华中科技大学

计算机视觉实验报告

姓　　名：王逸

学　　院：计算机科学与技术

专　　业：计算机科学与技术

班　　级：CS2011

学　　号：U202014774

指导教师：刘康

分数	
教师签名	

2023 年　4 月　14　日

目　录

实验三 基于剪枝算法的深度神经网络压缩

1.1 实验要求

对实验二构建的 CIFAR-10 数据集分类神经网络进行权重剪枝实现模型压缩。

1.1.1 报告要求

1. 画出最后一层卷积层（剪枝前）在整个测试数据集上的平均输出特征图。
2. 画出横坐标为 K，纵坐标为网络分类 accuracy 的折线图。
3. 实验报告包含网络设计、实验结果图，以及必要的分析等。

1.1.2 实验提示：

1. 可通过对最后一层卷积层，按照输出特征图的神经元激活均值排序，按从小到大依次进行对前 K（自定）个神经元进行剪枝；
2. 剪枝后卷积层权重大小为 Dx3x3x(P-K)，测试此时神经网络的分类准确性；
3. 剪枝神经元时，可将待剪枝的神经元权重、偏置直接设为 0，这样可以不用改变网络结构。

1.2 实验内容

本实验是在实验二的基础上添加两个部分：剪枝与特征图输出。并将其应用到 ResNet18 和 LeNet 上，作为本次实验分析的样例。

A. 剪枝

本实验通过计算每次 epoch 测试时，最后一个卷积层的输出神经元在整个测试集上的平均值，按从小到大排列，依次将前 K 个神经元的权重设置为 0，从而实现神经网络的剪枝。

为了实现这部分内容，首先我们需要获取测试集在最后一层卷积层上的输出：(以最简单的 LeNET 为例, ResNet 网络改动可看/Net/resNet.py)

```
self.features = nn.Sequential(
    nn.Conv2d(in_channels=3, out_channels=6, kernel_size=5),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=2, stride=2),
    nn.Conv2d(6, 16, 5),
    nn.ReLU(True),
)
self.e_features = nn.Sequential(
    nn.MaxPool2d(2, 2),
)
self.classifier = nn.Sequential(
    nn.Linear(16 * 5 * 5, 120),
    nn.ReLU(True),
    nn.Linear(120, 84),
    nn.ReLU(True),
    nn.Linear(84, num_classes),
)
```

首先将原来的 features 分为两个 Sequential, 方便在 forward 中计算特征的和与平均。

之后在 forward 中添加一个参数 test_flag：判断当前是否是测试集输入模型，若是，则对当前 batch 的输出特征图（128，16x10x10）在 dim=1 上求平均得到平均特征图；将特征矩阵求和变为（16）的特征向量。

```
if test_flag:
    mean_x = torch.mean(x, 1)
    sum_x = torch.sum(x, 0)
    # print(sum_x.shape)
    sum_x = torch.sum(sum_x, dim=1)
    # print(sum_x.shape)
    sum_x = torch.sum(sum_x, dim=1)
```

在./main.py 的 test 函数中，对每次 batch 得到的 sum_feats 进行求和，测试结束后求平均得到神经元激活的平均值：

```
avg_feats.add_(sum_feats)

avg_feats = avg_feats/10000
```

然后定义剪枝函数 prune(model, avg_feats, prune_num)：

参数	意义
model	测试完成后得到的网络模型
avg_feats	最后一层的平均输出
prune_num	需要剪枝的神经元个数

对 avg_feats 进行排序，返回排序后对应的原来下标；再在最后一层的权重矩阵(16, 6, 5, 5)中将前 prune_num 个神经元的权重和偏置设置为 0。最后将参数重新载入模型。

```python
def prune(model, avg_feats, prune_num):
    # 排序
    sort_feats, sort_ind = torch.sort(avg_feats, descending=False)
    # 获取参数
    for name, param in model.named_parameters():
        print(name)
        if 'features.3.weight' in name:
            weights = param
        if 'features.3.bias' in name:
            bias = param

    weights = weights.detach().numpy()
    bias = bias.detach().numpy()
    # print(bias.shape)
    # print(sort_ind.shape)

    for i in range(prune_num):
        weights[sort_ind[i]:, ] = 0
        bias[sort_ind[i]:, ] = 0

    # 剪枝后重新加载到模型中
    for name, param in model.named_parameters():
        if 'fearues.3.weight' in name:
            param.data = torch.from_numpy(weights)
        if 'features.3.bias' in name:
            param.data = torch.from_numpy(bias)
```

B. 平均特征图输出

平均特征图的输出需要拿到最后一层卷积层的输出并在输出通道的维度上求平均。由于每个 batch 一次处理 128 张图片，所以每次平均后得到的图片大小为 128x(5x5)。因为 10000 张图片的数量过多，若要一次全部输出则没有太大的观察意义，所以只选取了前 128 张图片的平均特征图进行输出。

