HRank：使用高阶特征图的过滤器修剪

杨卓凡

摘要

卷积神经网络 (CNN) 在计算机视觉应用中取得了巨大的成功，如分类、检测和分割。然而，复杂的网络架构需要大量的计算资源和内存占用成本，使得大多先进的 CNN 无法部署在边缘设备，如：智能手机等。我们希望在尽量不减少模型的精度情况下，进行神经网络的模型压缩。常用的技术包括滤波器压缩、参数量化和网络修剪。在不损害原始精度的情况下修剪和压缩各个层的权重。然而，基于大小的权重修剪减少了来自完全连接层的大量参数，并且修剪后网络的不规则稀疏性，可能不能充分降低卷积层中的计算成本。作者[1]通过探索特征图的高秩，提出了一种新的滤波器修剪方法。HRank 的灵感来自这样一个发现，即无论 CNN 接收到的图像批次数量如何，单个过滤器生成的多个特征图的平均排名总是相同的。基于 HRank，作者[1]开发了一种数学公式化的方法来修剪具有低秩特征图的滤波器。剪枝背后的原理是低秩特征图包含较少的信息，因此剪枝后的结果可以很容易地再现。在不引入任何额外约束的情况下，HRank 在 FLOP 和参数减少方面比现有技术有了显著的改进，在 VGG-16 上实现了 58.2% 的 FLOP 的减少，Top-1 精度仅下降 0.52%。

关键词：卷积神经网络；神经网络修剪；神经网络加速

1 引言

在过去几年中，我们见证了深度神经网络在计算机视觉领域的快速发展，从基本的图像分类任务，如 ImageNet 识别挑战[2]，到一些更高级的应用，如对象检测、语义分割、图像字幕等。与基于人工设计的视觉特征的传统方法相比，深度神经网络在这些领域取得了最先进的性能。

尽管它取得了巨大成功，但典型的深度模型很难部署在资源受限的设备上，例如移动电话或嵌入式小工具。资源受限意味着必须在有限的资源供应（如计算时间、存储空间等）下完成计算任务。深度神经网络的主要问题之一是其巨大的计算成本和存储开销，这对移动设备构成了严重挑战。例如，VGG-16 模型[3]具有 1.38 亿个参数，占用超过 500MB 的存储空间，并且需要 309.4 亿次 Flops（浮点运算）来对单个图像进行分类。这样一个巨大的模型很容易超过小型设备的计算极限。因此，网络压缩引起了学术界和工业界的极大兴趣。

网络压缩通常在有一定精度损失的情况下实现。在某些特殊情况下，精度甚至没有损失，乃至于会提高的情况。常用的技术有参数量化和网络修剪。参数量化涉及用宽度减小的数据类型替换现有数据类型。例如，用 8 位整数（INT8）替换 32 位浮点（FP32）。通常可以对值进行编码，以保存比简单转换更多的信息。参数分解是一种将高阶张量分解为低阶张量的技术，简化了内存访问并压缩了模型大小。它的工作原理是将大的层分解成许多小的层，从而减少计算次数。它可以应用于卷积层和完全连接层。该技术也可以应用于参数量化。网络修剪涉及删除不影响网络准确性的参数，修剪可以通过多种方式进行，典型的工作要么修剪滤波器权重以获得稀疏权重矩阵（权重修剪）[4]，要么从网络中移除整个滤波器（滤波器修剪）[5]。非结构化稀疏模型不能由现成的库支持，因此需要专门的硬件和软件来进行有效的推理，这在现实应用中是困难和昂贵的。另一方面，非结构化随机连接忽略了缓存和

内存访问问题。由于随机连接导致的缓存局部性差和内存访问跳跃，实际加速非常有限。但过滤器修剪方法没有这种限制。在 HRank 中，作者基于滤波器修剪以实现模型压缩（减少参数）和加速（减少 FLOP），旨在为具有低计算能力的设备提供通用解决方案。

滤波器修剪的核心在于滤波器的选择，滤波器的选择应该以最低的精度获得最高的压缩比为目标。基于滤波器评估函数的设计，可以将滤波器修剪分为两组。

属性重要性：根据 CNNs 的固有属性（如 L1-范数和梯度等）对过滤器进行修剪。这些修剪方法并不能修正网络训练损失。在修剪之后，通过微调来增强模型性能。但因为大多数滤波器评估函数的设计都是自适应的，这带来了时间复杂度低的优点，但也限制了加速度和压缩比。

