



南開大學
Nankai University

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	2
1.1	网络结构	2
1.2	训练结果	2
2	实现 ResNet	3
2.1	网络结构	3
2.2	训练结果	5
3	DenseNet	6
3.1	网络结构	6
3.2	训练结果	10
4	SE-ResNet	10
4.1	网络结构	10
4.2	训练结果	14
5	朴素卷积网络、ResNet、DenseNet、SE-ResNet 在训练过程的不同	15
5.1	ResNet	15
5.2	DenseNet	16
5.3	SE-ResNet	16

1 原始版本 CNN

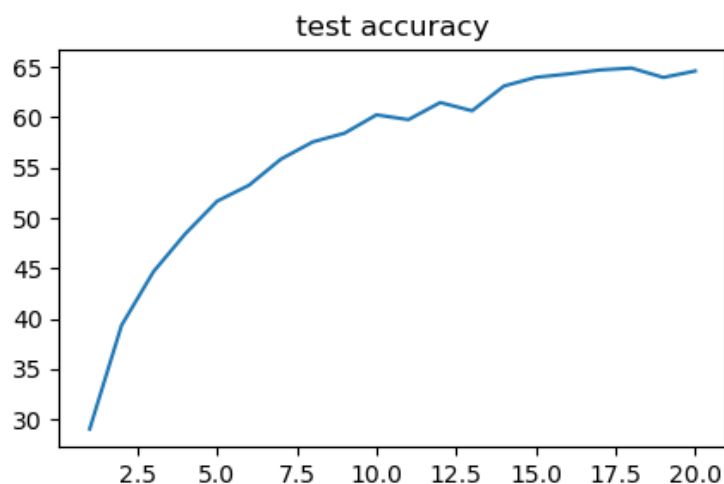
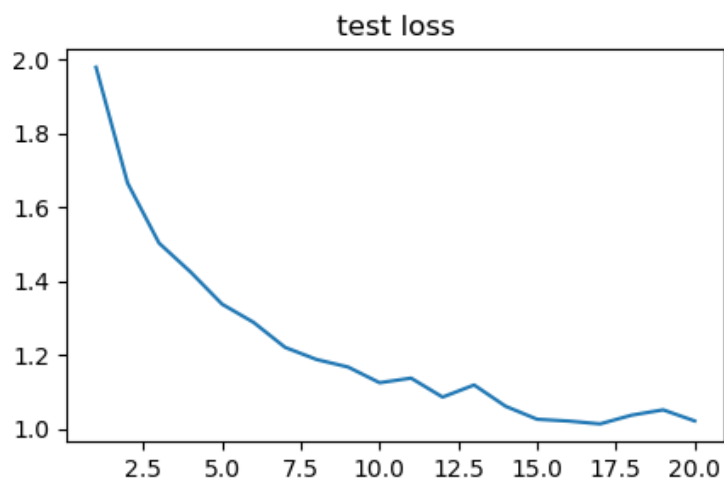
1.1 网络结构

使用 print 函数打印网络结构如下所示：

```
1 Net(  
2   (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))  
3   (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
4   (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))  
5   (fc1): Linear(in_features=400, out_features=120, bias=True)  
6   (fc2): Linear(in_features=120, out_features=84, bias=True)  
7   (fc3): Linear(in_features=84, out_features=10, bias=True)  
8 )
```

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：

```

1 ResNet(
2     (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
      bias=False)
3     (bn): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
      track_running_stats=True)
4     (relu): ReLU(inplace=True)
5     (layer1): Sequential(
6         (0): BasicBlock(
7             (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
              bias=False)
8             (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
9             (relu): ReLU(inplace=True)
10            (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
              bias=False)
11            (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
12        )
13        (1): BasicBlock(
14            (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
              bias=False)
15            (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
16            (relu): ReLU(inplace=True)
17            (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
              bias=False)
18            (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
19        )
20    )
21    (layer2): Sequential(
22        (0): BasicBlock(
23            (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
              bias=False)
24            (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
25            (relu): ReLU(inplace=True)
26            (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
              bias=False)
27            (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
28            (downsample): Sequential(
29                (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)

