计算机视觉实践实验报告

目录

一. 实验目的

- 改进SRCNN。
- 在Set-5数据集上进行测试，并对实验进行总结分析。

二. 实验原理

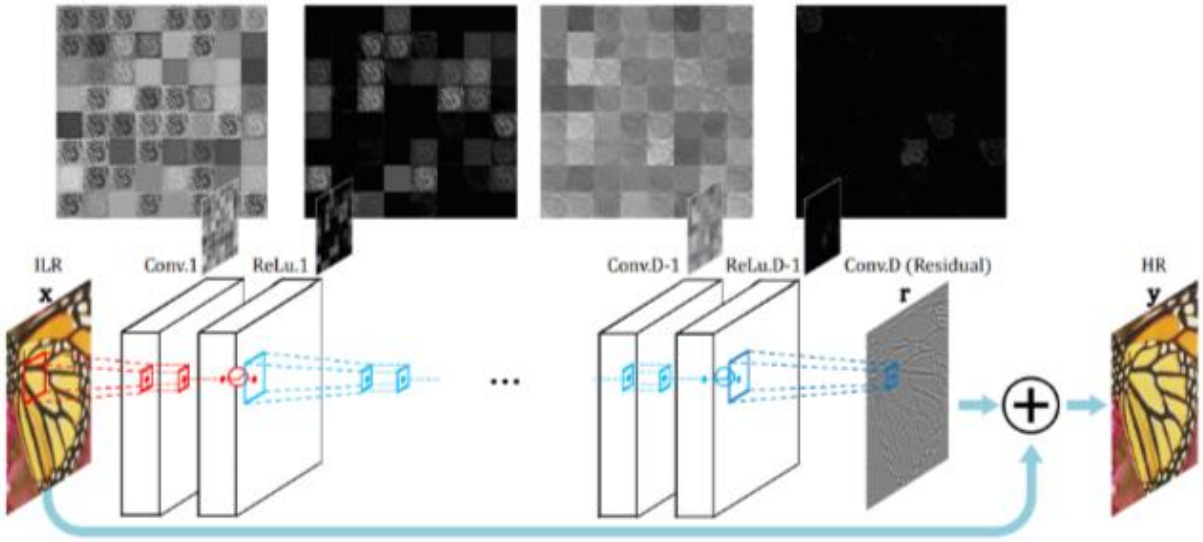
VDSR使用了一个非常深的卷积网络，灵感来自于用于ImageNet分类的VGG网络，网络深度的增加可以使得精度的显著提高。该模型的深度为20层，通过在深层网络结构中多次级联小型过滤器，可以有效地利用大型图像区域上的上下文信息。然而在非常深的网络中，收敛速度成为训练中的关键问题。VDSR提出一个简单而有效的训练程序。仅学习残差，使用极高的学习率（比SRCNN高出104倍），通过可调梯度裁剪实现。

2.1 SRCNN局限性

SRCNN在超分辨率（SR）问题中成功地引入了深入的学习技术，但在三个方面有局限性：第一，它依赖于小图像区域的上下文；第二，训练收敛太慢；第三，网络只适用于一个采样scale。即：训练层数少，没有足够的视野域；训练太慢，导致没有在深层网络上得到好的效果；不能支持多种倍数的高清化。VDSR可以解决以上问题。

2.2 VDSR网络结构

- 输入为插值后的低分辨率图像。
- 除第一层和最后一层之外，中间d层有相似结构：64个滤波器，尺寸为3 × 3 × 64，每一个滤波器跨64个通道，在3 × 3空间区域内操作。
- 第一层，对输入图像操作。
- 最后一层用于图像重建，包含一个滤波器，尺寸为3 × 3 × 64。
- 在每个卷积层之前补0保证特征图和输入图像尺寸一样。



2.3 VDSR解决问题方法

- 上下文信息（Context）：通过stack small filters来进行获得一个比较大的感受野，最大达到41x41，深网络可以使用较大的感受野，这可以充分考虑上下文信息。
- 收敛（Convergence）：加速训练：通过残差学习和极高学习率。
- 尺度（Scale Factor）：一个单一的神经网络可以针对多尺度超分辨率。

