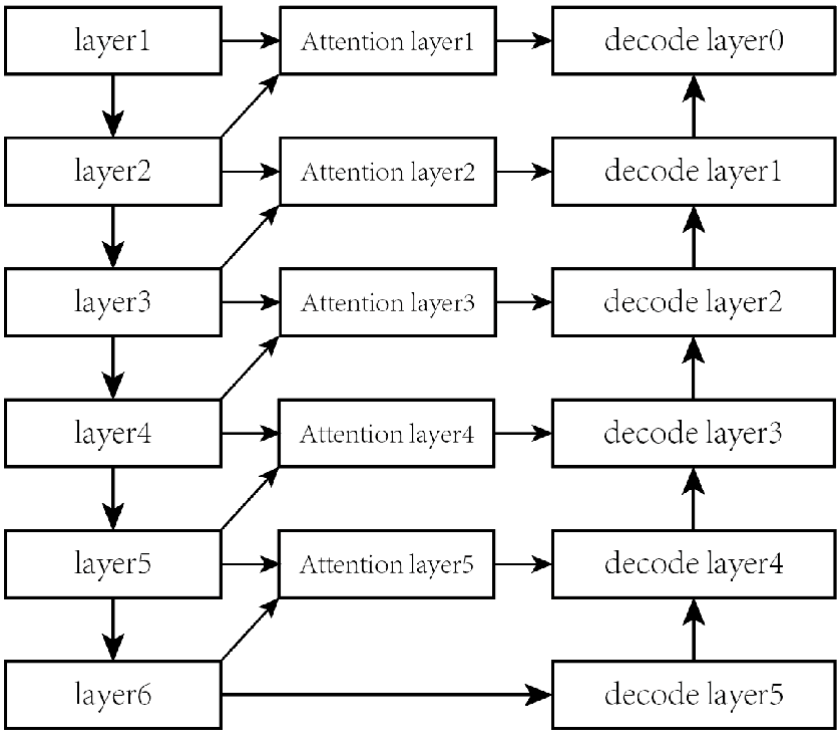
Lab 5 - 深度学习图像补全

一、实验要求

本次实验我们用深度学习给图片进行补全。

二、理论基础

按照下图的结构实现一个稍微复杂的图像补全网络。



三、运行环境

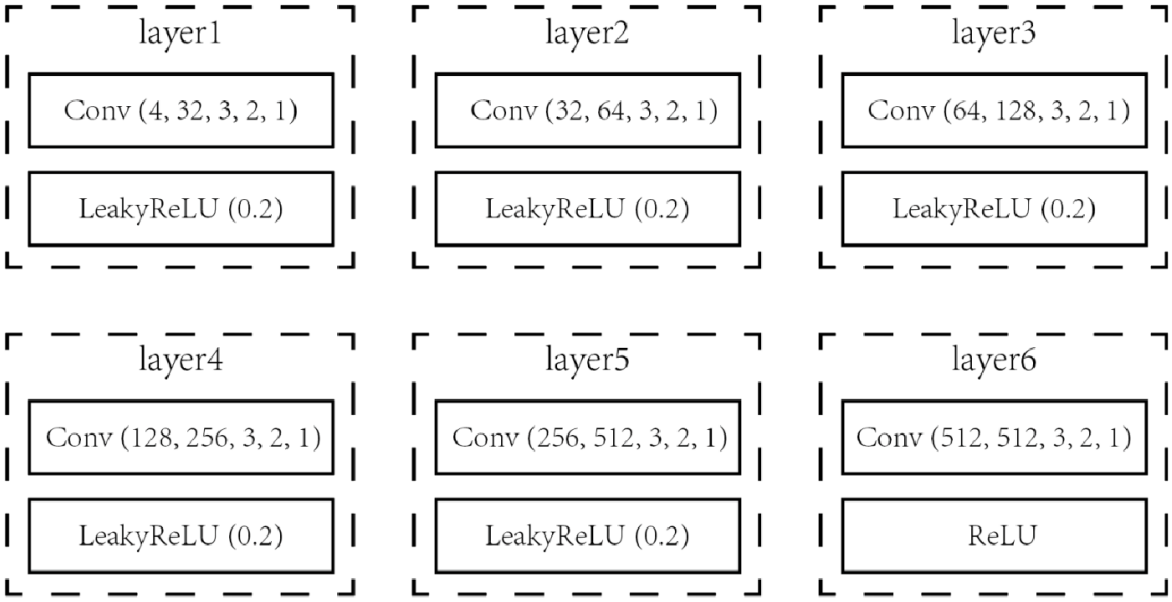
编译环境：`PyCharm Community Edition 2021.3.3`