```

```

30         (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
31             track_running_stats=True)
32     )
33     (1): BasicBlock(
34         (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
35             bias=False)
36         (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
37             track_running_stats=True)
38         (relu): ReLU(inplace=True)
39         (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
40             bias=False)
41         (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
42             track_running_stats=True)
43     )
44 )
45 (layer3): Sequential(
46     (0): BasicBlock(
47         (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
48             bias=False)
49         (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
50             track_running_stats=True)
51         (relu): ReLU(inplace=True)
52         (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
53             bias=False)
54         (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
55             track_running_stats=True)
56         (downsample): Sequential(
57             (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
58             (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
59                 track_running_stats=True)
60         )
61     )
62     (1): BasicBlock(
63         (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
64             bias=False)
65         (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
66             track_running_stats=True)
67         (relu): ReLU(inplace=True)
68         (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
69             bias=False)
70         (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
71             track_running_stats=True)
72     )
73 )
74 (layer4): Sequential(
75     (0): BasicBlock(
76         (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
77             bias=False)

```

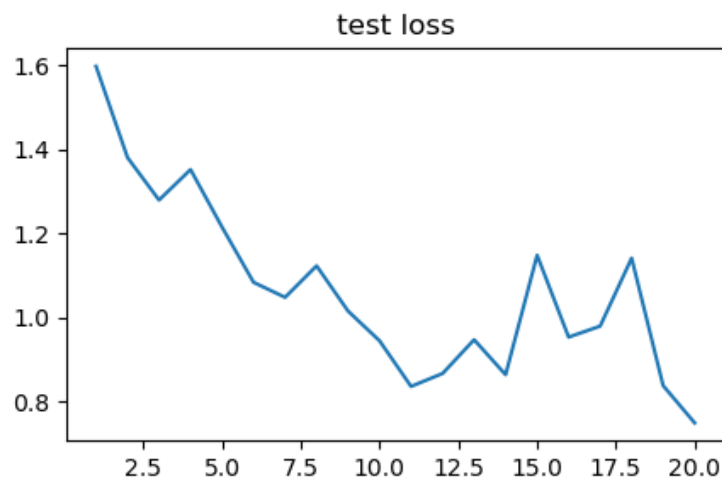
```

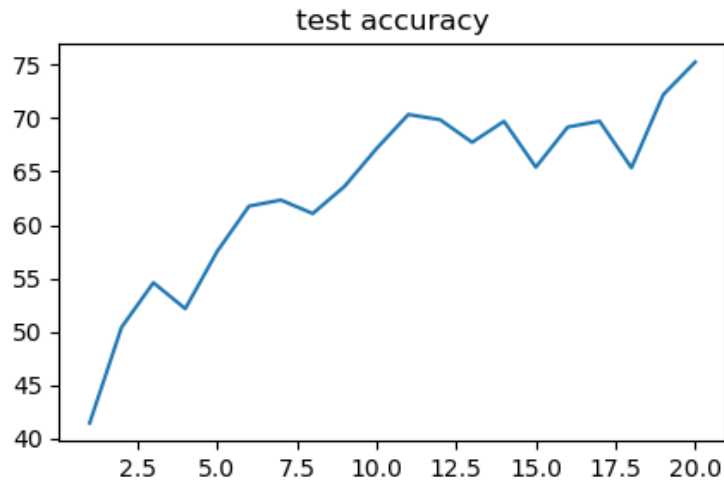
64         (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
65             track_running_stats=True)
66         (relu): ReLU(inplace=True)
67         (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
68             bias=False)
69         (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
70             track_running_stats=True)
71         (downsample): Sequential(
72             (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
73             (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
74                 track_running_stats=True)
75         )
76     )
77     (1): BasicBlock(
78         (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
79             bias=False)
80         (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
81             track_running_stats=True)
82         (relu): ReLU(inplace=True)
83         (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
84             bias=False)
85         (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
86             track_running_stats=True)
87     )
88 )
89 (avg_pool): AdaptiveAvgPool2d(output_size=(1, 1))
90 (fc): Linear(in_features=512, out_features=10, bias=True)
91 )

```

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：

```

1 DenseNet(
2   (conv1): Sequential(
3     (0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
4     (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
5       track_running_stats=True)
6     (2): ReLU(inplace=True)
7     (3): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
8   )
9   (layers): Sequential(
10    (0): DenseBlock(
11      (layers): Sequential(
12        (0): _DenseLayer(
13          (dense_layer): Sequential(
14            (0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
15              track_running_stats=True)
16            (1): ReLU(inplace=True)
17            (2): Conv2d(64, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
18            (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
19              track_running_stats=True)
20            (4): ReLU(inplace=True)
21            (5): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
22              bias=False)
23          )
24        (dropout): Dropout(p=0, inplace=False)
25      )
26    )
27    (1): _DenseLayer(
28      (dense_layer): Sequential(

