南開大學
Nankai University

网络空间安全学院

深度学习实验报告

实验二：卷积神经网络

姓名：武桐西

学号：2112515

专业：信息安全

指导教师：侯淇彬

2024 年 6 月 25 日

目录

1 实验目的

本次实验的主要内容是卷积神经网络（Convolutional Neural Network, CNN）。通过本次实验，需要掌握以下内容：

1. 掌握 PyTorch 框架的基础算子，如卷积、池化、激活等。
2. 学会使用 PyTorch 框架搭建简单的卷积神经网络并训练。
3. 了解经典的卷积神经网络的结构，如 LeNet、ResNet、DenseNet、MobileNet 等。

本次实验的实验要求如下：

1. 使用 PyTorch 框架，搭建卷积神经网络，在 CIFAR-10 数据集上进行训练与测试。
2. 实现课程中提供的原始版本的卷积神经网络并分析结果。
3. 实现 ResNet 网络并分析结果。
4. 实现 DenseNet 网络并分析结果。
5. 实现 MobileNet 网络并分析结果。

2 实验环境

本实验在 macOS 系统下测试，搭配 M3 Max 芯片，使用 Python`3.11.5`解释器，PyTorch 版本为`2.2.0`（使用 MPS 进行硬件加速），如表1所示。

表 1: 实验环境

操作系统	macOS	硬件	M3 Max (支持 MPS)
Python	3.11.5	PyTorch	2.2.0 (支持 MPS)

本次实验的代码已上传GitHub。

3 文件目录结构

本次实验的文件目录结构如下：

```
CNN
├── data ..... Data Path
├── models ..... Models
│   ├── cnn_base.py
│   ├── densenet.py
│   ├── mobilenet.py
│   └── resnet.py
├── trainer.py ..... Trainer
├── visualization.py ..... Visualization
└── main.py ..... Main Script
```

4 实验原理与实验过程

在本部分，首先介绍本次实验的目标任务和数据集，然后介绍四种卷积神经网络架构的实现，最后介绍整个实验的流程。

4.1 目标任务

本次实验需要实现**图像分类**任务，使用 CIFAR-10 数据集，该数据集中的图像为彩色图像（RGB 三通道），其形状为 32×32。

4.2 CNN Base

根据课程中提供的源代码，将其代码规范化后编写基础的 CNN Base 模型，其网络架构如代码1所示[1]。

代码 1: CNN Model Architecture

```
CNN(
  (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)
)
```

在卷积操作部分，每一个卷积操作块主要包含三种算子操作，按先后顺序依次为卷积操作算子、激活函数算子、池化操作算子；之后为全连接操作部分，每一个全连接层后面紧接一个激活函数层（最后一个全连接层除外，因为其一般和损失函数一起，在损失函数中会计算 Softmax 操作）；二者之间需要将张量的形状展平为一维张量（若考虑批处理，则实际为二维张量）。

4.3 ResNet

ResNet[1] 中创造性地提出了**残差**结构，将卷积块中的输出 $f(x)$ 与输入 x 相加（相减），使得在梯度反向传播时，卷积块的残差连接（即 x 所在线路）的梯度恒为单位 1，这样就解决了以往随着网络的层数加深后带来的**梯度消失**问题，如图4.1所示。

