山东大学

崇新学堂

2023－2024 学年第一学期

实 验 报 告

课程名称: 信息基础 II

专 业 班 级 崇新学堂 21 级

学 生 姓 名 刘浩

个 人 学 号 202120120312

实 验 名 称 ResNet-18 实现 CIFAR-10 图像分类

实验二：ResNet-18 实现 CIFAR-10 图像分类

一、 实验要求

搭建 ResNet-18 网络,自己编程实现网络结构，实现 CIFAR-10 数据集分类

二、 实验原理

首先是为何要引入 ResNet-18 残差网络:

最初人们的认知中网络越深那么准确率就越高，在网络逐步加深的过程中，实际上深层次的神经网络难以做到恒等映射，随着网络层数的加深，激活函数越来越多，实现的非线性变换也越来越多，所以对于简单的恒等映射是很难做到的，因而出现了网络退化现象，为了解决这个问题，引入了残差网络。

ResNet 最重要的思想是引入了残差块，允许某一层的输出直接跳过一个或多个层，连接到后续层的输入。这样做的好处是，即使某些层不做任何有意义的变换，它们仍然可以传递之前层的信息，而不会对梯度产生过多的损失。

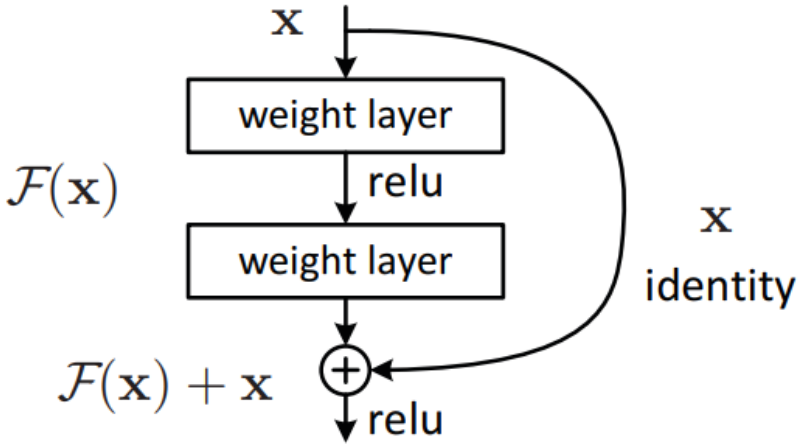


图 1 残差块示意图

其中论文中给出了两种残差块的结构，如图 2 所示：

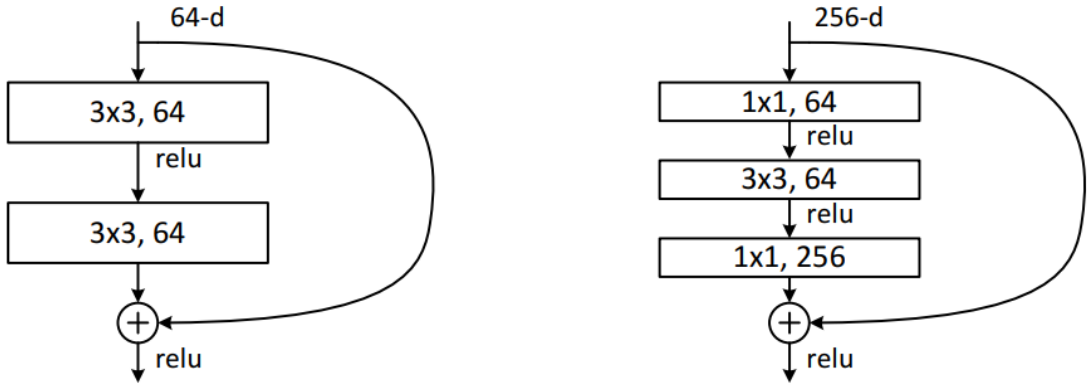


图 2 残差块结构

对于 ResNet-18 来说采用的是左边的形式，具体的结构如下所示：

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				
		[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	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 3 ResNet-X 结构