自适应重要性：将剪枝需求嵌入到网络训练损失中，并采用联合再训练优化来生成自适应剪枝决。然而，由于损失发生了变化，所需的再培训更加繁重，通常还需要进行另一轮超参数调优。对于某些方法，例如基于掩码的方案，修改后的损失需要专门的优化器，其影响了基于自适应重要性的方法的灵活性和易用性。

一方面，我们追求更高的压缩/加速比，而另一方面，受到繁重的机器时间和人力的限制（特别是基于自适应重要性的方法）。我们可以将这些问题归因于缺乏关于过滤器重要性和冗余性的实践/理论指导。而作者[1]提出了一种有效且高效的滤波器修剪方法，该方法探索了每个层中特征图的高秩（HRank），消除了引入额外辅助约束或重新训练模型的需要，从而简化了特征图生成、排名、微调的过程。

作者通过大量实验证明了 HRank 在各种技术状态下在模型压缩和加速方面的效率和有效性，而我们使用 VGG-16 模型[3]在 CIFAR-10 数据集[6]上进行了性能测试。

2 相关工作

2.1 L1-范数

基于 L1-范数判断滤波器的重要性[5]从第 i 个卷积层剪掉 m 个卷积核的算法过程：对每个卷积核 $F_{i,j}$，计算它的权重绝对值之和

$$s_j = \sum_{l=1}^{n_i} \sum |K_l|$$

，然后根据 s_j 排序，将 m 个权重绝对值之和最小的卷积核及对应的特征图剪掉。下一个卷积层中和剪掉的特征图相关的卷积核也要移除。一个对于第 i 层和第 i+1 层的新的权重矩阵被创建，最后剩下的权重参数被复制到新模型中

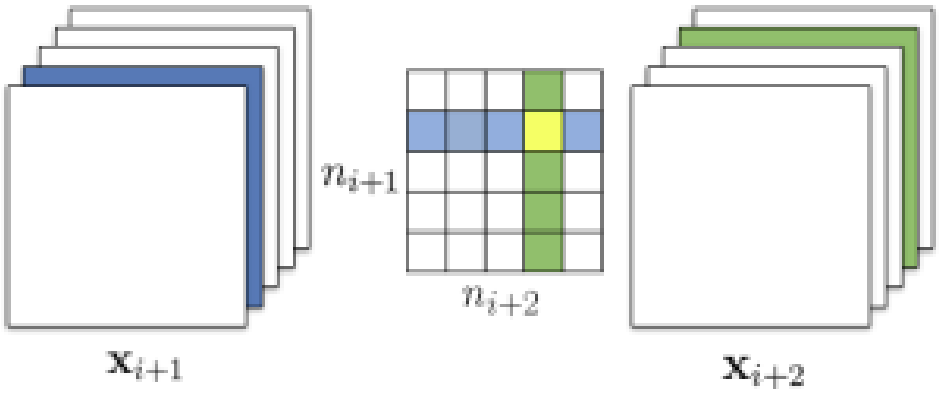

图 1: 跨连续层修剪过滤器。独立修剪策略计算过滤器和（以绿色标记的列），而不考虑前一层中移除的特征图（以蓝色显示），因此仍包含以黄色标记的内核权重。贪婪修剪策略不计算已经修剪的特征图的内核

而在涉及不同卷积层的剪枝时，有两种不同的策略。第一种：每一层独立剪枝，即在计算 (求权重绝对值之和) 时不考虑上一层的剪枝情况，所以计算时图中的黄色点权重仍然参与计算，第二种是采用贪心策略，计算时不计算已经剪枝过的，即黄色点不参与计算。实验结果证明采用贪心策略的计算方式精度会好一些

2.2 近似输出

ThiNet[7]认为一个第 i 层过滤器是否可以被修剪取决于它的下一层。如果第 i+1 层输入中的通道子集可以近似第 i+1 层的输出，那么其他输入通道可以安全地从 i+1 层的输入中删除。又因为第 i+1 层的一个输入通道是由第 i 层中的一个滤波器产生的，因此也可以安全地修剪第 i 层中相应的滤波器。也就是说，只要确保第 i+1 层的输出不改变太多，则这个裁剪后的模型效果也不会差太多。因为要比较输出对比，所以需要先输入一些数据去确定剪枝前后输出是否有变化，这个剪枝方法是数据驱动的。

