南开大学

Nankai University

南　　开　　大　　学

计　算　机　学　院

深度学习与应用实验报告

CNN 卷积神经网络

艾明旭　2111033

年级：2021 级

专业：信息安全

指导教师：侯淇彬

2024 年 4 月 8 日

目录

一、 jittor 分类

jittor 是一个很好的深度学习库，是少有的国内的可以用来作为深度学习框架进行训练的。本次实验我用 jittor 对 cifar 数据集实现了分类。

例如，我想要实现对车，轮船，鼠的分类：按照相应的方式设计网络

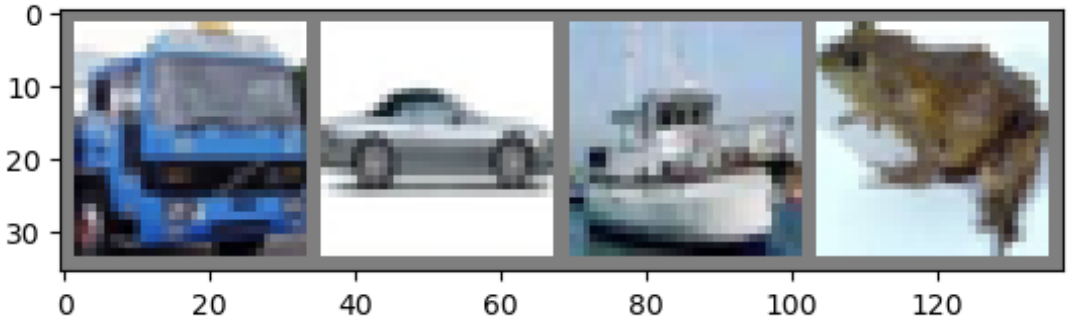

图 1: jittor 分类

```
class Net(nn.Module):
def __init__(self):
    super().__init__()
    self.conv1 = nn.Conv2d(3, 6, 5)  # kernel_size=5, padding=2, stride
        =1
    self.pool = nn.MaxPool2d(2, 2)
    self.conv2 = nn.Conv2d(6, 16, 5)
    self.fc1 = nn.Linear(16 * 5 * 5, 120)
    self.fc2 = nn.Linear(120, 84)
    self.fc3 = nn.Linear(84, 10)
def execute(self, x):
    x = self.pool(nn.relu(self.conv1(x)))
    x = self.pool(nn.relu(self.conv2(x)))
    x = jt.flatten(x, 1) # flatten all dimensions except batch
    x = nn.relu(self.fc1(x))
    x = nn.relu(self.fc2(x))
    x = self.fc3(x)
    return x
```

训练后，得到以下的图像:

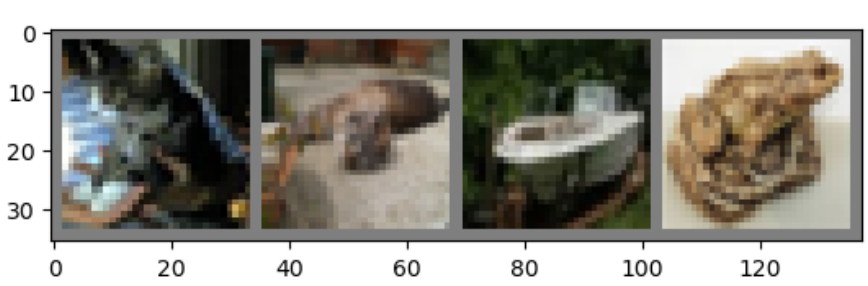

图 2: jittor 训练

二、 基础 CNN 结构

(一) 基础 CNN 的网络结构

```
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)
)
```