ResNet-18 采用的是图中红色框出的结构，在本实验中针对 CIFAR-10 数据集的 ResNet-18 如下图所示：

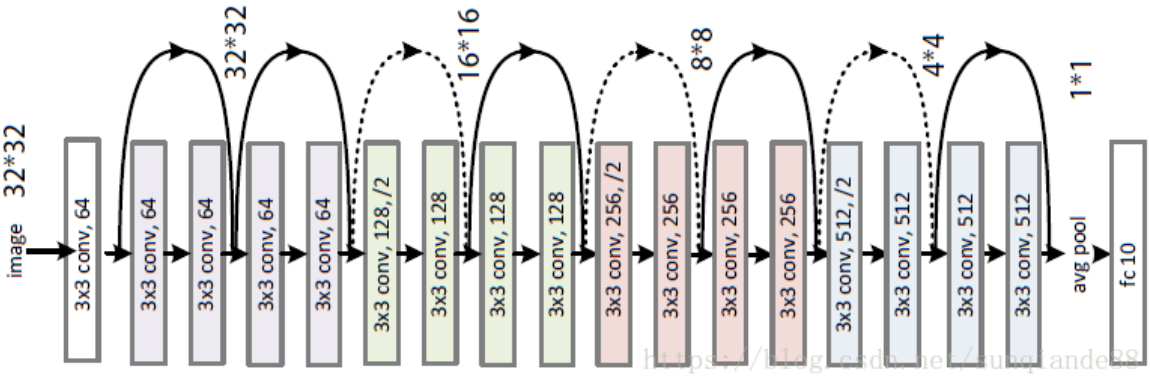


图 4 ResNet-18 结构

三、 实验步骤

首先数据集的导入和下载，采用 pytorch 内置的 CIFAR-10 数据集，我们只需要调用即可，我选择的 batch_size 大小是 128，即每次处理 128 张图片

同时，我对数据集进行了标准化，更好的提高了训练的准确率：

```
def read_data(batch_size):
    transform_train = transforms.Compose([
        transforms.RandomCrop(32, padding=4),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    ])
    transform_test = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    ])

    data_train = datasets.CIFAR10(
        root="./data",
        train=True,
        download=False,
        transform=transform_train
```

```
    )
    data_test = datasets.CIFAR10(
        root="./data",
        train=False,
        download=False,
        transform=transform_test,
    )

    dataloader_train = DataLoader(data_train, batch_size=batch_size, shuffle=True, num_wo
rkers=4)
    dataloader_test = DataLoader(data_test, batch_size=batch_size, shuffle=False, num_wor
kers=4)

    return dataloader_train, dataloader_test
```

下面就是网络搭建的过程

首先是残差块的搭建，按照图 2 左边的残差块结构搭建，值得注意的是恒等映射的构造，需要对于初始 identity 与输出维度不匹配的时候进行维度对齐操作，这样才可以相加。

```
# 搭建 block 部分
class Block(nn.Module):
    def __init__(self, inp_channel, out_channel, stride=1):
        super(Block, self).__init__()
        # 此处 bias 设 false 是为了避免和后面 BN 层的 bias 冲突
        self.conv1 = nn.Conv2d(inp_channel, out_channel, kernel_size=3, stride=stride, pa
dding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_channel)
        self.relu = nn.ReLU()
        self.conv2 = nn.Conv2d(out_channel, out_channel, kernel_size=3, stride=1, padding
=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channel)
        self.shortcut = nn.Sequential()
        # 维度对齐
        if stride != 1 or inp_channel != out_channel:
            self.shortcut = nn.Sequential(
                nn.Conv2d(inp_channel, out_channel, kernel_size=1, stride=stride, bias=Fa
lse),
                nn.BatchNorm2d(out_channel)
            )

    def forward(self, x):
        identity = x

        out = self.conv1(x)
        out = self.bn1(out)
        out = self.conv2(out)
        out = self.bn2(out)

        out = out + self.shortcut(identity)
        out = self.relu(out)

        return out
```

由于残差Block已经定义好，那么我们可以利用该Block进一步构建我们的ResNet，首先我定义了在实验原理中我们已经知道了 ResNet 由很多的 Block 构成，因此我构建了一个 make_layer 函数进行 Block 的倍增操作，使得 ResNet 的构造更加简单

```
# 此函数用来方便增加层数
def make_layer(self, num_block, inp_channel, out_channel, stride):
    net = []
    net.append(Block(inp_channel, out_channel, stride))

    for u in range(1, num_block):
        net.append(Block(out_channel, out_channel, stride))
    net = nn.Sequential(*net)
```

```
    return net
```