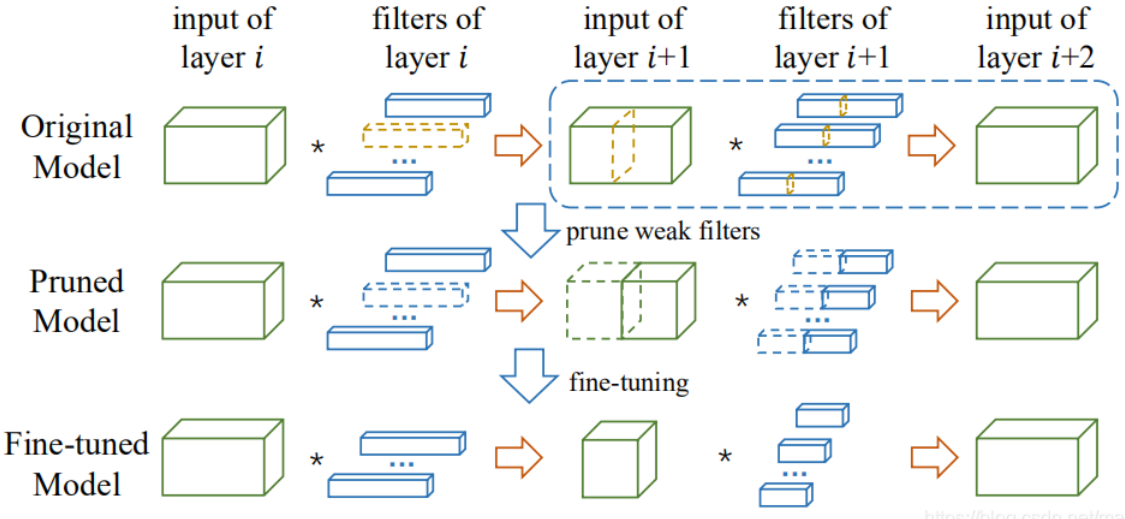

图 2: ThiNet 图解。首先关注虚线框部分，以确定几个弱通道及其相应的过滤器（第一行中以黄色突出显示）。这些通道（及其关联的过滤器）对整体性能几乎没有贡献，因此可以丢弃，从而导致修剪模型。

1. 【滤波器的选择】根据第 i+1 层的输入通道与剪枝率的统计信息来指导剪枝第 i 层的滤波器（同时第 i+1 层每个滤波器中相应通道删除）

2. 【剪枝】第 i+1 层的弱输入通道和它们在第 i 层中的对应滤波器将被删除，得到一个更小的模

型。注意，修剪后的网络具有完全相同的结构，但具有更少的滤波器和通道

3.【微调】为了恢复因滤波器剪枝而导致的泛化能力受损，需要进行微调。为了节省时间，我们在修剪一层后微调几个 Epoch。在所有的层都被修剪后，再将进行额外的 Epoch

4.【迭代到步骤 1 来修剪下一层】

2.3 讨论

与权重剪枝相比，过滤器剪枝在降低模型复杂度方面更为有利。此外，它的结构化修剪可以很容易地集成到高效 BLAS 库中，而无需专门的软件或硬件支持。修剪方法存在效率低下的加速和压缩（基于属性重要性的过滤器修剪）或机器和人工成本（基于自适应重要性的过滤器剪枝）。这两个问题给在资源有限的设备上部署深度卷积神经网络带来了根本挑战。可以将这种困境归因于缺少关于过滤器重要性和冗余的实践/理论指导。深度模型存在严重的过度参数化问题。例如，Denil 等人[8]证明了仅使用其原始参数的一小部分就可以有效地重建网络。然而，在模型训练期间，这种冗余似乎是必要的，因为高度非凸优化很难用现有技术解决。因此，非常需要在训练后减小模型尺寸。