2.4 残差学习

为了解决梯度弥散/梯度爆炸问题（vanishing/expoding gradients problem）；
定义残差图像：r=y-x;
损失函数定义为：

$$\frac{1}{2}\|\mathbf{r}-f(\mathbf{x})\|^2$$

f(x)是网络预测；
损失层有三个输入：残差估计；网络输入(interpolated low-resolution)图像和基准HR图像。

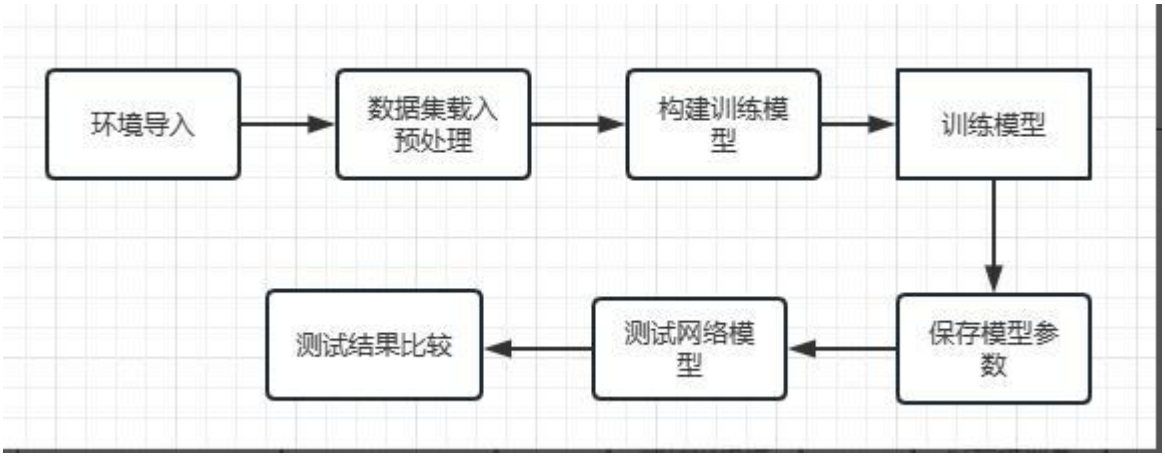
2.5 可调梯度裁剪

梯度裁剪是一种常用的训练递归神经网络的方法。但是，它在训练cnn时的使用是有限的。然而还是存在许多限制梯度的方法，一个常见的策略是裁剪独立在预定义的范围[-θ,θ]内。
利用训练中常用的随机梯度下降法，将学习率乘以调整步长。如果使用学习速率高,很可能θ是被调小以避免爆炸在高梯度学习速率的regime。但随着学习速率的减小，有效梯度(梯度乘以学习速率)趋近于零，如果学习速率呈几何级数下降，训练可能需要指数次迭代才能收敛。
为了最大化收敛速度,我们裁剪梯度在$[-\frac{\theta}{\gamma},\frac{\theta}{\gamma}]$,γ表示当前的学习速率。可调梯度裁剪使收敛过程非常快。20层网络训练可以在4小时内完成，而3层SRCNN需要几天的时间来训练。

2.6 多尺度

VDSR还训练了一个多尺度模型。使用这种方法，所有预定义的缩放因子都可以共享参数。训练一个多尺度模型很简单。针对几个指定规模的训练数据集被组合成一个大数据集。

三. 实验步骤



四. 数据集

训练数据集使用91-image，根据作者提供的matlab源码，生成训练用数据。

测试Set-5数据集。

五. 程序代码

- 定义数据集的读取方式，两个方法：getitem len。

```
class DatasetFromHdf5(data.Dataset):
    def __init__(self, file_path):
        super(DatasetFromHdf5, self).__init__()
        hf = h5py.File(file_path)
        self.data = hf.get('data')
        self.target = hf.get('label')

    def __getitem__(self, index):
        return torch.from_numpy(self.data[index, :, :, :]).float(), torch.from_numpy(
            self.target[index, :, :, :]).float()

    def __len__(self):
        return self.data.shape[0]
```

- 神经网络结构块（64*64*3）。

```python
class Conv_ReLU_Block(nn.Module):
    def __init__(self):
        super(Conv_ReLU_Block, self).__init__()
        # stride步长为1 padding填充为1 bias不添加偏置参数作为可学习参数
        self.conv = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride=1, padding=1, bias=False)
        # 对从上层网络Conv2d中传递下来的tensor直接进行修改，inplace变量替换能够节省运算内存，不用多存储其他变量
        self.relu = nn.ReLU(inplace=True)

    def forward(self, x):
        return self.relu(self.conv(x))
```

- VDSR主要网络结构。

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.residual_layer = self.make_layer(Conv_ReLU_Block, 18)
        self.input = nn.Conv2d(in_channels=1, out_channels=64, kernel_size=3, stride=1, padding=1, bias=False)
        self.output = nn.Conv2d(in_channels=64, out_channels=1, kernel_size=3, stride=1, padding=1, bias=False)
        self.relu = nn.ReLU(inplace=True)

        # Conv2d中参数的初始化 normal高斯
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                # 3 3 64  最后一次3 3 1
                # print(m.kernel_size[0], m.kernel_size[1], m.out_channels)
                n = m.kernel_size[0] * m.kernel_size[1] * m.out_channels
                m.weight.data.normal_(0, sqrt(2. / n))

    def make_layer(self, block, num_of_layer):
        layers = []
        for _ in range(num_of_layer):
            layers.append(block())
        # Sequential一个有序的容器，神经网络模块将按照在传入构造器的顺序依次被添加到计算图中执行
        return nn.Sequential(*layers)

    def forward(self, x):
        residual = x
        out = self.relu(self.input(x))
        out = self.residual_layer(out)
        out = self.output(out)
        out = torch.add(out, residual)
        return out
```

- VDSR训练模块。

