电子技术与系统 – 实验报告

Noflowerzzk

2025.5

分类	具体内容	完成情况
确保矩阵乘加速模块的功能正确（60%）		
	阵列的整体功能（20%）：通过 5 组测试数据	完成
	乘法器功能验证（20%）：通过测试集	完成
	加法器功能验证（20%）：通过测试集	完成
性能指标（40%）		
	硬件效率：所有同学的指标排序百分比	
加分项		
	扩展成动阵列以外的其他模块（∼20%）	无
	报告中完整的逻辑功能、面积、频率分析（∼10%）	部分
	对控制信号的分析、实现、验证和扩展（∼5%）	无
	输入数据的处理过程分析（∼5%）	无
减分项		
	未通过测试集	
	报告必备内容缺失	
	性能指标计算错误	

1 电路部分

1.1 电路结构设计

1.1.1 PE 单元的整体设计及电路图

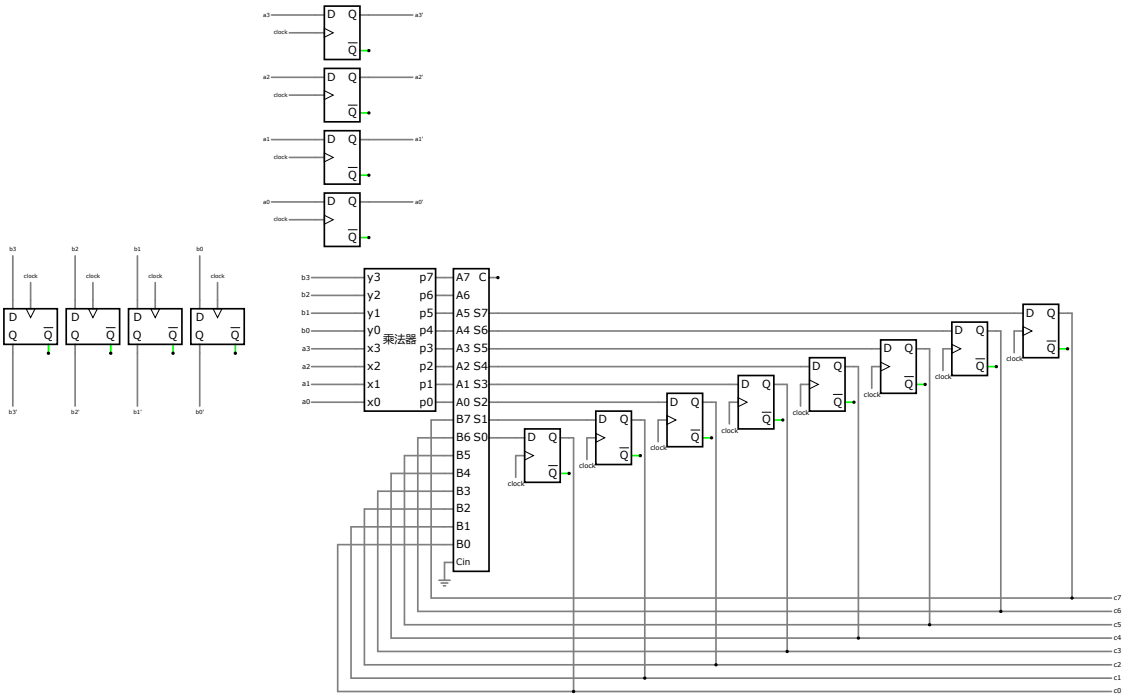

图 1: PE 单元电路结构设计

1.1.2 加法器电路结构设计

此处全加器采用 Liu's 半加器组合而成

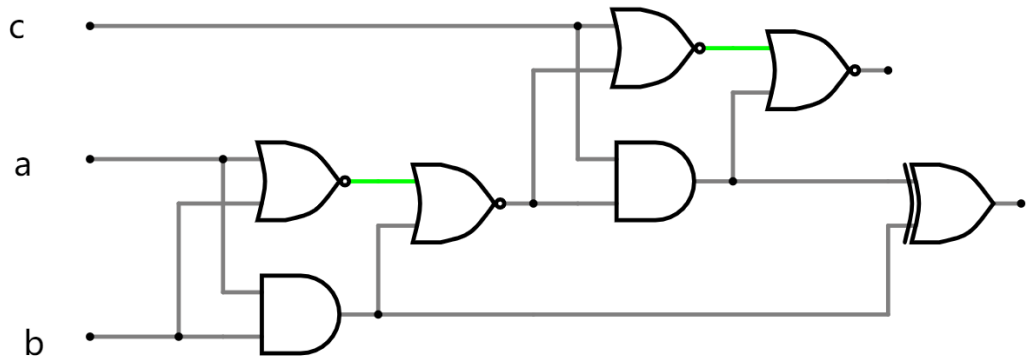

图 2: 全加器电路结构设计