3 本文方法

3.1 本文方法概述

一个预先训练的模型有 K 层，C^i 是第 i 个卷积层，滤波器可以用 $\mathcal{W}_{C^i}=\left\{\mathbf{w}_1^i,\mathbf{w}_2^i,\ldots,\mathbf{w}_n^i\right\}\in\mathbb{R}^{n_i\times n_i-1\times k_i\times k_i}$ 表示,其中第 j 个滤波器为 $\mathbf{w}_j^i\in\mathbb{R}^{n_{i-1}\times k_i\times k_i}$。对于特征图我们定义为 $\mathcal{O}^i=\left\{\mathbf{o}_1^i,\mathbf{o}_2^i,\ldots,\mathbf{o}_{n_i}^i\right\}\in\mathbb{R}^{n_i\times g\times h_i\times w_i}$，其中第 j 个特征图由 $\mathbf{w}_j^i$ 生成。过滤器的剪枝可以将分为 $\mathbf{w}_j^i\mathcal{W}_{C^i}$ 分为两组，保留的子集 $\mathcal{I}_{C^i}=\left\{\mathbf{w}_{\mathcal{I}_1^i}^i,\mathbf{w}_{\mathcal{I}_2^i}^i,\ldots,\mathbf{w}_{\mathcal{I}_{n_i}^i}^i\right\}$ 和一个不重要将要被裁剪的子集 $\mathcal{U}_{C^i}=\left\{\mathbf{w}_{\mathcal{U}_1^i}^i,\mathbf{w}_{\mathcal{U}_2^i}^i,\ldots,\mathbf{w}_{\mathcal{U}_{ni}^i}^i\right\}$，其中 $\mathcal{I}_j^i$ 和 $\mathcal{U}_j^i$ 分别是第 j 个重要的过滤器和第 j 个不重要的过滤器是重要和不重要滤波器的数量，于是有 $\mathcal{I}_{C^i}\cap\mathcal{U}_{C^i}=\varnothing,\mathcal{I}_{C^i}\cup\mathcal{U}_{C^i}=\mathcal{W}_{C^i}$ 和 $n_{i1}+n_{i2}=n_i$ 滤波器修剪的目的是找到不重要的滤波器并删除他们，这可以表示为一个优化问题。

作者认为，之前在滤波器上的特别设计忽略了输入图像和输出标签的分布，这可能是基于属性重要性的方法表现出次优性能的原因。相反，在本方法中，在特征映射上定义其的优化。其基本原理在于，特征映射是既能反映滤波器属性又能反映输入图像的中间步骤。特征图包含的信息越多，相应的滤波器就也越重要。利用了特征图的秩，它不仅被证明是一种有效的信息度量，而且在 P(I) 上也是一种稳定的表示。

3.2 秩

一个矩阵可以看作多个列向量的组合，如果一个列向量可以被其它列向量的通过一定的线性运算表达出来，就说这些向量是线性相关的。倘若一组向量相互不能够被表达，那么这组向量就是线性无关。一个矩阵的秩（rank）就是最大线性不相关的向量个数。一个矩阵的秩反应了矩阵所拥有的有效信息量，不相关的向量可以看作是固定的信息，而相关的向量可以用这些固定的信息来表达，可以说它是冗余的。

作者的灵感来自于这样一个发现，即由单个滤波器生成的多个特征图 (一个特征图就是一个矩阵) 的平均秩 (rank) 总是相同的，而不考虑接收到的 CNNs 图像 batch 的数量。在此基础上，提出了一种用数学方法对低秩特征图进行滤波的方法。剪枝背后的原则是，低秩特征图包含的信息较少，因此剪

枝的结果可以很容易地复制。

作者在做压缩的过程中发现这样一个规律：多个输入特征图经过一个过滤器之后的特征图的平均秩几乎是不变的；这似乎表明了输出特征图的秩主要由过滤器来决定。这个发现让作者激动不已，这给了作者启发：通过少量的特征图就可以估算一个过滤器生成特征图的秩的大小，那么这就给网络压缩带来了很小的代价，因为不用去计算更多的输入数据了。如下图，不管输入图片数量多少，各层计算出的秩基本上保持稳定，使得可以在不用输入太多图片的情况下完成秩的估算。

