
南開大學
Nankai University

南　　开　　大　　学

网 络 空 间 安 全 学 院

**深度学习平时作业**

---

# CNN 实验报告

---

穆禹宸　2012026

年级：2020 级

专业：信息安全、法学双学位班

指导教师：侯淇彬

2023 年 6 月 23 日

# 目录

# 一、 实验过程

## (一) 原始版本 CNN

### 1. 网络结构

```
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)
)
```

### 2. 实验结果

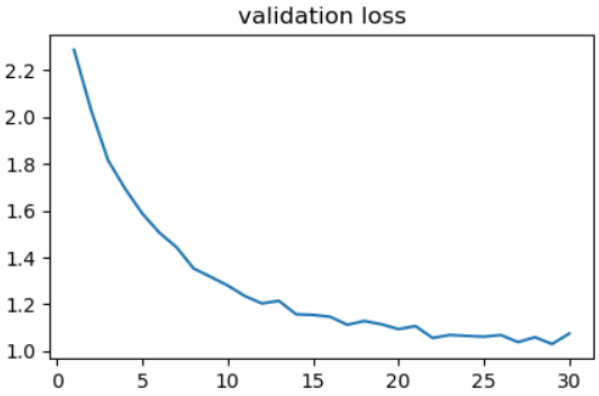


图 1: **原始 CNN 的损失**

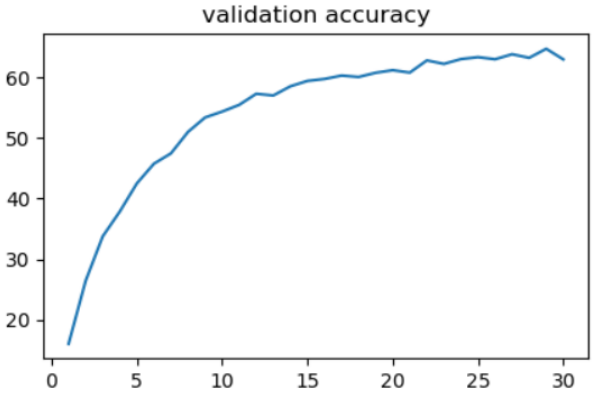


图 2: **原始 CNN 的准确率**

可以看到原始 CNN 的效果。

## (二) RESNET-18

### 1. 网络结构

```
ResNet(
  (conv1): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (conv2_x): Sequential(
    (0): BasicBlock(
      (residual_function): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (shortcut): Sequential()
    )
    (1): BasicBlock(
      (residual_function): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (shortcut): Sequential()
    )
  )
  (conv3_x): Sequential(
    (0): BasicBlock(
      (residual_function): Sequential(
        (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```
        (2): ReLU(inplace=True)
        (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (shortcut): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (residual_function): Sequential(
        (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (shortcut): Sequential()
    )
  )
  (conv4_x): Sequential(
    (0): BasicBlock(
      (residual_function): Sequential(
        (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (shortcut): 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(
      (residual_function): Sequential(
```

```
70          (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
                1), bias=False)
71          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
72          (2): ReLU(inplace=True)
73          (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
                1), bias=False)
74          (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
75        )
76        (shortcut): Sequential()
77      )
78    )
79    (conv5_x): Sequential(
80      (0): BasicBlock(
81        (residual_function): Sequential(
82          (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
                1), bias=False)
83          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
84          (2): ReLU(inplace=True)
85          (3): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
                1), bias=False)
86          (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
87        )
88        (shortcut): Sequential(
89          (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
90          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
91        )
92      )
93      (1): BasicBlock(
94        (residual_function): Sequential(
95          (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
                1), bias=False)
96          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
97          (2): ReLU(inplace=True)
98          (3): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
                1), bias=False)
99          (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
100       )
101       (shortcut): Sequential()
102     )
103   )
104   (avg_pool): AdaptiveAvgPool2d(output_size=(1, 1))
```

```
  (fc): Linear(in_features=512, out_features=10, bias=True)
)
```