进而我们可以构造 ResNet-18 如下：

```
class ResNet18(nn.Module):
    def __init__(self, num_class):
        super(ResNet18, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=True)
        self.bn1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU()
        self.maxpooling = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        self.layer1 = self.make_layer(2, 64, 64, stride=1)

        self.layer2 = self.make_layer(2, 64, 128, stride=2)

        self.layer3 = self.make_layer(2, 128, 256, stride=2)
        self.layer4 = self.make_layer(2, 256, 512, stride=2)

        self.avgpooling = nn.AdaptiveAvgPool2d(1)
        self.flatten = nn.Flatten()
        self.fc = nn.Linear(512, num_class)
```

然后实例化 ResNet 网络，让损失函数为交叉熵函数，优化器为 SGD→随机梯度下降的方式 ,值得注意的是原始 ResNet 是针对 ImageNet 数据集的图，而 CIFAR-10 数据集的图片大小只有 **32*32**，为此我们再采用 **kernel_size = 7** 会丢失很多信息，修改为 **kernel_size=3**，进而完成模型的设置。

```
dataloader_train, dataloader_test = read_data(batch_size=256)
num_class = 10
model = ResNet18(num_class)

# 由于 ResNet 是针对 ImageNet 的，而 CIFAR-10 数据集图片较小，修改网络结构，保留更多信息
model.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False)

model = model.to(device)
# 损失函数
lossfunc = nn.CrossEntropyLoss()
# 优化器
optimizer = torch.optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
```

训练过程和测试过程完全使用上次实验 LeNet5 的即可，此处不再赘述

四、 实验结果

训练 200 轮实验结果:

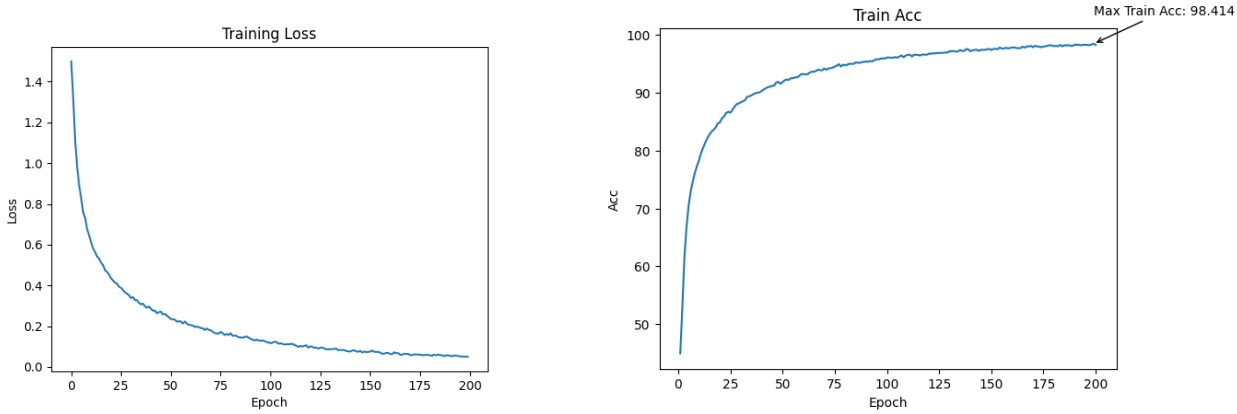


图 5 训练误差和训练准确率

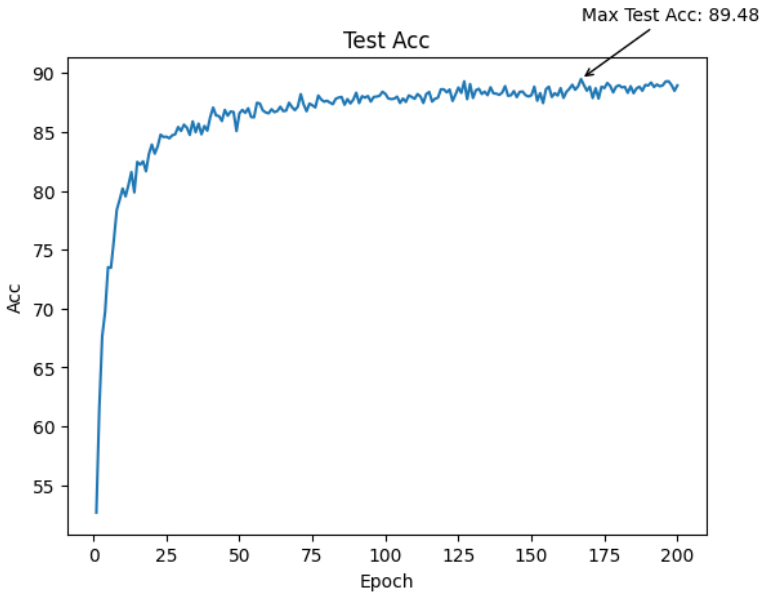


图 5 测试集准确率

最终模型的测试集准确率为 **89.48%**

在我写的测试模型精度的函数上进行测试(即从**测试集中选图片真正进行预测**),结果也是相当不错的：

五、 实验探究

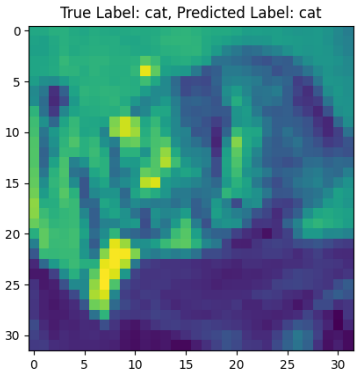
5.1 真实测试训练模型性能

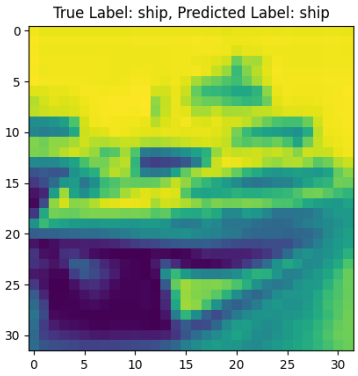
和 LeNet5 类似，在本实验中我也对测试数据集挑选图片进行测试，加载训练好的模型后，直接进行测试即可：

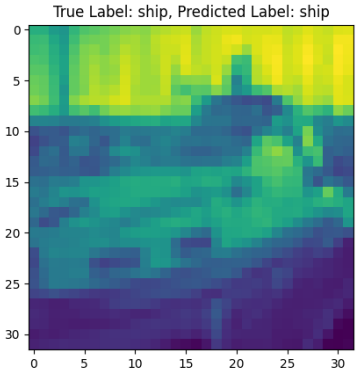
```
def test(model, test_num, test_loader):
    model.eval()
    count = 0  # 记录已处理的样本数量
    classes = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', '
ship', 'truck']
    output_folder = 'Test'
    os.makedirs(output_folder, exist_ok=True)
    with torch.no_grad():
        for i, (X, y) in enumerate(test_loader):
            if count >= test_num:
                break

            batch_size = X.size(0)  # 当前批次中的样本数量
            for j in range(batch_size):
                img_true, label = X[j][0].numpy(), y[j].item()
                X_batch = X.to(device)

                pred = model(X_batch)
                y_pred = torch.argmax(pred, dim=1)[j].item()

                true_label = classes[label]
                predicted_label = classes[y_pred]

                print("-----------验证模型性能开始------------")
                print("预测结果：", true_label)
                print("真实标签：", predicted_label)
```

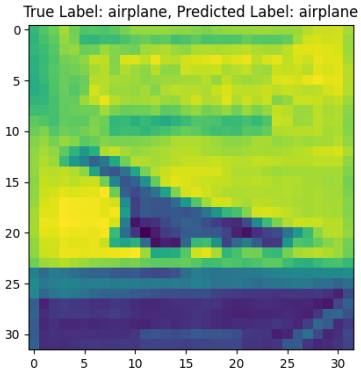
```
                # 显示图像
                plt.imshow(img_true)
                plt.title(f"True Label: {true_label}, Predicted Label: {predicted_label}"
)
                plt.savefig(os.path.join(output_folder, f"预测结果{count}.png"))
                # plt.show()

                count += 1
                if count >= test_num:
                    break
```

