卷积神经网络实验报告

姓名：林坤 学号：1911433

一、 实验要求

- 掌握卷积的基本原理
- 学会使用 PyTorch 搭建简单的 CNN 实现 Cifar10 数据集分类
- 学会使用 PyTorch 搭建简单的 ResNet 实现 Cifar10 数据集分类
- 学会使用 PyTorch 搭建简单的 DenseNet 实现 Cifar10 数据集分类

二、 报告内容

1. 老师提供的原始版本 CNN 网络结构（可用 print(net) 打印，复制文字或截图皆可）、在Cifar10 验证集上的训练 loss 曲线、准确度曲线图
2. 个人实现的 ResNet 网络结构在上述验证集上的训练 loss 曲线、准确度曲线图
3. 个人实现的 DenseNet 网络结构在上述验证集上的训练 loss 曲线、准确度曲线图
4. 解释没有跳跃连接的卷积网络、ResNet、DenseNet 在训练过程中有什么不同（重点部分）

三、 原始版本CNN网络

网络结构代码如下：

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
```

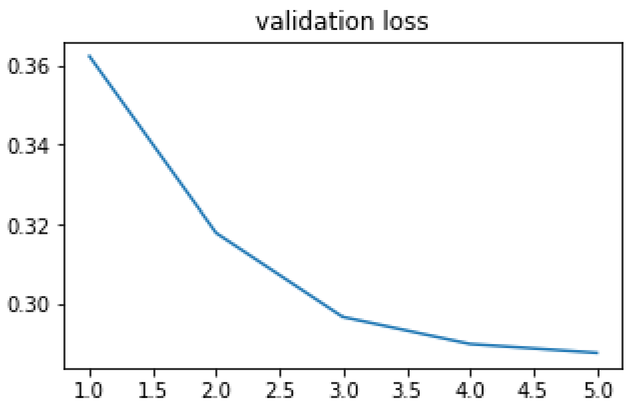
```
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

结构图显示：

```
Net(
    (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
    (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=
    (conv2) : Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
    (fc1) : Linear(in_features=400, out_features=120, bias=True)
    (fc2) : Linear(in_features=120, out_features=84, bias=True)
    (fc3) : Linear(in_features=84, out_features=10, bias=True)
)
```

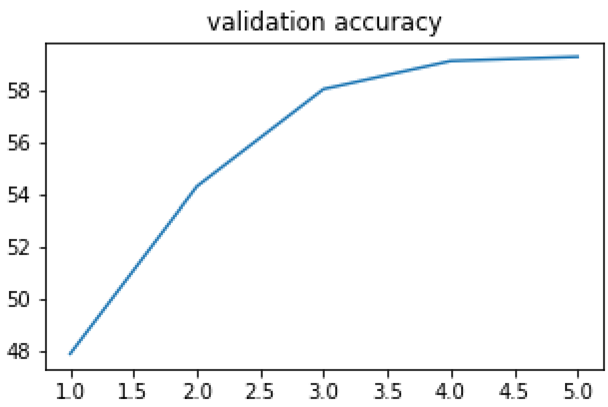
在 Cifar10 **训练集上的训练** loss **曲线** (5 epochs)**：**

Training set: Average loss: 0.29



在 Cifar10 **验证集上的准确度曲线图** (5 epochs)**：**

Validation set: Accuracy: 59%



四、个人实现Resnet网络

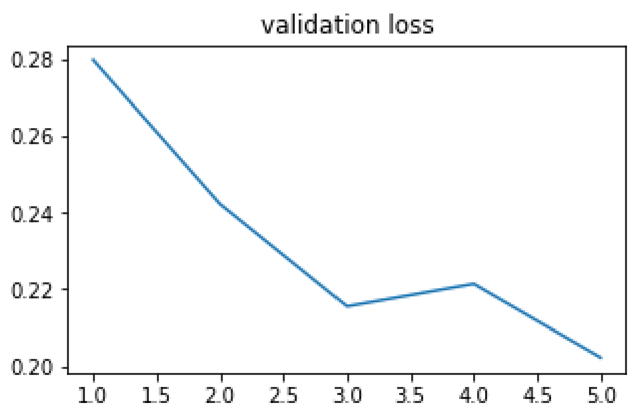
实现的微型 ResNet-18 **网络结构如下**

```
ResNet(
  (conv1): Conv2d(3, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (bn1): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (reslayer1): ResidualBlock(
    (res1): Sequential(
      (0): Conv2d(8, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
      (3): Conv2d(8, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (4): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (shortcut): Sequential()
  )
  (reslayer2): ResidualBlock(
    (res1): Sequential(
      (0): Conv2d(8, 16, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
      (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
      (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (shortcut): Sequential(
      (0): Conv2d(8, 16, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (reslayer3): ResidualBlock(
    (res1): Sequential(
      (0): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
      (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (shortcut): Sequential()
  )
  (reslayer4): ResidualBlock(
    (res1): Sequential(
      (0): Conv2d(16, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
      (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```
      (2): ReLU(inplace=True)
      (3): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (4): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (shortcut): Sequential(
      (0): Conv2d(16, 32, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (reslayer5): ResidualBlock(
    (res1): Sequential(
      (0): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
      (3): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (4): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (shortcut): Sequential()
  )
  (pool): AvgPool2d(kernel_size=4, stride=4, padding=0)
  (fc): Linear(in_features=128, out_features=10, bias=True)
)
```

