Lab 6 - 深度学习图像补全

学号：3210106034
姓名：王伟杰

> 实验环境：
>
> ```
> $ uname -a
> Linux ****** 5.15.0-101-generic #111~20.04.1-Ubuntu SMP Mon Mar 11
> 15:44:43 UTC 2024 x86_64 x86_64 x86_64 GNU/Linux
> $ python --version
> Python 3.9.19
> ```

实验内容

- 完成网络结构实现图像补全
- 找一张自己的图片，将自己从图片中扣去，然后用网络补全图像

理论分析

基础知识

卷积

卷积是深度学习中一种非常基础的操作，特别是在卷积神经网络（CNN）中。它通过一个称为“卷积核”或“滤波器”的小矩阵在输入图像上滑动，对每一小块区域进行加权求和，这个过程可以捕捉到局部的特征，如边缘、角点等。卷积操作可以保持空间关系，使得网络能够学习到图像中的空间层次结构。

ReLU激活函数

ReLU（Rectified Linear Unit，线性整流单元）是一种非常流行的激活函数，通常用于增加网络的非线性。其数学表达式很简单：`f(x) = max(0, x)`。这意味着对于所有的正输入值，输出与输入相同；而所有的负输入值都会被置为0。ReLU的优点是它能够在加速训练的同时减少梯度消失问题。

最大池化

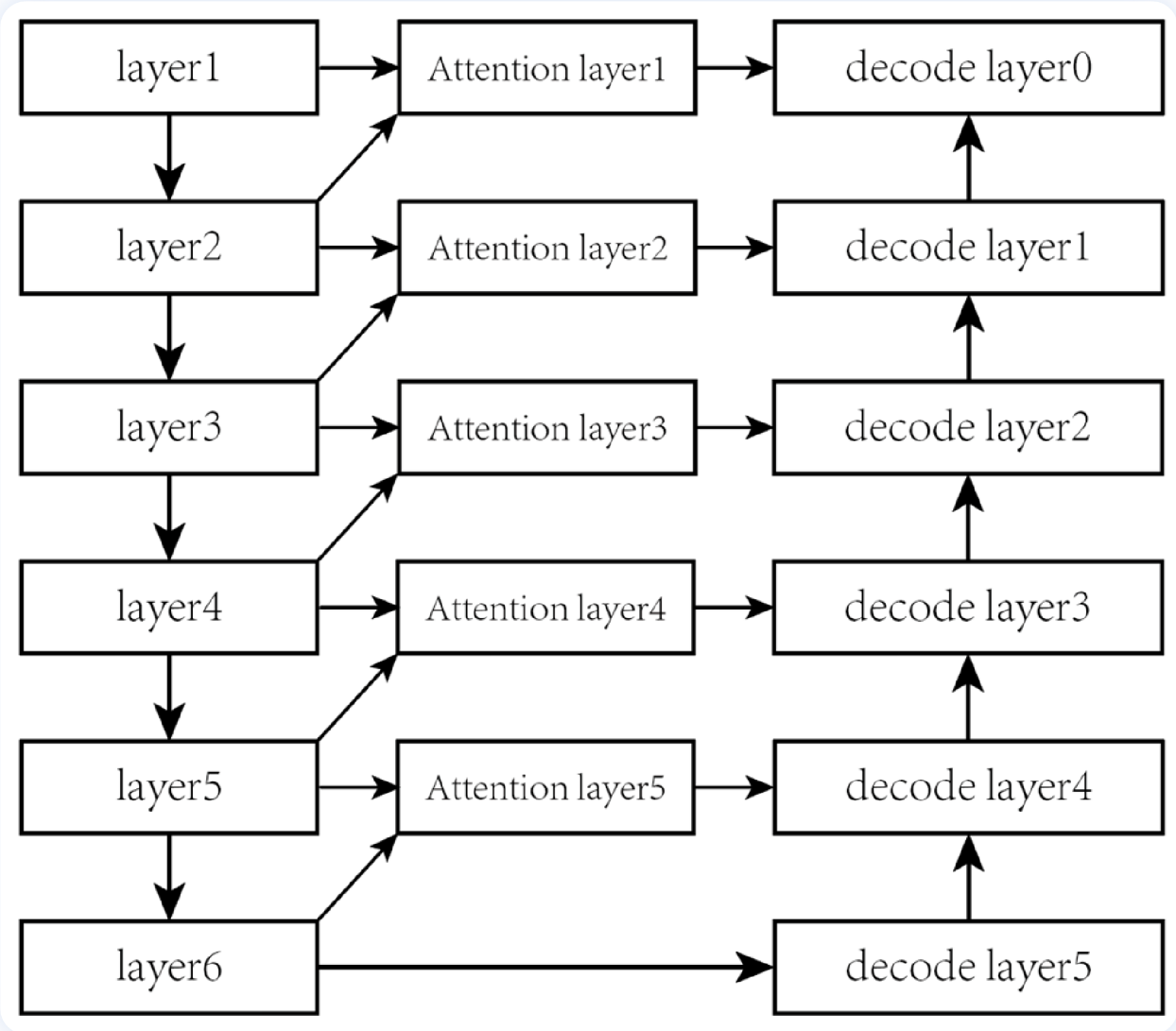
MaxPooling是一种池化操作，用于减少输入的维度和参数的数量，从而减少计算量并防止过拟合。在这个过程中，输入数据被分割成多个区域，每个区域只保留最大的值。这样可以在保留重要特征的同时，实现对输入数据的下采样。