我所实现的网络是 ResNet-18，这是由于资源有限，无法训练更大的神经网络，这里仅采取了 basic block。

### 2. 实验结果

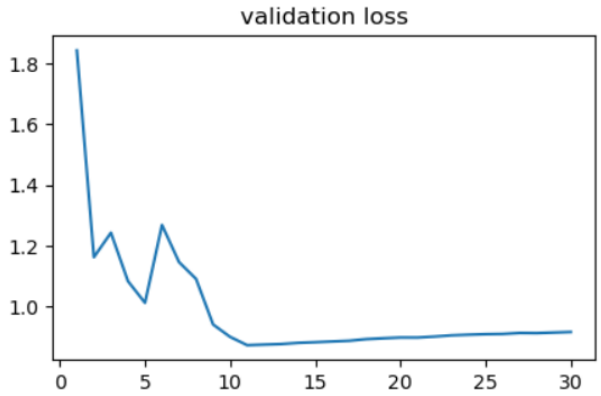


图 3: **RESNET-18 的损失**

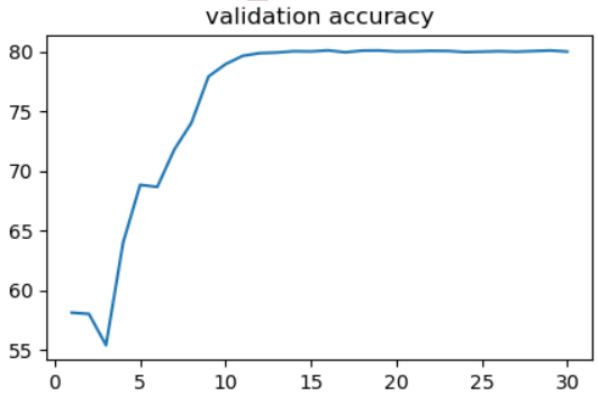


图 4: **RESNET-18 的准确率**

RESNET-18 的效果远远好于原始 CNN，大概在 80% 的正确率左右收敛。

## (三) DenseNet

### 1. 网络结构

```
DenseNet(
  (features): Sequential(
    (conv0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
        bias=False)
    (norm0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
    (relu0): ReLU(inplace=True)
```

```
(pool0): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
    ceil_mode=False)
(denseblock1): _DenseBlock(
  (denselayer1): _DenseLayer(
    (norm1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
    (relu1): ReLU(inplace=True)
    (conv1): Conv2d(64, 128, kernel_size=(1, 1), stride=(1, 1), bias=
        False)
    (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
    (relu2): ReLU(inplace=True)
    (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding
        =(1, 1), bias=False)
  )
  (denselayer2): _DenseLayer(
    (norm1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
    (relu1): ReLU(inplace=True)
    (conv1): Conv2d(96, 128, kernel_size=(1, 1), stride=(1, 1), bias=
        False)
    (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
    (relu2): ReLU(inplace=True)
    (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding
        =(1, 1), bias=False)
  )
  (denselayer3): _DenseLayer(
    (norm1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
    (relu1): ReLU(inplace=True)
    (conv1): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1), bias=
        False)
    (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
    (relu2): ReLU(inplace=True)
    (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding
        =(1, 1), bias=False)
  )
  (denselayer4): _DenseLayer(
    (norm1): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
    (relu1): ReLU(inplace=True)
    (conv1): Conv2d(160, 128, kernel_size=(1, 1), stride=(1, 1), bias=
        False)
    (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
    (relu2): ReLU(inplace=True)
```

```
      (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding
          =(1, 1), bias=False)
    )
  )
  (transition1): _Transition(
    (norm): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv): Conv2d(192, 96, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (pool): AvgPool2d(kernel_size=2, stride=2, padding=0)
  )
  (denseblock2): _DenseBlock(
    (denselayer1): _DenseLayer(
      (norm1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True,
          track_running_stats=True)
      (relu1): ReLU(inplace=True)
      (conv1): Conv2d(96, 128, kernel_size=(1, 1), stride=(1, 1), bias=
          False)
      (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
          track_running_stats=True)
      (relu2): ReLU(inplace=True)
      (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding
          =(1, 1), bias=False)
    )
    (denselayer2): _DenseLayer(
      (norm1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
          track_running_stats=True)
      (relu1): ReLU(inplace=True)
      (conv1): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1), bias=
          False)
      (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
          track_running_stats=True)
      (relu2): ReLU(inplace=True)
      (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding
          =(1, 1), bias=False)
    )
    (denselayer3): _DenseLayer(
      (norm1): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True,
          track_running_stats=True)
      (relu1): ReLU(inplace=True)
      (conv1): Conv2d(160, 128, kernel_size=(1, 1), stride=(1, 1), bias=
          False)
      (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
          track_running_stats=True)
      (relu2): ReLU(inplace=True)
      (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding
          =(1, 1), bias=False)
    )
```

```
72        (denselayer4): _DenseLayer(
73          (norm1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
74          (relu1): ReLU(inplace=True)
75          (conv1): Conv2d(192, 128, kernel_size=(1, 1), stride=(1, 1), bias=
                False)
76          (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
77          (relu2): ReLU(inplace=True)
78          (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding
                =(1, 1), bias=False)
79        )
80      )
81      (transition2): _Transition(
82        (norm): BatchNorm2d(224, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
