卷积神经网络

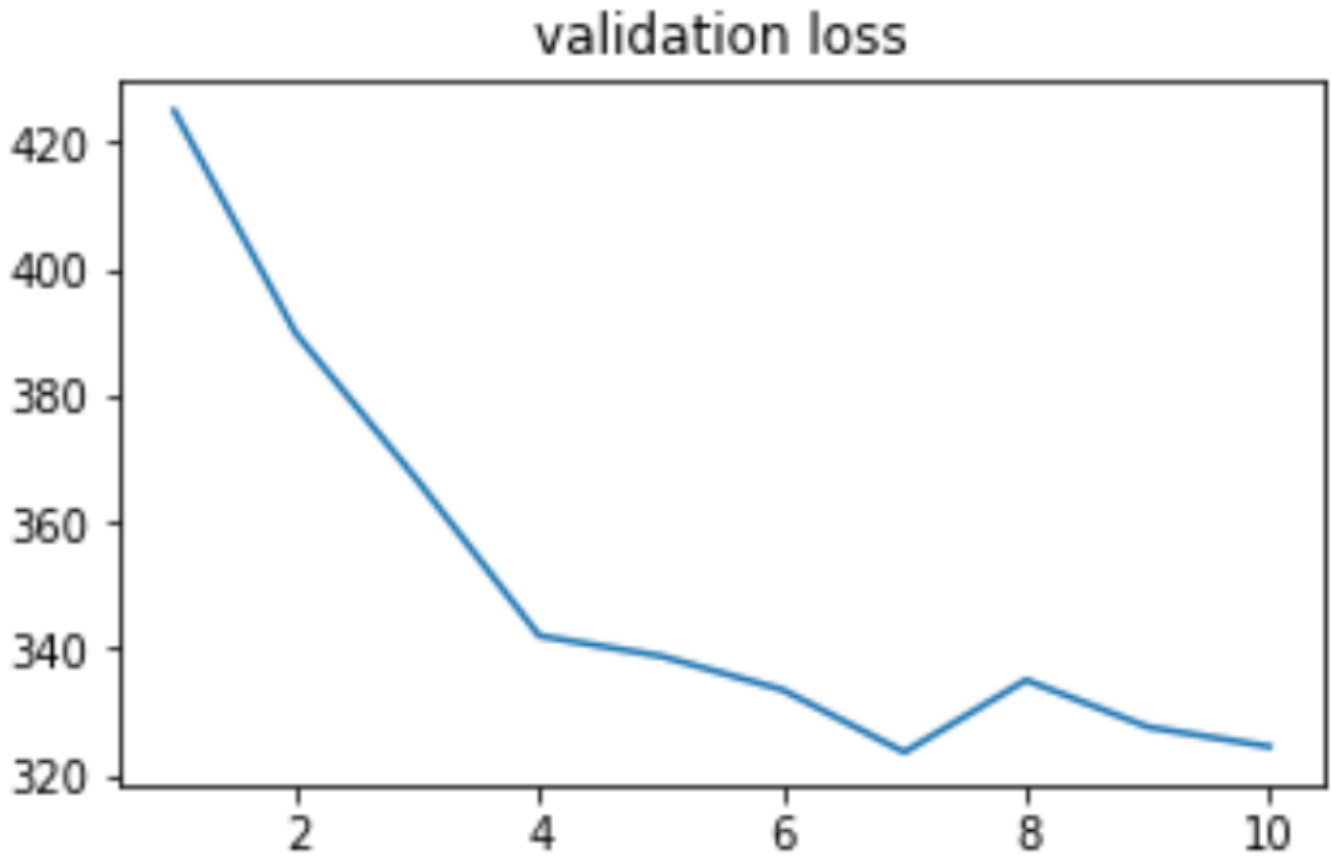
姓名：崔江浩
学号：2011915
专业：计算机科学与技术

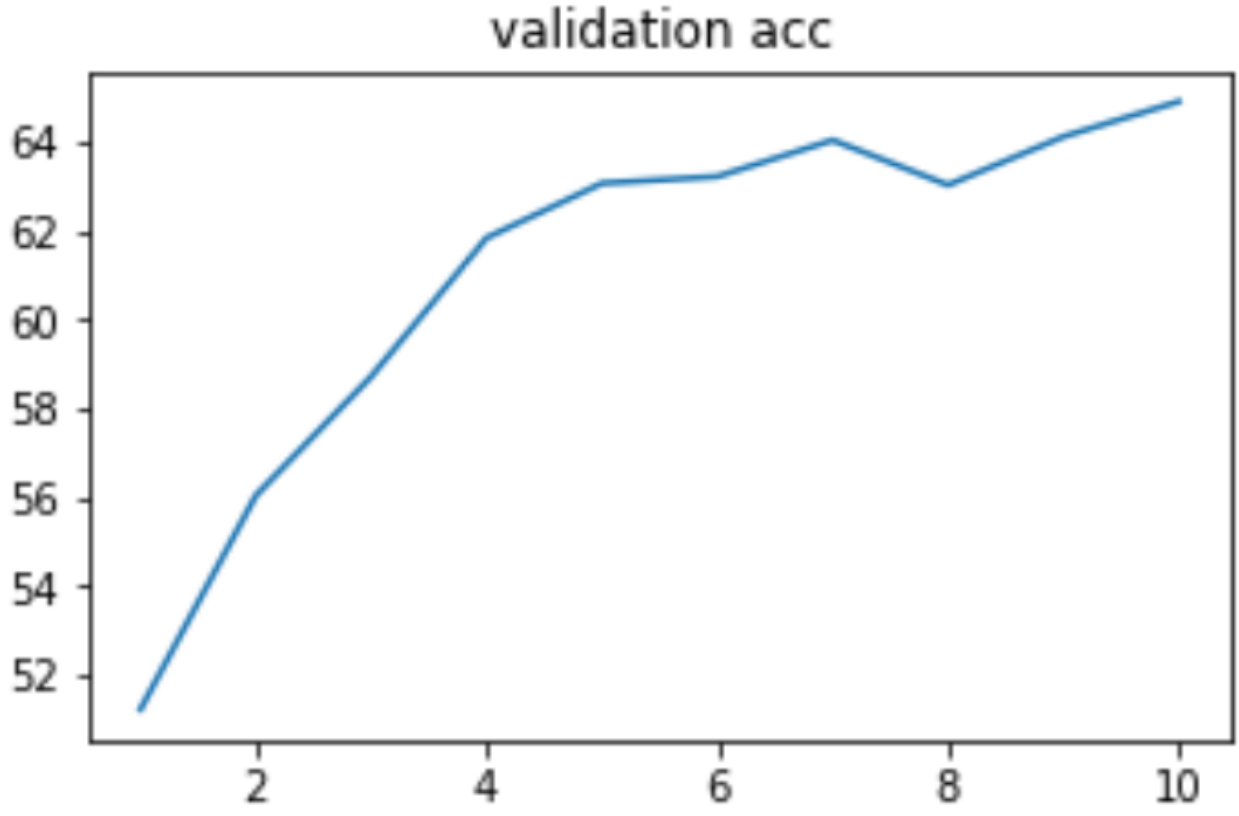
1 原始版本CNN

原始版本CNN定义了一个卷积神经网络（CNN）模型，用于处理 CIFAR-10 数据集。这个网络由两个卷积层和三个全连接层组成。卷积层使用 ReLU 激活函数和最大池化，用于提取和降低维度的特征。全连接层将这些特征映射到 10 个类别的输出上。这个网络通过 Adam 优化器进行训练，使用交叉熵损失函数计算损失。

```
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=False)
(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)
)
```

