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 SRGAN 基于 GAN 的图像超分辨率算法在 Set5 数据集上的测试

间：2023 年 4 月 20 日

1、实验目标

（1）、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。

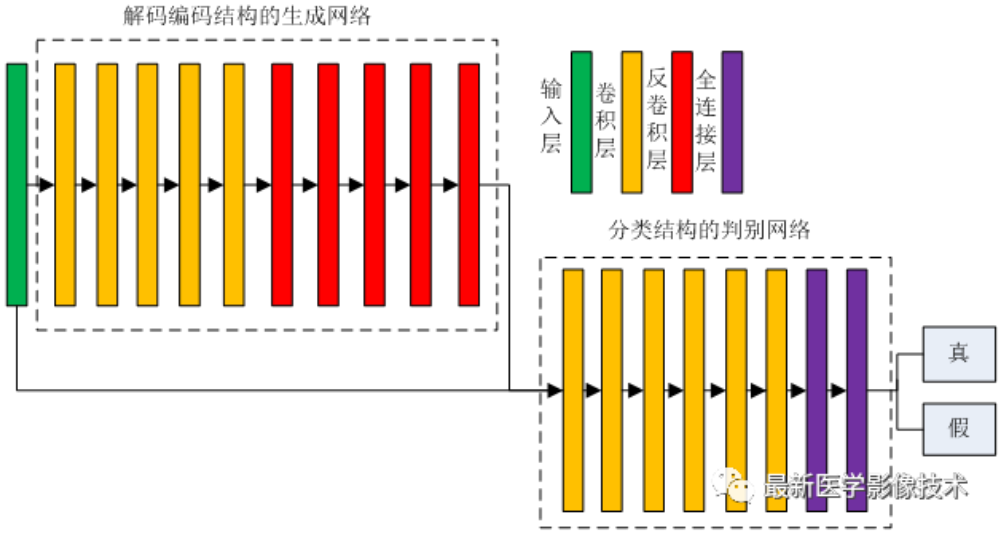
2、具体要求

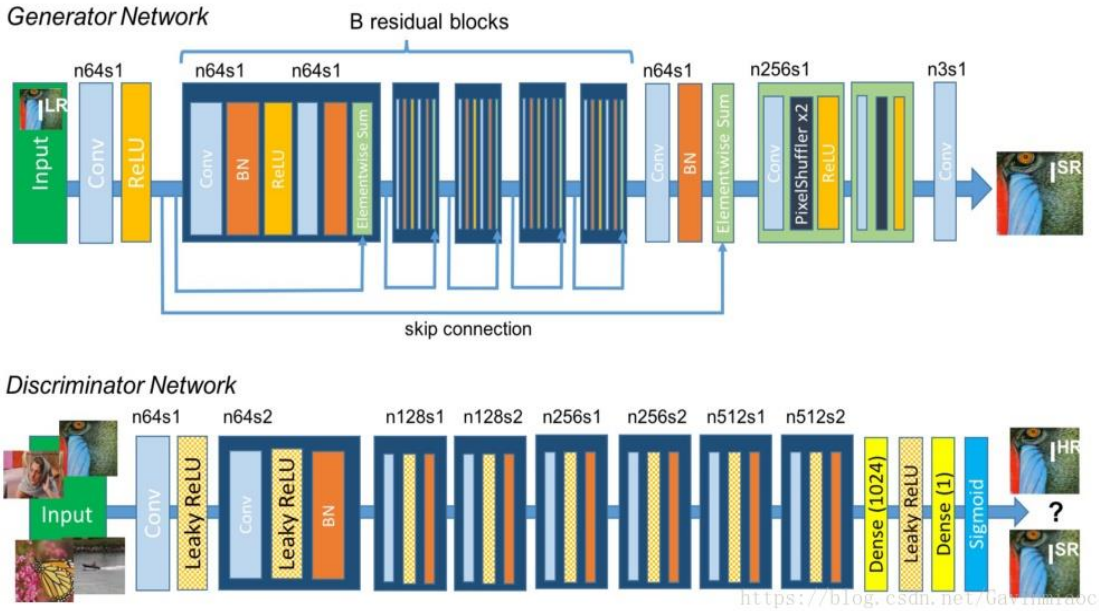
测试方式:先将图像用 Bicubic 插值进行下采样，再使用超分辨率算法处理，得到的超分辨率图像与真实的原始图像进行对比。数据集下载网址:https://blog.csdn.net/chen666250/article/details/116328919。将实验分析报告、生成的超分辨率图像结果及实现代码，上传到 Github，项目名称“计算机视觉实践-练习 3”。