83        (relu): ReLU(inplace=True)
84        (conv): Conv2d(224, 112, kernel_size=(1, 1), stride=(1, 1), bias=False)
85        (pool): AvgPool2d(kernel_size=2, stride=2, padding=0)
86      )
87      (denseblock3): _DenseBlock(
88        (denselayer1): _DenseLayer(
89          (norm1): BatchNorm2d(112, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
90          (relu1): ReLU(inplace=True)
91          (conv1): Conv2d(112, 128, kernel_size=(1, 1), stride=(1, 1), bias=
                False)
92          (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
93          (relu2): ReLU(inplace=True)
94          (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding
                =(1, 1), bias=False)
95        )
96        (denselayer2): _DenseLayer(
97          (norm1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
98          (relu1): ReLU(inplace=True)
99          (conv1): Conv2d(144, 128, kernel_size=(1, 1), stride=(1, 1), bias=
                False)
100         (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
101         (relu2): ReLU(inplace=True)
102         (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding
                =(1, 1), bias=False)
103       )
104       (denselayer3): _DenseLayer(
105         (norm1): BatchNorm2d(176, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
```

```
106         (relu1): ReLU(inplace=True)
107         (conv1): Conv2d(176, 128, kernel_size=(1, 1), stride=(1, 1), bias=
                False)
108         (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
109         (relu2): ReLU(inplace=True)
110         (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding
                =(1, 1), bias=False)
111       )
112       (denselayer4): _DenseLayer(
113         (norm1): BatchNorm2d(208, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
114         (relu1): ReLU(inplace=True)
115         (conv1): Conv2d(208, 128, kernel_size=(1, 1), stride=(1, 1), bias=
                False)
116         (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
117         (relu2): ReLU(inplace=True)
118         (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding
                =(1, 1), bias=False)
119       )
120     )
121     (transition3): _Transition(
122       (norm): BatchNorm2d(240, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
123       (relu): ReLU(inplace=True)
124       (conv): Conv2d(240, 120, kernel_size=(1, 1), stride=(1, 1), bias=False)
125       (pool): AvgPool2d(kernel_size=2, stride=2, padding=0)
126     )
127     (denseblock4): _DenseBlock(
128       (denselayer1): _DenseLayer(
129         (norm1): BatchNorm2d(120, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
130         (relu1): ReLU(inplace=True)
131         (conv1): Conv2d(120, 128, kernel_size=(1, 1), stride=(1, 1), bias=
                False)
132         (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
133         (relu2): ReLU(inplace=True)
134         (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding
                =(1, 1), bias=False)
135       )
136       (denselayer2): _DenseLayer(
137         (norm1): BatchNorm2d(152, eps=1e-05, momentum=0.1, affine=True,
                track_running_stats=True)
138         (relu1): ReLU(inplace=True)
139         (conv1): Conv2d(152, 128, kernel_size=(1, 1), stride=(1, 1), bias=
                False)
```

```
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
            track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding
            =(1, 1), bias=False)
      )
      (denselayer3): _DenseLayer(
        (norm1): BatchNorm2d(184, eps=1e-05, momentum=0.1, affine=True,
            track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(184, 128, kernel_size=(1, 1), stride=(1, 1), bias=
            False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
            track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding
            =(1, 1), bias=False)
      )
      (denselayer4): _DenseLayer(
        (norm1): BatchNorm2d(216, eps=1e-05, momentum=0.1, affine=True,
            track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(216, 128, kernel_size=(1, 1), stride=(1, 1), bias=
            False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
            track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding
            =(1, 1), bias=False)
      )
    )
    (norm5): BatchNorm2d(248, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
  )
  (classifier): Linear(in_features=248, out_features=10, bias=True)
)
```

