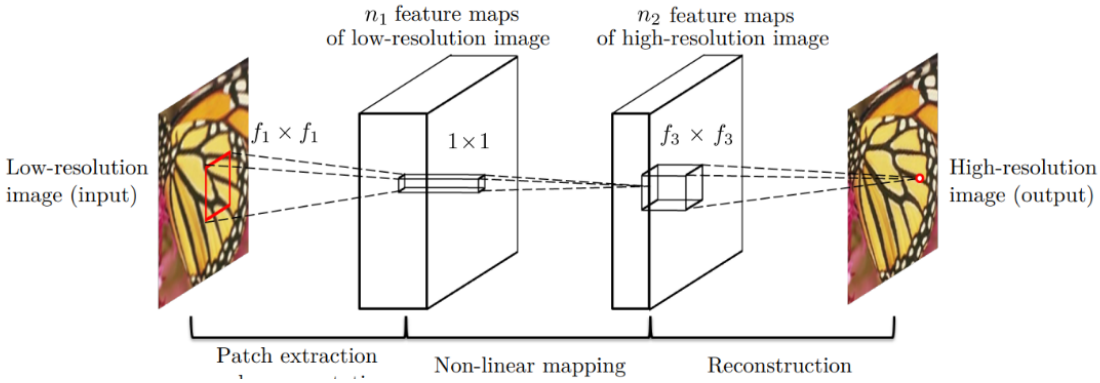
（1）、实现 SRCNN 或其他一种基于逐像素损失的图像超分辨率算法在 Set5 数据集上的测试，得到超分辨率图像，并进行分析。

2、具体要求

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，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://blog.csdn.net/chen666250/article/details/116328919。将实验分析报告、生成的超分辨率图像结果及实现代码，上传到 Github，项目名称“计算机视觉实践-练习 3”。

3、基于 SRCNN 逐像素损失的图像超分辨率算法

一、SRCNN 简介



①、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。

②、非线性映射层：把提取到的特征进一步做非线性映射，加大网络深度，提高网络复杂性。

③、重建层：结合了前面得到的补丁来产生最终的高分辨率图像。

1、 实验步骤

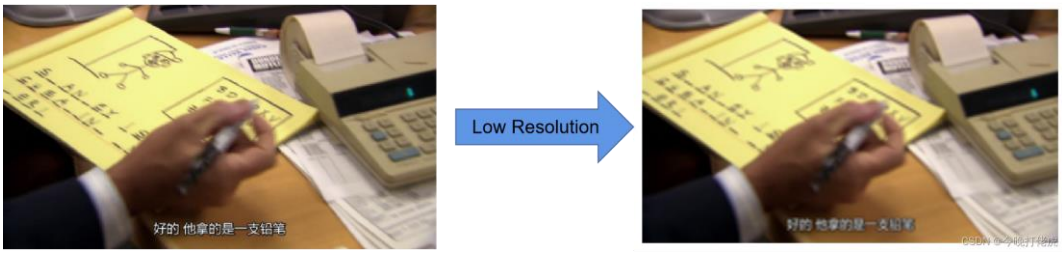
①、输入 LR 图像 X，经双三次(bicubic)插值，被放大成目标尺寸（如放大至 2 倍、3 倍、4 倍），得到 Y ，即低分辨率图像(Low-resolution image)。

②、通过三层卷积网络拟合非线性映射。

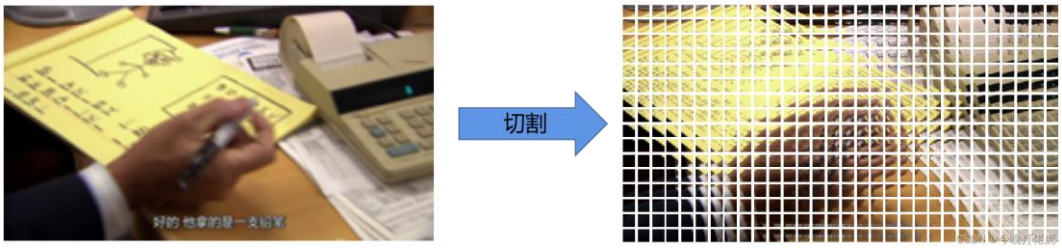
③、输出 HR 图像结果 F （ Y ）入的一种特征，然后每个 FeatureMap 有多个神经元。