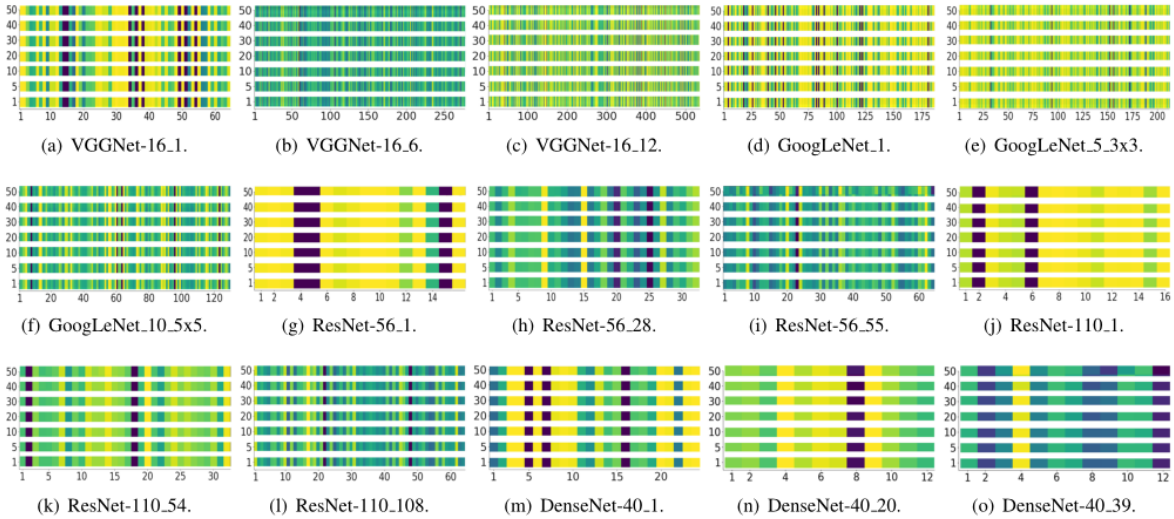

图 3: CIFAR-10 上不同卷积层和架构的特征图的平均秩统计。对于每个子图，x 轴表示特征图的索引，y 轴表示训练图像的批次（每个批次大小设置为 128）。不同的颜色表示不同的等级值。可以看出，无论图像批次如何，每个特征图（子图的列）的等级几乎不变（相同的颜色）。因此，即使是少量的图像也可以有效地估计不同架构中每个特征图的平均等级。

3.3 剪枝方法

整个裁剪流程就相对简单了：

1. 首先使用小批次数据计算特征图平均秩
2. 对秩值按从高到低排序
3. 找到高秩值对应的特征通道，裁剪低秩值通道
4. 使用高秩值通道作为预训练权重，对裁剪后的模型进行 fine-tune 训练