3、SRGAN 基于 GAN 的图像超分辨率算法

一、SRGAN 简介

SRGAN 的输入是低分辨图像和相应的高分率图像，低分辨图像经过解码编码网络结构的生层网络生成结果输出图像，然后将高分率图像和生成网络的输出图像一起输入到分类结构的判别网络中去，进行真假判别。





SRGAN 率先把 GAN 引入到超分辨领域。了解 GAN 的朋友可以快速理 SRGAN：与标准的 GAN 结构相比，SRGAN 生成器的输入不再是噪声，而是低分辨率图像；而判别器结构跟普通的 GAN 没有什么区别。

1 数据集预处理

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。

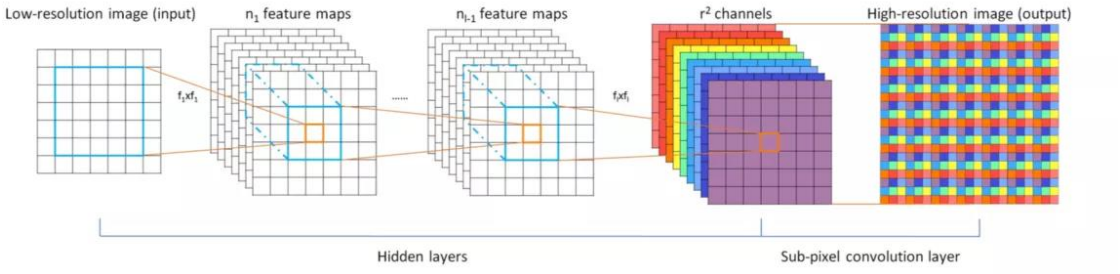
③、输出 HR 图像结果 F （ Y ）入的一种特征，然后每个 FeatureMap 有多个神经元。

首先准备好数据集，这里以 DIV2K_train_HR 作为训练集，DIV2K_vaild_HR 作为测试集

2、Subpixel 模块

PixelShuffler 操作，这里是用于 Upsampling，扩大图片尺寸 2 倍.

Subpixel 模块是一种在超分辨率中经常使用的 upscale 方法，又叫做 pixel shuffle。我们知道，对 CNN 的 feature map 进行放大的方法有转置卷积，但是如果直接用转置卷积的话，在超分辨率中通常会带入过多人工因素。而 Subpixel 模块会大大降低这个风险。



就是，如果我们想把图片放大 2 倍的话，那么我们需要生成 2^2=4 个一样大小的 feature map，最后将这 4 个特征图拼成一个大图。

```
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
```

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。

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

MSE/PSNR 不行，VGG feature map mse 行

作者认为 PSNR 高不一定好，PSNR 低一点的可能更加符合人的视觉效果

为此作者设计了 mean-opinion test（平均意见得分检验），请 26 个评分员对图片按 5 个梯度打分

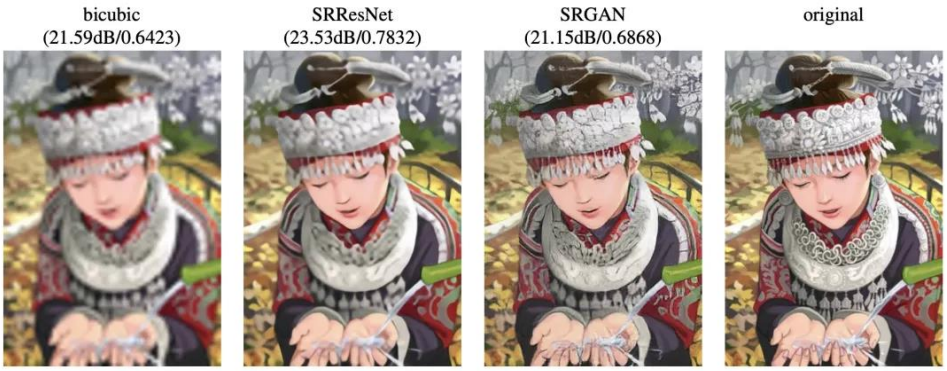


Figure 2: From left to right: bicubic interpolation, deep residual network optimized for MSE, deep residual generative adversarial network optimized for a loss more sensitive to human perception, original HR image. Corresponding PSNR and SSIM are shown in brackets. [4× upscaling]