```python
def train(training_data_loader, optimizer, model, criterion, epoch):
    lr = adjust_learning_rate(optimizer, epoch - 1)
    # param_groups的各项参数：[{'params','lr', 'momentum', 'dampening', 'weight_decay', 'nesterov'},{……}]
    for param_group in optimizer.param_groups:
        param_group["lr"] = lr
    print("Epoch = {}, lr = {}".format(epoch, optimizer.param_groups[0]["lr"]))
    # 接下来是训练部分 model.train()保证Batch Normalization层用每一批数据的均值和方差，对于Droupout:model.train()是随机取一部分网
    # model.eval()是保证Batch Normalization用全部训练数据的均值和方差，对于Droupout:model.eval()是利用到了所有网络连接
    model.train()
    # 遍历training_data_loader，从下标1开始
    for iteration, batch in enumerate(training_data_loader, 1):
        # running_loss = 0.0
        # Variable是一种可以不断变化的变量，符合反向传播，参数更新的属性
        input, target = Variable(batch[0]), Variable(batch[1], requires_grad=False)

        if opt.cuda:
            input = input.cuda()
            target = target.cuda()

        loss = criterion(model(input), target)

        if iteration % 100 == 0:
            output = "===> Epoch[{}]({}/{}): Loss: {:.10f}".format(epoch, iteration, len(training_data_loader),
                                                                   loss.item())
            with open("loss.txt", "a+") as f:
                f.write(output + '\n')
                f.close

        optimizer.zero_grad()
        loss.backward()
        # 梯度裁剪
        nn.utils.clip_grad_norm_(model.parameters(), opt.clip)
        optimizer.step()

        if iteration % 100 == 0:
            print(
                "===> Epoch[{}]({}/{}): Loss: {:.10f}".format(epoch, iteration, len(training_data_loader), loss.ite
```

- 通过MSE来计算PSNR峰值信噪比。

```python
def PSNR(pred, gt, shave_border=0):
    height, width = pred.shape[:2]
    pred = pred[shave_border:height - shave_border, shave_border:width - shave_border]
    gt = gt[shave_border:height - shave_border, shave_border:width - shave_border]
    imdff = pred - gt
    rmse = math.sqrt(np.mean(imdff ** 2))
    if rmse == 0:
        return 100
    return 20 * math.log10(255.0 / rmse)
```

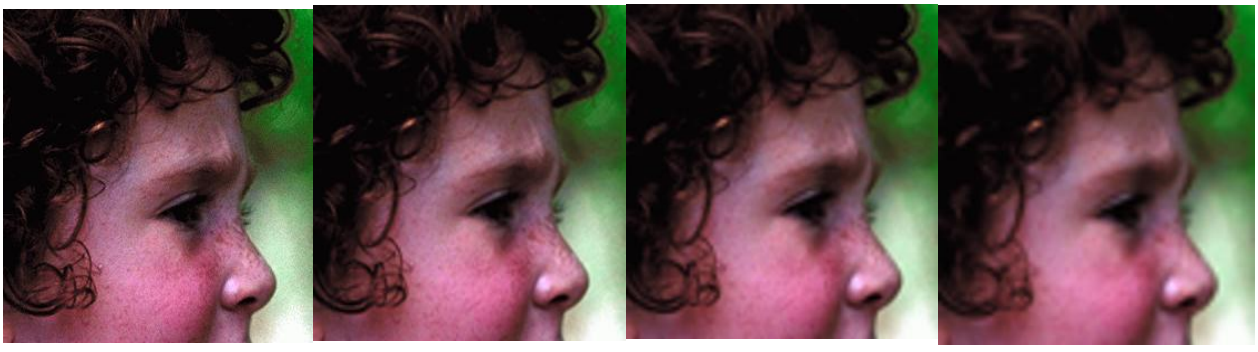
六. 实验结果

- 在Set-5数据集上的实验结果（从前到后缩放倍数分别为2，3，4）：

37.15dB　　33.93dB　　31.78dB

40.40dB　　34.92dB　　31.46dB

32.58dB　　28.22dB　　25.34dB



34.47dB　　32.53dB　　30.97dB

34.34dB　　30.73dB　　27.99dB

七. 实验分析与总结

- 可以看到VDSR的实验结果不仅psnr相比于SRCNN要高，而且视觉效果上更好。网络的加深确实起到了一定的作用。
- VDSR在缩放倍数为2倍的时候效果最好。