运行环境：`Windows10`

四、具体实现

4.1 `Layer` 实现

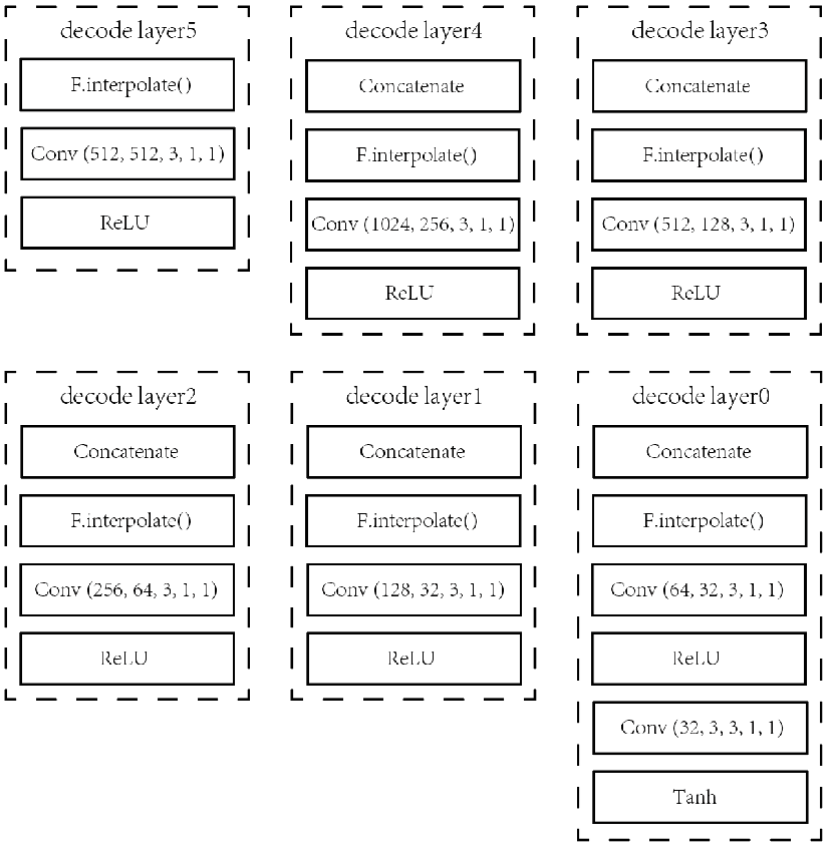
根据下图完善 `layer1~6` 的代码，实现非常简单，只需要将每一步依样画葫芦写出即可。




```
self.dw_conv01 = nn.Sequential(
    # layer 1
    nn.Conv2d(4,32,kernel_size=3,stride=2,padding=1),
    nn.LeakyReLU(0.2)
)
self.dw_conv02 = nn.Sequential(
    # layer 2
    nn.Conv2d(32, 64, kernel_size=3, stride=2, padding=1),
    nn.LeakyReLU(0.2)
)
self.dw_conv03 = nn.Sequential(
    # layer 3
    nn.Conv2d(64, 128, kernel_size=3, stride=2, padding=1),
    nn.LeakyReLU(0.2)
)
self.dw_conv04 = nn.Sequential(
    # layer 4
    nn.Conv2d(128,256, kernel_size=3, stride=2, padding=1),
    nn.LeakyReLU(0.2)
)
self.dw_conv05 = nn.Sequential(
    # layer 5
    nn.Conv2d(256, 512, kernel_size=3, stride=2, padding=1),
    nn.LeakyReLU(0.2)
)
self.dw_conv06 = nn.Sequential(
    # layer 6
    nn.Conv2d(512, 512, kernel_size=3, stride=2, padding=1),
    nn.ReLU()
)
```