在本次实验中，选用Adam作为优化器，学习率为0.001，训练了10个epoch，得到训练结果
如图：





可以观察到在训练了10个epoch后已经接近收敛并且准确率达到64%。

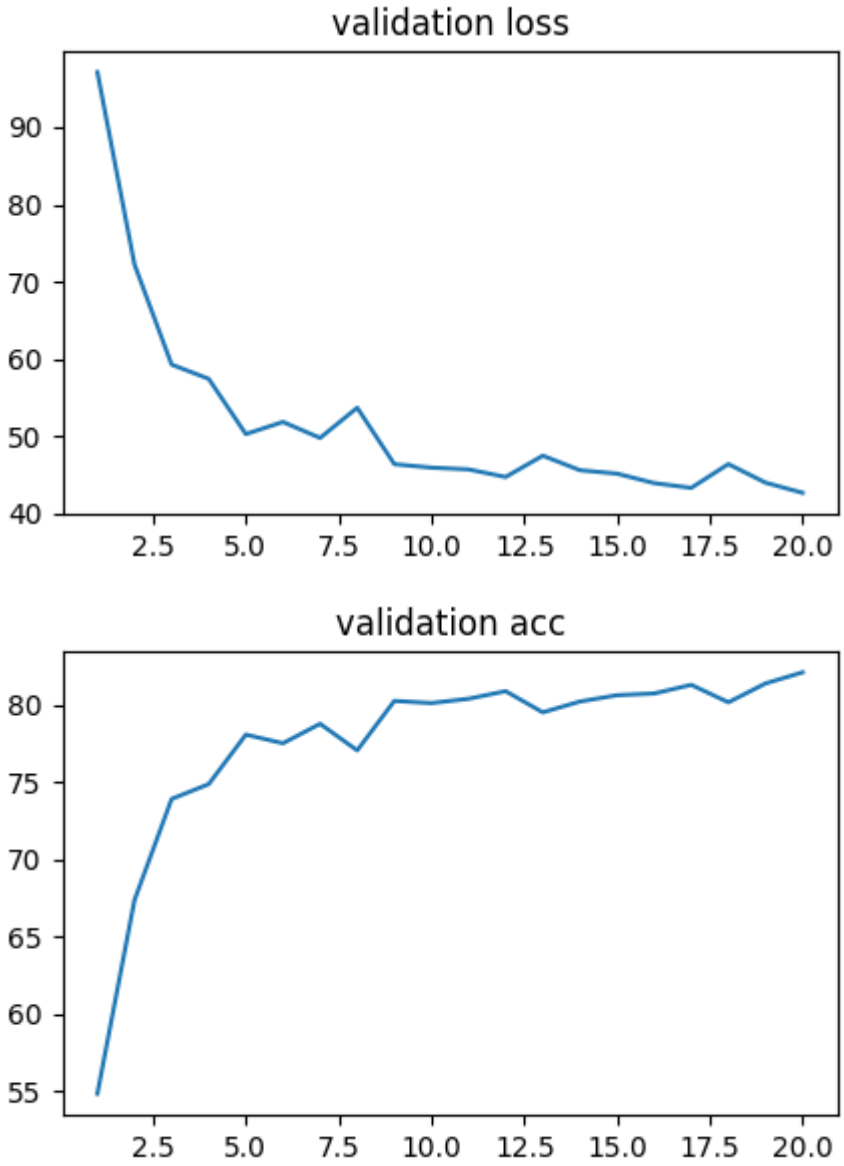
2 ResNet网络

ResNet，全名为残差网络（Residual Network），是由Kaiming He等人在2015年提出的一种深度学习模型。该模型的主要特点在于其独特的"shortcut connection"设计，这种设计允许网络的某一层直接访问前面层的输出，使得模型可以有效地学习输入和输出之间的"残差"函数。这种机制有效地解决了深度神经网络中的梯度消失和表示瓶颈问题，使得网络可以设计得更深，从而在许多任务上获得了更好的性能。