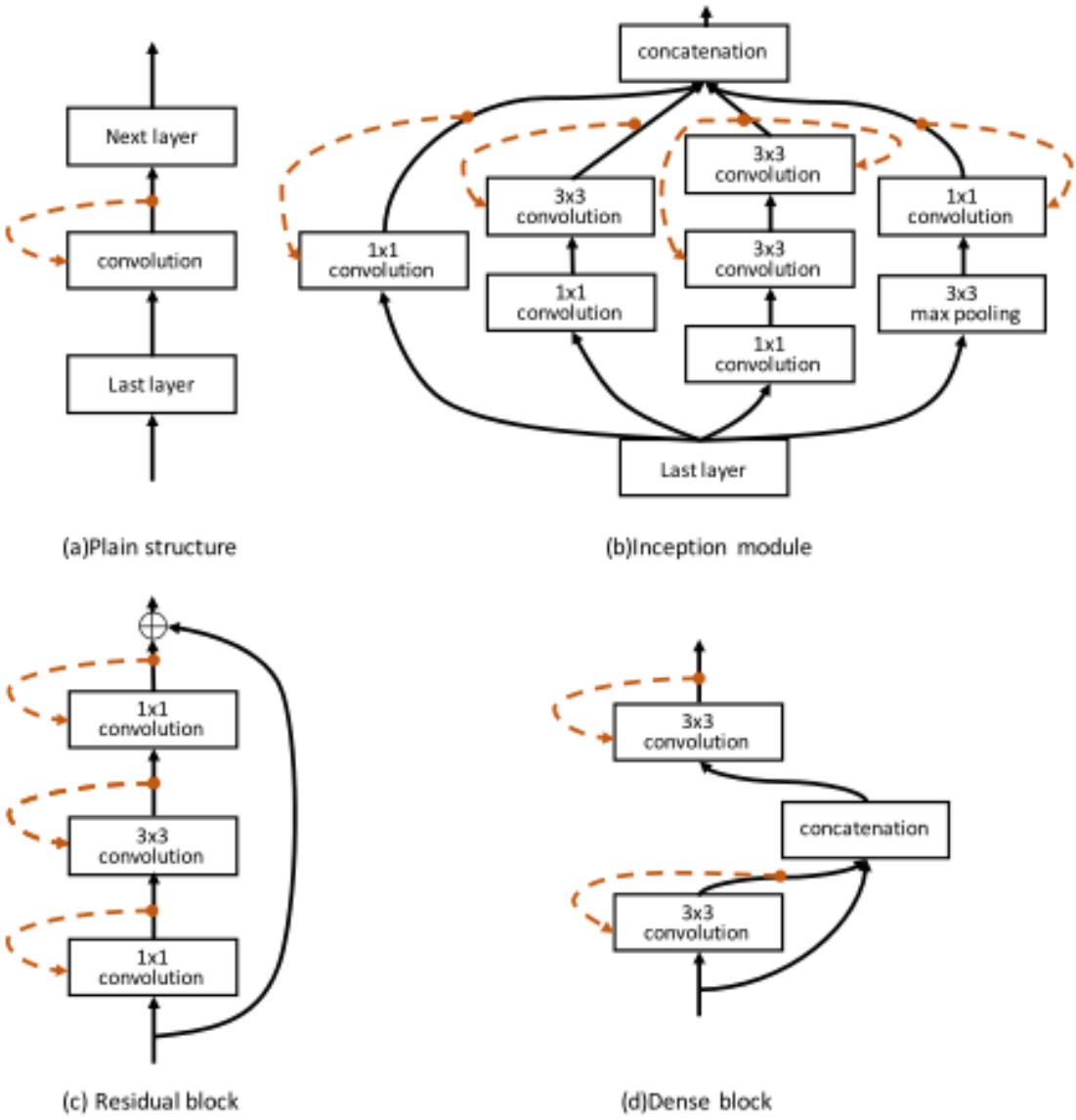

图 4: 修剪的主流网络结构的图示，包括平滑结构、Inception Block、Residual Block 和 Dense Block。黑线表示神经网络的推理流。红点表示卷积层的输出（即特征图）。红色虚线表示在观察要素地图的等级后要修剪的图层。注意，对于 1×1 卷积和 n×n 卷积（（b）中 n ＝ 3），我们不考虑修剪 1×1 滤波器，因为与 n×n 滤波器相比，它包含更少的参数和更少的计算

4 复现细节

4.1 与已有开源代码对比

作者在 https://github.com/lmbxmu/HRank 上公布了 Hrank[1]的代码，其主要是对四种常见网络 VGG16、ResNet、GoogleLeNet、DenseNet 的裁剪与性能测试。在本次复现实验中，首先是将作者多个实现不同功能的文件整合在一个文件，便于阅读与理解，其次是实现了作者在《HRank: Filter Pruning using High-Rank Feature Map》论文[1]中的对比实验，最后是实现了模型中参数量与 Flops 的计算，并与当前最新的其他剪枝方法进行性能对比。

首先是作者在《HRank: Filter Pruning using High-Rank Feature Map》论文[1]中的对比存在着两种相应的对比实验——随机删除和与 HRank 相反的删除，用以体现 HRank 的效果，在作者代码的基础上加了新的类实现该对比实验。其次在作者的具体的实现中，过滤器的裁剪是没有实际删除的，仅仅是将其中的的权值都赋值为 0，所以假如使用现有的 Torchstat、Ptflops、Thop 等第三方库去计算参数量

与 Flops 的话，结果将于原模型的值一样，无法体现作者的裁剪效果。所以为了对应作者过滤器裁剪的处理，相应地实现了对神经网络模型的参数与 Flops 的计算。最后的是与相关工作中的 Thinet[7]裁剪方法进行比对实验，查看该剪枝方法的效果如何。

4.2 实验设置

为了证明在降低模型复杂性方面的效率，在小型数据集和主流卷积神经网络上进行了实验，即 CIFAR-10[6]和 VGG16[3]。前面的图 4 中 Plain structure 即是修剪 VGG16[3]的标准。而在秩的计算中，随机抽样 10 个 Batch 的训练集，以评估每个特征图的平均排名。

在性能测试中，使用采用广泛使用的协议，即参数数量和所需浮点运算（Flops），以评估模型大小和计算需求。而在裁剪效果的中，在 CIFAR-10 上提供了修剪模型的 top-1 精度和修剪率。

使用 PyTorch[9]来实现 HRank 方法。首先使用 Google 提供的预训练模型作为基准，在此基础上进行裁剪与正确率对比，而在每一层裁剪后的与裁剪完成后的 fine-tune 中使用初始学习率为 0.001 的随机梯度下降算法（SGD）来解决优化问题。批量大小、权重衰减和动量分别设置为 32、0.0005 和 0.9。

5 实验结果分析

首先对 HRank[1]、随机删除、Anti-HRank（与 HRank 相反，删除过滤器中秩高的）在不同的压缩率进行测试。结果如下图 5，我们可以发现 HRank 在不同的压缩率下，效果都是最好的，并且随着压缩率的下降，差距越来越明显。Anti-HRank 的效果是最差的，随机删除的效果也很差。随着压缩率的提高，可以看出 HRank 效果的稳定性，并且在与 Anti-HRank 的对比中，我们可以证明 HRank 的作用。