4.2 `Decode Layer` 实现

根据下图完善 `decode layer` 的代码，我们发现这里多了 `F.interpolate()` 和 `Concatenate` 函数，如果直接将其加入到 `nn.Sequential()` 中会出现问题，同时发现这些操作都在 `Conv()`、`ReLU`、`Tanh` 等操作的前面，且操作是依次线性进行的，因此我们可以将这两个函数放在后面的部分，即 `def forward(self, img, mask)` 中。当我们想得到正确的结果时，只需要先依次进行上述两个操作，再调用相应的 `self.up_conv01~05()` 和 `self.decoder()` 函数即可。




```
self.up_conv05 = nn.Sequential(
    # decode layer 5
    nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
    nn.ReLU()
)
self.up_conv04 = nn.Sequential(
    # decode layer 4
    nn.Conv2d(1024, 256, kernel_size=3, stride=1, padding=1),
    nn.ReLU()
)
self.up_conv03 = nn.Sequential(
    # decode layer 3
    nn.Conv2d(512, 128, kernel_size=3, stride=1, padding=1),
    nn.ReLU()
)
self.up_conv02 = nn.Sequential(
    # decode layer 2
    nn.Conv2d(256, 64, kernel_size=3, stride=1, padding=1),
    nn.ReLU()
)
self.up_conv01 = nn.Sequential(
    # decode layer 1
    nn.Conv2d(128, 32, kernel_size=3, stride=1, padding=1),
    nn.ReLU()
)
self.decoder = nn.Sequential(
    # decode layer 0
    nn.Conv2d(64, 32, kernel_size=3, stride=1, padding=1),
    nn.ReLU(),
    nn.Conv2d(32, 3, kernel_size=3, stride=1, padding=1),
    nn.Tanh()
)
```

下面是完善`def forward(self, img, mask)`中的部分，我们需要在这里实现`F.interpolate()`和`Concatenate`函数的功能。

```
# decoder
# decode layer 5
up_x5 = F.interpolate(x6, scale_factor=2, mode='bilinear', align_corners=True)
up_x5 = self.up_conv05(up_x5)
# decode layer 4
up_x4 = torch.cat([up_x5,x5], dim=1)
up_x4 = F.interpolate(up_x4, scale_factor=2, mode='bilinear',
align_corners=True)
up_x4 = self.up_conv04(up_x4)
# decode layer 3
up_x3 = torch.cat([up_x4, x4], dim=1)
up_x3 = F.interpolate(up_x3, scale_factor=2, mode='bilinear',
align_corners=True)
up_x3 = self.up_conv03(up_x3)
# decode layer 2
up_x2 = torch.cat([up_x3, x3], dim=1)
up_x2 = F.interpolate(up_x2, scale_factor=2, mode='bilinear',
align_corners=True)
up_x2 = self.up_conv02(up_x2)
# decode layer 1
up_x1 = torch.cat([up_x2, x2], dim=1)
up_x1 = F.interpolate(up_x1, scale_factor=2, mode='bilinear',
align_corners=True)
up_x1 = self.up_conv01(up_x1)
# decode layer 0
output = torch.cat([up_x1, x1], dim=1)
output = F.interpolate(output, scale_factor=2, mode='bilinear',
align_corners=True)
output = self.decoder(output)
```