[1]由于篇幅的所限以及其他网络层数较深，因此后面的网络架构在本报告中省略，详细可参考源代码

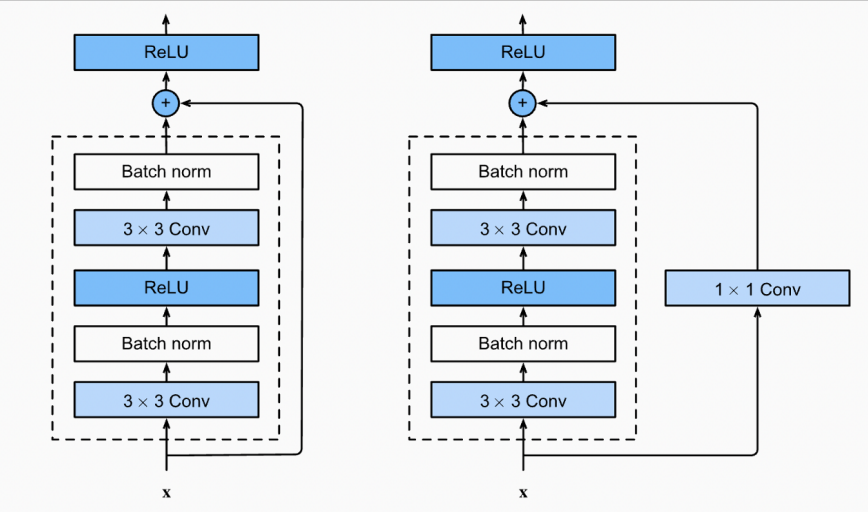


图 4.1: ResNet 残差结构 [1]（以 BasicBlock 块结构为例）

4.3.1 实现过程

ResNet 共有五种参数大小的版本，分别为 ResNet-18、ResNet-34、ResNet-50、ResNet-101 和 ResNet-152。ResNet 的网络架构如图4.2所示。

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2 $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

图 4.2: ResNet 网络结构 [1]

可以看到，每一种参数版本的网络结构均有四层卷积块层，其中前两种参数版本的 ResNet 采用的是 BasicBlock 块结构（如图4.1所示），另外三种 ResNet 采用的是 Bottleneck 块结构（参见附录B中的图B.8）。二者之中后者更为复杂，参数量也更多，因此也用在更深层的网络中。

在实现时，分别编写 BasicBlock 块结构和 Bottleneck 块结构的类，在 ResNet 类中封装`_make_layer`函数用以构建 ResNet 中的四个卷积块层。最后，编写函数，根据层数的不同，调用不同的基本卷积块（BasicBlock 块结构或 Bottleneck 块结构）并提供不同的块数参数，便可构建五种深度的 ResNet，如附录A中代码3所示。

4.3.2 与原论文 [1] 的区别

本实验中实现的 ResNet 与原论文 [1] 基本一致，但是参数略有差异。由于原论文中的 ResNet 结构是在 ImageNet 数据集中使用的，数据预处理时会同一将图片的形状 Resize 为 224×224，而本实验中使用的 CIFAR-10 数据集中图像的形状为 32×32，而图像的大小每次通过 ResNet 的一层（包括最开始的卷积和池化层）后会减半，这会导致图像的大小提前减为 1×1，因此本次实验中去掉了四层卷积块层之后的平均池化层，如附录A中代码2所示。

4.4 DenseNet

DenseNet[?] 是在 ResNet 的基础上进行改进的，与 ResNet 的最大的不同在于 DenseNet 将 ResNet 中的残差的相加操作 $\bigoplus$ 变为拼接操作`Concat`，这也是其名称中"Dense" 的含义。

DenseNet 的基本架构与 ResNet 类似，其主要由两部分构成：

1. **稠密块**（Dense Block）：利用拼接操作`Concat`实现稠密连接。
2. **过渡层**（Transition Layer）：控制通道数量，缓解由于稠密连接而导致通道数过多的问题，使其不会太复杂。

4.4.1 与原论文的区别 [3]

与 ResNet 类似，由于数据集图像大小的问题（224×224 v.s. 32×32），本次实验中实现的 DenseNet 与原论文 [3] 中略有区别，但是与 ResNet 的解决方式不同，本实验中将 DenseNet 的卷积层的相关参数进行修改，使得最终的全连接层前的图像大小变为 1×1，而不是 ResNet 中直接去掉最后的平均池化层。

除此之外，DenseNet 的实现与前面 ResNet 的实现基本类似，这里不再赘述，详情可参见源代码。

4.5 MobileNet

MobileNet[2, 5] 中，采用了深度可分离卷积（Depthwise Separable Convolution），与传统卷积相比具有更低的的参数量和计算量，因此更适合于移动端等计算能力较差的嵌入式设备。

深度可分离卷积可以看做由深度卷积（Depthwise Convolution）和逐点卷积（Pointwise Convolution）组成。Depthwise Convolution 是一种轻量级卷积操作，其中每个输入通道分别与单独的过滤器进行卷积，从而显著减少参数和计算量，如图C.9所示；之后可以通过 Pointwise Convolution（即 1×1 卷积）增加特征图的通道数，提取更多的特征，如图C.10所示。