2、训练过程

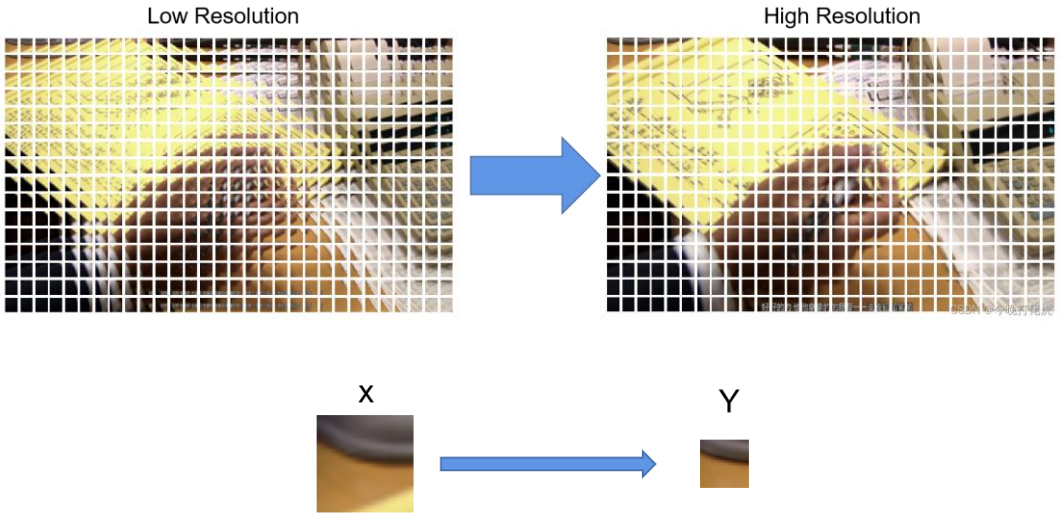
（1）、降低分辨率：



（2）、切割图片，补丁之间有重复：



（3）、训练模型，学习低分辨率 → to→ 高分辨率的映射关系：



3、损失函数

损失哈数：MES（均方误差），选择 MSE 作为损失函数的一个重要原因是 MSE 的格式和我们图像失真评价指标 PSNR 很像：

F(Y; θ)：得到的超分辨率图像，X：原高分辨率图像。

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

4、SRCNN 训练过程

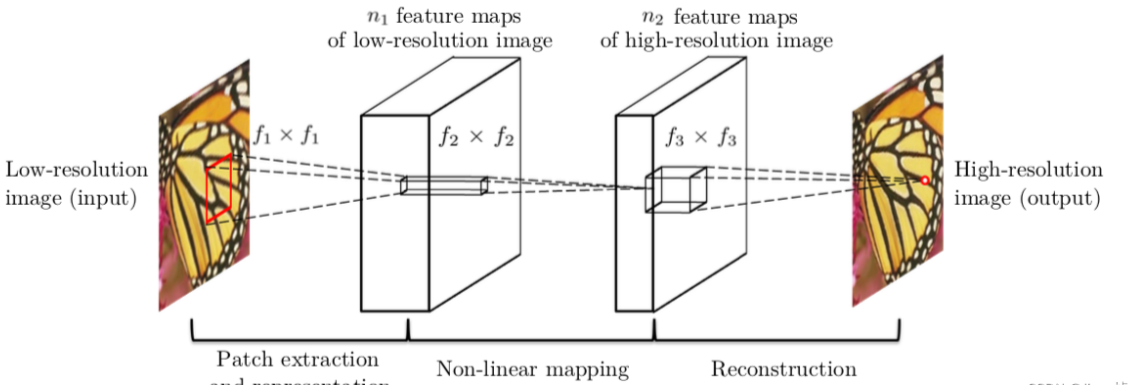
（1）、构建训练集，含有低分辨率图像和高分辨图像，其中图像需要将其从 RGB 图像转为 YCBCR 图像，并且对图像进行分割为小块进行存储，高分辨率图像为未下采样前的图像，低分辨率图像为下采样，上采样后的图像。

