计算机学院

深度学习实验报告

CNN 实验报告

姓名：徐文斌

学号：2010234

专业：计算机科学与技术

2023 年 6 月 24 日

目录

1 原始版本 CNN

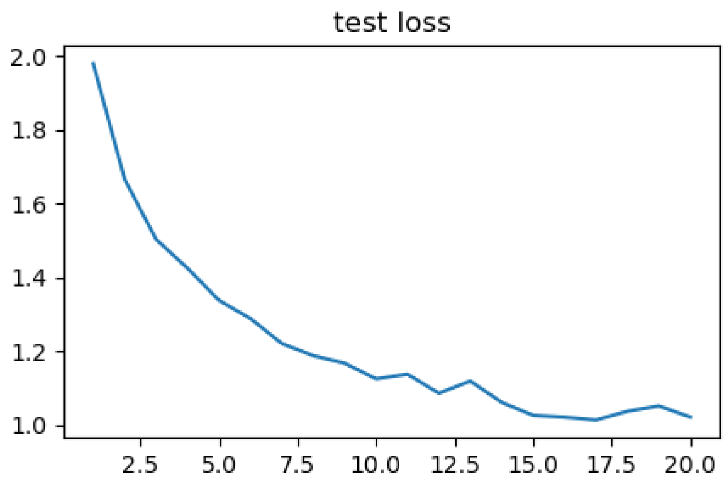
1.1 网络结构

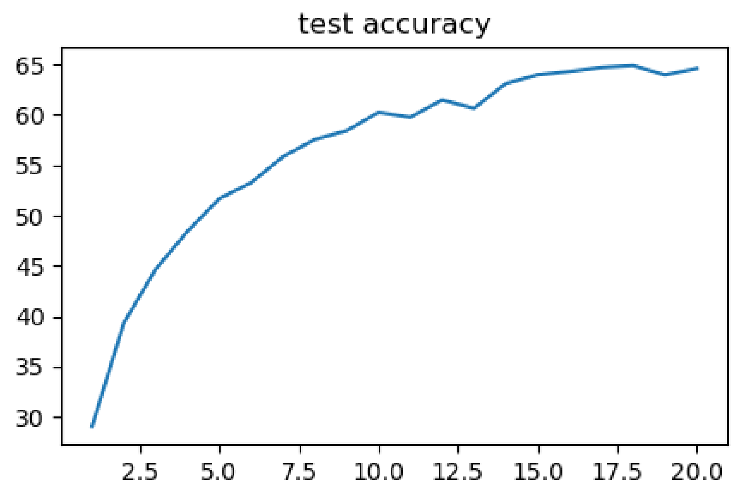
使用 print 函数打印网络结构如下所示：

```
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=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)
)
```

1.2 训练结果

训练网络，得到每轮的损失和准确度曲线如下，在经过 20 轮训练后，在测试集上达到了 64.59% 准确率。





2 实现 ResNet

2.1 网络结构

实现 ResNet18 残差网络，网络结构如下：

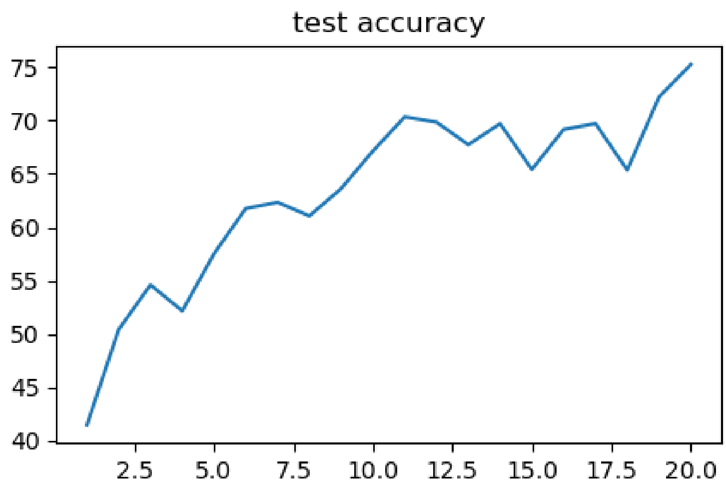
```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
     bias=False)
  (bn): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True)
  (relu): ReLU(inplace=True)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, 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)
      (relu): ReLU(inplace=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)
    )
    (1): BasicBlock(
      (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)
      (relu): ReLU(inplace=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)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, 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)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
         bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
```

```
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
```

```
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (avg_pool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=10, bias=True)
)
```