上采样

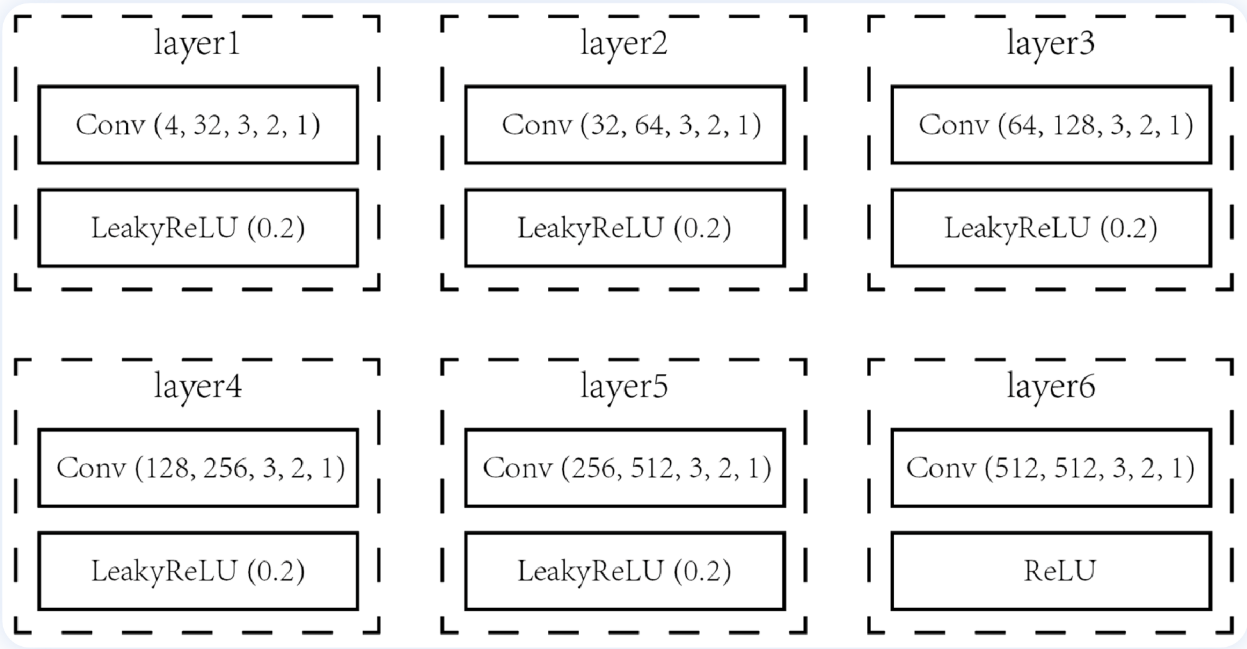
上采样是将数据从低维空间映射到高维空间的过程。`F.interpolate`是PyTorch中的一个函数，用于实现上采样，它可以通过不同的方法（如最近邻插值、线性插值等）来增加数据的尺寸。上采样常用于深度学习中的图像分割和超分辨率等任务，其中模型需要从低分辨率的特征图重构出高分辨率的输出。