实现了一个ResNet-18的变体用于CIFAR-10数据集的分类。ResNet-18是由原始的ResNet论文提出的多种ResNet架构之一，包含5个阶段，每个阶段包括一个或多个卷积层和一个shortcut连接，总共有18层（不包括激活函数和池化层）。网络的主要构建块是BasicBlock，它包含两个3x3的卷积层和一个shortcut连接。在实现中，首先进行一个7x7的卷积和最大池化，然后通过四个阶段（每个阶段包含两个BasicBlock），最后通过全局平均池化和全连接层产生最终的分类输出。

每一个BasicBlock在做两次卷积操作后，会把原始的输入（shortcut）与卷积结果相加，这就构成了一个“残差”，这个设计允许网络在学习过程中拟合出输入和输出之间的差别，有助于优化模型性能，特别是在处理深层网络时，能有效缓解梯度消失问题，加速收敛。此外，对于网络的深度，由于ResNet采取了跳跃连接和恒等映射的设计，使得添加更多的层不会影响网络的性能，这为训练更深的网络提供了可能。

2.1 损失和准确率图片



可以看到，在训练了20个epoch时准确率已经超过了80%。并且到此为止，并没有过拟合情况的发生。

2.2 网络结构

```
ResNet(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (layer1): Sequential(
    (0): ResidualBlock(
      (conv1): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (shortcut): Sequential()
    )
    (1): ResidualBlock(
      (conv1): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (shortcut): Sequential()
    )
  )
  (layer2): Sequential(
    (0): ResidualBlock(
      (conv1): Conv2d(16, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): 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)
      )
    )
    (1): ResidualBlock(
      (conv1): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (shortcut): Sequential()
    )
  )
  (layer3): Sequential(
    (0): ResidualBlock(
      (conv1): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (shortcut): Sequential(
        (0): Conv2d(32, 64, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): ResidualBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (shortcut): Sequential()
    )
```

```
    )
    (fc): Linear(in_features=64, out_features=10, bias=True)
  )
```