五、实验结果及分析

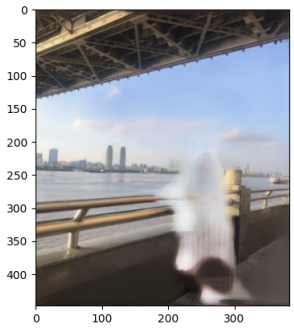
5.1运行结果

下面三张是测试样例，我自己的图片和扣去自己的图片，相应的mask在文件夹中，未放在报告文档中。

下面是通过不同模型跑出来的结果。

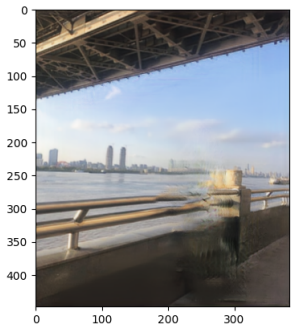
5.1.1`celebahq`模型

对于题目给出的测试样例，本模型效果一般，对于我自己的图片，本模型效果较差，可以明显地看到一个人型的糊化图案。

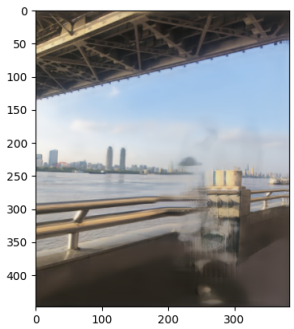
5.1.2 `dtd` 模型

对于题目给出的测试样例，本模型效果较差，可以在结果中看到一大块白色的区域，对于我自己的图片，本模型效果较好，基本实现了扣除指定区域的任务，除了一些区域和原图的衔接效果较差，其余部分完成地比较好。

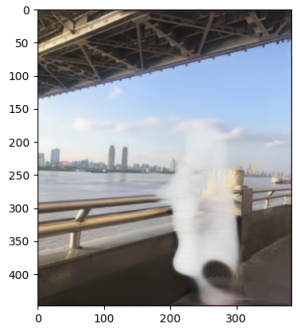
5.1.3 `facade` 模型

对于题目给出的测试样例，本模型效果较好，原人型区域在处理后不明显，对于我自己的图片，本模型效果介于上面两种模型之间，但结果也属于我比较满意的范围，可以大致地达到目标。

5.1.4 `places2` 模型

对于题目给出的测试样例，本模型效果和上一种不分伯仲，对于我自己的图片，本模型效果较差，在处理之后仍然可以看到一大块白色的区域。



5.2结果分析与改进

可以看到不同网络结构训练后得到的结果也不同，即使是同一个模型，输入不同其效果也不同。对于题目给出的样例和我自己选择的图片，不同模型展现的效果都有很大差异，以`dtd`和`places2`为例，这两个模型针对两个输入便给出了效果截然相反的输出图像。

当我们用不同网络时结果出现了不同的人型印记，这一点我觉得也可能是由于训练数据不够或是由于图像本身造成的，如果输入的图像本身mask周围的各部分对比度较高或是像素差异较大，补全的效果可能会更好一些。

六、心得体会

本次实验的代码量较小，在理解透彻原理后可以较快地完成代码工作，利用代码进行图像补全得到结果显得更加重要和有趣味性，因此本次实验我认为应该以体验和学习为主。对于不常使用如此大规模训练数据网络的我而言，在进行本次实验相关的任务时，用本机的CPU跑模型时常会出现内存不够的窘况，因此总要把图片压缩到一定程度才能正常进行补全。总体而言，本次实验还是十分具有趣味性的，涉及的范围也比较广，虽然对网络的结构及其他知识点方面浅尝辄止，但是比之前有了更深入的理解，最重要的是有了实操经验，相信如果之后再有相关的任务，上手会比现在快很多。

七、参考资料

[1]. https://www.cnblogs.com/askayoyoo/p/11589717.html

[2]. https://www.cnblogs.com/shoufengwei/p/8526105.html