（2）、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）。

（3）、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。

二、具体步骤及过程

1、模型的主体（model.py）



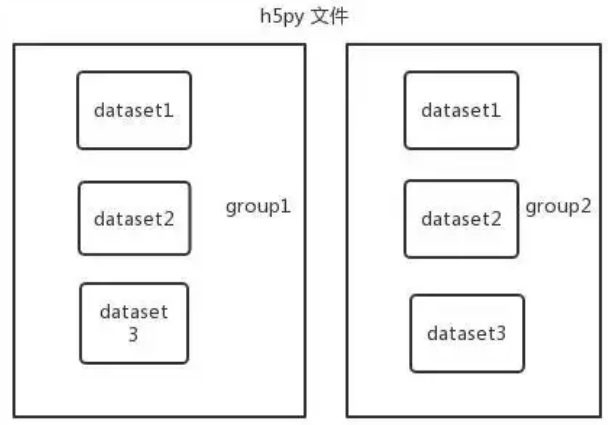

```
from torch import nn

class SRCNN(nn.Module):  #搭建 SRCNN 3 层卷积模型，Conve2d（输入层数，输出层数，卷积核大小，步长，填充层）
    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、准备数据部分（dataset.py）




```
class TrainDataset(Dataset):  # 构建训练数据集，通过 np.expand_dims 将 h5 文件中的 lr（低分辨率图像）和 hr（高分辨率图像）组合为训练集
    def __init__(self, h5_file):
        super(TrainDataset, self).__init__()
        self.h5_file = h5_file
    def __getitem__(self, idx): #通过 np.expand_dims 方法得到组合的新数据
        with h5py.File(self.h5_file, 'r') as f:
            return np.expand_dims(f['lr'][idx] / 255., 0), np.expand_dims(f['hr'][idx] / 255., 0)
    def __len__(self):    #得到数据大小
        with h5py.File(self.h5_file, 'r') as f:
            return len(f['lr'])

# 与 TrainDataset 类似
class EvalDataset(Dataset):    # 构建测试数据集，通过 np.expand_dims 将 h5 文件中的 lr（低分辨率图像）和 hr（高分辨率图像）组合为验证集
    def __init__(self, h5_file):
        super(EvalDataset, self).__init__()
        self.h5_file = h5_file
    def __getitem__(self, idx):
        with h5py.File(self.h5_file, 'r') as f:
            return np.expand_dims(f['lr'][str(idx)][:, :] / 255., 0), np.expand_dims(f['hr'][str(idx)][:, :] / 255., 0)

    def __len__(self):
        with h5py.File(self.h5_file, 'r') as f:
            return len(f['lr'])
```

3、train.py（训练 SRCNN 模型，得到最优参数）

```
args.outputs_dir = os.path.join(args.outputs_dir, 'x{}'.format(args.scale))
# 没有该文件夹就新建一个文件夹
if not os.path.exists(args.outputs_dir):
    os.makedirs(args.outputs_dir)

# benckmark 模式，加速计算，但寻找最优配置，计算的前馈结果会有差异
cudnn.benchmark = True
# gpu 或者 cpu 模式，取决于当前 cpu 是否可用
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

# 每次程序运行生成的随机数固定
torch.manual_seed(args.seed)

# 构建 SRCNN 模型，并且放到 device 上训练
model = SRCNN().to(device)
# 恢复训练，从之前结束的那个地方开始
# model.load_state_dict(torch.load('outputs/x3/epoch_173.pth'))