```

```

24         (0): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True,
25             track_running_stats=True)
26         (1): ReLU(inplace=True)
27         (2): Conv2d(96, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
28         (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
29             track_running_stats=True)
30         (4): ReLU(inplace=True)
31         (5): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
32             bias=False)
33     )
34     (dropout): Dropout(p=0, inplace=False)
35 )
36 (2): _DenseLayer(
37     (dense_layer): Sequential(
38         (0): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
39             track_running_stats=True)
40         (1): ReLU(inplace=True)
41         (2): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
42         (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
43             track_running_stats=True)
44         (4): ReLU(inplace=True)
45         (5): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
46             bias=False)
47     )
48     (dropout): Dropout(p=0, inplace=False)
49 )
50 )
51 (1): _TransitionLayer(
52     (transition_layer): Sequential(
53         (0): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True,
54             track_running_stats=True)
55         (1): ReLU(inplace=True)
56         (2): Conv2d(160, 80, kernel_size=(1, 1), stride=(1, 1), bias=False)
57         (3): AvgPool2d(kernel_size=2, stride=2, padding=0)
58     )
59 )
60 (2): DenseBlock(
61     (layers): Sequential(
62         (0): _DenseLayer(
63             (dense_layer): Sequential(
64                 (0): BatchNorm2d(80, eps=1e-05, momentum=0.1, affine=True,
65                     track_running_stats=True)
66                 (1): ReLU(inplace=True)
67                 (2): Conv2d(80, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
68                 (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
69                     track_running_stats=True)
70                 (4): ReLU(inplace=True)
71                 (5): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),

```



```

        bias=False)
64     )
65     (dropout): Dropout(p=0, inplace=False)
66 )
67 (1): _DenseLayer(
68     (dense_layer): Sequential(
69         (0): BatchNorm2d(112, eps=1e-05, momentum=0.1, affine=True,
70             track_running_stats=True)
71         (1): ReLU(inplace=True)
72         (2): Conv2d(112, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
73         (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
74             track_running_stats=True)
75         (4): ReLU(inplace=True)
76         (5): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
77             bias=False)
78     )
79     (dropout): Dropout(p=0, inplace=False)
80 )
81 (2): _DenseLayer(
82     (dense_layer): Sequential(
83         (0): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True,
84             track_running_stats=True)
85         (1): ReLU(inplace=True)
86         (2): Conv2d(144, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
87         (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
88             track_running_stats=True)
89         (4): ReLU(inplace=True)
90         (5): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
91             bias=False)
92     )
93     (dropout): Dropout(p=0, inplace=False)
94 )
95 )
96 (3): _TransitionLayer(
97     (transition_layer): Sequential(
98         (0): BatchNorm2d(176, eps=1e-05, momentum=0.1, affine=True,
99             track_running_stats=True)
100         (1): ReLU(inplace=True)
101         (2): Conv2d(176, 88, kernel_size=(1, 1), stride=(1, 1), bias=False)
102         (3): AvgPool2d(kernel_size=2, stride=2, padding=0)
103     )
104 )
105 (4): DenseBlock(
106     (layers): Sequential(
107         (0): _DenseLayer(
108             (dense_layer): Sequential(
109                 (0): BatchNorm2d(88, eps=1e-05, momentum=0.1, affine=True,
110                     track_running_stats=True)