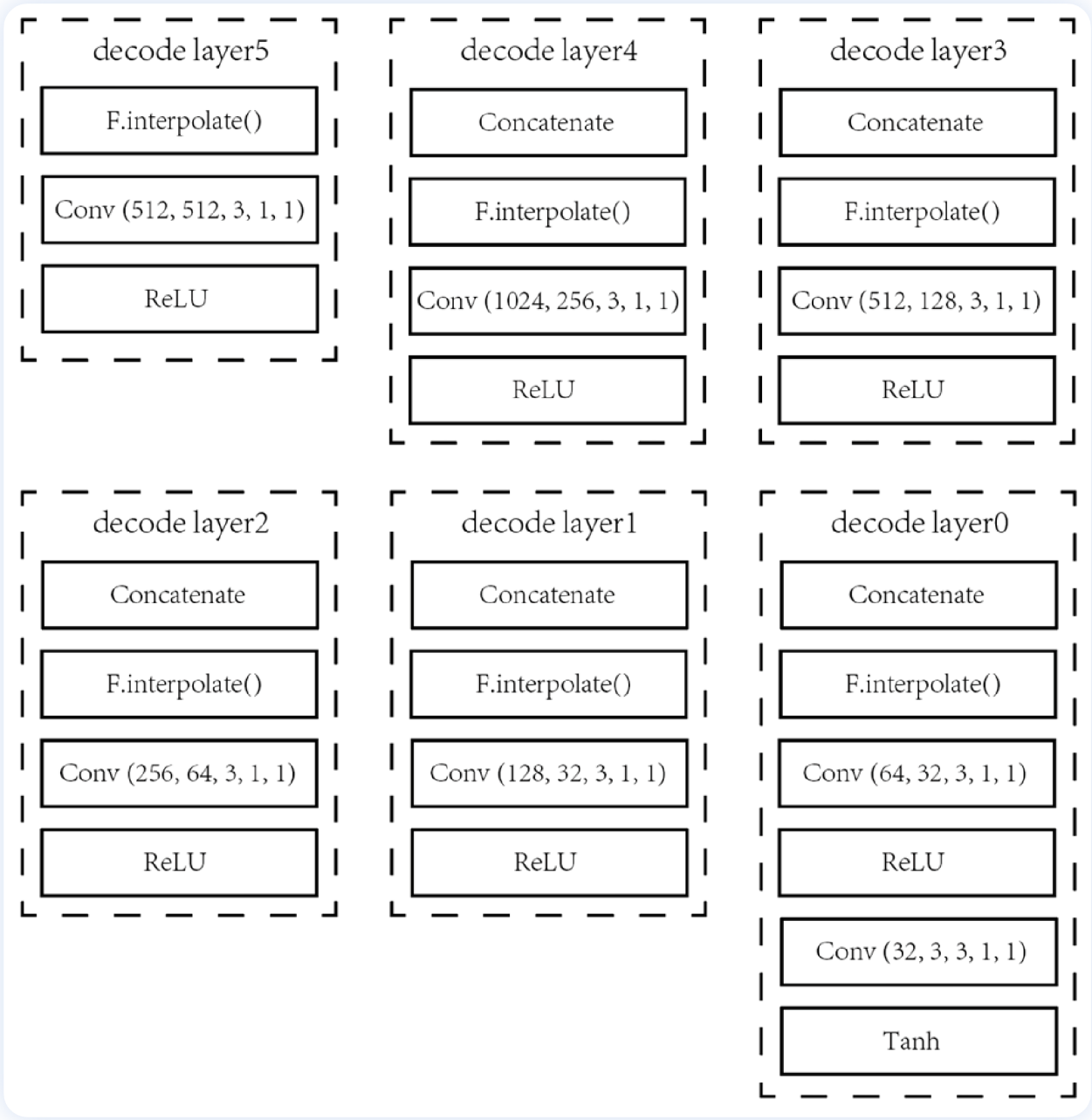
网络结构

网络输入为masked_image和mask图片，这次使用的网络结构如下图所示



其中各层和解码器的详细结构解释如下图：





实验细节

补全网络结构

按照理论分析中的结构图补全网络结构：

```
def __init__(self):
    super(Generator, self).__init__()

    self.dw_conv01 = nn.Sequential(
```

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

        # attention module
        cnum = 32
        self.at_conv05 = AttentionConv(cnum * 16, cnum * 16, ksize=1, fuse=False)
        self.at_conv04 = AttentionConv(cnum * 8, cnum * 8)
        self.at_conv03 = AttentionConv(cnum * 4, cnum * 4)
        self.at_conv02 = AttentionConv(cnum * 2, cnum * 2)
        self.at_conv01 = AttentionConv(cnum, cnum)

        # decoder
        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()
        )
```

在 `forward` 函数中，注意使用 `F.interpolate()` 进行上采样，并用 `torch.cat` 进行拼接。