我所实现的版本,是以 ResNet-18 作为 backbone 的 DenseNet 中,仅使用 basicblock 的"DenseNet-B"。在所有版本的 DenseNet 中，基本块是网络的基本组建。DenseNet 的基本块比 ResNet 的基础块更为复杂。它包括了三个卷积层，其中第一个卷积层输出的特征图将被连接到后续卷积层的输入中。这意味着后续卷积层将密集地集成来自前一个卷积层的特征信息。因此，每个基本块的输出包含所有该层之前的特征信息。然后将其输出作为下一个基本块的输入。

DenseNet-B 与 ResNet-18 相同，包含 4 个卷积块，每个卷积块包含两个 basicblock。基本块的输入和输出大小仍然是 32 * 32，每个 basicblock 中都有 2 个 3x3 卷积层与恒等映射。DenseNet-B 相对于 ResNet-18 的不同之处在于，在基本块之间使用密集连接的方式。

### 2. 实验结果

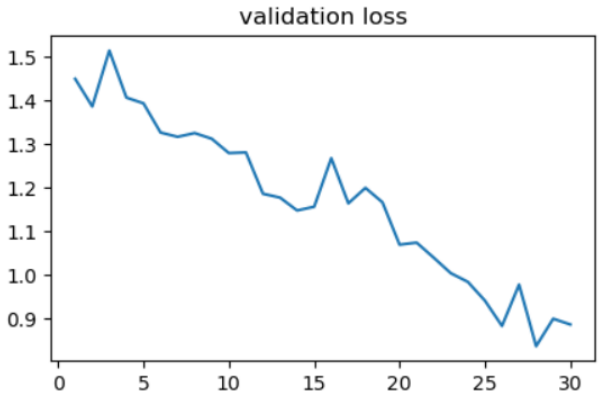


图 5: **DenseNet 的损失**

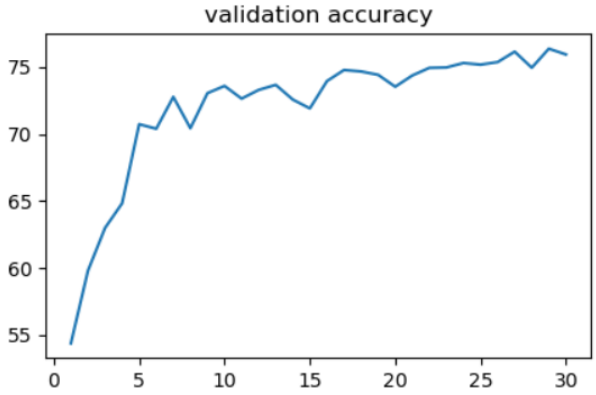


图 6: **DenseNet 的准确率**

奇怪的是 DenseNet 的效果并没有优于 ResNet，大概在 77% 的正确率左右收敛，这里猜测是由于我使用的 backbone 是 ResNet-18，而在原始论文之中至少都是 100 层，因此 DenseNet 的性能没有完全得到体现。

## (四) SE-ResNet

### 1. 网络结构

```
SENet(
  (conv1): Conv2d(3, 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)
  (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=(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(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=(2, 2), 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(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)
      (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(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)
      (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(
```

```
74        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding
              =(1, 1), bias=False)
75        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
76        (relu): ReLU(inplace=True)
77        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding
              =(1, 1), bias=False)
78        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
              track_running_stats=True)
79      )
80    )
81    (avg_pool): AdaptiveAvgPool2d(output_size=1)
82    (fc): Linear(in_features=512, out_features=10, bias=True)
83    (se): SEBlock(
84      (avg_pool): AdaptiveAvgPool2d(output_size=1)
85      (fc1): Conv2d(512, 32, kernel_size=(1, 1), stride=(1, 1), bias=False)
86      (relu): ReLU(inplace=True)
87      (fc2): Conv2d(32, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
88      (sigmoid): Sigmoid()
89    )
90  )
```

SENet 的主要思想是在每个卷积块中插入 SE 块，以自适应地重新校准每个信道的重要性，从而增强信道间的关联性。在 SE 块中，首先对每个特征张量进行全局平均池化，以获得描述其平均激活的向量。然后，将该向量通过两个全连接层，其中第一个全连接层将向量压缩为原始信道数的 1/16，并使用非线性 ReLU 激活函数，第二个全连接层将向量恢复到原始信道数，并使用 Sigmoid 激活函数。最后，使用计算出来的重要性向量来对输入的特征进行重新校准，使得网络能够更好地学习到特征。