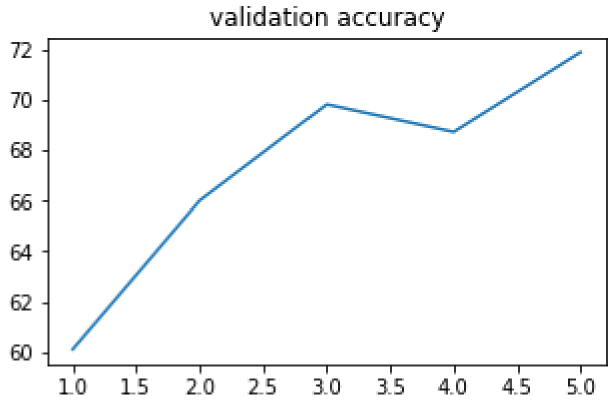
在 Cifar10 **训练集上的训练** loss **曲线** (5 epochs)**：**

最终 loss 值：Train set: Average loss: 0.202



在 Cifar10 **验证集上的** accuracy **曲线** (5 epochs)**：**

最终 accuracy 值：Accuracy: 7188/10000 (72%)



五、个人实现Densenet网络

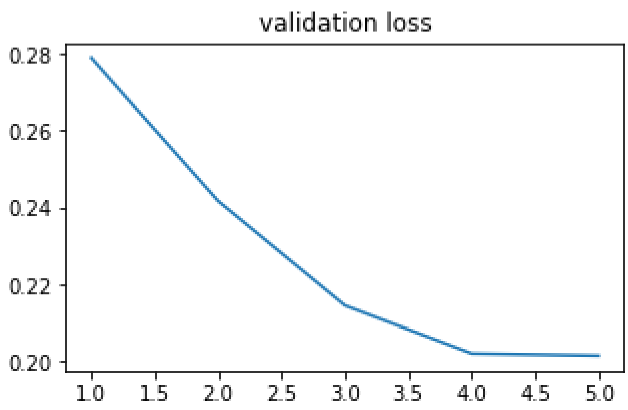
个人实现的微型 DensNet **网络结构如下：**

```
DenseNet(
 (block1): Sequential(
  (0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3))
  (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): ReLU(inplace=True)
  (3): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
 )
 (block2): Sequential(
  (0): dense_block(
   (net): Sequential(
    (0): Sequential(
     (0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
     (1): ReLU(inplace=True)
     (2): Conv2d(64, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
    (1): Sequential(
     (0): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
     (1): ReLU(inplace=True)
     (2): Conv2d(96, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
   )
  )
  (1): Sequential(
   (0): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
   (1): ReLU(inplace=True)
   (2): Conv2d(128, 64, kernel_size=(1, 1), stride=(1, 1))
   (3): AvgPool2d(kernel_size=2, stride=2, padding=0)
  )
  (2): dense_block(
```

```
    (net): Sequential(
      (0): Sequential(
        (0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (1): ReLU(inplace=True)
        (2): Conv2d(64, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (1): Sequential(
        (0): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (1): ReLU(inplace=True)
        (2): Conv2d(96, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (2): Sequential(
        (0): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (1): ReLU(inplace=True)
        (2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
    )
  )
  (bn): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (avg_pool): AvgPool2d(kernel_size=3, stride=3, padding=0)
 )
 (classifier): Linear(in_features=160, out_features=10, bias=True)
)
```

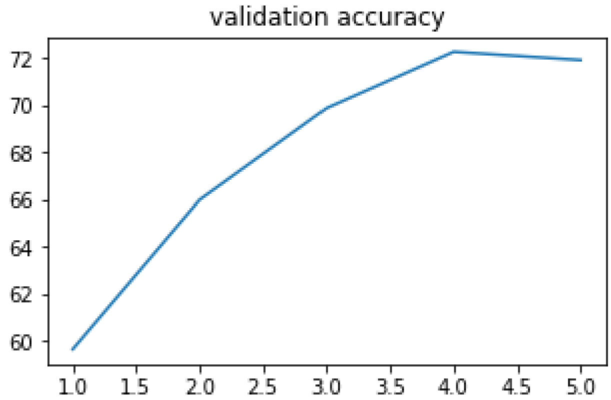
在 Cifar10 **训练集上的训练** loss **曲线** (5 epochs)**：**