代码可见./utils.py，其中定义了特征可视化函数。

C. 训练

训练部分较上次并未作出太大改变，本报告中不过多赘述。

D. 画图分析

本次实验需要画出 acc-k 的折线图：

一是模型测试准确率随着剪枝神经元个数的变化曲线。该部分与实验一中类似，代码可见文件 draw_fig.py。

1.3 实验结果分析

a. LeNet

a) 平均特征输出图：每个特征图的大小为 10x10，选取前 128 张进行输出展示。每行 16 张图片，共 128/16 = 8 行。

编号	神经网络种类	学习率	迭代次数	Prune_num
1	LeNet	0.01	20	3
2		0.01	20	5
3		0.01	20	10
4		0.01	20	13

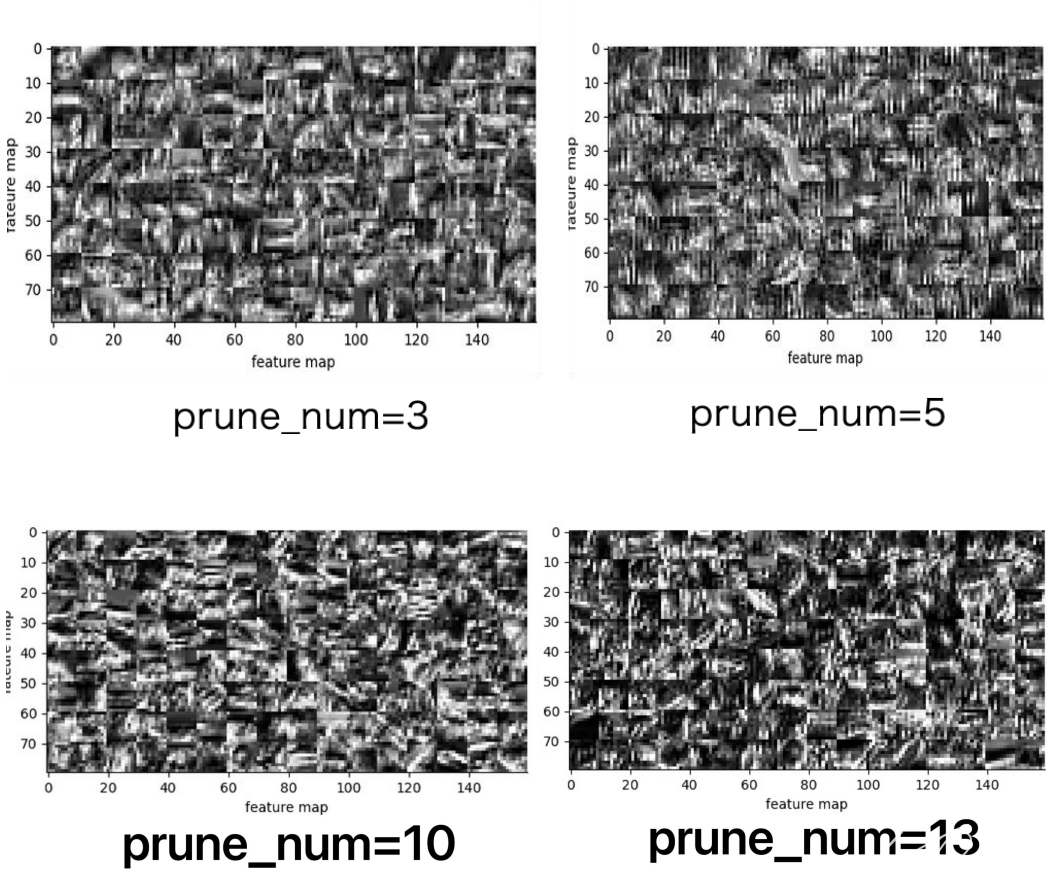
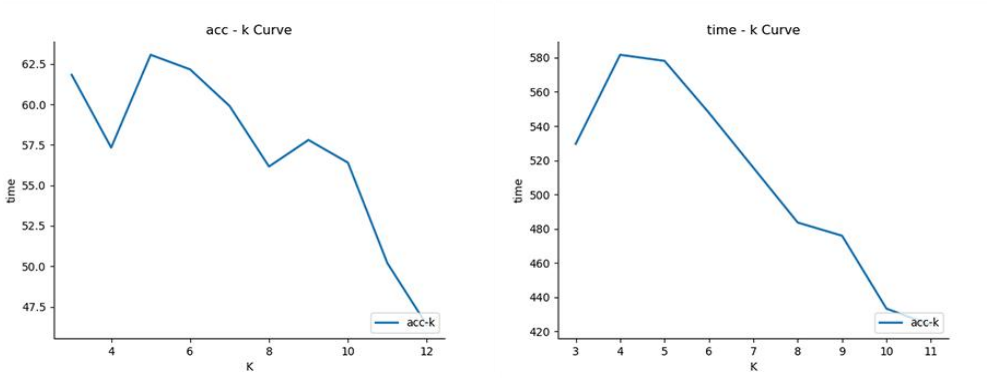