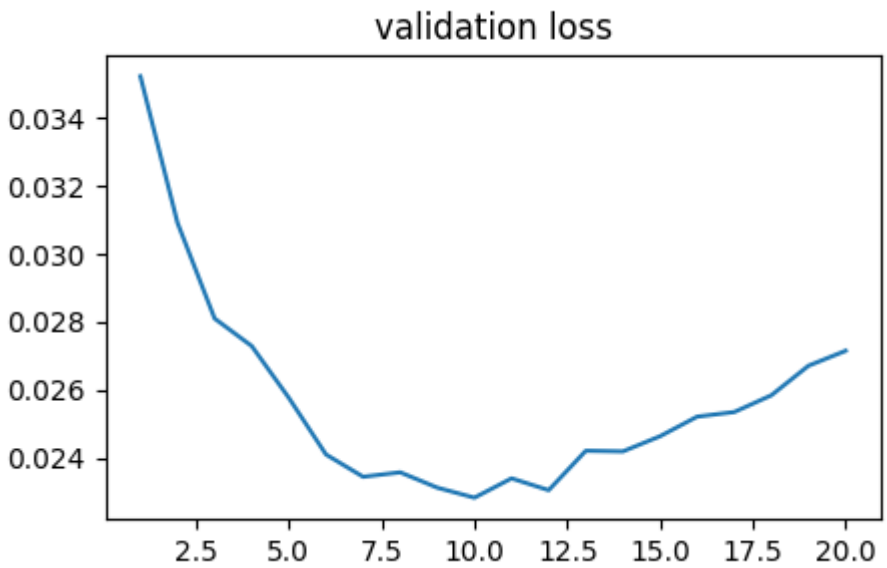
3 DenseNet网络

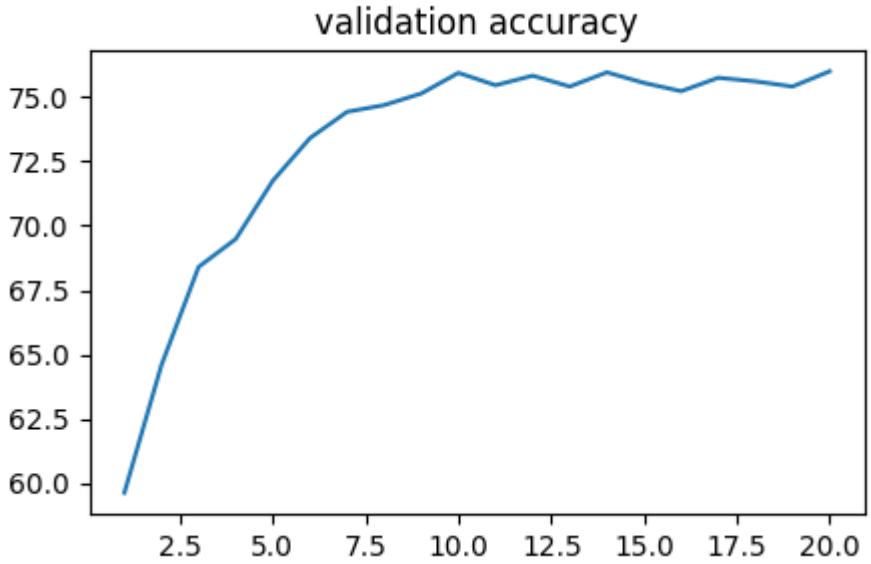
DenseNet，即Densely Connected Convolutional Networks（密集连接卷积网络），是一个由Gao Huang等人在2016年提出的网络架构。相较于传统的深度卷积神经网络架构，DenseNet的主要创新在于其引入了密集连接机制。在DenseNet中，每个层都与前面所有的层相连，并且将所有前面层的特征信息作为输入，而它自己的特征信息也会被所有后续的层作为输入。这种设置允许了前向传播、反向传播以及梯度的直接流动，有效解决了训练过程中的梯度消失问题。

对于实现的SimpleDenseNet，它是一个对DenseNet的简化版设计。首先，它包含一个初始的卷积层和最大池化层，以降低输入图像的分辨率，并增加特征的深度。接着，网络由多个DenseBlock和TransitionLayer交替组成，其中DenseBlock由多个DenseLayer构成，DenseLayer是一个包含了批归一化、ReLU激活函数以及卷积的小模块，每个DenseLayer都会产生固定数量的特征图，并将这些特征图与前面所有层的特征图进行拼接。TransitionLayer则用于降低特征图的数量以及分辨率。最后，网络通过一个全局平均池化层和全连接层进行分类。

这种设计确保了所有的层都在一定程度上接收到了原始输入的信息，使得网络可以更好地利用所有层的特征。在保持网络深度的同时，它有效地降低了模型的参数数量，使得模型在实践中更为高效且易于训练。

3.1 损失和准确率图片





可以观察到，准确率超过75%高于原始CNN但是低于ResNet，并且loss上升，可能发生过过拟合显现，因此DenseNet整体效果明显差于ResNet，分析原因可能是因为DenseNet网络较复杂，而CIFAR-10数据集分类比较简单，因此可能产生了过拟合。