```

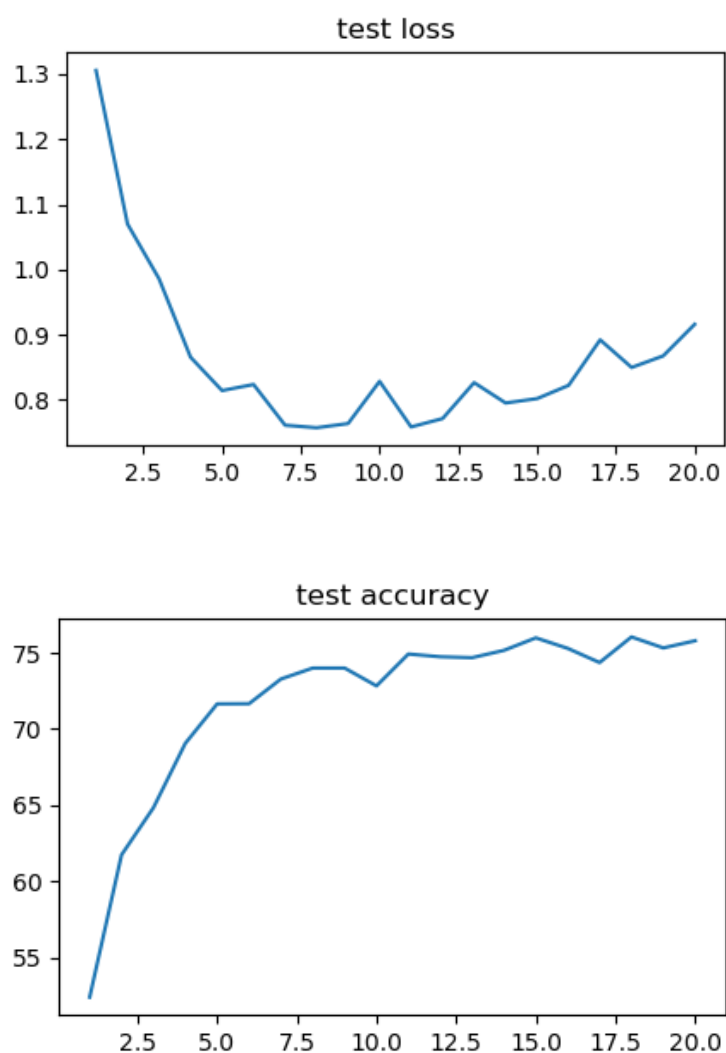
```

104         (1): ReLU(inplace=True)
105         (2): Conv2d(88, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
106         (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
            track_running_stats=True)
107         (4): ReLU(inplace=True)
108         (5): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
            bias=False)
109     )
110     (dropout): Dropout(p=0, inplace=False)
111 )
112 (1): _DenseLayer(
113     (dense_layer): Sequential(
114         (0): BatchNorm2d(120, eps=1e-05, momentum=0.1, affine=True,
            track_running_stats=True)
115         (1): ReLU(inplace=True)
116         (2): Conv2d(120, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
117         (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
            track_running_stats=True)
118         (4): ReLU(inplace=True)
119         (5): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
            bias=False)
120     )
121     (dropout): Dropout(p=0, inplace=False)
122 )
123 (2): _DenseLayer(
124     (dense_layer): Sequential(
125         (0): BatchNorm2d(152, eps=1e-05, momentum=0.1, affine=True,
            track_running_stats=True)
126         (1): ReLU(inplace=True)
127         (2): Conv2d(152, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
128         (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
            track_running_stats=True)
129         (4): ReLU(inplace=True)
130         (5): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
            bias=False)
131     )
132     (dropout): Dropout(p=0, inplace=False)
133 )
134 )
135 )
136 )
137 (avgpool): AvgPool2d(kernel_size=2, stride=1, padding=0)
138 (fc): Linear(in_features=184, out_features=10, bias=True)
139 )

```

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

```

1 ResNet(
2   (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
      bias=False)
3   (bn): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
      track_running_stats=True)
4   (relu): ReLU(inplace=True)
5   (layer1): Sequential(
6     (0): BasicBlock(
7       (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
          bias=False)

```

```

8         (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
9             track_running_stats=True)
10        (relu): ReLU(inplace=True)
11        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
12            bias=False)
13        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
14            track_running_stats=True)
15        (se): SELayer(
16            (avg_pool): AdaptiveAvgPool2d(output_size=1)
17            (fc): Sequential(
18                (0): Linear(in_features=64, out_features=4, bias=False)
19                (1): ReLU(inplace=True)
20                (2): Linear(in_features=4, out_features=64, bias=False)
21                (3): Sigmoid()
22            )
23        )
24    (1): BasicBlock(
25        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
26            bias=False)
27        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
28            track_running_stats=True)
29        (relu): ReLU(inplace=True)
30        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
31            bias=False)
32        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
33            track_running_stats=True)
34        (se): SELayer(
35            (avg_pool): AdaptiveAvgPool2d(output_size=1)
36            (fc): Sequential(
37                (0): Linear(in_features=64, out_features=4, bias=False)
38                (1): ReLU(inplace=True)
39                (2): Linear(in_features=4, out_features=64, bias=False)
40                (3): Sigmoid()
41            )
42        )
43    )
44    (layer2): Sequential(
45        (0): BasicBlock(
46            (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
47                bias=False)
48            (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
49                track_running_stats=True)
49            (relu): ReLU(inplace=True)
50            (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
51                bias=False)
52            (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
53                track_running_stats=True)