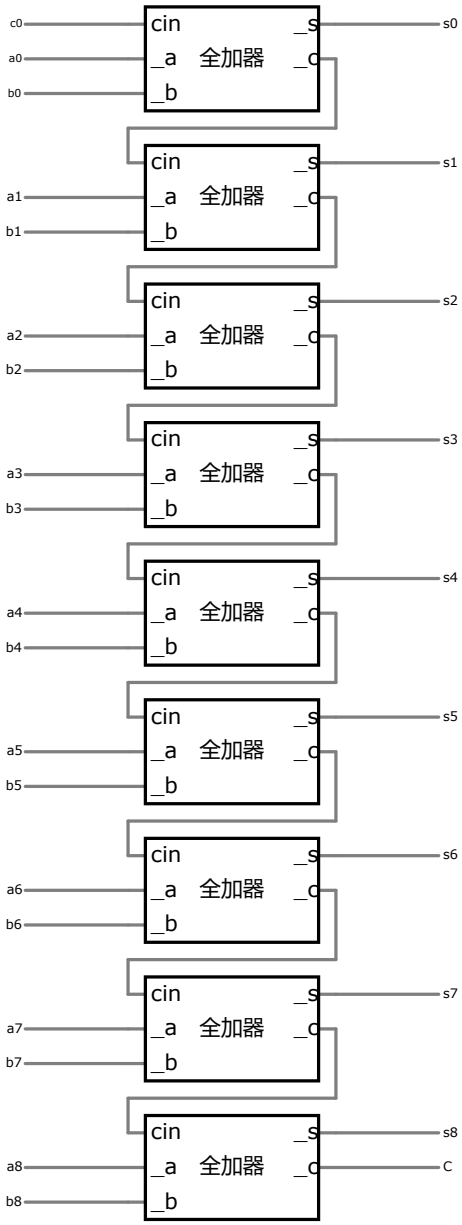

图 3: 加法器电路结构设计

1.1.3 乘法器电路结构设计

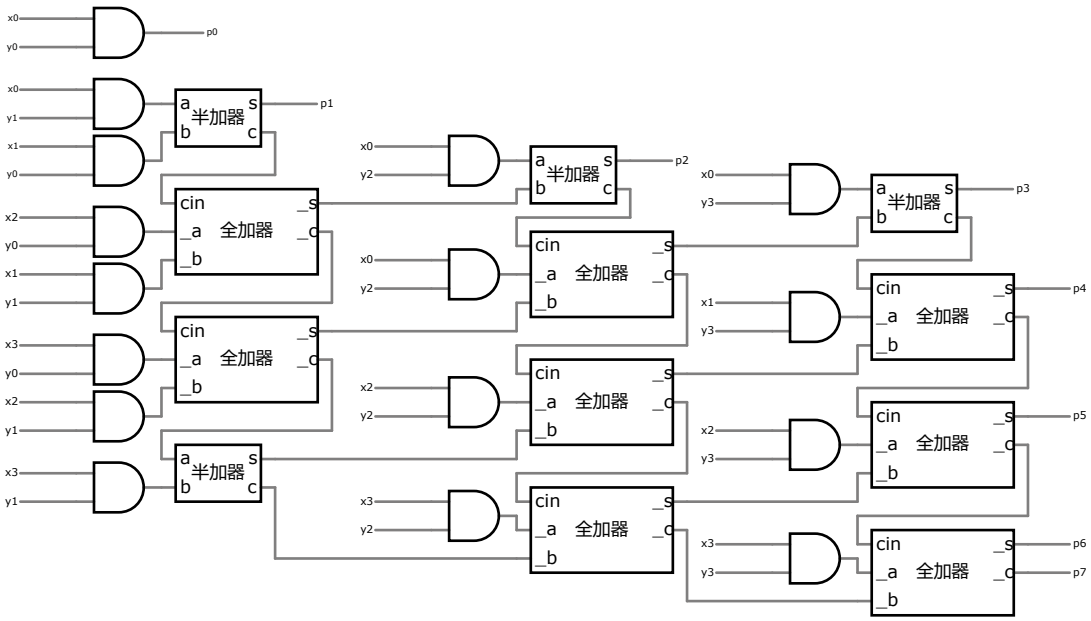

图 4: 乘法器电路结构设计

1.2 电路结构验证

1.2.1 PE array 功能验证

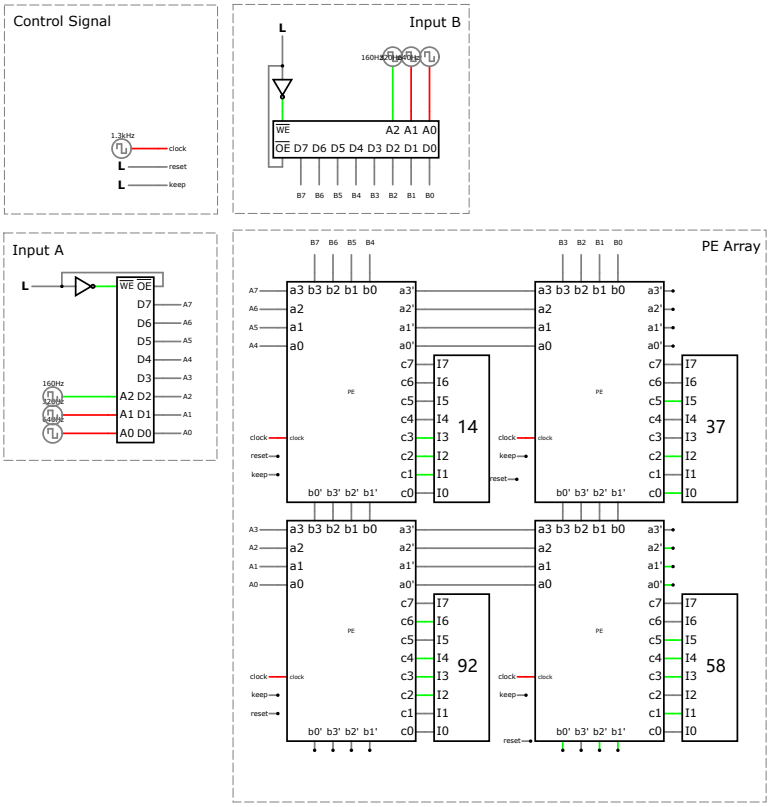

图 5: PE array 功能验证

1.2.2 乘法器功能验证

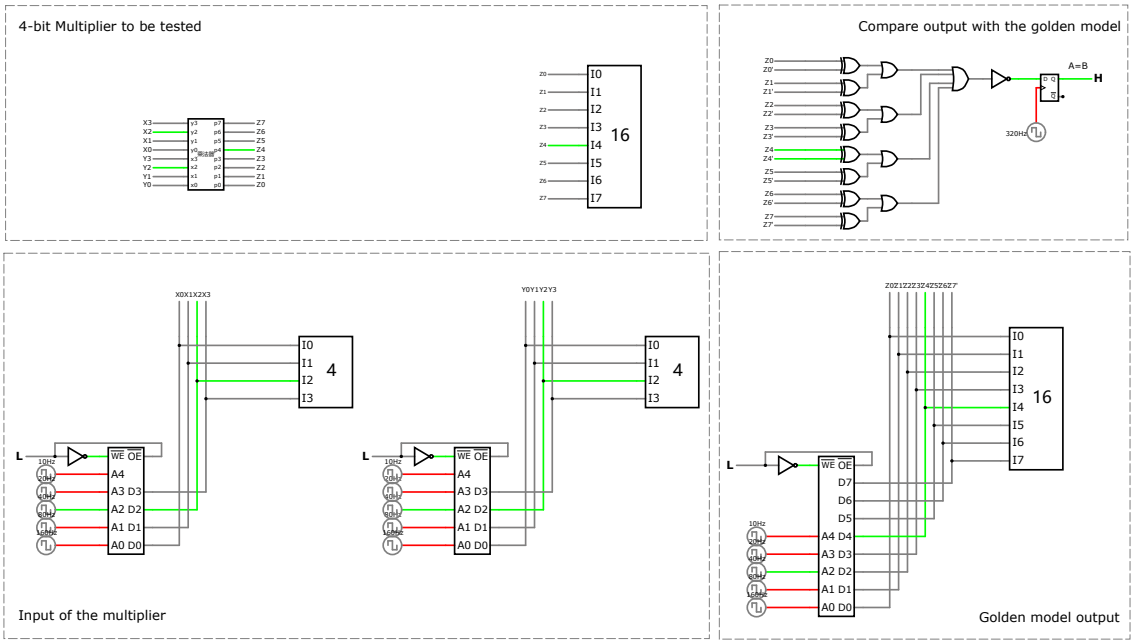

图 6: 乘法器功能验证

1.2.3 加法器功能验证

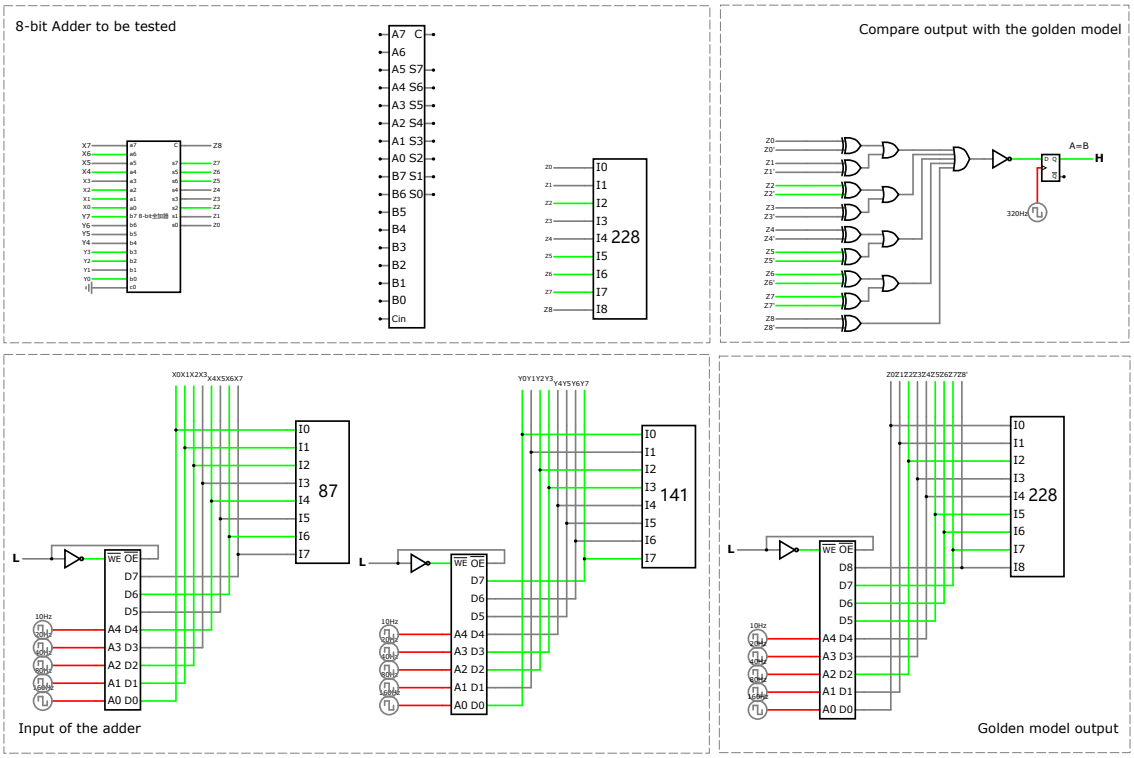

图 7: 加法器功能验证

1.3 电路指标评估

1.3.1 总面积开销

单个 PE 总面积开销为 148, 四个 PE 总面积开销为 592

1.3.2 时序性能

单个 PE 关键路径延迟为 6 单位，时钟周期 $T \geq 6$

1.3.3 硬件效率计算

单个效率为 $\frac{1}{120}$, 四个 PE 硬件效率为 $\frac{1}{480}$

2 神经网络稀疏部分

2.1 模型中权重的分布

2.1.1 问题一解答

共同特点：均为单峰分布，且较为集中到 0 帮助：能大致确定稀疏度大小变化时模型精度的变化。

2.1.2 权重分布图

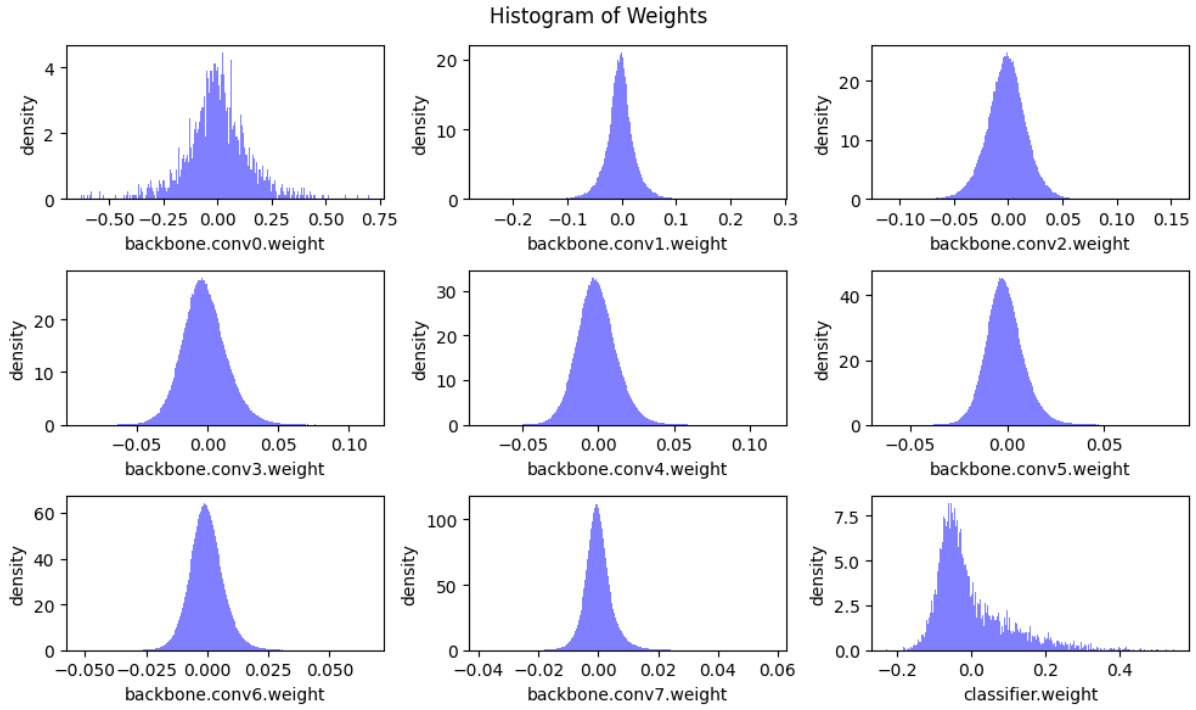

图 8: 权重分布图

2.2 基于数值大小进行细粒度剪枝

2.2.1 代码任务一代码

```python
def fine_grained_prune(tensor: torch.Tensor, sparsity : float) -> torch.Tensor:
    """
    对单个张量进行基于数值大小的修剪
    :param tensor: torch 中的 Tensor, 线性层/卷积层的权重 传的是引用，有做修改
    :param sparsity: float, 目标稀疏度
        稀疏度 = 张量中 0 的数目 / 张量中的总元素数目 = 1 - 张量中非 0 的数目 / 张量中的总元素数目
    :return:
        torch.(cuda.)Tensor, 返回掩码；掩码中的 True(1) 代表保留相应元素；False(0) 代表对相应元素执行剪枝 (置为 0)
    """
    sparsity = min(max(0.0, sparsity), 1.0) # 确保稀疏度在 [0, 1] 之间
    # 处理一些边界情况
    if sparsity == 1.0: # 稀疏度为 1: 全裁掉
        tensor.zero_()
        return torch.zeros_like(tensor) # 注意：该函数返回的是掩码，全裁掉那就返回一个和原张量形状相同、全为 0 的张量
                                        # torch.zeros_like 方法接受一个张量，返回一个和输入张量形状相同、但数值全为 0 的张量
    elif sparsity == 0.0: # 稀疏度为 0: 全保留
        return torch.ones_like(tensor) # 注意：该函数返回的是掩码，全保留那就返回一个和原张量形状相同、全为 1 的张量
                                       # torch.ones_like, 和 zeros_like 类似哦，相信你会举一反三 ~

    num_elements = tensor.numel() # 张量中总元素数目

    ###################### YOUR CODE STARTS HERE ######################
    # Z3dKFpREy9
    # 第一步：对权重张量的每个元素进行打分，得到分数 (重要性) 张量
    importance = tensor.abs()
    # importance = torch.abs(tensor) 也行
    # 第二步：根据分数和目标稀疏度，寻找阈值
    # threshold = importance.reshape(-1).sort(descending=False)[0][round(num_elements * sparsity) - 1]
    # threshold = importance.kthvalue()[0]   # 不指定维度的话默认在最后一个维度找 k-th, 相当于降维，返回时为最小值张量以及对应索引张量。[0] 指取值，[1] 指取索引
    threshold = importance.reshape(-1).kthvalue(round(sparsity * num_elements))[0]    # reshape 把改张量转化为一行的（自动推导） reshape(a.numel()) 也行
    # 第三步：得到掩码
    mask = importance > threshold
    # Z3dKFpREy9
    ###################### YOUR CODE ENDS HERE ######################

    # 第四步：将掩码作用于原权重张量
```

```
    # 注：我们这里使用了“原位操作”，即方法名后面带了个下划线 `_`
    # 在 Lab0.2 中，我们知道多数方法会返回一个“新”张量，并不会作用到原张量上
    # 而原位操作，允许我们将计算结果，直接作用到原张量上
    tensor.mul_(mask)

    return mask # 返回掩码
```

2.2.2 输出结果

图 9: 输出结果

```
* Test fine_grained_prune()
目标稀疏度: 0.75
    剪枝前的稀疏度: 0.04
    剪枝后的稀疏度: 0.76
    掩码的稀疏度: 0.76
* 测试通过
```

2.3 敏感度扫描

2.3.1 敏感度曲线图

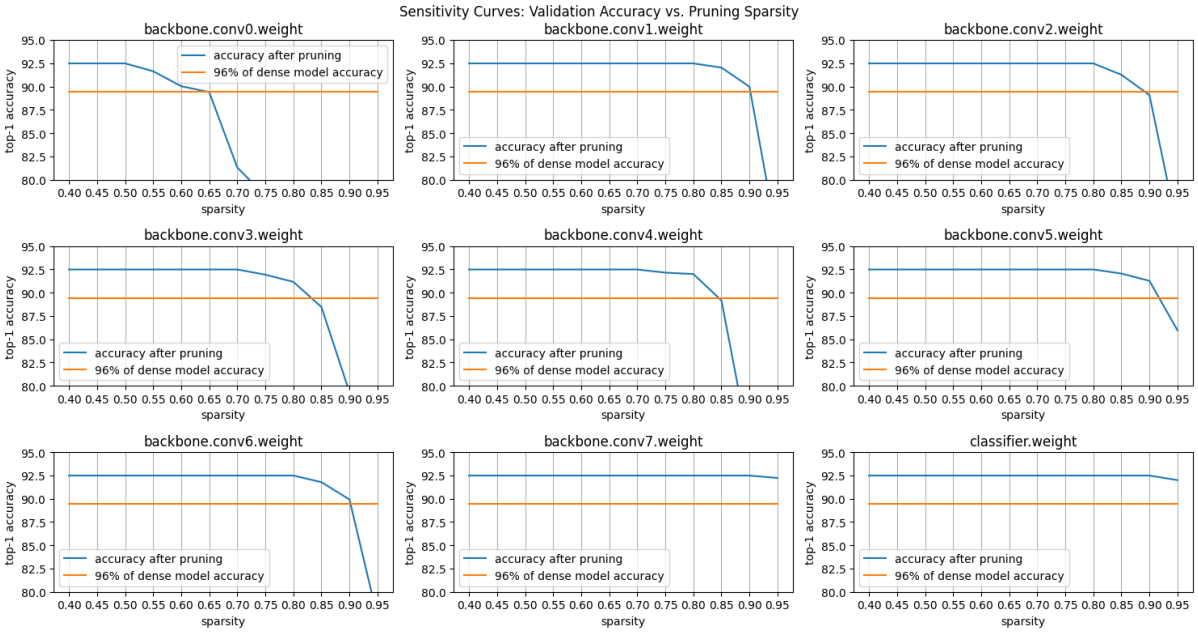

图 10: 敏感度曲线图

2.3.2 问题二回答

1. 稀疏度高，准确性低

2. 不同

3. 第一层最敏感，最后一层（分类器）最不敏感

2.4 指定各层的稀疏度

2.4.1 各层参数量图

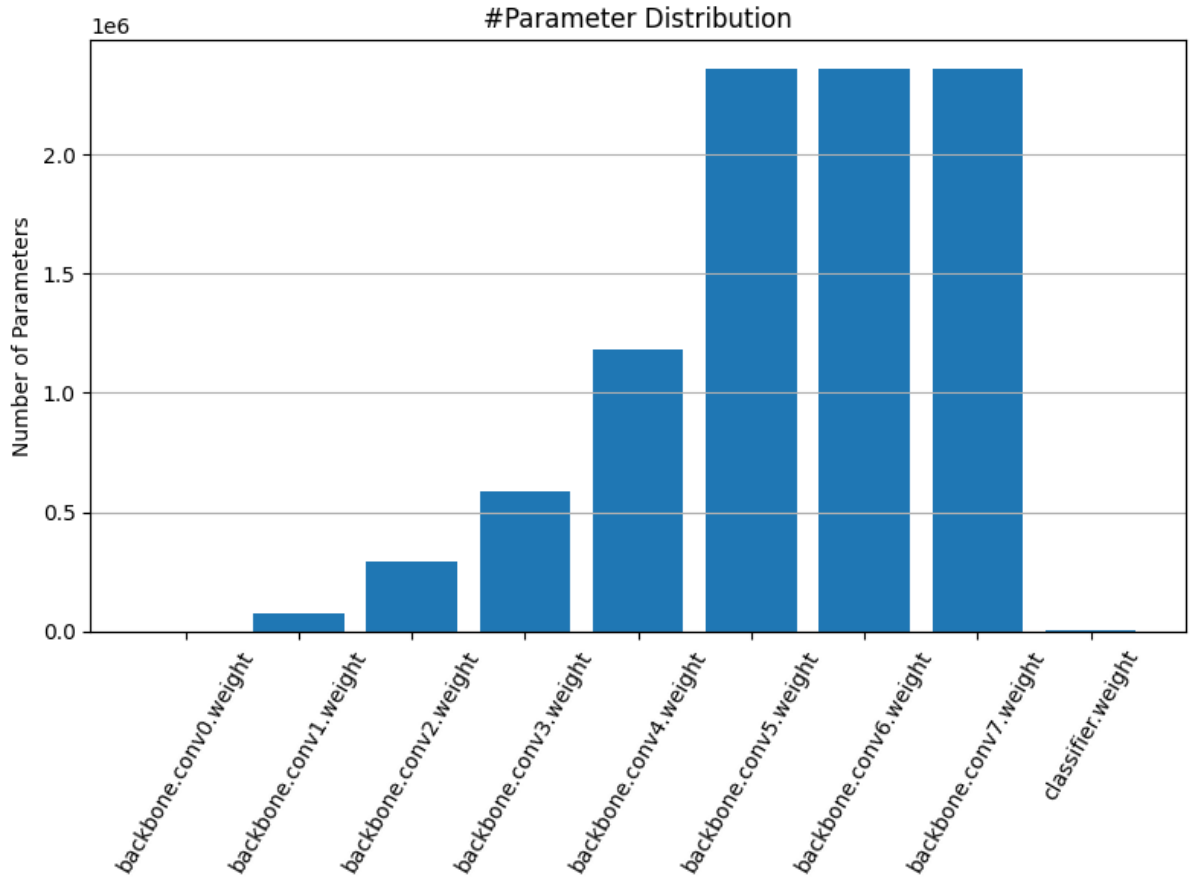

图 11: 各层参数量图

2.4.2 为各层指派的稀疏度

```
'backbone.conv0.weight': 0.5,
'backbone.conv1.weight': 0.85,
'backbone.conv2.weight': 0.85,