4、SRGAN 训练过程

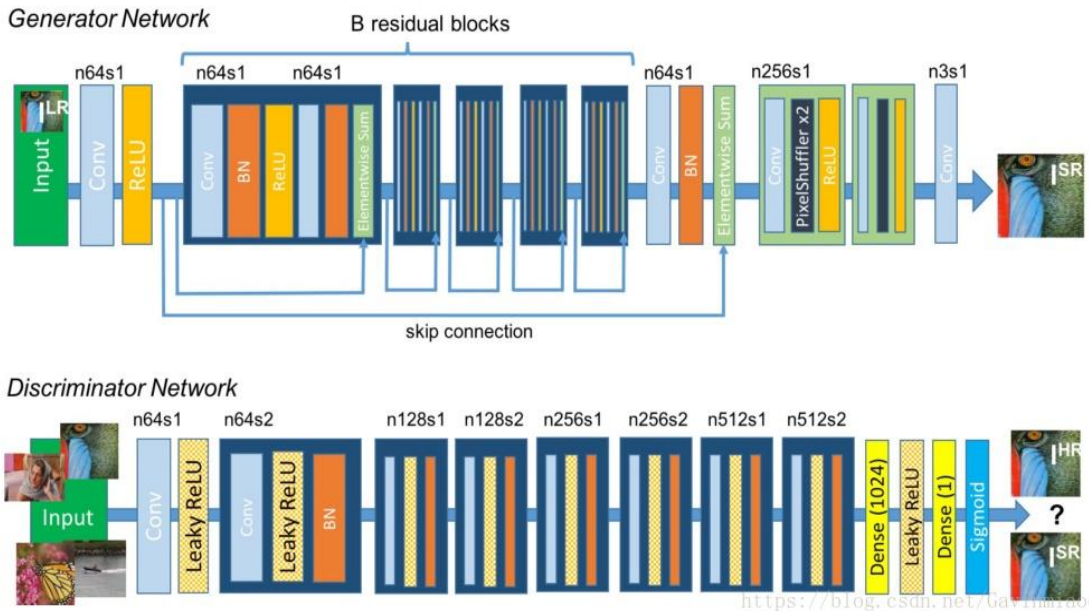
（1）、构建训练集，含有低分辨率图像和高分辨图像，其中图像需要将其从 RGB 图像转为 YCBCR 图像，并且对图像进行分割为小块进行存储，高分辨率图像为未下采样前的图像，低分辨率图像为下采样，上采样后的图像。

（2）、构建 SRGAN 模型，SRGAN 率先把 GAN 引入到超分辨领域。了解 GAN 的朋友可以快速理解 SRGAN：与标准的 GAN 结构相比，SRGAN 生成器的输入不再是噪声，而是低分辨率图像；而判别器结构跟普通的 GAN 没有什么区别。

（3）、训练模型 SRGAN，即学习低分辨率图像到高分辨率图像的映射关系。

二、具体步骤及过程

1、模型的主体（model.py）




```
# 生成器
class Generator(nn.Module):
```

```
    def __init__(self, scale_factor):
        # 上采样块数，8 倍就有 3 个
        upsample_block_num = int(math.log(scale_factor, 2))

        super(Generator, self).__init__()
        # 连接卷积层和激活函数层
        self.block1 = nn.Sequential(
            # 3 个通道，64 个卷积核，卷积大小为 9，需要扩充
            nn.Conv2d(3, 64, kernel_size=9, padding=4),
            nn.PReLU()
        )
        # 残差层
        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),
            # BN 层
            nn.BatchNorm2d(64)
        )
        # 上采样层
        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

# 判别器，较为简单，VGG 在计算损失时使用，这里没有。
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))
```