图 1 lr=0.01, epoch=40,LeNET

b) ACC-K\ TIME-K 曲线：

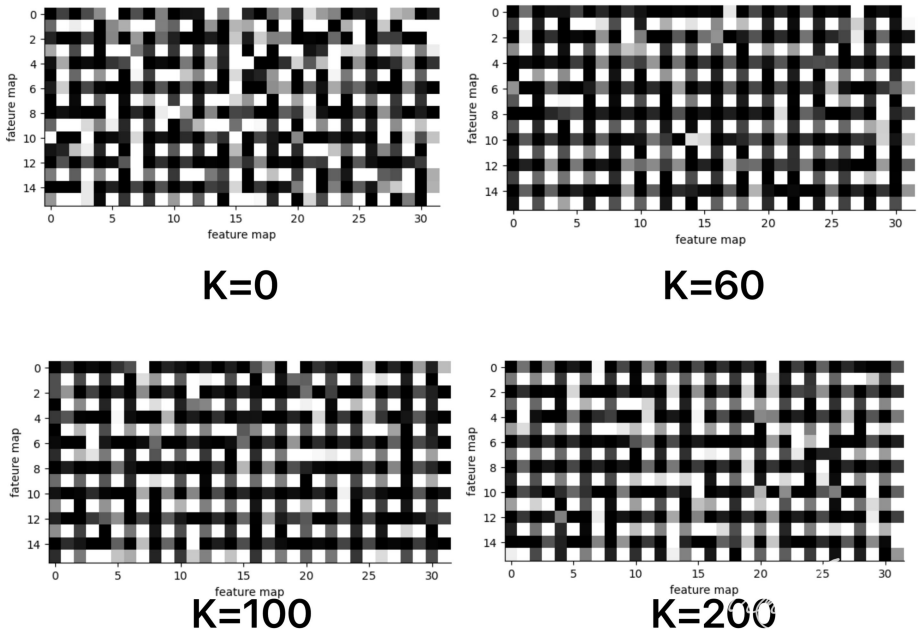


b. ResNet18:

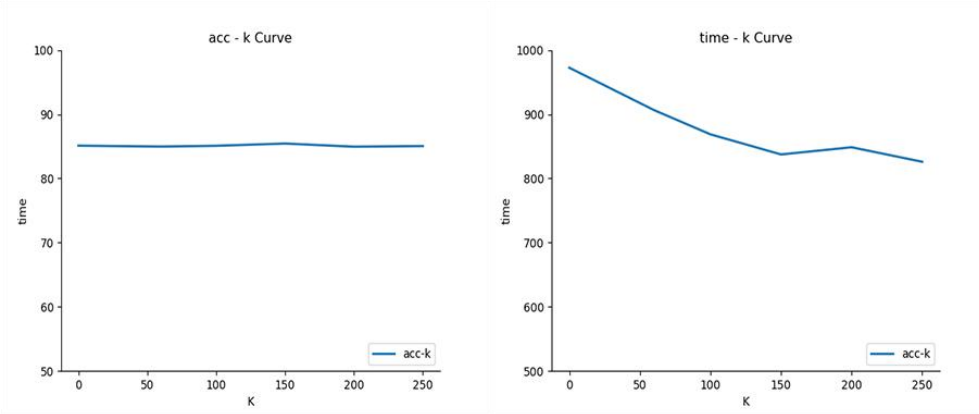
a) 平均输出特征图：输出特征个数为 512，每个特征图的大小为 2x2，选取前 128 张进行输出展示。每行 16 张图片，共 128/16 = 8 行。

编号	神经网络种类	学习率	迭代次数	Prune_num
1	ResNet	0.01	30	0
2		0.01	30	60
3		0.01	30	100

4		0.01	20	200



b) ACC-K\ TIME-K 曲线：



通过以上两组网络的实验，可以看出：

- 对于比较简单的神经网络，剪枝对于降低训练时间的作用不大，反而可能因为减少神经元个数而严重影响正确率；
- 对于稍微复杂一点、卷积层数多、神经元个数也多的网络，一般可以一次减少几十、甚至几百个神经元，在大幅减少运行时间的情况下，几乎不影响网络自身的准确率。
- 另外，我还可以从两组网络的最后一层卷积层的特征图看出，神经网络越往后训练，其提取的特征图就越抽象；而卷积核的大小与个数也会影响提取效果，ResNet 最后一层的特征图比 LeNet 更加抽象。

1.4 实验小结

本次实验过程中比较麻烦的是何时计算最后一层的输出平均值、权重矩阵的获取以及重载入。由于后面是通过 GPU 训练 ResNet，在模型参数改变重新载入时也需要将模型重新送到 GPU 上，否则会造成类型的不一致而报错。基于硬件以及时间的限制，这次实验还是选择了两个相对来说比较简单的网络进行对比，剪枝效果虽有但提升效果不尽如人意，之后会继续尝试 ResNet50 或者更为复杂的网络测试剪枝效果。