基础的 MLP 网络为一个三层的网络，其映射关系如上面的结构所示。

(二) 图形展示实验结果

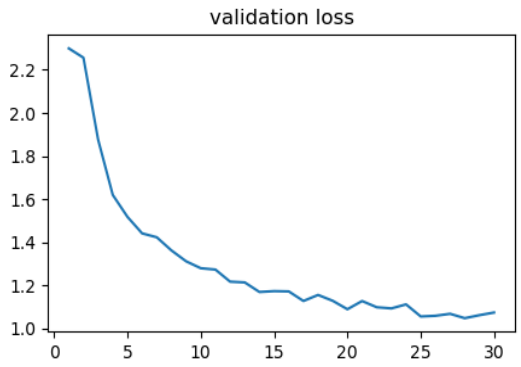

图 3: 基础 CNN validation loss

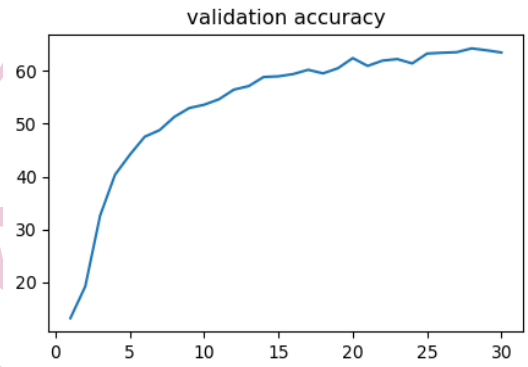

图 4: 基础 CNN validation accuracy

三、 ResNet

(一) ResNet 的实现

ResNet（Residual Neural Network）是一种深度神经网络结构，由微软研究院的研究员提出，旨在解决深度神经网络训练过程中的梯度消失和梯度爆炸等问题。ResNet 通过引入残差模块（Residual Block）来构建深层网络，使得网络可以更深更容易训练。

在 ResNet 中，残差模块通过增加跨层连接（skip connection）来实现残差学习。传统的神经网络会将输入信号传递到激活函数，而在 ResNet 中，跨层连接使得输入信号可以直接绕过若干层的非线性变换，从而保留输入信号的信息，这样有助于减轻梯度消失的问题，使得网络更容易训练。

```
class ResNet(nn.Module):
 def __init__(self, block, num_block, num_classes=10):
     super().__init__()
     self.in_channels = 64
```

```
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True))
        self.conv2_x = self._make_layer(block, 64, num_block[0], 1)
        self.conv3_x = self._make_layer(block, 128, num_block[1], 2)
        self.conv4_x = self._make_layer(block, 256, num_block[2], 2)
        self.conv5_x = self._make_layer(block, 512, num_block[3], 2)
        self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512 * block.expansion, num_classes)
    def _make_layer(self, block, out_channels, num_blocks, stride):
        strides = [stride] + [1] * (num_blocks - 1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_channels, out_channels, stride))
            self.in_channels = out_channels * block.expansion
        return nn.Sequential(*layers)
    def forward(self, x):
        output = self.conv1(x)
        output = self.conv2_x(output)
        output = self.conv3_x(output)
        output = self.conv4_x(output)
        output = self.conv5_x(output)
        output = self.avg_pool(output)
        output = output.view(output.size(0), -1)
        output = self.fc(output)
        return output
```

(二) 结果图像展示

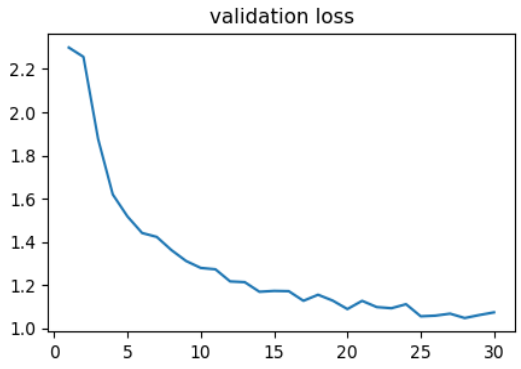

图 5: ResCNN validation loss

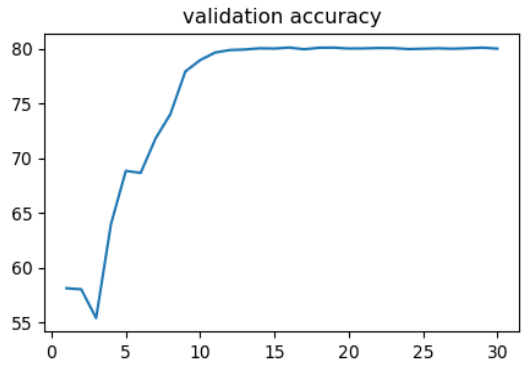

图 6: ResCNN validation accuracy

可以看到，在引入残差块之后，整个深度学习的网络层次都发生了一定的变化。

四、 DenseNet

(一) DenseNet 的实现

相比 ResNet，DenseNet 提出了一个更激进的密集连接机制：即互相连接所有的层，具体来说就是每个层都会接受其前面所有层作为其额外的输入。ResNet 是每个层与前面的某层（一般是 2-3 层）短路连接在一起，连接方式是通过元素级相加。而在 DenseNet 中，每个层都会与前面所有层在 channel 维度上连接在一起，并作为下一层的输入。对于一个 L 层的网络，DenseNet 共包含 $L(L+1)/2$ 个连接，相比 ResNet，这是一种密集连接。而且 DenseNet 是直接 concat 来自不同层的特征图，实现特征重用，提升效率，这一特点是 DenseNet 与 ResNet 最主要的区别。