2.2 训练结果

训练网络，得到每轮的损失和准确度曲线如下，在经过 20 轮训练后，在测试集上达到了 75.26% 准确率。



3 DenseNet

3.1 网络结构

实现 DenseNet 残差网络，由于机器性能有限，这里只设置三个 DenseBlock，每个 DenseBlock 中设置三个 DenseLayer。网络结构如下：

```
DenseNet(
  (conv1): Sequential(
    (0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    (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)
  )
  (layers): Sequential(
    (0): DenseBlock(
      (layers): Sequential(
        (0): _DenseLayer(
          (dense_layer): Sequential(
            (0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (1): ReLU(inplace=True)
            (2): Conv2d(64, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (4): ReLU(inplace=True)
            (5): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          )
          (dropout): Dropout(p=0, inplace=False)
        )
        (1): _DenseLayer(
          (dense_layer): Sequential(
```

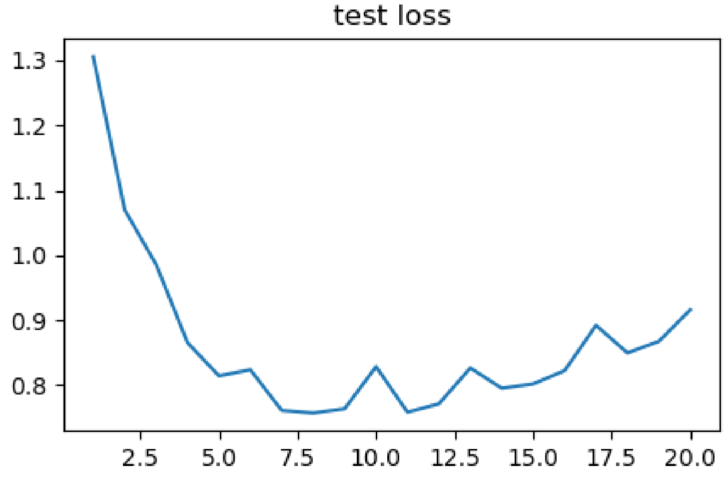
```
            (0): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (1): ReLU(inplace=True)
            (2): Conv2d(96, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (4): ReLU(inplace=True)
            (5): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          )
          (dropout): Dropout(p=0, inplace=False)
        )
        (2): _DenseLayer(
          (dense_layer): Sequential(
            (0): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (1): ReLU(inplace=True)
            (2): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (4): ReLU(inplace=True)
            (5): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          )
          (dropout): Dropout(p=0, inplace=False)
        )
      )
    )
    (1): _TransitionLayer(
      (transition_layer): Sequential(
        (0): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (1): ReLU(inplace=True)
        (2): Conv2d(160, 80, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (3): AvgPool2d(kernel_size=2, stride=2, padding=0)
      )
    )
    (2): DenseBlock(
      (layers): Sequential(
        (0): _DenseLayer(
          (dense_layer): Sequential(
            (0): BatchNorm2d(80, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (1): ReLU(inplace=True)
            (2): Conv2d(80, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (4): ReLU(inplace=True)
            (5): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
```