2、训练数据部分（train.py）

```
# 训练细节
```

```
        for data, target in train_bar:
            # 这个不太懂，知道的可以评论一下
            g_update_first = True
            batch_size = data.size(0)
            running_results['batch_sizes'] += batch_size

            ############################
            # (1) Update D network: maximize D(x)-1-D(G(z))
            # 最大化判别器判别原图(HR)概率，最小化生成图(SR)判别概率
            ###########################

            # HR
            real_img = Variable(target)
            if torch.cuda.is_available():
                real_img = real_img.cuda()
            # LR
            z = Variable(data)
            if torch.cuda.is_available():
                z = z.cuda()
            # SR
            fake_img = netG(z)

            #梯度清零
            netD.zero_grad()

            # 反向传播过程
            real_out = netD(real_img).mean()
            fake_out = netD(fake_img).mean()
            # 损失函数
            d_loss = 1 - real_out + fake_out
            # 反向传播
            d_loss.backward(retain_graph=True)
            # 参数更新
            optimizerD.step()

            ############################
            # (2) Update G network: minimize 1-D(G(z)) + Perception Loss + Image Loss + TV
Loss
            # 最小化生成网络中 SR 被认出概率、感知损失（VGG 计算）、图像损失(MSE)、
平滑损失
            ###########################

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

```
                ## The two lines below are added to prevent runetime error in Google Colab ##
                fake_img = netG(z)
                fake_out = netD(fake_img).mean()

                ##  计算损失及反向传播
                g_loss = generator_criterion(fake_out, fake_img, real_img)
                g_loss.backward()

                fake_img = netG(z)
                fake_out = netD(fake_img).mean()



                optimizerG.step()

                # loss for current batch before optimization
                running_results['g_loss'] += g_loss.item() * batch_size
                running_results['d_loss'] += d_loss.item() * batch_size
                running_results['d_score'] += real_out.item() * batch_size
                running_results['g_score'] += fake_out.item() * batch_size

                train_bar.set_description(desc='[%d/%d] Loss_D: %.4f Loss_G: %.4f D(x): %.4f
D(G(z)): %.4f' % (
                    epoch, NUM_EPOCHS, running_results['d_loss'] /
running_results['batch_sizes'],
                    running_results['g_loss'] / running_results['batch_sizes'],
                    running_results['d_score'] / running_results['batch_sizes'],
                    running_results['g_score'] / running_results['batch_sizes']))
```

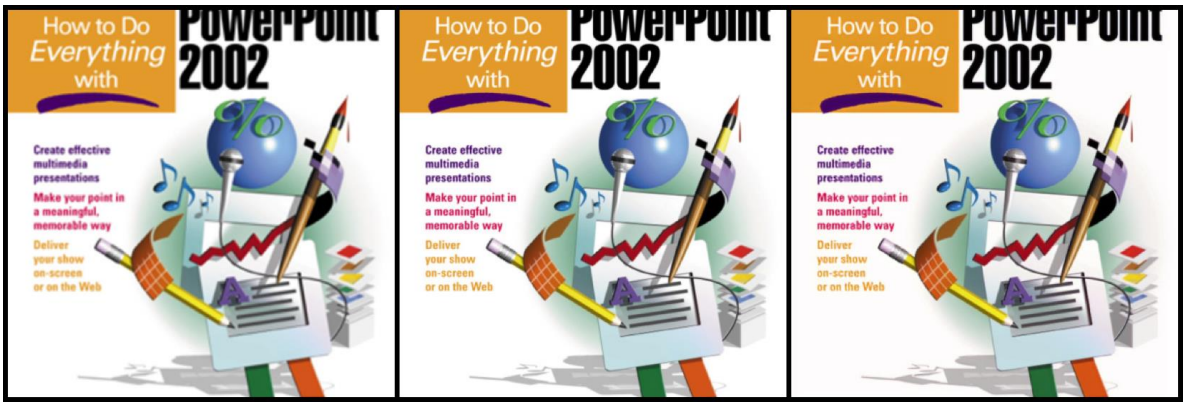
3、test.py（测试 SRGAN 模型）

