```
import argparse
from torch import nn
import torch
import torch.backends.cudnn as cudnn
import numpy as np
import PIL.Image as pil_image

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__()
        #第一层卷积层（实现数据读入）
        #输入：低分辨率补丁
        self.conv1 = nn.Conv2d(num_channels, 64, kernel_size=9, padding=9
// 2)
        #第二层为 conv 层(实现非线性多个映射)
        #输入：第一层输入
        self.conv2 = nn.Conv2d(64, 32, kernel_size=5, padding=5 // 2)
        #第三层为 conv 层(实现重建)
        #输入：第二层输出
        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

"""
        只操作 y 通道
        因为我们感兴趣的不是颜色变化(存储在 CbCr 通道中的信息)而只是其亮度(Y 通
道);
        根本原因在于相较于色差，人类视觉对亮度变化更为敏感。
"""
def convert_rgb_to_y(img):
    if type(img) == np.ndarray:
        return 16. + (64.738 * img[:, :, 0] + 129.057 * img[:, :, 1] + 25.064
* img[:, :, 2]) / 256.
    elif type(img) == torch.Tensor:
        if len(img.shape) == 4:
            img = img.squeeze(0)
        return 16. + (64.738 * img[0, :, :] + 129.057 * img[1, :, :] + 25.064
```

```
* img[2, :, :]) / 256.
    else:
        raise Exception('Unknown Type', type(img))

"""
        RGB 转 YCBCR
        Y=0.257*R+0.564*G+0.098*B+16
        Cb=-0.148*R-0.291*G+0.439*B+128
        Cr=0.439*R-0.368*G-0.071*B+128
"""
def convert_rgb_to_ycbcr(img):
    if type(img) == np.ndarray:
        y = 16. + (64.738 * img[:, :, 0] + 129.057 * img[:, :, 1] + 25.064
* img[:, :, 2]) / 256.
        cb = 128. + (-37.945 * img[:, :, 0] - 74.494 * img[:, :, 1] + 112.439
* img[:, :, 2]) / 256.
        cr = 128. + (112.439 * img[:, :, 0] - 94.154 * img[:, :, 1] - 18.285
* img[:, :, 2]) / 256.
        return np.array([y, cb, cr]).transpose([1, 2, 0])
    elif type(img) == torch.Tensor:
        if len(img.shape) == 4:
            img = img.squeeze(0)
        y = 16. + (64.738 * img[0, :, :] + 129.057 * img[1, :, :] + 25.064
* img[2, :, :]) / 256.
        cb = 128. + (-37.945 * img[0, :, :] - 74.494 * img[1, :, :] + 112.439
* img[2, :, :]) / 256.
        cr = 128. + (112.439 * img[0, :, :] - 94.154 * img[1, :, :] - 18.285
* img[2, :, :]) / 256.
        return torch.cat([y, cb, cr], 0).permute(1, 2, 0)
    else:
        raise Exception('Unknown Type', type(img))

# ycbr 转 rgb
def convert_ycbcr_to_rgb(img):
    if type(img) == np.ndarray:
        r = 298.082 * img[:, :, 0] / 256. + 408.583 * img[:, :, 2] / 256.
- 222.921
        g = 298.082 * img[:, :, 0] / 256. - 100.291 * img[:, :, 1] / 256.
- 208.120 * img[:, :, 2] / 256. + 135.576
        b = 298.082 * img[:, :, 0] / 256. + 516.412 * img[:, :, 1] / 256.
- 276.836
        return np.array([r, g, b]).transpose([1, 2, 0])
    elif type(img) == torch.Tensor:
        if len(img.shape) == 4:
```

```
            img = img.squeeze(0)
        r = 298.082 * img[0, :, :] / 256. + 408.583 * img[2, :, :] / 256.
- 222.921
        g = 298.082 * img[0, :, :] / 256. - 100.291 * img[1, :, :] / 256.
- 208.120 * img[2, :, :] / 256. + 135.576
        b = 298.082 * img[0, :, :] / 256. + 516.412 * img[1, :, :] / 256.
- 276.836
        return torch.cat([r, g, b], 0).permute(1, 2, 0)
    else:
        raise Exception('Unknown Type', type(img))

# 计算psnr
def calc_psnr(img1, img2):
    return 10. * torch.log10(1. / torch.mean((img1 - img2) ** 2))


# 计算 平均数，求和，长度
class AverageMeter(object):
    def __init__(self):
        self.reset()

    def reset(self):
        self.val = 0
        self.avg = 0
        self.sum = 0
        self.count = 0

    def update(self, val, n=1):
        self.val = val
        self.sum += val * n
        self.count += n
        self.avg = self.sum / self.count


if __name__ == '__main__':
    print("success used")
    # 初始参数设定，处理图像目录，放大倍数
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights-file', type=str,
default="C:\\Users\\86136\\Desktop\\SRCNN-pytorch-master\\Weight-File
\\srcnn_x3.pth")
    parser.add_argument('--image-file', type=str,
default="D:\\AIPicture\\OriginalPicture\\OriginalPicture.png")
    parser.add_argument('--scale', type=int, default=3)
```

```
    args = parser.parse_args()

    print("1")
    # 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
    # 使用 gpu
    device = torch.device('cuda:0' if torch.cuda.is_available() else
'cpu')
    # 新建一个模型
    model = SRCNN().to(device)
    # 通过 model.state_dict()得到模型有哪些 parameters and persistent
buffers
    # torch.load('tensors.pth', map_location=lambda storage, loc:
storage)  使用函数将所有张量加载到 CPU（适用在 GPU 训练的模型在 CPU 上加载）
    state_dict = model.state_dict()
    print("use model")
    for n, p in torch.load(args.weights_file).items():
        print("test" + n)
        if n in state_dict.keys():
            state_dict[n].copy_(p)
        else:
            raise KeyError(n)
    # for n, p in torch.load(args.weights_file, map_location=lambda
storage, loc: storage).items():
    #     print("test"+n)
    #     if n in state_dict.keys():
    #         state_dict[n].copy_(p)
    #     else:
    #         raise KeyError(n)
    print("load weight")
    # 切换为测试模式 ，取消 dropout
    model.eval()
    # 训练完 train 样本后，生成的模型 model 要用来测试样本了。在 model(test) 之
前，需要加上 model.eval()，
    # 否则的话，有输入数据，即使不训练，它也会改变权值。这是 model 中含有 BN 层和
Dropout 所带来的的性质。
    # 将图片转为 RGB 类型
    image = pil_image.open(args.image_file).convert('RGB')
    print("read image")
    # 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)))
    # 双三次插值
    # 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
    #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，并且将参数放到 device 上
    y = torch.from_numpy(y).to(device)
    # 增加两个维度
    y = y.unsqueeze(0).unsqueeze(0)
    # 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)
    # 计算 y 通道的 psnr 值
    psnr = calc_psnr(y, preds)
    # 插值图和结果图。
    # 格式化输出 PSNR 值
    print('PSNR: {:.2f}'.format(psnr))
    # 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 转化为 ndarray
    # 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)
    # 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)
    # array 转换成 image，即将矩阵转为图像
    output = pil_image.fromarray(output)
    # 对图像进行保存
```

```
    output.save(args.image_file.replace('.',
'_srcnn_x{}.'.format(args.scale)))
    # replace 旧字符串换新字符
    print("success output")
```