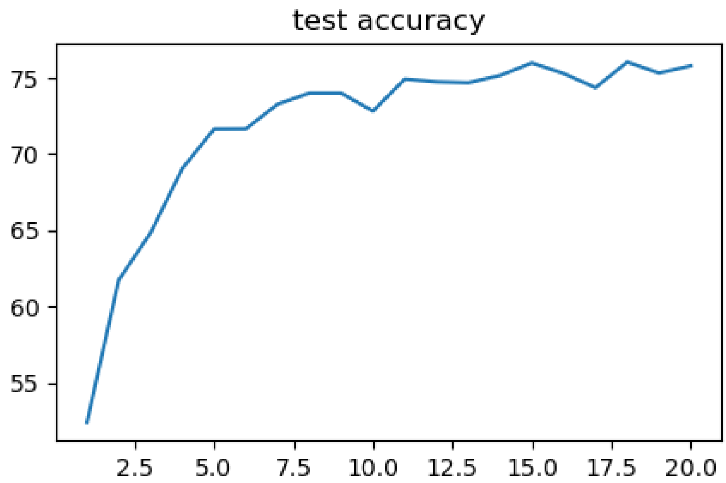
```
                bias=False)
          )
          (dropout): Dropout(p=0, inplace=False)
        )
        (1): _DenseLayer(
          (dense_layer): Sequential(
            (0): BatchNorm2d(112, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (1): ReLU(inplace=True)
            (2): Conv2d(112, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (4): ReLU(inplace=True)
            (5): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          )
          (dropout): Dropout(p=0, inplace=False)
        )
        (2): _DenseLayer(
          (dense_layer): Sequential(
            (0): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (1): ReLU(inplace=True)
            (2): Conv2d(144, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (4): ReLU(inplace=True)
            (5): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          )
          (dropout): Dropout(p=0, inplace=False)
        )
      )
    )
    (3): _TransitionLayer(
      (transition_layer): Sequential(
        (0): BatchNorm2d(176, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (1): ReLU(inplace=True)
        (2): Conv2d(176, 88, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (3): AvgPool2d(kernel_size=2, stride=2, padding=0)
      )
    )
    (4): DenseBlock(
      (layers): Sequential(
        (0): _DenseLayer(
          (dense_layer): Sequential(
            (0): BatchNorm2d(88, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```
            (1): ReLU(inplace=True)
            (2): Conv2d(88, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (4): ReLU(inplace=True)
            (5): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          )
          (dropout): Dropout(p=0, inplace=False)
        )
        (1): _DenseLayer(
          (dense_layer): Sequential(
            (0): BatchNorm2d(120, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (1): ReLU(inplace=True)
            (2): Conv2d(120, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (4): ReLU(inplace=True)
            (5): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          )
          (dropout): Dropout(p=0, inplace=False)
        )
        (2): _DenseLayer(
          (dense_layer): Sequential(
            (0): BatchNorm2d(152, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (1): ReLU(inplace=True)
            (2): Conv2d(152, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (4): ReLU(inplace=True)
            (5): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          )
          (dropout): Dropout(p=0, inplace=False)
        )
      )
    )
  )
  (avgpool): AvgPool2d(kernel_size=2, stride=1, padding=0)
  (fc): Linear(in_features=184, out_features=10, bias=True)
)
```

3.2 训练结果

训练网络，得到每轮的损失和准确度曲线如下，在经过 20 轮训练后，在测试集上达到了 75.79% 准确率。





4 SE-ResNet

4.1 网络结构

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
      bias=False)
  (bn): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
      track_running_stats=True)
  (relu): ReLU(inplace=True)
  (layer1): Sequential(
    (0): BasicBlock(
      (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)
      (relu): ReLU(inplace=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): SELayer(
        (avg_pool): AdaptiveAvgPool2d(output_size=1)
        (fc): Sequential(
          (0): Linear(in_features=64, out_features=4, bias=False)
          (1): ReLU(inplace=True)
          (2): Linear(in_features=4, out_features=64, bias=False)
          (3): Sigmoid()
        )
      )
    )
    (1): BasicBlock(
      (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)
      (relu): ReLU(inplace=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): SELayer(
        (avg_pool): AdaptiveAvgPool2d(output_size=1)
        (fc): Sequential(
          (0): Linear(in_features=64, out_features=4, bias=False)
          (1): ReLU(inplace=True)
          (2): Linear(in_features=4, out_features=64, bias=False)
          (3): Sigmoid()
        )
      )
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, 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)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

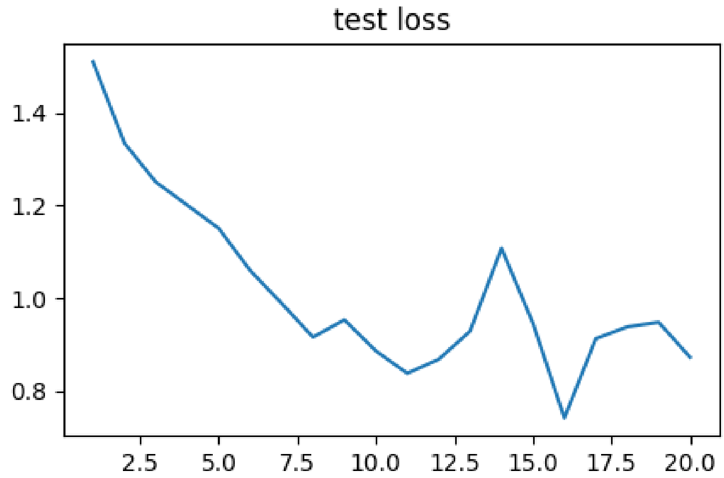
```
      (se): SELayer(
        (avg_pool): AdaptiveAvgPool2d(output_size=1)
        (fc): Sequential(
          (0): Linear(in_features=128, out_features=8, bias=False)
          (1): ReLU(inplace=True)
          (2): Linear(in_features=8, out_features=128, bias=False)
          (3): Sigmoid()
        )
      )
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
            track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
          bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
          track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
          bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
          track_running_stats=True)
      (se): SELayer(
        (avg_pool): AdaptiveAvgPool2d(output_size=1)
        (fc): Sequential(
          (0): Linear(in_features=128, out_features=8, bias=False)
          (1): ReLU(inplace=True)
          (2): Linear(in_features=8, out_features=128, bias=False)
          (3): Sigmoid()
        )
      )
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
          bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
          track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
          bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
          track_running_stats=True)
      (se): SELayer(
        (avg_pool): AdaptiveAvgPool2d(output_size=1)
```