对于 SENet 的结构，它包含了多个卷积层和 SE 块。整个网络以一组卷积层开始，其中卷积层的深度从低到高逐渐增加，以便提取越来越高级别的特征。在每一个卷积块中，都会插入一个 SE 块。在每个卷积块中，还包含两个 3*3 的卷积层和一个恒等映射来改变通道的数量，以确保输入和输出的维数和尺寸相同。整个网络的最后一层是全局平均池化层，用于将特征压缩成一个向量，并将该向量输入到一个完全连接层中进行分类。

**2. 实验结果**

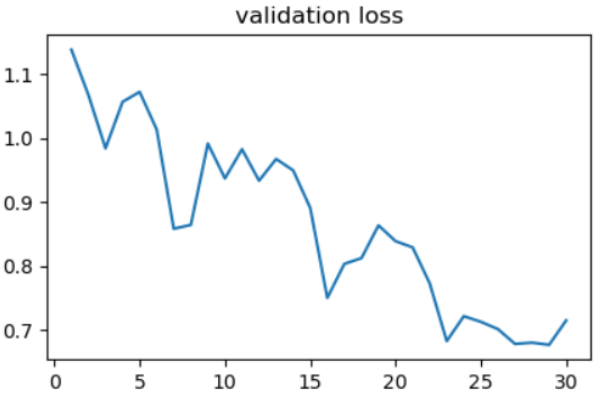


图 7: **SE-ResNet 的损失**

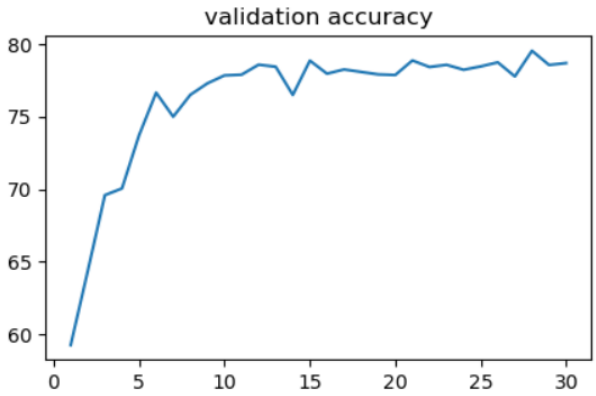


图 8: **SE-ResNet 的准确率**

SE-Resnet 的性能也大概达到了 80% 左右。

# 二、 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

## (一) 各个网络的比较

**没有跳跃连接的卷积神经网络** 在没有跳跃连接的卷积神经网络中，网络主要由卷积层和池化层构成，相邻的层之间没有跳跃连接，信息只能从前向后依次流动。这种结构的另一种常见形式是全连接网络，是最早的深度神经网络之一。没有跳跃连接的卷积神经网络具有计算效率高和参数较少的优点，但大部分不能训练得很深，因为随着层数的逐渐增加，梯度消失或梯度爆炸等问题会导致模型难以收敛。

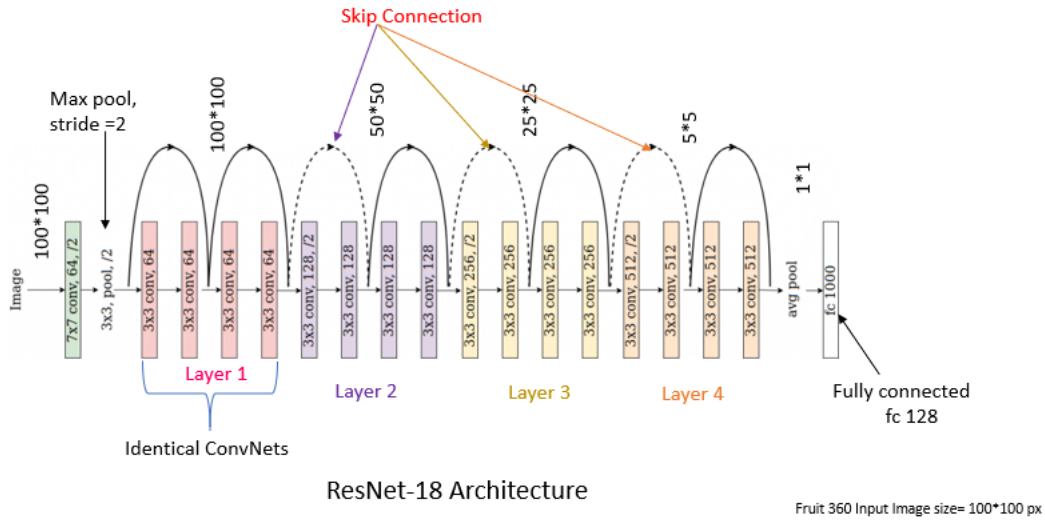


图 9: **ResNet**

[1]