# 设置损失函数为 MSE
criterion = nn.MSELoss()

# 优化函数 Adam，lr 代表学习率，
optimizer = optim.Adam([
    {'params': model.conv1.parameters()},
    {'params': model.conv2.parameters()},
    {'params': model.conv3.parameters(), 'lr': args.lr * 0.1}
], lr=args.lr)

# 预处理训练集
train_dataset = TrainDataset(args.train_file)
train_dataloader = DataLoader(
    # 数据
    dataset=train_dataset,
    # 分块
    batch_size=args.batch_size,
    # 数据集数据洗牌,打乱后取 batch
    shuffle=True,
    # 工作进程，像是虚拟存储器中的页表机制
    num_workers=args.num_workers,
    # 锁页内存，不换出内存，生成的 Tensor 数据是属于内存中的锁页内存区
    pin_memory=True,
    # 不取余，丢弃不足 batchSize 大小的图像
    drop_last=True)
```

```
# 预处理验证集
eval_dataset = EvalDataset(args.eval_file)
eval_dataloader = DataLoader(dataset=eval_dataset, batch_size=1)

# 拷贝权重
best_weights = copy.deepcopy(model.state_dict())
best_epoch = 0
best_psnr = 0.0

# 画图用
lossLog = []
psnrLog = []

# 恢复训练
# for epoch in range(args.num_epochs):
for epoch in range(1, args.num_epochs + 1):
    # for epoch in range(174, 400):
    # 模型训练入口
    model.train()

    # 变量更新，计算 epoch 平均损失
    epoch_losses = AverageMeter()

    # 进度条，就是不要不足 batchsize 的部分
    with tqdm(total=(len(train_dataset) - len(train_dataset) % args.batch_size)) as t:
        # t.set_description('epoch:{}/{}'.format(epoch, args.num_epochs - 1))
        t.set_description('epoch:{}/{}'.format(epoch, args.num_epochs))

        # 每个 batch 计算一次
        for data in train_dataloader:
            # 对应 datastes.py 中的__getItem__，分别为 lr,hr 图像
            inputs, labels = data

            inputs = inputs.to(device)
            labels = labels.to(device)
            # 送入模型训练
            preds = model(inputs)

            # 获得损失
            loss = criterion(preds, labels)

            # 显示损失值与长度
            epoch_losses.update(loss.item(), len(inputs))
```

```
            # 梯度清零
            optimizer.zero_grad()

            # 反向传播
            loss.backward()

            # 更新参数
            optimizer.step()

            # 进度条更新
            t.set_postfix(loss='{:.6f}'.format(epoch_losses.avg))
            t.update(len(inputs))
    # 记录 lossLog 方面画图
    lossLog.append(np.array(epoch_losses.avg))
    # 可以在前面加上路径
    np.savetxt("lossLog.txt", lossLog)

    # 保存模型
    torch.save(model.state_dict(),
     os.path.join(args.outputs_dir, 'epoch_{}.pth'.format(epoch)))
    # 是否更新当前最好参数
    model.eval()
    epoch_psnr = AverageMeter()

    for data in eval_dataloader:
        inputs, labels = data
        inputs = inputs.to(device)
        labels = labels.to(device)

        # 验证不用求导
        with torch.no_grad():
            preds = model(inputs).clamp(0.0, 1.0)

        epoch_psnr.update(calc_psnr(preds, labels), len(inputs))
    print('eval psnr: {:.2f}'.format(epoch_psnr.avg))

    # 记录 psnr
    psnrLog.append(Tensor.cpu(epoch_psnr.avg))
    np.savetxt('psnrLog.txt', psnrLog)
    # 找到更好的权重参数，更新
    if epoch_psnr.avg > best_psnr:
        best_epoch = epoch
        best_psnr = epoch_psnr.avg
        best_weights = copy.deepcopy(model.state_dict())
```