3.2 网络结构

```
SimpleDenseNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    (1): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (2): DenseBlock(
      (layers): ModuleList(
        (0): DenseLayer(
          (conv): Sequential(
            (0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (1): ReLU()
            (2): Conv2d(64, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          )
        )
        (1): DenseLayer(
          (conv): Sequential(
            (0): BatchNorm2d(76, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (1): ReLU()
            (2): Conv2d(76, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          )
        )
        (2): DenseLayer(
          (conv): Sequential(
            (0): BatchNorm2d(88, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (1): ReLU()
            (2): Conv2d(88, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          )
        )
      )
    )
    (3): TransitionLayer(
      (downsample): Sequential(
        (0): BatchNorm2d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (1): ReLU()
        (2): Conv2d(100, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (3): AvgPool2d(kernel_size=2, stride=2, padding=0)
      )
    )
    (4): DenseBlock(
      (layers): ModuleList(
        (0): DenseLayer(
          (conv): Sequential(
            (0): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (1): ReLU()
            (2): Conv2d(128, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          )
        )
        (1): DenseLayer(
          (conv): Sequential(
            (0): BatchNorm2d(140, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (1): ReLU()
            (2): Conv2d(140, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
```

```
          )
        )
        (2): DenseLayer(
          (conv): Sequential(
            (0): BatchNorm2d(152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (1): ReLU()
            (2): Conv2d(152, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          )
        )
      )
    )
    (5): TransitionLayer(
      (downsample): Sequential(
        (0): BatchNorm2d(164, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (1): ReLU()
        (2): Conv2d(164, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (3): AvgPool2d(kernel_size=2, stride=2, padding=0)
      )
    )
    (6): DenseBlock(
      (layers): ModuleList(
        (0): DenseLayer(
          (conv): Sequential(
            (0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (1): ReLU()
            (2): Conv2d(64, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          )
        )
        (1): DenseLayer(
          (conv): Sequential(
            (0): BatchNorm2d(76, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (1): ReLU()
            (2): Conv2d(76, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          )
        )
        (2): DenseLayer(
          (conv): Sequential(
            (0): BatchNorm2d(88, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (1): ReLU()
            (2): Conv2d(88, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          )
        )
      )
    )
    (7): AdaptiveAvgPool2d(output_size=(1, 1))
  )
  (classifier): Linear(in_features=100, out_features=10, bias=True)
)
```

4 Se-ResNet网络

带有SE模块（Squeeze-and-Excitation Networks）的ResNet网络是一种在ResNet结构中加入了注意力机制的改进版本。SE模块通过学习每个通道的重要性权重，自适应地调整通道间的特征重要性，从而提高网络的表达能力和分类性能。

SE模块由两个主要组件组成：Squeeze和Excitation。Squeeze操作通过全局平均池化将每个通道的特征图转换为一个数值，以获得全局感知力。然后，Excitation操作利用全连接层学习到的权重来生成通道相关的重要性权重。这些权重被应用于输入特征图，以加权调节每个通道的响应，从而突出重要的特征。

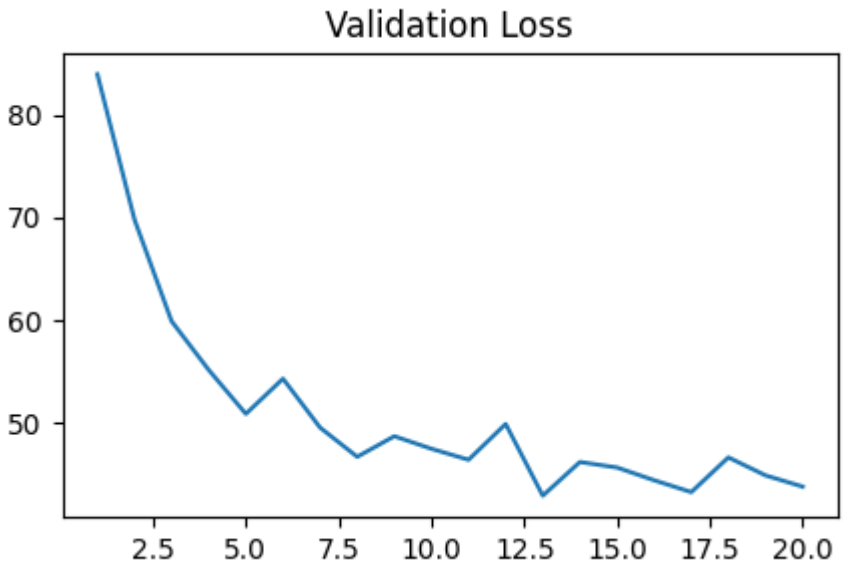
在实现的带有SE模块的ResNet网络中，SE模块被添加到每个残差块（Residual Block）中。每个残差块包含两个卷积层和一个SE模块。卷积层负责提取特征，而SE模块通过学习特征通道之间的相关性，动态地调整特征的重要性。SE模块中的Squeeze操作通过全局平均池化将特征图转换为一个数值，然后通过两个全连接层学习到权重，最后将这些权重应用于特征图，以产生加权调节后的特征响应。