**ResNet** ResNet 是一种具有跳跃连接的卷积神经网络，主要解决的问题是网络退化的问题。当网络层数增加时，网络的训练误差开始上升，这意味着网络的性能开始下降。让网络变得更深并不总是比浅模型好。ResNet 引入了跨层次跳跃连接，通过将输入直接提供到输出层，来有效地解决这个问题。这种跳跃连接不仅增强了网络的逐层表征能力，而且还可以有效地缓解梯度消失问题，并提高网络的训练速度与准确性。

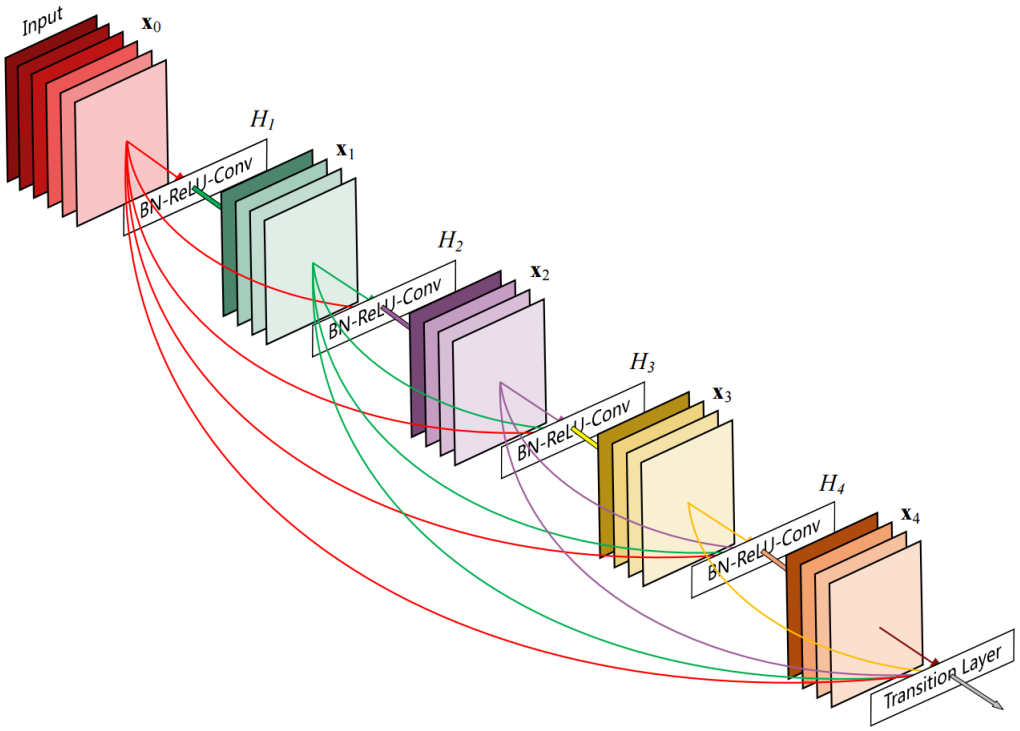


图 10: **DenseNet**

[3]

**DenseNet** DenseNet 是一种密集跳跃连接的卷积神经网络。节点之间的连接不是像 ResNet 那样跨越层的边缘，而是在同一层中直接连接。DenseNet 利用了浅层和深层之间的信息流，通过密集连接可以使特征更好地传递和重用，进一步提高网络性能。DenseNet 的架构设计使得该模型在训练过程中需要比 ResNet 更少的参数，而且训练时还能缓解梯度消失的问题，并且在训练过程中可以减轻过度拟合问题。

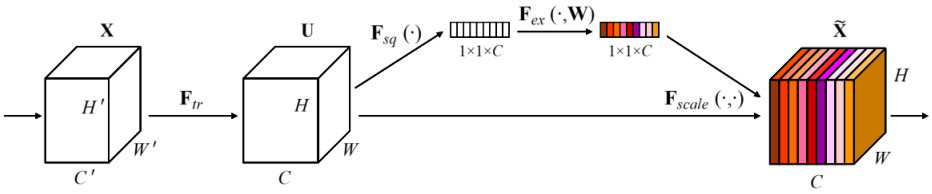


图 11: **SE-ResNet**

[2]