```
            print('best epoch: {}, psnr: {:.2f}'.format(best_epoch, best_psnr))
            torch.save(best_weights, os.path.join(args.outputs_dir, 'best.pth'))
    print('best epoch: {}, psnr: {:.2f}'.format(best_epoch, best_psnr))
    torch.save(best_weights, os.path.join(args.outputs_dir, 'best.pth'))
```

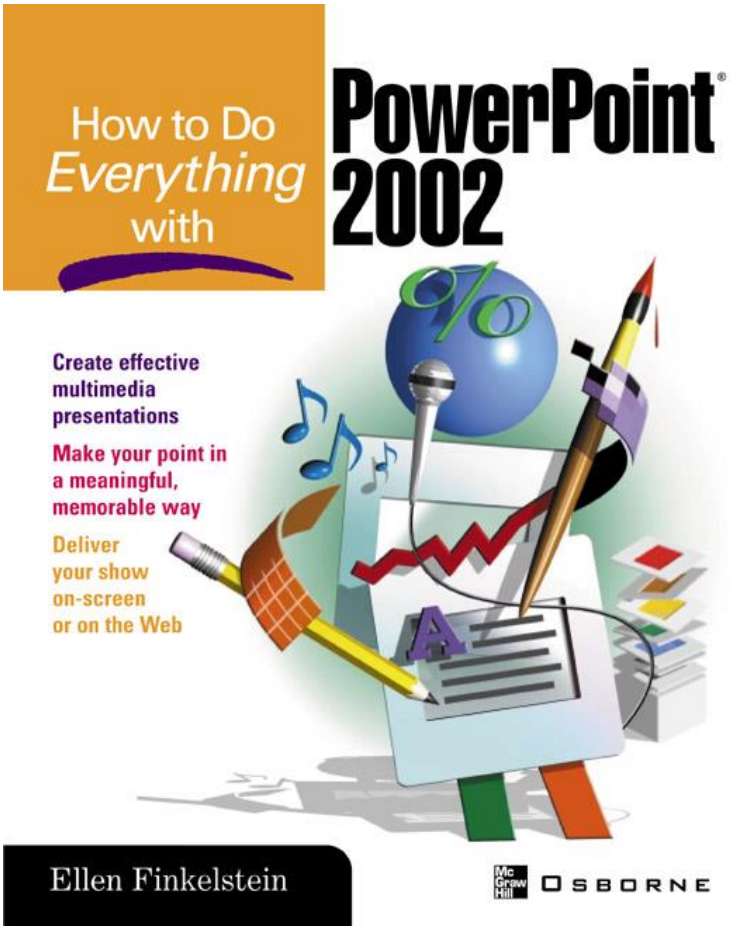
三、结果展示

(a)是原图 (b)是 bicubic (c)是 SRCNN

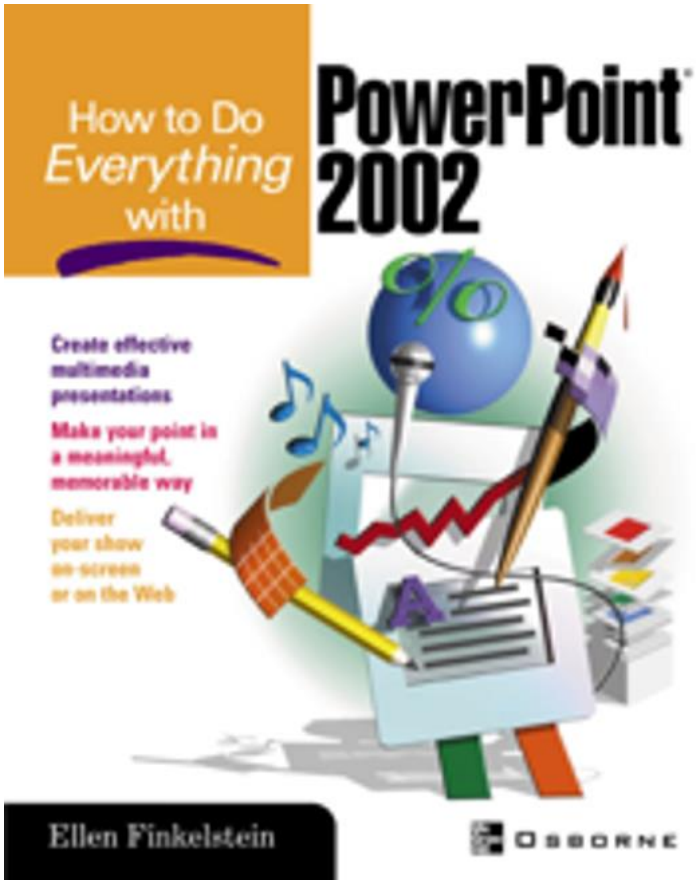
(a)

(b)

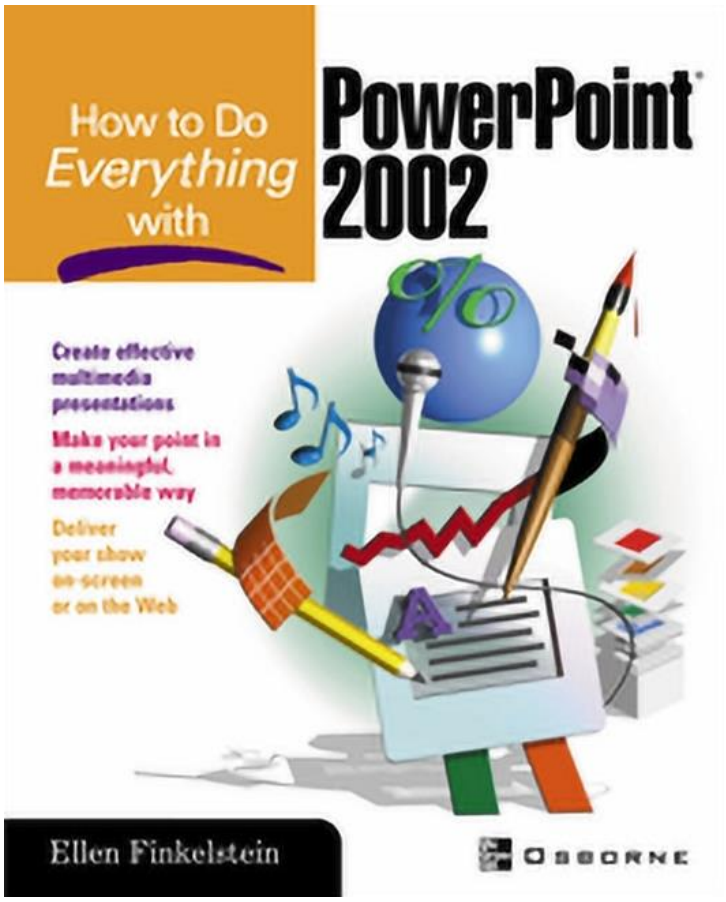
(c)


How to Do
Everything
with
PowerPoint®
2002
Create effective multimedia presentations
Make your point in a meaningful, memorable way
Deliver your show on-screen or on the Web
Ellen Finkelstein
Osborne

(a)


How to Do
Everything
with
PowerPoint
2002
Create effective multimedia presentations
Make your point in a meaningful, memorable way
Deliver your show on-screen or on the Web
Ellen Finkelstein
OSBORNE

(b)



(c)

4、代码运行说明

（1)、代码使用 python 编写，运行时请使用 pycharm 等 python 编译器。

（2)、SRCNN_image 中共包含：data, datasets, outputs, model.py, prepare.py, test.py, train.py 等文件

（3)、使用时可以直接运行 test.py,对 butterfly 图片进行超分辨率，并且生成 butterfly_bicubic 以及 butterfly_srcnn 两张图片。此外也可以运行 train.py，该操作使用 91-image_x4 作为模型的训练集，Set5_x4 作为测试集，所得到的模型权重参数保存在 outputs 文件下。