'backbone.conv3.weight': 0.8,
'backbone.conv4.weight': 0.8,
'backbone.conv5.weight': 0.9,
'backbone.conv6.weight': 0.9,
'backbone.conv7.weight': 0.9,
'classifier.weight': 0.9
```

根据敏感度曲线图指派并根据后续代码微调

2.4.3 稀疏剪枝后的权重分布图

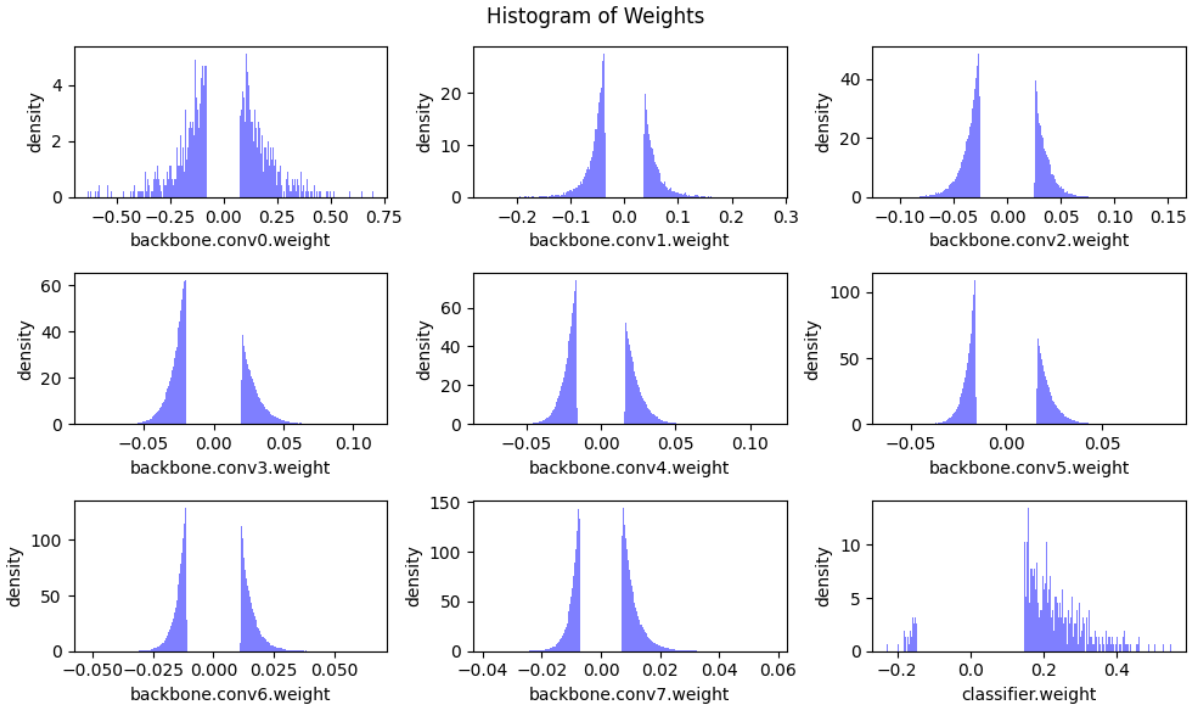

图 12: 稀疏剪枝后的权重分布图

2.4.4 稀疏度和准确率

状态	稀疏度	准确率
微调前	87.82%	40.06%
微调后	87.82%	92.91%