```

```

46         (se): SELayer(
47             (avg_pool): AdaptiveAvgPool2d(output_size=1)
48             (fc): Sequential(
49                 (0): Linear(in_features=128, out_features=8, bias=False)
50                 (1): ReLU(inplace=True)
51                 (2): Linear(in_features=8, out_features=128, bias=False)
52                 (3): Sigmoid()
53             )
54         )
55         (downsample): Sequential(
56             (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
57             (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
                    track_running_stats=True)
58         )
59     )
60     (1): BasicBlock(
61         (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
                    bias=False)
62         (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
                    track_running_stats=True)
63         (relu): ReLU(inplace=True)
64         (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
                    bias=False)
65         (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
                    track_running_stats=True)
66         (se): SELayer(
67             (avg_pool): AdaptiveAvgPool2d(output_size=1)
68             (fc): Sequential(
69                 (0): Linear(in_features=128, out_features=8, bias=False)
70                 (1): ReLU(inplace=True)
71                 (2): Linear(in_features=8, out_features=128, bias=False)
72                 (3): Sigmoid()
73             )
74         )
75     )
76 )
77 (layer3): Sequential(
78     (0): BasicBlock(
79         (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
                    bias=False)
80         (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
                    track_running_stats=True)
81         (relu): ReLU(inplace=True)
82         (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
                    bias=False)
83         (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
                    track_running_stats=True)
84         (se): SELayer(
85             (avg_pool): AdaptiveAvgPool2d(output_size=1)

```

```

86         (fc): Sequential(
87             (0): Linear(in_features=256, out_features=16, bias=False)
88             (1): ReLU(inplace=True)
89             (2): Linear(in_features=16, out_features=256, bias=False)
90             (3): Sigmoid()
91         )
92     )
93     (downsample): Sequential(
94         (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
95         (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
          track_running_stats=True)
96     )
97 )
98 (1): BasicBlock(
99     (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
      bias=False)
100    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
      track_running_stats=True)
101    (relu): ReLU(inplace=True)
102    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
      bias=False)
103    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
      track_running_stats=True)
104    (se): SELayer(
105        (avg_pool): AdaptiveAvgPool2d(output_size=1)
106        (fc): Sequential(
107            (0): Linear(in_features=256, out_features=16, bias=False)
108            (1): ReLU(inplace=True)
109            (2): Linear(in_features=16, out_features=256, bias=False)
110            (3): Sigmoid()
111        )
112    )
113 )
114 )
115 (layer4): Sequential(
116     (0): BasicBlock(
117         (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
          bias=False)
118         (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
          track_running_stats=True)
119         (relu): ReLU(inplace=True)
120         (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
          bias=False)
121         (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
          track_running_stats=True)
122         (se): SELayer(
123             (avg_pool): AdaptiveAvgPool2d(output_size=1)
124             (fc): Sequential(
125                 (0): Linear(in_features=512, out_features=32, bias=False)