最终 loss 值：Train set: Average loss: 0.2016



在 Cifar10 **验证集上的** accuracy **曲线** (5 epochs)**：**

最终 accuracy 值：Accuracy: 7191/10000 (72%)

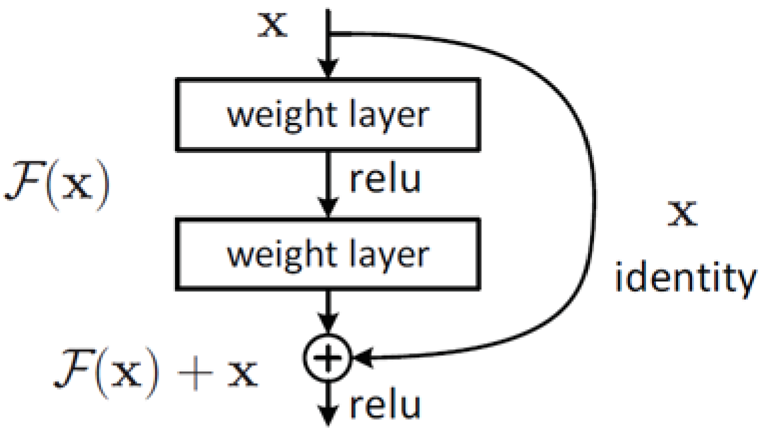


六、卷积网络和 ResNet、DenseNet

解释没有跳跃连接的卷积网络、ResNet、DenseNet **在训练过程中有什么不同?**

1. 普通卷积网络和 ResNet

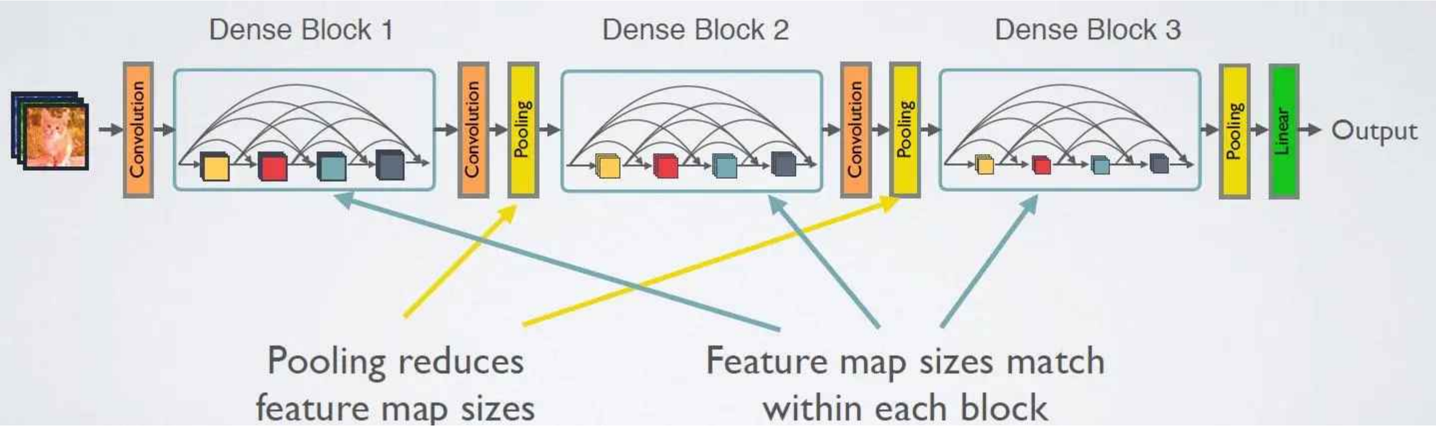
在传统的 CNN 网络中，网络的深度对其性能至关重要，随着网络层数的增加，网络可以提取更加复杂的特征。但是，由于梯度消失和梯度爆炸问题的存在，使得深度网络模型的性能退化。ResNet 采用了残差学习的方法来解决深度网络的退化问题，相比传统 CNN 网络，ResNet 具有更低的复杂度和参数量需求；同时，增加了网络的深度，但不会出现梯度消失现象，从而提高了分类准确度。



2. ResNet 和 DenseNet

DenseNet 的网络结构主要由 DenseBlock 和 Transition 组成，通过上文实验结果可以看到，DeseNet 的分类结果均要优于普通 CNN 和 ResNet 网络。DenseNet 网络的优势主要体现在：

- 由于密集连接方式，DenseNet 提升了梯度的反向传播，使得网络更容易训练
- 由于 DenseNet 是通过 concat 特征来实现短路连接，实现了特征重用，并且采用较小的growth rate，使得参数更小且计算更高效
- 由于特征复用，最后的分类器使用了低级特征



DenseNet 的 Block 结构如上图所示，DenseNet 采用更密集的连接方式，确保各层之间的信息流动达到最大。因为 DenseNet 的每一次卷积输入的 Chanel 数少于 ResNet 并且全连接层的参数也比 ResNet 少，所以相较于ResNet 网络有参数量上的优势。