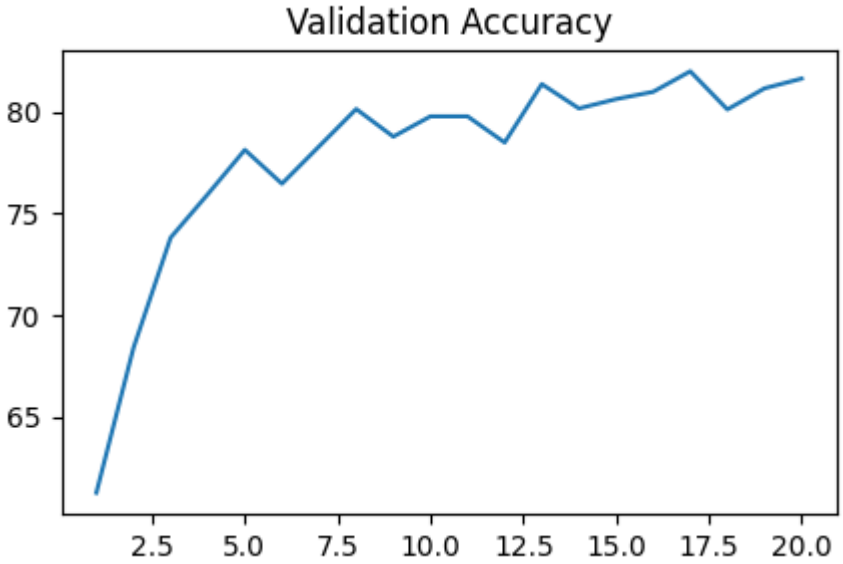
整个网络的架构与传统的ResNet相似，包括卷积层、批归一化层、残差块和全连接层。在网络的开始，有一个卷积层和一个批归一化层，用于对输入图像进行特征提取和归一化。然后，通过堆叠多个残差块，构建深层网络结构。每个残差块都包含了带有SE模块的卷积层和批归一化层，以及残差连接来处理输入和输出之间的维度不匹配。最后，通过全局平均池化和全连接层进行分类。

在训练过程中，使用交叉熵损失函数和随机梯度下降优化器进行网络的训练。学习率调度器（scheduler）用于动态调整学习率。训练过程中，计算每个epoch的训练损失和准确率，并将其记录下来。最后，使用matplotlib库将训练过程中的损失和准确率曲线可视化。

通过引入SE模块，带有SE模块的ResNet网络能够自适应地关注和增强重要的特征，从而提高网络的表达能力和分类性能。

4.1 损失和准确率图片





由上图可以观察到，其准确率超过80%，并且已经完成收敛，其效果于原始ResNet相差不大，原因可能是此次分类的数据集较简单，难以体现Se-ResNet网络的优势。