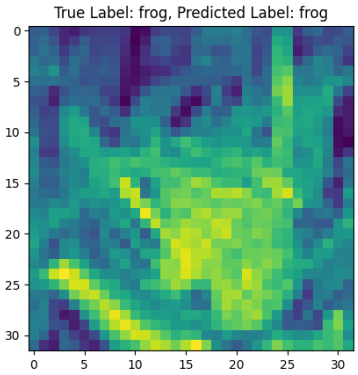
结果如下：

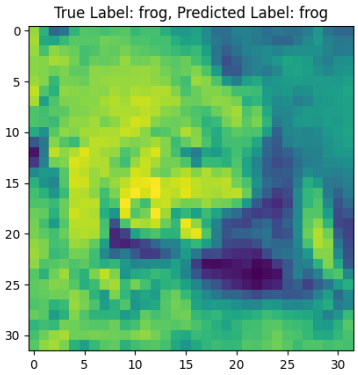












5. 2 调整网络结构策略

修改网络结构，**去掉**最大池化层，同样训练 200 轮的基础上，得到的结果如下：

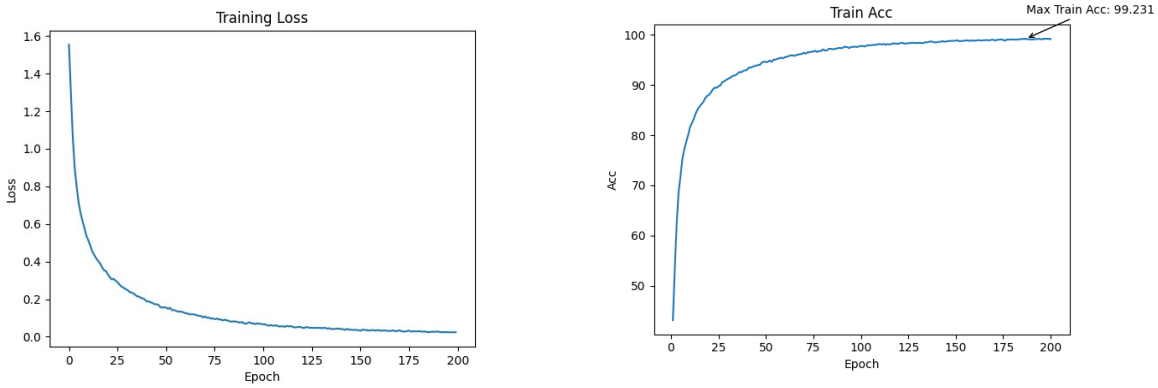


图 7 修改网络结构后的训练结果

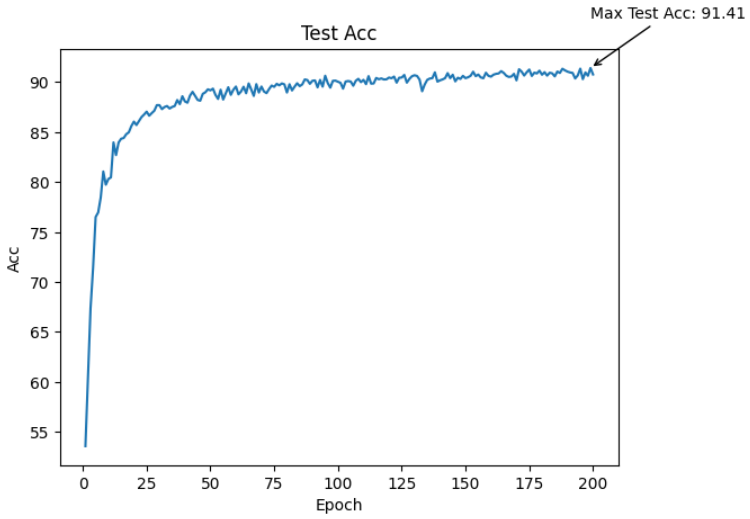


图 8 修改网络结构的测试集准确率

与原始的 ResNet-18 对比,准确率有所提高：

	最高准确率
原始 ResNet-18	89.48%
修改网络结构的 ResNet-18	**91.41%**

六、 实验感想

本次实验的残差网络有效的解决了深层网络产生的网络退化现象，在这个基础上我们理论上可以构建很深的神经网络去提高精确度。

在实验原理中我也提到了 ResNet 本身是针对 ImageNet 数据集的，在本实验中是对 CIFAR-10 数据集进行分类，图片大小由原来的 224*224 变成了 32*32，对网络结构我也做出了一定的修改，最终感受到了 ResNet 在深层网络中的作用，达到了不错的效果，在后续的很多工作中我们都可以继续采用 ResNet 的这种思想！

七、 参考资料

[1]. http://t.csdnimg.cn/p880q
[2]. http://t.csdnimg.cn/T8wtw
[3]. https://zhuanlan.zhihu.com/p/157134695