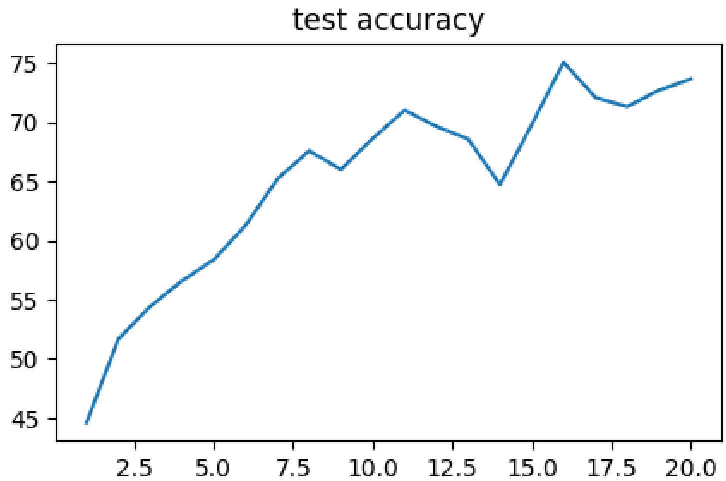
```
        (fc): Sequential(
          (0): Linear(in_features=256, out_features=16, bias=False)
          (1): ReLU(inplace=True)
          (2): Linear(in_features=16, out_features=256, bias=False)
          (3): Sigmoid()
        )
      )
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
            track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
          bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
          track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
          bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
          track_running_stats=True)
      (se): SELayer(
        (avg_pool): AdaptiveAvgPool2d(output_size=1)
        (fc): Sequential(
          (0): Linear(in_features=256, out_features=16, bias=False)
          (1): ReLU(inplace=True)
          (2): Linear(in_features=16, out_features=256, bias=False)
          (3): Sigmoid()
        )
      )
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
          bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
          track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
          bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
          track_running_stats=True)
      (se): SELayer(
        (avg_pool): AdaptiveAvgPool2d(output_size=1)
        (fc): Sequential(
          (0): Linear(in_features=512, out_features=32, bias=False)
```

```
          (1): ReLU(inplace=True)
          (2): Linear(in_features=32, out_features=512, bias=False)
          (3): Sigmoid()
        )
      )
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
           track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
         bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
         bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (se): SELayer(
        (avg_pool): AdaptiveAvgPool2d(output_size=1)
        (fc): Sequential(
          (0): Linear(in_features=512, out_features=32, bias=False)
          (1): ReLU(inplace=True)
          (2): Linear(in_features=32, out_features=512, bias=False)
          (3): Sigmoid()
        )
      )
    )
  )
  (avg_pool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=10, bias=True)
)
```

4.2 训练结果

训练网络，得到每轮的损失和准确度曲线如下，在经过 20 轮训练后，在测试集上达到了 73.63% 准确率。





5 朴素卷积网络、ResNet、DenseNet、SE-ResNet 在训练过程的不同

5.1 ResNet

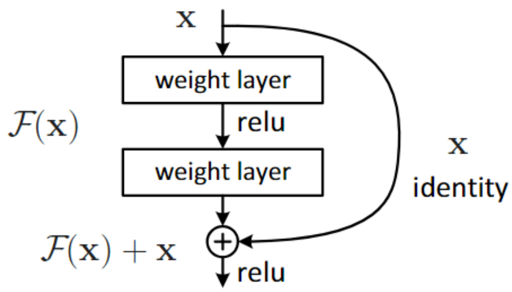


图 5.1: resnet 基础结构 [1]