4.2 网络结构

```
ResNetSE(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (layer1): Sequential(
    (0): ResidualBlockSE(
      (conv1): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (se): SEBlock(
        (avg_pool): AdaptiveAvgPool2d(output_size=1)
        (fc1): Linear(in_features=16, out_features=1, bias=True)
        (fc2): Linear(in_features=1, out_features=16, bias=True)
      )
      (shortcut): Sequential()
    )
    (1): ResidualBlockSE(
      (conv1): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (se): SEBlock(
        (avg_pool): AdaptiveAvgPool2d(output_size=1)
        (fc1): Linear(in_features=16, out_features=1, bias=True)
        (fc2): Linear(in_features=1, out_features=16, bias=True)
      )
      (shortcut): Sequential()
    )
  )
  (layer2): Sequential(
    (0): ResidualBlockSE(
      (conv1): Conv2d(16, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (se): SEBlock(
        (avg_pool): AdaptiveAvgPool2d(output_size=1)
        (fc1): Linear(in_features=32, out_features=2, bias=True)
        (fc2): Linear(in_features=2, out_features=32, bias=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)
      )
    )
    (1): ResidualBlockSE(
      (conv1): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (se): SEBlock(
        (avg_pool): AdaptiveAvgPool2d(output_size=1)
        (fc1): Linear(in_features=32, out_features=2, bias=True)
        (fc2): Linear(in_features=2, out_features=32, bias=True)
      )
      (shortcut): Sequential()
```

```
    )
  )
  (layer3): Sequential(
    (0): ResidualBlockSE(
      (conv1): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (se): SEBlock(
        (avg_pool): AdaptiveAvgPool2d(output_size=1)
        (fc1): Linear(in_features=64, out_features=4, bias=True)
        (fc2): Linear(in_features=4, out_features=64, bias=True)
      )
      (shortcut): Sequential(
        (0): Conv2d(32, 64, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): ResidualBlockSE(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (se): SEBlock(
        (avg_pool): AdaptiveAvgPool2d(output_size=1)
        (fc1): Linear(in_features=64, out_features=4, bias=True)
        (fc2): Linear(in_features=4, out_features=64, bias=True)
      )
      (shortcut): Sequential()
    )
  )
  (fc): Linear(in_features=64, out_features=10, bias=True)
)
```

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

在详细解释这些不同类型的神经网络在训练过程中的差异之前，先概述它们各自的基本结构和工作原理。

1. **没有跳跃连接的卷积网络**：这种类型的网络，比如经典的AlexNet和VGGNet，通常由卷积层、全连接层和激活函数组成。信息在网络中依次向前传播，没有任何形式的跳跃连接。这种网络的主要问题是，随着网络深度的增加，反向传播的梯度会逐渐消失，导致深层网络难以训练。

2. **ResNet (残差网络)**：为了解决上述深度网络的训练问题，ResNet引入了跳跃连接（或称为短路连接）。这些连接使得一部分信息能直接跳过一些层，从而保持反向传播的梯度。这种机制使得深度学习模型可以有效地训练更深的网络。

3. **DenseNet (密集连接网络)**：DenseNet是在ResNet的基础上的改进，它通过建立更紧密的跳跃连接——每一层都与前面所有层以及后续的所有层相连。这样的连接结构保证了前向和反向信号的高效传播，还能大幅度减少参数的数量。

4. **SE-ResNet (Squeeze-and-Excitation Residual Network)**：SE-ResNet是ResNet的一个改进版本，引入了注意力机制的概念。SE模块通过对各通道的信息进行"压缩"和"激发"，赋予不同通道不同的重要性，进一步提升了网络的性能。

那么，这些网络在训练过程中的主要差异体现在以下几个方面：

- **梯度传播**：没有跳跃连接的卷积网络在训练深度模型时会遇到梯度消失问题，这会导致网络在训练过程中效果不佳。而ResNet和DenseNet通过引入跳跃连接，保证了梯度能够有效地向前反向传播，大大缓解了梯度消失的问题。
- **参数数量**：通常，没有跳跃连接的卷积网络和ResNet需要更多的参数来达到相同的效果。而DenseNet通过其特殊的结构减少了参数数量，节省了存储和计算资源。
- **注意力机制**：SE-ResNet在训练过程中引入了注意力机制，即通过学习给不同的特征通道分配不同的权重，进一步提升了模型的表现。

总的来说，随着跳跃连接和注意力机制的引入，神经网络的训练过程变得更加有效和灵活，使得我们能够训练更深、更强大的模型。