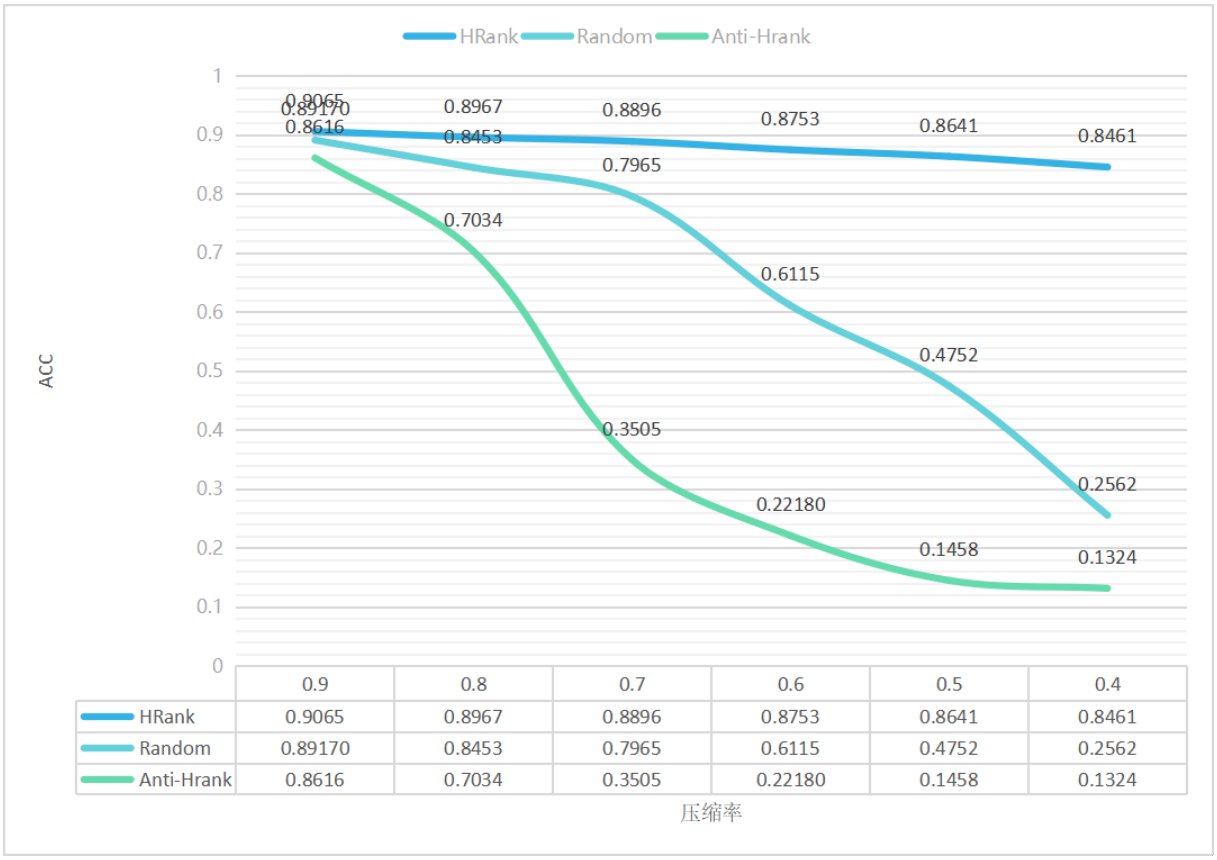

图 5

其次，我们将 HRank[1]与 ThiNet[7]、随机删除在不同的压缩率进行性能测试。我们可以发现 HRank 的效果也是最好的，并且随着压缩率的提高，HRank 的稳定性也是最高的。