残差网络添加了一个恒等映射的部分，残差块最终的输出是 $f(x)+x$ 而非 $f(x)$。这种结构可以很好地解决梯度消失，可以使网络层数更深而仍具有学习能力，从而提高网络的效果。而且还可以比较好的解决网络退化的问题，因为新加入一层最差我们也可以让他是一个恒等映射（或极其接近恒等映射的）层，不会对后续效果有影响，那么自然不会比加入之前差，从而解决网络退化问题。

5.2 DenseNet

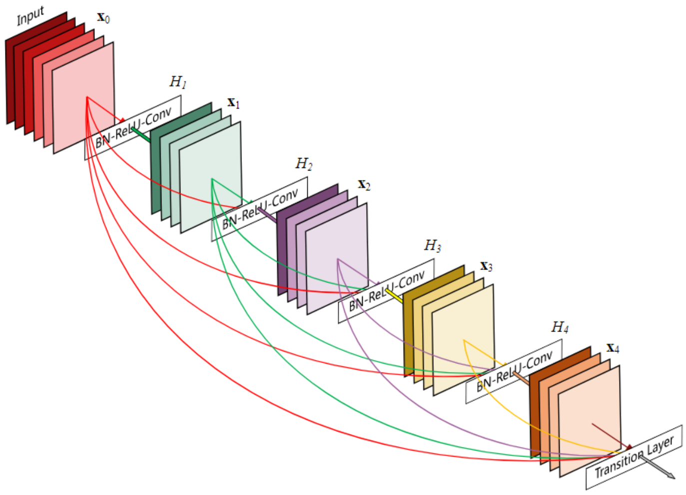


图 5.2: densenet 基础结构 [2]

当前馈卷积神经网络的深度过深的时候，信息通过许多层后，可能会消失或被“洗掉”。Denset 也提出了一种残差结构，这种结构的每个层都从前面的层获得额外的输入，并将自己的特征图传递给所有后续层。与 ResNet 最大的不同是，ResNet 是将残差和输出进行相加，而 Desnet 则是将输出和残差进行拼接。

虽然看起来更密集的连接会增加参数数量，但实际上 DenseNet 比传统的卷积网络需要的参数反而更少，因为密集的连接带来了特征重用，不需要重新学习冗余的特征图。而且维度拼接的操作，带来了丰富的特征信息，利用更少的卷积就能获得很多的特征图。

5.3 SE-ResNet

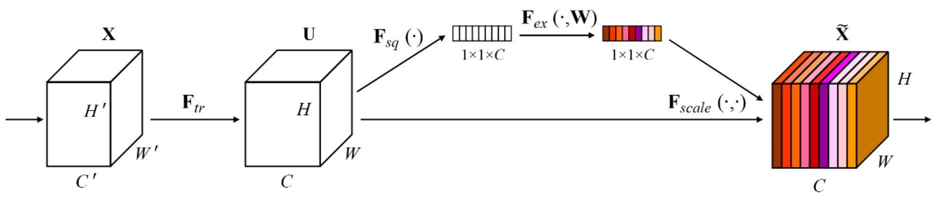


图 5.3: se block 基础结构 [3]

SENet 工作机制如下：

对于每一输出通道，先进行全剧平均池化，每个通道得到 1 个标量，C 个通道得到 C 个数，然后经过 FC-ReLU-FC-Sigmoid 得到 C 个 0 1 之间的标量，作为通道的加权，然后原来的输出通道每个通

道用对应的权重进行加权（对应通道的每个元素与权重分别相乘），得到新的加权后的特征。

SE 模块对于每个输出 channel，预测一个常数权重，对每个 channel 加权一下，本质上，SE 模块是在 channel 维度上做 attention 或者 gating 操作，这种注意力机制让模型可以更加关注信息量最大的 channel 特征，而抑制那些不重要的 channel 特征。SENet 一个很大的优点就是可以很方便地集成到现有网络中，提升网络性能，并且代价很小。

参考文献

[1] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

[2] Huang G, Liu Z, Van Der Maaten L, et al. Densely connected convolutional networks[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2017: 4700-4708.

[3] Hu J, Shen L, Sun G. Squeeze-and-excitation networks[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2018: 7132-7141.