```
    # 测试模式，无需更新网络。
        netG.eval()
        # 模型保存
        out_path = 'training_results/SRF_' + str(UPSCALE_FACTOR) + '/'
        if not os.path.exists(out_path):
            os.makedirs(out_path)
        # 参数计算
        with torch.no_grad():
            val_bar = tqdm(val_loader)
            valing_results = {'mse': 0, 'ssims': 0, 'psnr': 0, 'ssim': 0, 'batch_sizes': 0}
            val_images = []
            for val_lr, val_hr_restore, val_hr in val_bar:
                batch_size = val_lr.size(0)
                valing_results['batch_sizes'] += batch_size
                lr = val_lr
                hr = val_hr
```

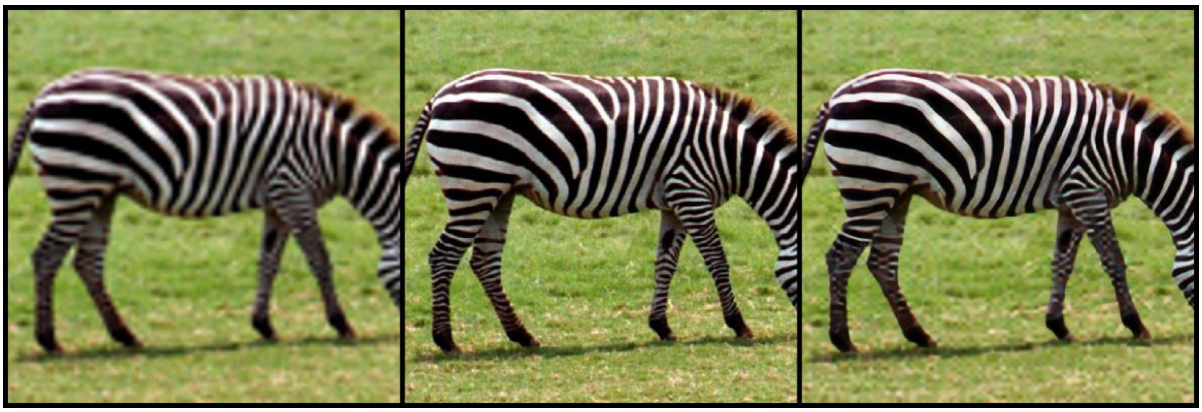
```
                    if torch.cuda.is_available():
                        lr = lr.cuda()
                        hr = hr.cuda()
                    sr = netG(lr)

                    batch_mse = ((sr - hr) ** 2).data.mean()
                    valing_results['mse'] += batch_mse * batch_size
                    batch_ssim = pytorch_ssim.ssim(sr, hr).item()
                    valing_results['ssims'] += batch_ssim * batch_size
                    valing_results['psnr'] = 10 * log10((hr.max()**2) / (valing_results['mse'] / valing_results['batch_sizes']))
                    valing_results['ssim'] = valing_results['ssims'] / valing_results['batch_sizes']

                    val_bar.set_description(
                        desc='[converting LR images to SR images] PSNR: %.4f dB SSIM: %.4f' % (
                            valing_results['psnr'], valing_results['ssim']))
                    # 拼接三张图片 如果想提高训练速度 下面到 index += 1 可以注释
                    val_images.extend(
                        [display_transform()(val_hr_restore.squeeze(0)), display_transform()(hr.data.cpu().squeeze(0)),
                         display_transform()(sr.data.cpu().squeeze(0))])
                val_images = torch.stack(val_images)
                val_images = torch.chunk(val_images, val_images.size(0) // 15)
                val_save_bar = tqdm(val_images, desc='[saving training results]')
                index = 1
                for image in val_save_bar:
                    image = utils.make_grid(image, nrow=3, padding=5)
                    utils.save_image(image, out_path + 'epoch_%d_index_%d.png' % (epoch, index), padding=5)
                    index += 1

            # save model parameters
            torch.save(netG.state_dict(), 'epochs/netG_epoch_%d_%d.pth' % (UPSCALE_FACTOR, epoch))
            torch.save(netD.state_dict(), 'epochs/netD_epoch_%d_%d.pth' % (UPSCALE_FACTOR, epoch))
```

三、结果展示

（1）、示例一



（2）、示例二



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, test.py, train.py 等文件，其中 netD_epoch_4_1.pth 是已经训练好的参数。

（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 epoch 文件下。