4.6 实验流程

本次实验的实验流程如下：

1. **数据预处理**：加载 CIFAR-10 数据集，并进行数据预处理（归一化操作等）。
2. **模型选择**：根据命令行参数选择要使用的模型。
3. **模型训练**：使用 Trainer 类进行模型训练。
4. **模型评估**：在验证集上评估模型，并保存模型。
5. **结果可视化**：绘制训练过程中的损失以及准确率曲线（训练集和验证集），并绘制混淆矩阵。

5 实验结果与分析

5.1 实验设置

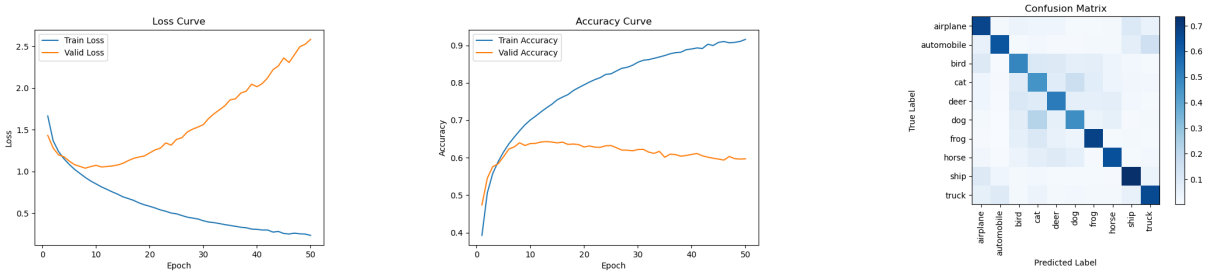
本实验中，所有的超参数设置均相同，如表2所示。

表 2: 实验设置

batch size	64	# of epoch	50
loss function	CrossEntropyLoss	learning rate	0.001
优化器	Adam	硬件加速	MPS

5.2 结果与分析

5.2.1 CNN Base



(a) 损失曲线 (b) 准确率曲线 (c) 混淆矩阵

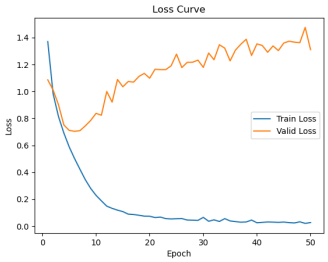
图 5.3: CNN Base Model 结果

可以看到，CNN Base 模型在第 10 个 Epoch 左右达到临界点，在此之前训练集与验证集的损失共同下降，在该点之后训练集的损失虽然仍逐渐降低，但是验证集的损失并没有明显降低，甚至不降反升，说明此后模型发生**过拟合现象**。而训练集上的损失在前 50 个 Epoch 的训练中并没有表现出收敛的趋势，从图5.3中可以看出预计在第 50 个 Epoch 左右收敛。从混淆矩阵上看，模型的表现整体较好，验证集上的最佳准确率为 64.19%。其余结果信息综合在表3所示。

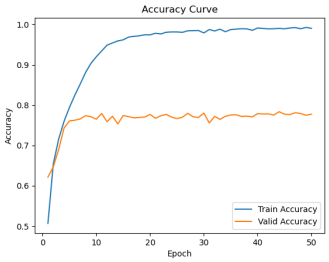
5.2.2 ResNet

在 ResNet 中，尽管笔者实现了 ResNet-18、ResNet-34、ResNet-50、ResNet-101 和 ResNet-152 五种不同深度的 ResNet 架构，但在这里仅展示 ResNet-18 与 ResNet-50 的结果。

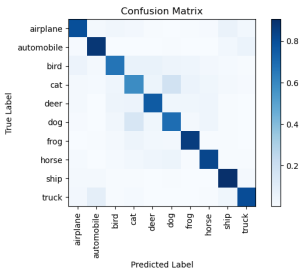
可以看到，ResNet-18 模型在第 5 个 Epoch 左右达到临界点，在此之前训练集与验证集的损失共同下降，在该点之后训练集的损失虽然仍逐渐降低，但是验证集的损失并没有明显降低，甚至不降反升，说明此后模型发生**过拟合现象**。而训练集上的损失在第 20 个 Epoch 之后下降比较缓慢，说明模型逐渐收敛。从混淆矩阵上看，模型的表现整体较好，验证集上的最佳准确率为 77.98%。其余结果信息综合在表3所示。



(a) 损失曲线



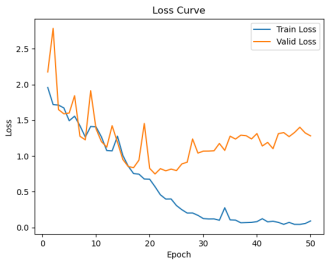
(b) 准确率曲线



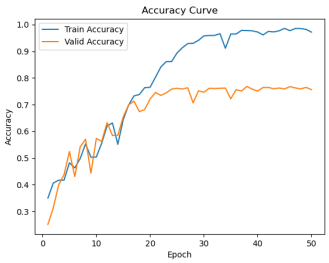
(c) 混淆矩阵

图 5.4: ResNet-18 Model 结果

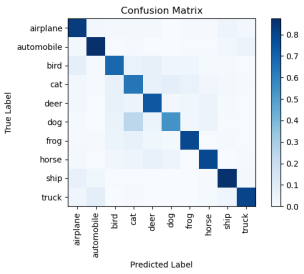
可以看到，ResNet-50 模型在第 20 个 Epoch 左右达到临界点，在此之前训练集与验证集的损失共同下降，在该点之后训练集的损失虽然仍逐渐降低，但是验证集的损失并没有明显降低，甚至不降反升，说明此后模型发生**过拟合现象**。而训练集上的损失在第 30 个 Epoch 之后下降比较缓慢，说明模型逐渐收敛。从混淆矩阵上看，模型的表现整体较好，验证集上的最佳准确率为 76.83%。其余结果信息综合在表3所示。



(a) 损失曲线



(b) 准确率曲线



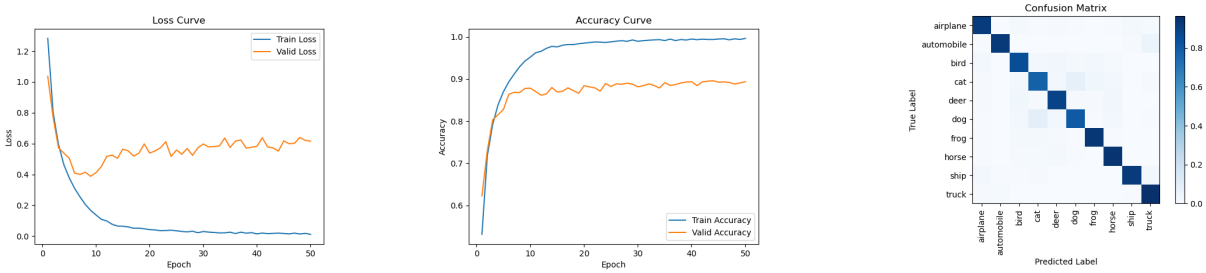
(c) 混淆矩阵

图 5.5: ResNet-50 Model 结果

5.2.3 DenseNet

在 DenseNet 中，尽管笔者实现了 DenseNet121、DenseNet169、DenseNet201 和 DenseNet161 四种不同深度的 DenseNet 架构，但在这里仅展示 DenseNet121 的结果。

可以看到，DenseNet121 模型在第 10 个 Epoch 左右达到临界点，在此之前训练集与验证集的损失共同下降，在该点之后训练集的损失虽然仍逐渐降低，但是验证集的损失并没有明显降低，甚至不降反升，说明此后模型发生**过拟合现象**。而训练集上的损失在第 20 个 Epoch 之后下降比较缓慢，说明模型逐渐收敛。从混淆矩阵上看，模型的表现整体较好，验证集上的最佳准确率为 89.58%。该模型 DenseNet121 是本次实验中性能表现最好的模型，在验证集上的准确率能够达到接近 90%。其余结果信息综合在表3所示。

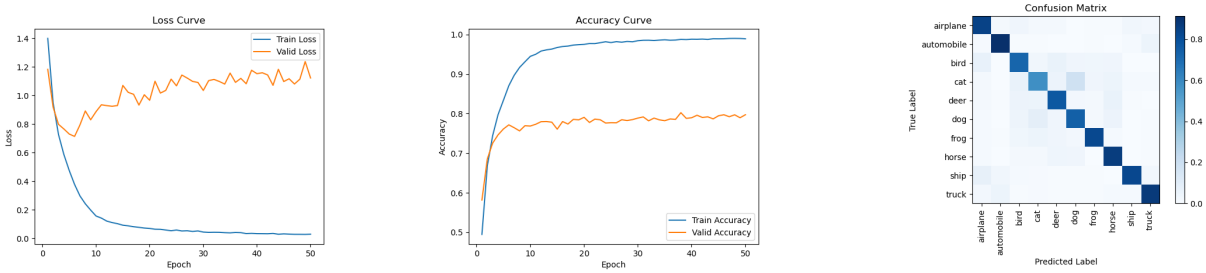


(a) 损失曲线 (b) 准确率曲线 (c) 混淆矩阵

图 5.6: DenseNet Model 结果

5.2.4 MobileNet

可以看到，MobileNet 模型在第 10 个 Epoch 左右达到临界点，在此之前训练集与验证集的损失共同下降，在该点之后训练集的损失虽然仍逐渐降低，但是验证集的损失并没有明显降低，甚至不降反升，说明此后模型发生**过拟合现象**。而训练集上的损失在第 20 个 Epoch 之后下降比较缓慢，说明模型逐渐收敛。从混淆矩阵上看，模型的表现整体较好，验证集上的最佳准确率为 79.73%。其余结果信息综合在表3所示。



(a) 损失曲线 (b) 准确率曲线 (c) 混淆矩阵

图 5.7: MobileNet Model 结果

5.2.5 对比总结

实验结果汇总如表3所示。

表 3: 实验结果

Model	临界 Epoch	Accuracy
CNN Base	10	64.19%
ResNet-18	5	77.98%
ResNet-50	20	76.83%
DenseNet121	10	89.58%
MobileNet	10	79.73%

由实验结果可知,本次实验中在验证集上的性能表现 DenseNet121>MobileNet>ResNet-18≈ResNet-50>CNN Base。DenseNet121 的效果最好可能是因为其参数量最大、层数最深，因而模型的表达能力在这五个模型中最强；MobileNet 由于其引入了深度可分离卷积，在深度相同时具有更少的参数量和计算量，模型的训练和推理速度更快，适合于算力较差的嵌入式系统；ResNet 模型由于引入残差连接

结构，缓解了梯度消失问题和退化问题，在一定程度上随着层数的加深性能也会有一定的提高；以上模型均好于纯粹的卷积神经网络模型。

6 总结与体会

在本次实验中，笔者深入了解卷积神经网络的原理，并编程实现基础 CNN 网络以及现代经典的 CNN 模型（ResNet、DenseNet、MobileNet），进一步熟悉了 PyTorch 中的基本算子单元（卷积算子、激活函数、池化操作等）。

经过本次实验，笔者理解并掌握了以下深度学习技巧：

- 残差连接结构：可以用来缓解梯度消失以及退化问题，帮助网络向更深层发展。
- 稠密连接结构：采用拼接操作代替残差连接，可以得到更加稠密的网络结构。
- 深度可分离卷积：通过将常规的卷积分解为 Depthwise Convolution 和 Pointwise Convolution 操作，可以减少参数量和计算量，适用于嵌入式系统。

笔者在实验过程中遇到的最重要的问题是过拟合问题，可以通过**提前停止**、**正则化方法**（批归一化 BatchNorm、Dropout 等）来缓解。

参考文献

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

[2] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

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

[4] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[5] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[6] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[7] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. Cambridge University Press, 2023. https://D2L.ai.

附录 A 部分代码

代码 2: ResNet 前向传播

```python
def forward(self, x):
    """
    ResNet Forward Pass

    :param x: input tensor of shape (N, C, H, W)
    :return: output tensor of shape (N, num_classes)
    """
    out = F.relu(self.bn1(self.conv1(x)))
    out = self.max_pool(out)
    out = self.layer1(out)
    out = self.layer2(out)
    out = self.layer3(out)
    out = self.layer4(out)
    # out = F.avg_pool2d(out, 4)  # For CIFAR-10, the output shape has already been 1x1
        ↪ before this avg_pool layer
    out = out.view(out.size(0), -1)
    out = self.linear(out)
    return out
```

代码 3: 构建不同深度的 ResNet

```python
def ResNet(num, num_classes=10):
    if num == 18:
        return _ResNet(BasicBlock, [2, 2, 2, 2], num_classes)
    elif num == 34:
        return _ResNet(BasicBlock, [3, 4, 6, 3], num_classes)
    elif num == 50:
        return _ResNet(Bottleneck, [3, 4, 6, 3], num_classes)
    elif num == 101:
        return _ResNet(Bottleneck, [3, 4, 23, 3], num_classes)
    elif num == 152:
        return _ResNet(Bottleneck, [3, 8, 36, 3], num_classes)
    else:
        raise NotImplementedError
```

附录 B ResNet Bottleneck

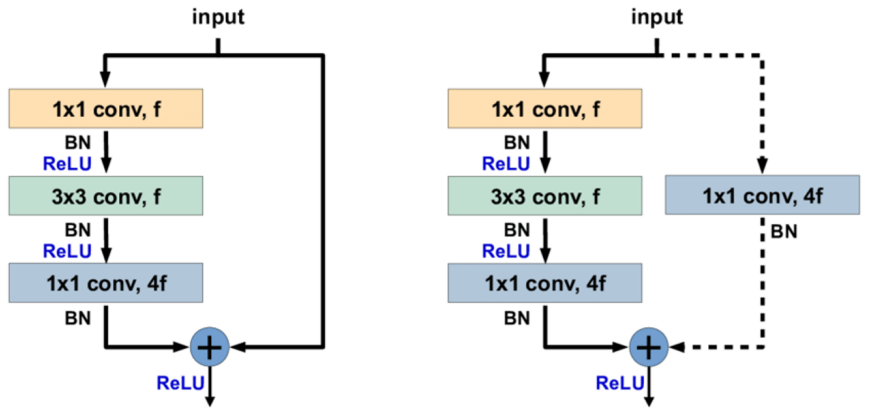


图 B.8: ResNet 的 Bottleneck 块结构 [1]

附录 C 深度可分离卷积

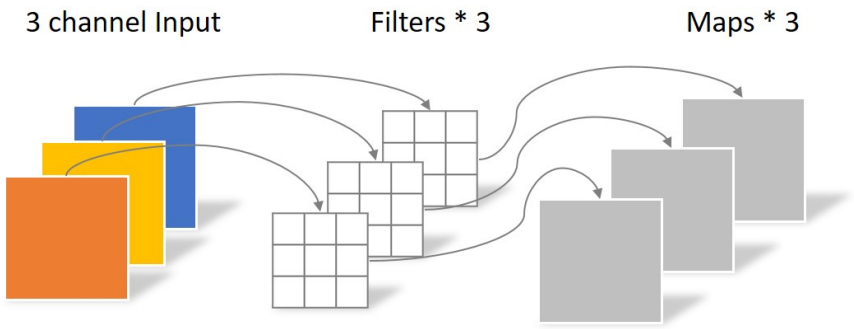


图 C.9: 深度卷积

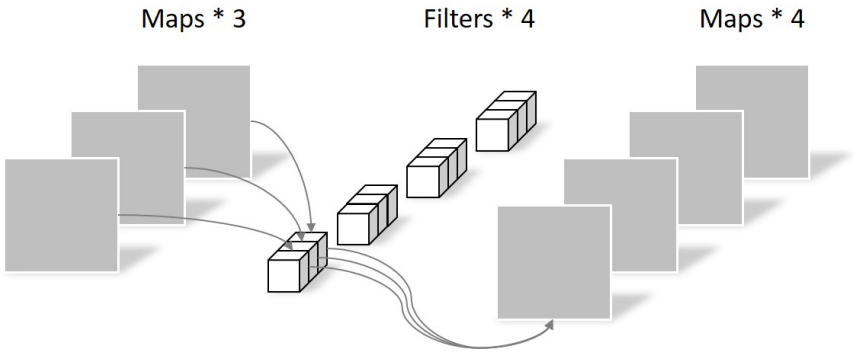


图 C.10: 逐点卷积