编写的网络结构代码如下：

```
class _DenseLayer(nn.Module):
def __init__(
    self,
    num_input_features: int,
    growth_rate: int,
    bn_size: int,
    drop_rate: float,
    memory_efficient: bool = False
) -> None:
    super(_DenseLayer, self).__init__()
    self.norm1: nn.BatchNorm2d
    self.add_module('norm1', nn.BatchNorm2d(num_input_features))
    self.relu1: nn.ReLU
    self.add_module('relu1', nn.ReLU(inplace=True))
    self.conv1: nn.Conv2d
    self.add_module('conv1', nn.Conv2d(num_input_features, bn_size *
     growth_rate, kernel_size=1, stride=1, bias=False))
    self.norm2: nn.BatchNorm2d
    self.add_module('norm2', nn.BatchNorm2d(bn_size * growth_rate))
    self.relu2: nn.ReLU
    self.add_module('relu2', nn.ReLU(inplace=True))
    self.conv2: nn.Conv2d
    self.add_module('conv2', nn.Conv2d(bn_size * growth_rate, growth_rate
        ,kernel_size=3, stride=1, padding=1, bias=False))
    self.drop_rate = float(drop_rate)
    self.memory_efficient = memory_efficient
```

(二) 结果图形展示

五、 带有 Depthwise conv 的 MobileNets

(一) MobileNets 的实现

Mobilenets 的最主要思想是利用 depthwise 卷积来构建轻权重的深度网络。常规的卷积（标准卷积）是把每一个卷积核作用在输入图像的所有通道，并将他们整合在一起，形成一个通道输出。假设输入图像大小为 $C*H*W$，卷积核为 $N*C*sz*sz$，则输出图像为 $N*H*W$。计

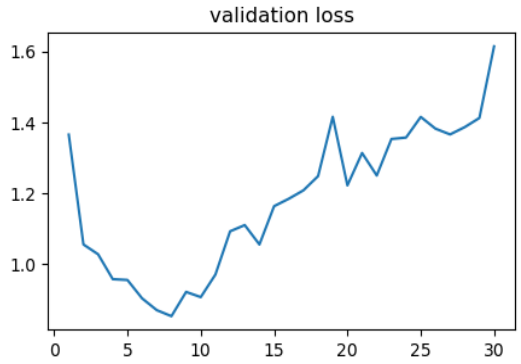

图 7: DenseCNN validation loss

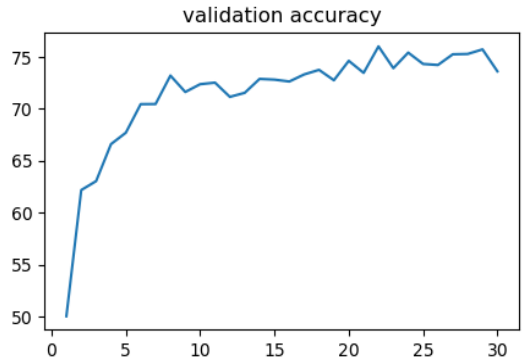

图 8: DenseCNN validation accuracy

算花费 $N*C*sz*sz*H*W$。mobilenets 中只有第一层用了标准卷积，其余层都 depthwise 卷积 +pointwise 卷积，也就是 depthwise separable convolution。

Depthwise 卷积与标准卷积不同，不是把输入通道整合在一起，而是分别对每一个通道进行操作。每个通道都有自己的权重，卷积核的通道数与输入图像的通道数一致。假设输入图像为 $C*H*W$，卷积核为 $C*sz*sz$，则输出图像为 $C*H*W$。计算花费为 $C*sz*sz*H*W$。depthwise 卷积之后是 pointwise 卷积，本质上是 1x1 卷积，也就是将所有的输入通道对应位置加权求和。Pointwise 卷积的目的就是整合 depthwise 卷积中所有输出通道来创建新特征。假设输入图像为 $C*H*W$，卷积核为 $N*1*1$，则输出图像为 $N*H*W$。计算花费为 $N*C*H*W$。