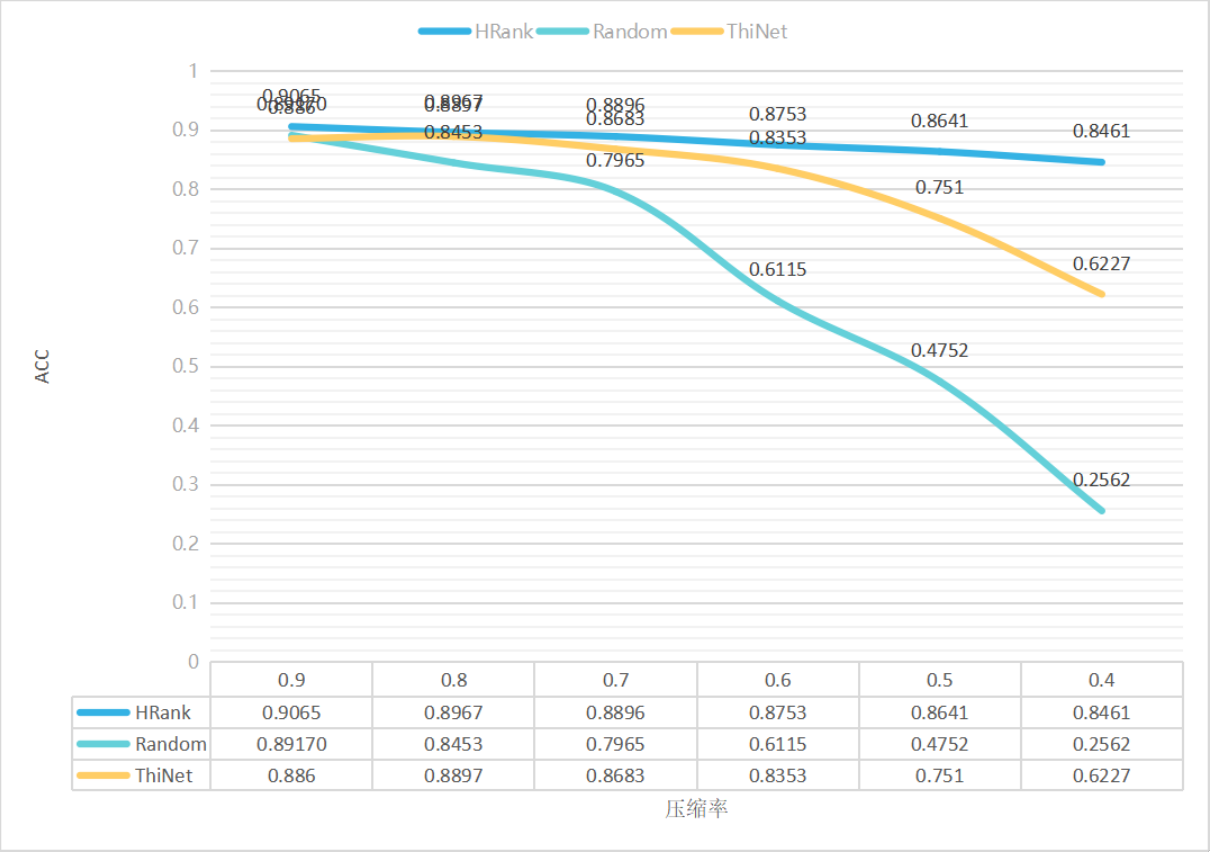

图 6: 方法对比

6 总结与展望

作者提出了一种新的过滤器修剪方法，称为 HRank[1]，该方法通过观察特征图的秩来确定过滤器的相对重要性，这也通过实验得到了验证。此外，在大量统计验证的基础上，证明了单个滤波器生成的特征图的平均秩几乎没有变化。在 VGG16[3]卷积神经网络上的大量实验证明了 HRank 在降低计算复杂性和模型大小方面的有效性。但作者并未解释为什么由单个过滤器生成的特征图的平均秩总是相同的。同时在实验中发现：在神经网络不使用 Relu 的情况下，裁剪效果不太理想。并且每层的裁剪率也是一个超参数，其的设置也需要其他方法的辅助，不然该压缩方法也仅是模型压缩在确定裁剪率的一种次优解。

参考文献

[1] MINGBAO LIN Y W, Rongrong Ji. HRank: Filter Pruning using High-Rank Feature Map[J]. CVPR, 2020, 1.

[2] A. KRIZHEVSKY I S, HINTON G E. Imagenet classification with deep convolutional neural networks [J]. NIPS, 2012, 1: 1097-1105.

[3] SIMONYAN K, ZISSERMAN A. Very deep convolutional networks for large-scale image recognition [J]. ICLR, 2015, 1: 1-14.

[4] CARREIRA-PERPINÁN M A, IDELBAYEV Y. Learning-compression algorithms for neural net pruning [J]. CVPR, 2018, 1.

[5] HAO LI I D, Asim Kadav. Pruning filters for efficient convnets[J]. In-ternational Conference of Learning

Representation (ICLR), 2017, 1.

[6] ALEX KRIZHEVSKY E A, Geoffrey Hinton. Learning multiple layers of features from tiny images[J]. Technical report, 2009, 2,5.

[7] JIAN-HAO LUO W L, Jianxin Wu1. ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression[J]. CVPR, 2017.

[8] M. DENIL L D, B. Shakibi. Predicting parameters in deep learning[J]. NIPS, 2013, 2.

[9] ADAM PASZKE S C, Sam Gross. Automatic differentiation in pytorch[J]. Neural Information Processing Systems (NeurIPS), 2017, 5.