```
    def forward(self, img, mask):
        x = img

        # layer 1
        x1 = self.dw_conv01(x)
        # layer 2
        x2 = self.dw_conv02(x1)
        # layer 3
        x3 = self.dw_conv03(x2)
        # layer 4
        x4 = self.dw_conv04(x3)
        # layer 5
        x5 = self.dw_conv05(x4)
        # layer 6
        x6 = self.dw_conv06(x5)

```

```
        # attention
        x5 = self.at_conv05(x5, x6, mask)
        x4 = self.at_conv04(x4, x5, mask)
        x3 = self.at_conv03(x3, x4, mask)
        x2 = self.at_conv02(x2, x3, mask)
        x1 = self.at_conv01(x1, x2, mask)

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

        return output
```

结果展示

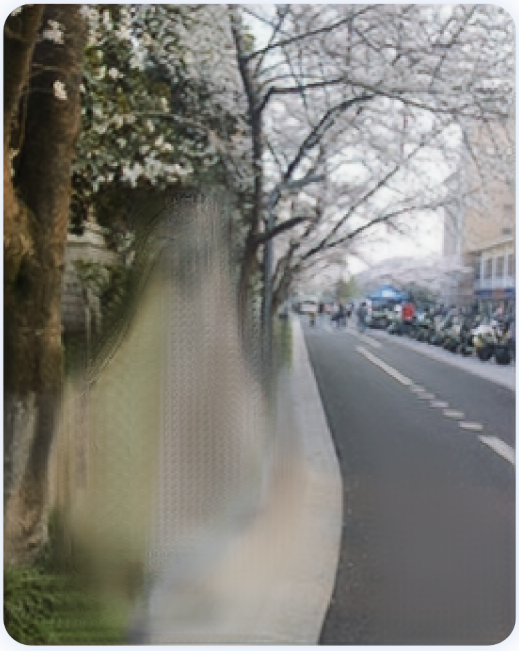
原图

扣去自己的图片

相应的mask

补全后的图片

- celebahq:



- dtd:

- facade:

- places2: