NOVAUtO
超 星 未 来

"鲁班"全流程开发工具链用户手册

剪枝工具

北京超星未来科技有限公司

更改历史

版本	更改描述	更改日期	更改人
V1.0	初始版本	2023.03.15	

目录

1. 剪枝工具(NPruner)

1.1. 简介

深度学习目前已经在自动驾驶领域取得了广泛应用。高精度的神经网络模型通常都十分庞大，由数百万甚至以亿计的参数构成，运行这些模型需要消耗大量的计算资源。虽然神经网络专用推理芯片在不断发展，算力也在不断增长，但把神经网络模型在资源受限的嵌入式计算设备上运行起来，并满足自动驾驶应用的实时性要求，这仍然是一个巨大的挑战。

NPruner是超星未来自主研发的神经网络模型自动剪枝工具，帮助用户对分类、检测、分割等各类模型进行剪枝压缩优化，支持视觉、点云感知等常见自动驾驶任务的优化。NPruner具有下列特点：

- 高能效：数量级地提升神经网络推理能效
- 高易用性：用户无需专业知识
- 高通用性：支持各类神经网络架构，适配各类计算平台
- 高度自动化：相比于手工压缩，能够减少数倍的研发时间

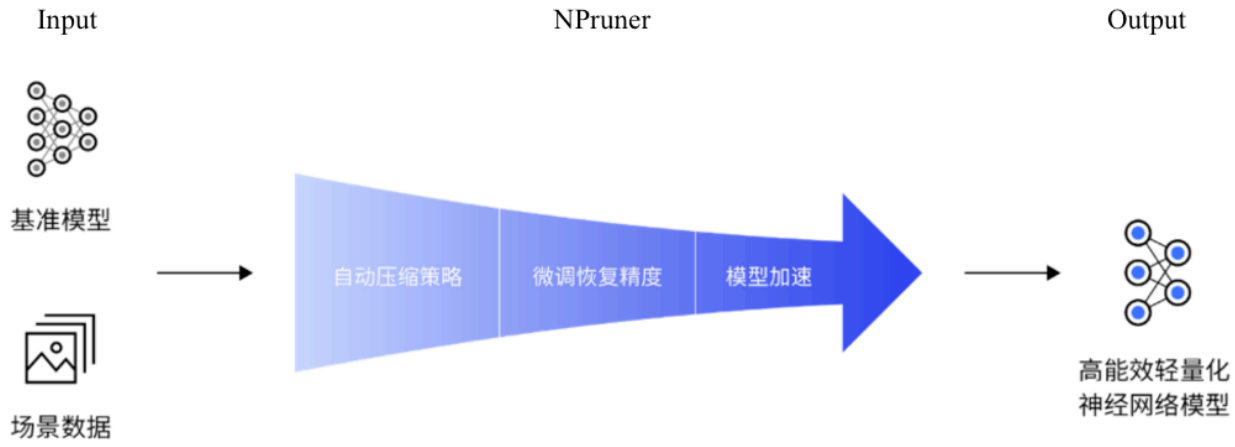

图1–1 剪枝流程示意

1.2. 功能

1.2.1. 已支持模型结构

目前支持的Pytorch Function:

```
['torch.add', '+', 'torch.mul', '*', 'torch.cat', 'torch.flatten',
'torch.stack', 'torch.mean', 'tensor.permute', 'tensor.view']
```

目前支持的Pytorch Module:

```
['nn.Conv2d', 'nn.AvgPool2d', 'nn.AdaptiveAvgPool2d', 'nn.MaxPool2d',
'nn.BatchNorm2d', 'nn.Dropout', 'nn.Linear', 'nn.ReLU', 'nn.ReLU6',
'nn.LeakyReLU', 'nn.SiLU', 'nn.Sigmoid', 'nn.Upsample', 'F.interpolate',
'nn.ConvTranspose2d', 'nn.ZeroPad2d', 'nn.Dropout2d', 'nn.PixelShuffle']
```

1.2.2. 典型模型的压缩率和精度损失范围

下面是NPruner对典型模型的压缩率和精度损失范围：

表1–1 YOLOv3压缩率和精度损失信息

YOLOv3	Dataset	Input Size	mAP	FLOPs	Params
原始版本	BDD100K	1×3×416×416	56.3%	65.3G	247MB
压缩版本	BDD100K	1×3×416×416	53.8%	9.8G	7.6MB

表1–2 UNet压缩率和精度损失信息

UNet	Dataset	Input Size	mIoU	PixACC	FLOPs	Params
原始版本	Cityscapes	1×3×420×420	61.1%	94.8%	258.6G	115.8MB
压缩版本1	Cityscapes	1×3×420×420	61.3%	94.6%	14.4G	2.7MB
压缩版本2	Cityscapes	1×3×420×420	52.0%	93.7%	7.9G	840KB

表1–3 YOLOv5的压缩率和精度损失信息

YOLOv5m	Dataset	Input Size	car	bus	bicycle	motorbike	person	mAP	Flops	Params
原始版本	VOC	1×3×640×640	92.7%	89.9%	91.5%	88.7%	88.6%	90.3%	50.6G	84.5MB
压缩版本	VOC	1×3×640×640	90.8%	87.4%	89.2%	85.3%	87.1%	88.0%	7.2G	12.6MB

表1–4 Pointpillars压缩率和精度损失信息

Pointpillars	Dataset	car	pedestrian	cyclist	mAP	Flops	Params
原始版本	Kitti	86.3%	58.1%	66.8%	70.4%	274.9G	20.01MB
超星未来版本	Kitti	85.1%	63.4%	72.7%	73.8%	274.9G	20.01MB
压缩版本	Kitti	83.9%	59.3%	70.4%	71.2%	40.0G	944KB

1.3. 安装

剪枝工具提供两种部署方式，可以直接通过whl包安装，或者使用docker进行部署。

1.3.1. whl安装

1.3.1.1. 运行环境

已通过以下环境测试：

- Ubuntu16.04/18.04/20.04, CentOS7.4
- Python >= 3.6
- Pytorch 1.4/1.5/1.6/1.7/1.8

1.3.1.2. 安装步骤

压缩工具包含两个wheel文件，下面是安装示例：

```
pip install n_graph-2.4.0-cp38-cp38-linux_x86_64.whl

pip install npruner-2.0.0-cp38-cp38-linux_x86_64.whl
```

安装上面wheel文件过程中，默认会安装这些Python环境依赖包: thop, schema, torch>=1.8.0 torchvision>=0.9.0 ntplib, onnx, onnx-simplifier==0.4.13, netifaces, pycryptodome。

1.3.2. docker部署

1.3.2.1. 获取docker镜像

下载docker镜像`npruner-v1.9.0-cuda11.1.tar.gz`，并载入镜像。

```
gunzip -c npruner-v1.9.0-cuda11.1.tar.gz | sudo docker load
```

查看镜像是否成功载入：

```
sudo docker images

# 显示结果如下
# [sudo] password for novauto:
# REPOSITORY                TAG                         IMAGE ID       CREATED
        SIZE
# npruner                   v1.9.0-cuda11.1             9e3a86bb16d4   21 hours
ago    8.08GB
```

1.3.2.2. 启动容器

容器启动脚本示例：

```
#/bin/bash
export MY_CONTAINER="npruner_test_v1.9.0"
export DOCKER_IMAGE="npruner:v1.9.0-cuda11.1"
num=`docker ps -a|grep "$MY_CONTAINER"|wc -l`

echo "name:   $MY_CONTAINER"
```

```
echo "image:  $DOCKER_IMAGE"
if [ 0 -eq $num ];then
    echo "Create new container."
    docker run --gpus all \
                    --net=host \
                    --shm-size 1G \
                    -v
/ic/toolchain/SharedDatasets/:/ic/toolchain/SharedDatasets/ \
                    -it --name $MY_CONTAINER \
                    $DOCKER_IMAGE /bin/bash
else
    echo "Start container."
    docker start $MY_CONTAINER
    docker exec -ti $MY_CONTAINER /bin/bash
fi

# MY_CONTAINER: docker 容器名称，可以自定义为您想要的名称
# DOCKER_IMAGE: 创建容器使用的镜像

# 可以在上面 docker run 命令中添加 -v 选项将本地的目录挂载到容器中，
# 例如 -v /home/SharedDatasets/:/opt/datasets/ ，
# 将 /home/SharedDatasets 目录中内容映射到了容器中的 /opt/datasets 下。
```

1.4. 快速上手

1.4.1. 压缩流程概述

通常，神经网络模型的剪枝流程包含三个阶段。为了对模型进行充分地压缩优化，STEP.2和STEP.3可迭代进行。

- STEP.1 Train from scratch
  基于用户已有的训练代码，训练原始模型。
- STEP.2 Automatic compression
  使用NPruner进行自动压缩。
- STEP.3 Finetune
  基于用户已有的训练代码，对压缩后的模型进行微调训练。

下面是一个自动迭代压缩的流程图:

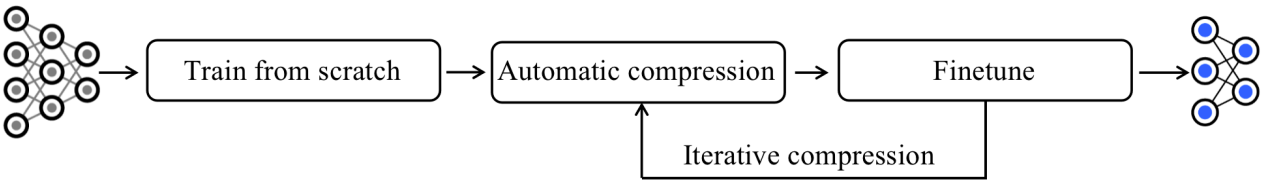

图1–2 自动迭代剪枝流程

1.4.2. NPruner许可证

Npruner需要商用许可证方可运行，试用版本的商用许可证有效期限为30天。配置好环境后，用户测试压缩工具附带的示例时会打印出设备ID信息，商用许可证的生成需要用户提供该ID信息，下图为设备ID信息示例：

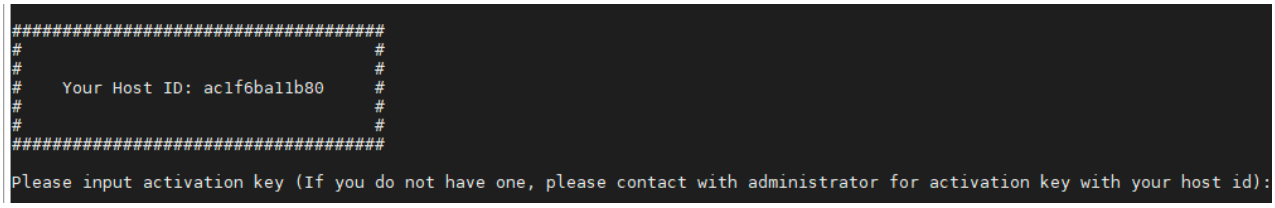

图1–3 设备ID信息

用户将得到的激活码输入到命令行，压缩工具会在主目录下生成许可证文件，工具就可以正常使用了。

1.4.3. 模型压缩

下面提供了模型压缩的参考示例：

```
python ./samples/npruner/automatic_compression.py
```

下面截取了上述示例中的部分代码来进行说明：

- STEP.1 基于用户已有的训练代码，训练原始模型。

```
# STEP.1 Train from scratch
optimizer = torch.optim.SGD(
    model.parameters(), lr=0.1, momentum=0.9, weight_decay=1e-4)
for epoch in range(args.pretrain_epochs):
    train(model, train_loader)
```

- STEP.2 基于NPruner进行压缩。

```
# STEP.2 Automatic compression
logging.info('start model pruning...')

dummy_input = [torch.randn([1, 3, 32, 32])]
exclude_layers = ['conv3']

model = compression(model=model,
                    val_func=test,
                    val_loader=val_dataloader,
                    exclude_layers=exclude_layers,
                    dummy_input=dummy_input,
                    ori_metric=0.99,
                    metric_thres=0.01,
                    single_process_mode=True)
```

```
pruned_model_path = os.path.join(
    args.checkpoints_dir, 'pruned_{}_{}.pth'.format(model_name, dataset_name))
torch.save(model, pruned_model_path)
```

对模型进行压缩需要用户提供自定义模型验证函数"test"，下面是模型验证函数的定义。

```
def test(model, test_loader, **kwargs):
    print(kwargs, "nothing happened to these args.")
    model.eval()
    model.cuda()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.cuda(), target.cuda()
            output = model(data)
            test_loss += F.cross_entropy(output, target,
reduction='sum').item()
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()
    test_loss /= len(test_loader.dataset)
    acc = correct / len(test_loader.dataset)
    logging.info('Loss: {}  Accuracy: {})\n'.format(test_loss, acc))
    return acc
```

- STEP.3 基于用户已有的训练代码，对压缩后的模型进行微调训练。

```
# STEP.3 Finetune
optimizer_finetune = torch.optim.SGD(
    model.parameters(), lr=0.1, momentum=0.9, weight_decay=1e-4)
for epoch in range(args.finetune_epochs):
    train(model, train_loader)
```

1.5. 常用接口说明

1.5.1. 函数接口

```
from npruner.utils.compress_utils import compression

dummy_input = [torch.randn([1, 3, 640, 640]).to("cpu")]

def test(model, test_loader, **kwargs):
    print(kwargs, "nothing happened to these args.")
    model.eval()
    model.cuda()
    test_loss = 0
```

```
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.cuda(), target.cuda()
            output = model(data)
            test_loss += F.cross_entropy(output, target,
reduction='sum').item()
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()
    test_loss /= len(test_loader.dataset)
    acc = correct / len(test_loader.dataset)
    logging.info('Loss: {}  Accuracy: {})\n'.format(test_loss, acc))
    return acc

model = compression(model=model,
                    val_func=test,
                    val_loader=val_dataloader,
                    dummy_input=dummy_input,
                    ori_metric=0.99,
                    metric_thres=0.09,
                    exclude_layers=['conv3'],
                    single_process_mode=True,
                    sparsities=None,
                    sparsities_dict=None,
                    early_stop_mode='dropped',
                    config_list=None,
                    channel_alignment=2,
                    forward_name='forward')
```

1.5.2. 参数说明

表1–5 NPruner接口参数说明

参数	类型	参数说明
model	torch.nn.Module	待压缩的模型。
val_func	function	在敏感度分析阶段，该函数返回模型的精度或损失。
val_loader	torch.utils.data.DataLoader	一个可迭代对象，代表数据集。
dummy_input	list	列表中包含了模型的输入Tensor。
ori_metric	float	原始模型的精度。
metric_thres	float	精度下降的阈值，该参数用于控制模型压缩大小。
exclude_layers	list	包含不需要执行敏感度分析的卷积层名字，这些卷积层不会被压缩。
single_process_mode	bool	是否开启多进程敏感度分析模式，试用版本工具只支持单进程模式。
sparsities	list	列表形式的参数，用于指定敏感度分析的稀疏度，默认是[0.1, 0.2, 0.3, 0.4, 0.5]。
sparsities_dict	dict	字典形式的参数支持粗粒度自定义稀疏，key(str)表示对应卷积的名字，value(list)是按顺序排列的自定义稀疏度。例如：{'conv1' : [0.25, 0.55, 0.90,]}
early_stop_mode	str	如果参数不为None，当验证指标（例如准确率/损失）已经达到阈值时，卷积层的敏感度分析将提前停止。 我们支持两种不同的提前停止模式：'dropped'、'raised'。默认值为 None，这意味着在测试所有给定的稀疏性之前分析不会停止。'dropped' 模式表示当验证指标减小的值超过'metric_thres'时，停止分析。'raised' 模式表示当验证指标增加的值超过'metric_thres'时，停止分析。
config_list	list	压缩配置列表。
channel_alignment	int	指定压缩过程中卷积输出通道需要按数字多少对齐，例如，channel_alignment=16，channel_aligment默认等于2。
forward_name	str	支持'forward'和'dummy_forward'两种方式，通常用于在压缩阶段去除后处理操作。

1.5.3. 敏感度分析文件

模型压缩包括**敏感度分析，压缩和微调训练**三步，敏感度分析阶段会在当前目录下生成一个sens_analysis文件夹， 完成敏感度分析后会在sens_analysis下生成sens.csv文件，该文件第一行由稀疏性列表构成，第二行是对应的test_func的返回值。

```
layername,0.1,0.2,0.3,0.4,0.5,0.7,0.8,0.9
features.0,0.46308,0.06978,0.0374,0.03024,0.01512,0.00866,0.00492,0.00184
layername,0.1,0.2,0.3,0.4,0.5,0.7,0.8,0.9
features.3,0.51184,0.37978,0.19814,0.07178,0.02114,0.00438,0.00442,0.00142
layername,0.1,0.2,0.3,0.4,0.5,0.7,0.8,0.9
features.6,0.53566,0.4887,0.4167,0.31178,0.19152,0.08612,0.01258,0.00236
layername,0.1,0.2,0.3,0.4,0.5,0.7,0.8,0.9
features.8,0.54194,0.48892,0.42986,0.33048,0.2266,0.09566,0.02348,0.0056
layername,0.1,0.2,0.3,0.4,0.5,0.7,0.8,0.9
features.10,0.5394,0.49576,0.4291,0.3591,0.28138,0.14256,0.05446,0.01578
```

⚠️用户定义val_func函数时，参照上文test函数的定义，参数列表必须按顺序依次包含：model, test_loader。test函数的参数列表中除了model，test_loader以外，可能还需要传入其他参数，这些额外参数可以通过**键值对方式**传入compression接口。

⚠️目前使用多进程并行来加速敏感度分析，**多进程要求model和test_loader对象是可pickle的**，否则无法开启多进程。 代码中对model和test_loader都进行了pickle检查，如果对象中有无法pickle的属性（例如lambda函数）则会报错。

⚠️压缩后的模型要通过torch.save(model, "compression.pth")保存，这是因为模型经过压缩后，它

的冗余通道已经被剪掉，得到的新模型输入输出通道和定义的原始模型不匹配，所以需要保存压缩后模型结构和权重。压缩后的模型需要微调来恢复它的精度，**保存的压缩模型文件并不能跨平台加载**，它需要模型定义才能被反序列化，在加载压缩模型时，可以通过“import”方式导入模型定义。

1.6. 注意事项 & 常见问题

NPruner需要模型建立有向无环图来分析依赖关系，因此对模型定义有一定的限制。

1.6.1. 模型定义注意事项

1.6.1.1. Function

1.permute

`permute`会改变剪枝Mask所在的维度，例如`.permute(0, 2, 3, 1)`将`[N, C, H, W]`转化为`[N, H, W, C]`, Mask则会从`[None, Mask, None, None]`变成`[None, None, None, Mask]`，会影响后续模块的剪枝，如果模块里面有`permute`操作，请预先评估`permute`操作后的模块是否需要剪枝。

2.view

目前`view`操作不允许对需要剪枝的维度进行操作。例如剪枝的维度为`C`, 可以进行的操作为：

```
[N, C, H, W].view([N, C, H*W])
[N, C, H, W].view([N, C, X, Y, Z]) #X * Y * Z = H * W
[N, C, H, W].view([N1, N2, C, X, Y, Z]) # N1 * N2 = N
```

⚠️不允许的操作为

```
[N, C, H, W].view([N*C1, C2, H, W]) # C1 * C2 = C
```

如果需要使用`tensor.view(N, -1)`，请使用`torch.flatten`

1.6.1.2. Module

1.共享参数

module要避免出现共享参数的情况，例如：在不同地方使用激活函数，请分别初始化后再使用。

```
# 正确例子
class Custom(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(C1, C2)
        self.bn1 = nn.BatchNorm2d(C2)
        self.act1 = nn.ReLU()
        self.conv2 = nn.Conv2d(C2, C3)
        self.bn2 = nn.BatchNorm2d(C3)
        self.act2 = nn.ReLU()
    def forward(self, inputs):
        out1 = self.act1(self.bn1(self.conv1(inputs)))
        out2 = self.act2(self.bn2(self.conv2(out1)))
```

```
        return out2

# ⚠️错误例子
class Custom(nn.Module):
    def __init__(self):
        super().__init__()
        self.act = nn.ReLU()
        self.conv1 = nn.Conv2d(C1, C2)
        self.bn1 = nn.BatchNorm2d(C2)
        self.conv2 = nn.Conv2d(C2, C3)
        self.bn2 = nn.BatchNorm2d(C3)
    def forward(self, inputs):
        out1 = self.act(self.bn1(self.conv1(inputs)))
        out2 = self.act(self.bn2(self.conv2(out1)))
        return out2
```

2.自定义module

自定义module建议通过继承nn.Module来实现：

```
class PadConv(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size, stride=1,
                 padding=0, dilation=1, groups=1, bias=True):
        super(PadConv, self).__init__()
        self.pad = padding
        self.s = stride
        self.ori_conv = nn.Conv2d(
            in_channels, out_channels, kernel_size, stride, 0, dilation,
groups, bias)

    def forward(self, x):
        if self.pad == 1 and self.s == 2:
            x = F.pad(x, (0, 1, 0, 1))
        elif self.pad == 1 and self.s == 1:
            x = F.pad(x, (1, 1, 1, 1))
        return self.ori_conv(x)
```

⚠️避免使用这样的自定义module(继承nn.Conv2d)：

```
class PadConv(nn.Conv2d):
    def __init__(self, in_channels, out_channels, kernel_size, stride=1,
                 padding=0, dilation=1, groups=1, bias=True):
        super(PadConv, self).__init__(in_channels, out_channels,
                                      kernel_size, stride, 0, dilation, groups,
bias)
        self.pad = padding
        self.s = stride
```

```
    def forward(self, x):
        if self.pad == 1 and self.s == 2:
            x = F.pad(x, (0, 1, 0, 1))
        elif self.pad == 1 and self.s == 1:
            x = F.pad(x, (1, 1, 1, 1))
        x = super().forward(x)
        return x
```

1.6.2. AMP支持

目前模型的敏感度分析和finetune与AMP解耦，理论上支持AMP包装后的model进行敏感度分析和finetune，但是由于Apex官方代码中有未知Bug会导致AMP模式下，敏感度分析出错， **建议使用torch官方的AMP模块** `torch.cuda.amp`。经过测试，加速效果与Apex模块一致，并且可以支持多进程敏感度分析与finetune。

表1–6 混合精度模式的UNet推理时间对比

UNet	Train(ms)	Test(ms)
original model	66	22
apex.amp	43	13
torch.cuda.amp	39	12

使用方式如下：

```
from torch.cuda.amp import autocast as autocast

# 创建model，默认是torch.FloatTensor
model = Net().cuda()
optimizer = optim.SGD(model.parameters(), ...)

# 在训练最开始之前实例化一个GradScaler对象
scaler = GradScaler()

for epoch in epochs:
    for input, target in data:
        optimizer.zero_grad()

        # 前向过程(model + loss)开启 autocast
        with autocast():
            output = model(input)
            loss = loss_fn(output, target)

        # Scales loss. 为了梯度放大.
        scaler.scale(loss).backward()

        # scaler.step() 首先把梯度的值unscale回来.
```

```
        # 如果梯度的值不是 infs 或者 NaNs，那么调用optimizer.step()来更新权重，
        # 否则，忽略step调用，从而保证权重不更新（不被破坏）
        scaler.step(optimizer)

        # 准备着，看是否要增大scaler
        scaler.update()
```

1.6.3. 常见问题

(1) metric_thres的设置

建议一次压缩模型不超过模型当前大小的50%，通过查找敏感度分析文件，找到大多数conv在50%压缩率情况下的精度，设置近似的metric_thres。

工具新的版本中也实现了设置全局比例结合敏感度分析进行压缩，实现自动迭代压缩，将在测试完毕之后发布。

(2) 敏感度分析的并行度设置

由于每个进程都会有一个独立的dataloader，dataloader又会开启num_workers个子进程，建议`N * (num_workers + 1)`不超过CPU核心的两倍，N为敏感度分析阶段开启的进程数量。

(3) 模型中有不支持的算子

1. NPruner是从整体的角度考虑模型压缩的，需要对模型中**全部可压缩算子**进行敏感度分析才能生成模型的压缩策略。如果不支持算子位于模型尾部，或者不支持算子后面的算子不需要被压缩， 那么就可以将包含不支持算子的子模型从原模型中分离，单独对可压缩部分的模型进行剪枝。

2. 如果不支持算子在模型中位置不满足上述情况, 请提出需求和场景模型, 我们会提供针对性支持。

(4) 剪枝不友好模型

一般情况下，剪枝不友好模型都包含大量的分离卷积，比如：SSD（backbone是MobileNet）,客户在对这类模型压缩时，压缩阈值不要设置过大，建议第一轮压缩掉的参数量不要超过原模型20%。目前工具不支持包含有ShuffleNet结构的模型。

(5) 剪枝后的模型精度低

剪枝后的模型一定要经过finetune过程来恢复模型的精度，如果finetune后精度依然很低，考虑是由这些原因造成的：

① 由于模型剪枝空间有限（比如:MobileNet），模型被剪掉部分太多。针对这种情况，客户需要减小压缩阈值。

② 模型中某些层剪枝比例过大。针对这种情况，用户需要根据敏感度分析文件（sens_analysis目录下的sens.csv）判断是否存在某些层剪枝比例过大，如果存在，可以通过compression接口中的sparsities_dict参数设置某些层的稀疏度。

③ finetune的超参数设置存在问题。通常finetune后压缩模型精度低，考虑是由超参数设置不合理引起的，如果finetune过程采用训练baseline时的超参数不能恢复模型精度，那么需要用户调整训练超参数，比如：学习率，学习率更新策略等。

(6) 剪枝过程中出现推理报错

如果输入的Tensor形状和算子不匹配就会出现推理报错，出现这种报错信息时客户首先需要检查是否有不能压缩的算子被压缩了（比如：YOLOv3的Detect部分的卷积输出通道和数据集类别数有关，所以这些卷积不能被压缩），如果模型包含不能参与压缩的卷积，客户需要将这些卷积的名字传到exclude_layers参数对应的列表里。如果模型出现推理报错，并且推理报错部分不包含不能压缩的算子，请联系我们, 我们会提供针对性支持。

文件状态：
[ ]草稿
[✓]正式发布
[ ]正在修改

文件起草分工：
唐长成 张志永, 陆天翼

责任人	签名	日期
编制：张志永		
审核：唐长成		
批准：陈忠民		

所有权声明

该文档及其所含信息是北京超星未来科技有限公司的资产。该文档及其所含信息的复制、使用及披露必须得到北京超星未来科技有限公司的书面授权。