基于全局无结构剪枝（Global Unstructured Pruning）和知识蒸馏(Knowledge Distillation)的模型压缩应用

为了将学习到的先进知识应用到实际中，我参考综述报告中提到的如下文献内容，对之前实现的一个具有非常良好性能的医学图像分类模型进行了更加深度的模型压缩。涉及到的参考文献如下：

1) **[4] Hou, Y., Ma, Z., Liu, C., Wang, Z., & Loy, C. C. (2021). Network pruning via resource reallocation. arXiv preprint arXiv:2103.01847. ：（主要参考）**
2) [7] Iofinova, E., Peste, A., & Alistarh, D. (2023). Bias in pruned vision models: In-depth analysis and countermeasures. arXiv preprint arXiv:2304.12622.
3) [8] Han, S., Pool, J., Tran, J., & Dally, W. (2015). Learning both weights and connections for efficient neural network. In Advances in neural information processing systems (pp. 1135-1143).：
4) [14] Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531.

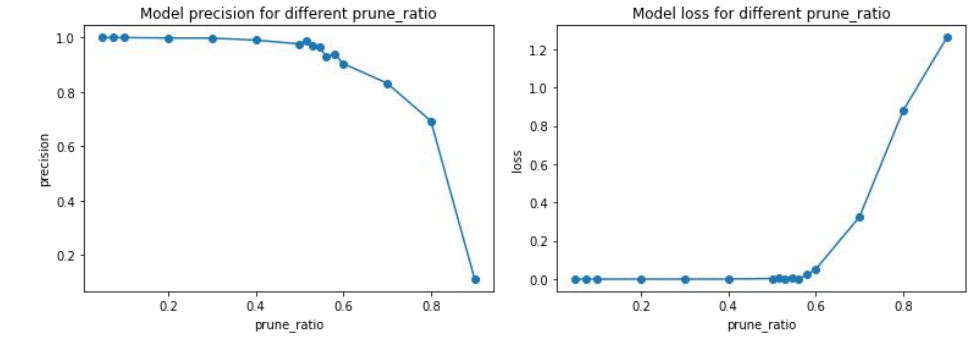
本次采用的模型压缩方法相比实验三中所采用的方法而言更加有效且压缩后的模型与原模型相比几乎没有精度损失。

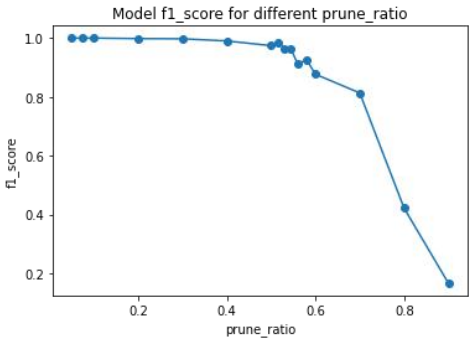
具体方法如下（主要参考[4]中所提到的**PEEL 技术：剪枝+知识蒸馏**）：

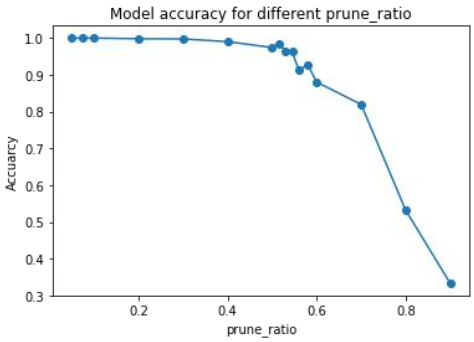
第一步，结合[4]、[7]、[8]中提出的剪枝方法与原模型的特点，首先采用全局无结构剪枝的方法对原模型进行剪枝。在这种方法中，剪枝过程针对整个模型的所有参数（通常是权重和偏置），而不是特定的层或结构（无结构）。

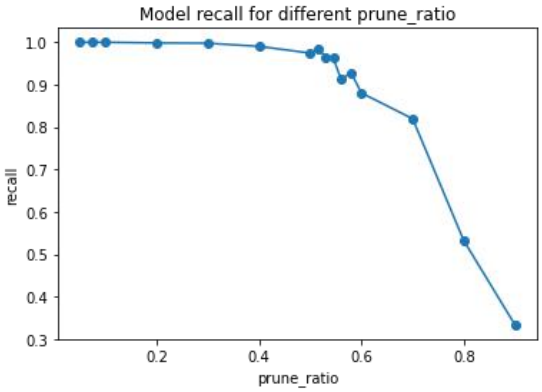
这种剪枝方法的关键思想是通过将一定比例的参数值设置为零，实现在保持模型性能的同时，创建一个稀疏的模型，从而降低计算和存储需求实现模型大小和计算开销的降低。

在全局无结构剪枝中，通常会设定一个剪枝比例，然后根据权重的绝对值大小对权重进行排序，并将最小的一定比例权重设置为零。本次实验中，我尝试了多个剪枝比例，尝试结果见下图：









综合剪枝比例以及剪枝后的模型性能，我选择了 0.55 这一剪枝比例对模型进行剪枝。

第二步，由于剪枝后，模型可能会出现性能下降，因此需要进行微调，以恢复或提高模型性能。微调是剪枝后的一个重要步骤，其目的在于调整剪枝后模型的权重，以便在保持模型大小和计算复杂度较低的同时，尽可能地恢复或提高模型性能。

在进行微调时，我们通常使用较小的学习率，以避免权重发生剧烈变化，从而导致过拟合。微调可以帮助模型在剪枝后适应新的架构，使其更适合解决原始任务。因此，在剪枝后进行微调是有意义的，可以实现在压缩模型大小和计算复杂度的同时，仍然保持较高的性能。

本实验中，我通过模型的微调将模型的性能恢复到与之前的原模型接近的水平：

原模型性能：

```
# 载入最佳模型作为当前模型
model = torch.load('./best-1.000.pth')
model.eval()
print(evaluate_testset())
```

```
{'test_loss': 7.575038e-05, 'test_accuracy': 1.0, 'test_precision': 1.0, 'test_recall': 1.0, 'test_f1-score': 1.0}
```

剪枝后性能：

```
model = torch.load('./best-1.000.pth')
# 对 ResNet50 的每个卷积层和线性层进行剪枝
for name, module in model.named_modules():
    if isinstance(module, torch.nn.Conv2d):
        prune.l1_unstructured(module, name='weight', amount=0.55)
        prune.remove(module, 'weight')  # 移除以恢复原始权重
    elif isinstance(module, torch.nn.Linear):
        prune.l1_unstructured(module, name='weight', amount=0.55)
        prune.remove(module, 'weight')  # 移除以恢复原始权重
model.eval()
print(evaluate_testset())
```

```
{'test_loss': 0.00500649, 'test_accuracy': 0.9586666666666667, 'test_precision': 0.963226571767497, 'test_recall': 0.9586666666666667, 'test_f1-score': 0.9585071682213094}
```

微调后性能：

```
# 载入最佳模型作为当前模型
teacher_model = torch.load('./fine_tuned_pruned_resnet50-0.999.pth')
teacher_model.eval()
print(evaluate_testset(teacher_model))
```

```
{'epoch': 0, 'test_loss': 0.0009464224, 'test_accuracy': 0.9993333333333333, 'test_precision': 0.9993346640053226, 'test_recall': 0.9993333333333334, 'test_f1-score': 0.9993333326666659}
```

第三步，根据[4]、[14]中提出的蒸馏方法，对全局无结构剪枝过的模型进行知识蒸馏，将原本的模型作为教师模型，采用 torchvision.models 中的 resnet18 作为学生模型，

```
from torchvision import models
import torch.optim as optim
from torch.optim import lr_scheduler

student_model = models.resnet18(pretrained=False, num_classes=n_class)
student_model = student_model.to(device)
```

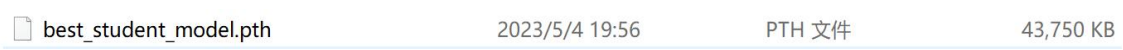
训练 50 个 epoch 保证知识蒸馏完毕后，resnet18 用更小的体积实现了甚至略微超过原模型的准确率

```
{'epoch': 50, 'test_loss': 0.0113918865, 'test_accuracy': 0.9996666666666667, 'test_precision': 0.9996669996669997, 'test_recall': 0.9996666666666667, 'test_f1-score': 0.9996666665833334}
```

模型压缩结果：

原模型：

压缩后模型：



压缩率=(原始模型大小- 压缩后模型大小)/原始模型大小

=(92180-43750)/92180

≈52.5%

可以看到，模型压缩在实现了 52.5%的压缩率的同时，仅付出了些许精度损失作为代价。根据上述实验结果，可以看到这种模仿[4]中的 PEEL 技术进行模型压缩的方法具有非常好的效果。