```

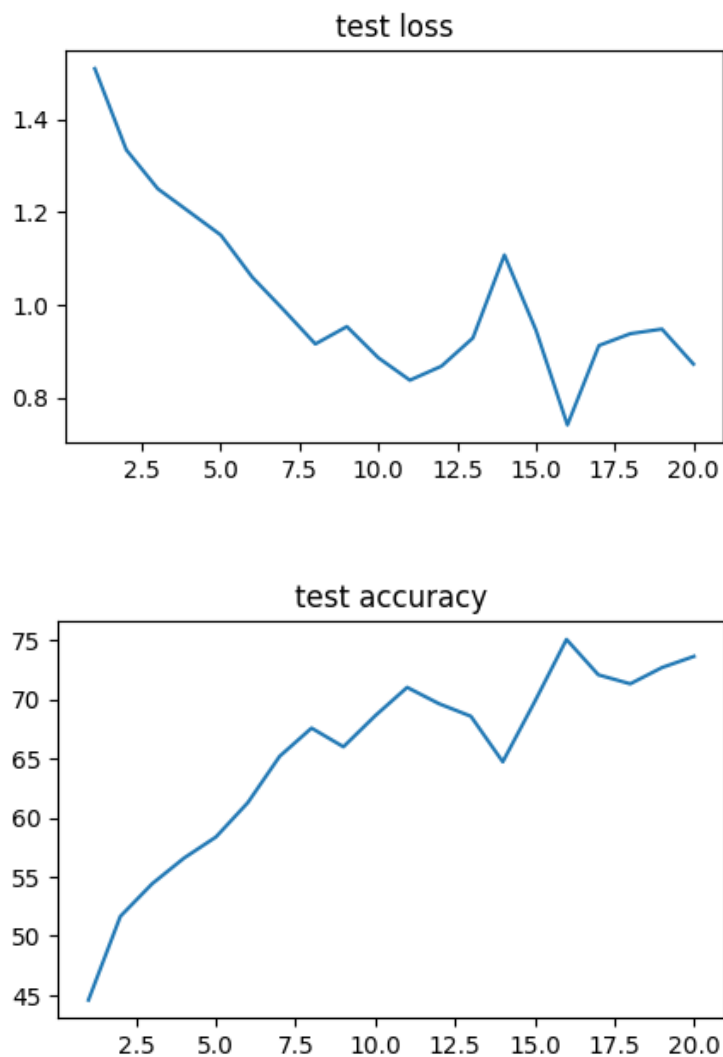
```

126         (1): ReLU(inplace=True)
127         (2): Linear(in_features=32, out_features=512, bias=False)
128         (3): Sigmoid()
129     )
130 )
131 (downsample): Sequential(
132     (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
133     (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
134 )
135 )
136 (1): BasicBlock(
137     (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
        bias=False)
138     (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
139     (relu): ReLU(inplace=True)
140     (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
        bias=False)
141     (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
142     (se): SELayer(
143         (avg_pool): AdaptiveAvgPool2d(output_size=1)
144         (fc): Sequential(
145             (0): Linear(in_features=512, out_features=32, bias=False)
146             (1): ReLU(inplace=True)
147             (2): Linear(in_features=32, out_features=512, bias=False)
148             (3): Sigmoid()
149         )
150     )
151 )
152 )
153 (avg_pool): AdaptiveAvgPool2d(output_size=(1, 1))
154 (fc): Linear(in_features=512, out_features=10, bias=True)
155 )

```

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

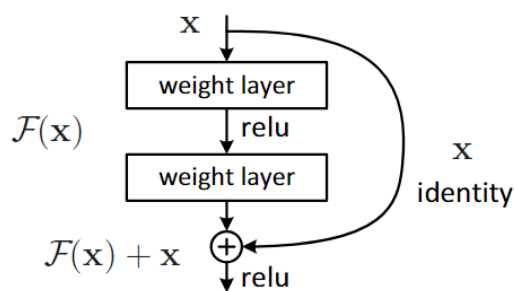


图 5.1: resnet 基础结构 [1]

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

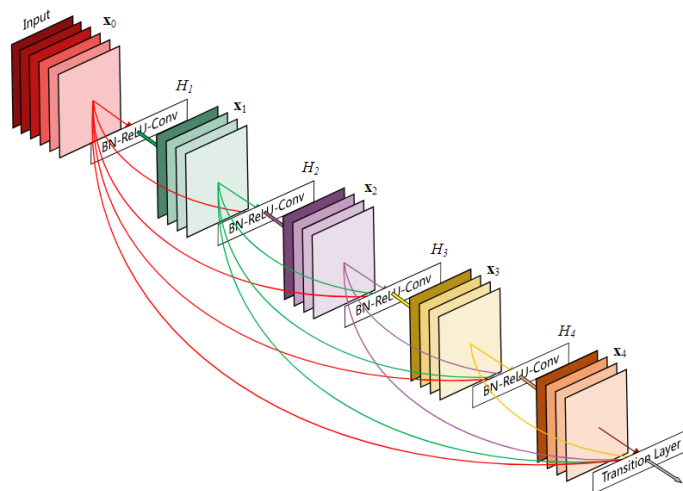


图 5.2: densenet 基础结构 [2]

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”。DenseNet 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 DenseNet 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

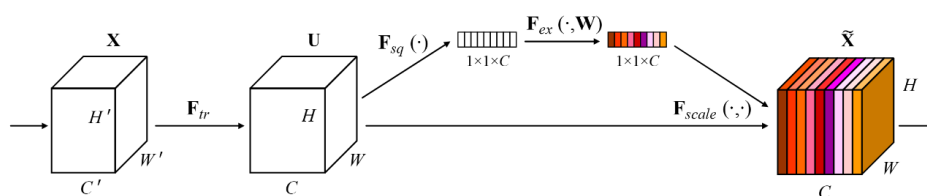


图 5.3: se block 基础结构 [3]

SENet 工作机制如下：

对于每一输出通道，先进行全局平均池化，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.