depthwise 卷积 +pointwise 卷积计算花费为 $C*sz*sz*H*W+N*C*H*W$。效率上，3x3 卷积中，depthwise 卷积 +pointwise 卷积比标准卷积大约快 9 倍

```
class MobileNet(nn.Module):
def __init__(self, num_classes=10):
    super().__init__()
    self.conv1 = nn.Conv2d(3, 32, 3, 2, 1, bias=False)
    self.dwconv1 = DepthwiseSeparableConv(32, 32, 3, 1, 1)
    self.dwconv2 = DepthwiseSeparableConv(32, 64, 3, 2, 1)
    self.dwconv3 = DepthwiseSeparableConv(64, 128, 3, 1, 1)
    self.dwconv4 = DepthwiseSeparableConv(128, 128, 3, 2, 1)
    self.dwconv5 = DepthwiseSeparableConv(128, 256, 3, 1, 1)
    self.dwconv6 = DepthwiseSeparableConv(256, 256, 3, 2, 1)
    self.dwconv7 = DepthwiseSeparableConv(256, 512, 3, 1, 1)
    self.dwconv8 = DepthwiseSeparableConv(512, 512, 3, 2, 1)
    self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
    self.fc = nn.Linear(512, num_classes)
```

(二) 结果图形展示

六、 总结

卷积神经网络是三维的网络，具有更高的准确率。ResNet，DenseNet 和 MobileNet 都是深度学习中的卷积神经网络（CNN）架构，它们在设计和应用场景上有所不同。

ResNet（残差网络）：ResNet 的主要特点是引入了“残差块”，通过跳跃连接解决了深度神经网络中的梯度消失和表示瓶颈问题。这使得网络可以安全地增加深度以提高性能。ResNet 在

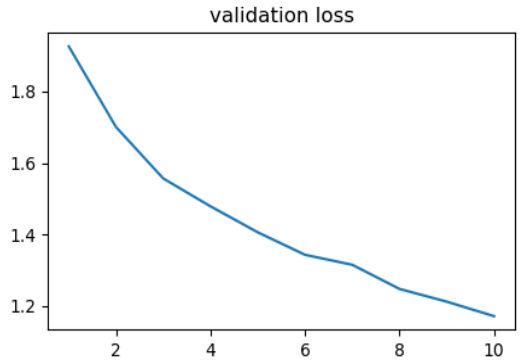

图 9: MobileCNN validation loss

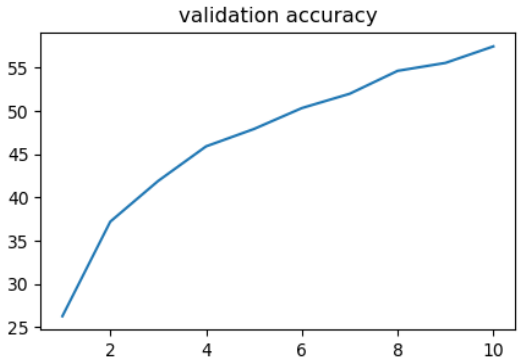

图 10: MobileCNN validation accuracy

图像分类、物体检测和语义分割等任务中表现出色。

DenseNet（密集连接网络）：DenseNet 的主要特点是每个层都直接连接到后面的每个层（称为密集连接）。这种设计改善了信息流，使得网络可以使用较少的参数和计算量达到相同的性能。DenseNet 在图像分类和语义分割等任务中表现良好。

MobileNet：MobileNet 的主要特点是使用深度可分离卷积，这大大减少了计算量和模型大小，使其适合在资源受限的设备（如移动设备和嵌入式系统）上运行。MobileNet 在图像分类、物体检测和人脸识别等任务中表现良好，尤其适合需要轻量级模型的场景。

在结果优劣方面，ResNet 和 DenseNet 在大多数任务中可以达到最先进的性能，但它们的模型大小和计算量较大。相比之下，MobileNet 的性能可能稍逊一筹，但其模型大小和计算量显著减少，使其在资源受限的设备上运行更为高效。

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。MobileNet 则是采用了两种卷积方式混合使用的方式进行训练。ResNet 中跳跃连接是从输入到输出直接跨越层的边，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。

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。MobileNet 利用了降维，使得训练的次数也很少，因此参数数量是最少的。

本次实验当中，我对卷积神经网络有了更加深刻的理解，也能够编写出相应的代码，希望可以在未来的学下当中继续进步，在深度学习这门课当中学习到更多有用而充满挑战性的知识。