**SE-ResNet** SE-ResNet 结合了跳跃连接和 SE 块，通过自适应地重新校准每个信道的重要性来增强网络性能。在 SE-ResNet 中，Squeeze-and-Excitation（SE）块被引入用于对每个信道重新校准其重要性，这使得 SE-ResNet 在训练过程中能够更好地学习到不同的特征，从而提高了泛化性能。此外，跳跃连接可以帮助传递学到的特征和加速模型的优化过程。在训练过程中，SE-

ResNet 表现出更快的收敛速度和更好的泛化性能，相对于没有跳跃连接的卷积神经网络而言，在 ResNet 的基础上，带有 SE 模块的 ResNet 进一步增强了 ResNet 的表示能力。SE 模块的主要作用是动态地学习通道之间的相关性并对不相关的通道进行清理。SE-ResNet 模型的架构可以用于多种图像分类和检测任务，并在各种数据集上实现了最先进的性能。相比于传统的 ResNet，SE-ResNet 在保持高精度的同时，具有更少的参数。

## (二) 更具体的分析

对于网络结构来说，没有跳跃连接的卷积神经网络主要由一系列卷积层和池化层构成，相邻的层之间没有跳跃连接。而 ResNet 和 DenseNet 都使用了跨层的跳跃连接来促进前向信号和梯度的流动。SE-Resnet 则是采取了 SE 块进行训练。ResNet 中跳跃连接是从输入到输出直接跨越层的边缘，而 DenseNet 则采用了密集的跳跃连接，将当前层的所有输入都连接到下一层的所有输出上。这种连接方式可以有效地提高网络的表示能力，使得网络更容易训练和优化。

从梯度传递的角度来看，在没有跳跃连接的卷积神经网络中，由于网络是顺序执行的，由于信息只能从前向后依次流动，而无法从某些层向前或向后传递。这样就会导致在网络的深层次结构中，梯度信号会逐渐消失而难以传递。梯度的传递会因为层数增多而逐渐消失或爆炸，导致模型难以收敛。而 ResNet 和 DenseNet 都采用了跳跃连接，提升了梯度和信息的传递。ResNet 跳跃连接从输入到输出直接跨越边缘，可以缓解梯度消失的问题，同时提高网络的表达能力，这使得通过 ResNet 训练深层网络变得更加容易。而 DenseNet 使用密集连接来使每一层都能够访问与前一层的所有特征，这样每一个层的梯度都可以传递到所有的后继层，不会有梯度消失的问题。

以信息提取角度观察，CNN 由于学习到的特征会被多个层重复使用，因此网络难以学习到全局的特征信息。DenseNet 的网络结构是通过密集连接实现的，每一层都能够访问前一层的全部输出，在逐层的处理中，深度和宽度的维度的信息都得到了充分的传递和提取。这使得 DenseNet 可以充分挖掘出数据的大量特征，使得网络的训练过程更快、效果更好。通过全局共享特征来增加网络中的信息流，将每一层的输出和前面所有层的输出拼接起来，从而使得网络可以更好地学习到全局特征。相比之下，ResNet 的网络结构中仍然存在跨越多层的信息流，难以挖掘数据的深度特征信息。在 SE-ResNet 中，SE 块用于学习每个信道的重要性，以使得网络学习到的特征更加丰富，从而使得特征的重要性得到进一步提升。

而对训练速度来说，由于跳跃连接对梯度和信息的传递起了积极的作用，DenseNet 和 ResNet 的训练速度都相对较快，但 DenseNet 的训练速度还可以更快一些，因为它使用了密集的连接而且参数更少，因此 DenseNet 模型不仅速度快，而且训练结果比 ResNet 更透彻。带有 SE 模块的 ResNet 进一步增强了 ResNet 的表示能力，使得网络训练效果更加显著。因此 SE 模块的其中一个直接作用就在于减少训练参数，加快训练速度。SE-ResNet 利用了共享权重的密集块特点，使得 SE 模块的参数数量也很少，因此参数数量是最少的。

# 参考文献

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016:770–778, 2016.

[2] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[3] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017:4700–4708, 2017.