Software Engineering and Applications 软件工程与应用, 2021, 10(4), 595-605
Published Online August 2021 in Hans. http://www.hanspub.org/journal/sea
https://doi.org/10.12677/sea.2021.104064

Hans汉斯

贪心剪枝：无需搜索的自动剪枝算法

王　勇[1*]，戚浩金[1]，刘　宇[2]，琚小明[2#]

[1]国网宁波供电公司，信息通信分公司，浙江　宁波
[2]华东师范大学，软件工程学院，上海
Email: 3614149@qq.com, qihaojin1987@163.com, 1290109033@qq.com, #xmju@sei.ecnu.edu.cn

收稿日期：2021年6月22日；录用日期：2021年8月23日；发布日期：2021年8月30日

摘　要

运行功能强大的神经网络需要消耗大量的存储空间和计算资源，这对资源有限的移动设备和嵌入式设备是无法接受的。针对这个问题，本文基于贪心策略提出了一个高效的GreedyPruner算法用来自动修剪网络模型。该算法首先预训练一个超网络，这个超网络可以预测任意给定网络结构的性能；其次，引入精度队列和压缩池分别保存性能较好和剪枝率较高的网络结构，提出贪心训练策略对超网络进行二次训练，将训练空间从全体解空间贪婪地转移到精度队列和压缩池中；最后，取精度队列和压缩池中的最优网络结构进行权值微调，得到修剪后的网络模型。实验结果表明，GreedyPruner可以在网络性能几乎不变的情况下，大幅压缩模型的参数和运算量，压缩后的网络模型更有利于部署在移动设备和嵌入式设备中。

关键词

神经网络模型修剪，自动剪枝，超网络，贪心训练

GreedyPruner: Automatic Pruning Algorithm without Searching

Yong Wang[1*], Haojin Qi[1], Yu Liu[2], Xiaoming Ju[2#]

[1]Information and Communication Branch, State Grid Ningbo Power Supply Company, Ningbo Zhejiang
[2]School of Software Engineering, East China Normal University, Shanghai
Email: 3614149@qq.com, qihaojin1987@163.com, 1290109033@qq.com, #xmju@sei.ecnu.edu.cn

Received: Jun. 22nd, 2021; accepted: Aug. 23rd, 2021; published: Aug. 30th, 2021

Abstract

A powerful neural network needs to consume a lot of storage space and computing resources,

*第一作者。
#通讯作者。

文章引用: 王勇, 戚浩金, 刘宇, 琚小明. 贪心剪枝: 无需搜索的自动剪枝算法[J]. 软件工程与应用, 2021, 10(4): 595-605. DOI: 10.12677/sea.2021.104064

which is unacceptable for mobile devices and embedded devices with limited resources. In response to this problem, this paper proposes an efficient GreedyPruner algorithm based on the greedy strategy to automatically prune the network model. The algorithm first pretrains a SuperNet, which can predict the performance of any given network structure; secondly, the precision queue and the compression pool are introduced to preserve the network structure with better performance and higher pruning rate respectively, and the greedy training strategy is proposed to train the SuperNet work twice, and the training space is greedily transferred from the whole solution space to the precision queue and the compression pool; finally, take the optimal network structure in the precision queue and the compression pool to fine-tune the weights to obtain the pruned network model. Experimental results show that GreedyPruner can greatly compress the parameters and calculations of the model while the network performance is almost unchanged. The compressed network model is more conducive to deployment on mobile devices and embedded devices.

Keywords

Neural Network Model Pruning, Automatic Pruning, SuperNet, Greedy Training

1. 引言

近年来，随着深度学习算法研究的不断深入，深度神经网络模型已在各行各业取得了显著的成效。虽然，这些训练有素的网络性能强大，但运行如此庞大的网络需要消耗大量的存储空间和带宽。例如，ResNet152 拥有超过 6000 万个参数，在推断 224 × 224 的图像时需要超过 20 千兆个浮点运算，这在资源受限的移动端和嵌入式设备上，是无法接受的。为了减小保存模型需要的内存以及进行推理需要的计算量，使深度神经网络能更好地应用到移动设备和嵌入式设备，网络剪枝成为研究的主要方向。网络剪枝方法主要分为两类：传统的手工剪枝和基于 AutoML 的自动剪枝。现有的网络剪枝方法已经取得了较多的研究成果，但仍然不能在精度几乎不变的情况下大幅压缩模型的参数量(Parameters)和计算量(FLOPs)。

传统的手工剪枝需要领域专家设计规则判断卷积核通道的重要性，修剪不重要的卷积核通道。Song Han 等[1] [2]提出以权值矩阵的范数衡量神经元的重要性，删除范数小于阈值的神经元，将原始的密集网络变成稀疏网络达到网络剪枝的目的。Hao Li 等[3]以权重绝对值之和来衡量卷积核通道的重要性，裁剪数值低于阈值的卷积核通道。Zhuang Liu 等[4]提出使用 BN 层的可学习参数 γ 衡量卷积核通道的重要性。JianHao Luo 等[5]和 Yihui He 等[6]都通过最小化特征重建误差来确定需要裁剪的通道。上述方法都依赖领域专家设计的规则，同时需要大量的实验才能确定网络模型每一层的剪枝率，剪枝后模型的精度损失严重，无法大幅压缩参数量和计算量。

基于 AutoML 的自动剪枝几乎不需要领域专家的参与，可以自动修剪网络模型，逐渐成为网络剪枝的主流方法。Yihui He 等[7]用强化学习中的 DDPG 算法自动探索模型每一层的剪枝率，将领域专家从繁琐的实验中解放。Ning Liu 等[8]将 ADMM [9] [10]结构化剪枝算法作为核心，采用智能搜索算法代替的 DDPG 自动压缩模型。Shaohui Lin 等[11]采用对抗生成学习的思想，提出训练一个 GAN 网络[12]，通过 genetator 生成压缩后的网络。上述基于 AutoML 的自动剪枝虽然都可以自动确定模型每一层的剪枝率，但是需要维护大量的超参数或者训练额外的辅助网络，算法开销大，实现困难。

与本文工作比较类似的是 MetaPruning [13]和 GreedyNas [14]。MetaPruning 通过训练 PruningNet 预测给定网络结构的权重，然后使用遗传算法搜索最优的网络结构，最后将搜索到的网络结构进行重训练得到压缩后的网络模型。但是，PruningNet 参数多且结构复杂，PruningNet 的训练会花费大量的时间和计算资源。并且 PruningNet 训练完成后预测的是网络结构的权重，还需要再进行一次前向传播才能得到给定网络结构的性能。GreedyNas 虽然可以直接预测网络结构的性能。但是在网络剪枝任务中，上一个卷积层的通道数会对下一个卷积层的通道产生影响，这会导致 GreedyNas 无法应用在网络剪枝任务中。同时，GreedyNas 在超网络训练结束后，使用智能搜索算法搜索最优的网络结构，这也花费了大量的时间。

为了解决上述问题，本文提出了 GreedyPruner 算法。该算法首先设计三种共享通道块用来构建超网络，然后基于平等策略对超网络进行预训练；其次，引入精度队列和压缩池分别保存性能较好和剪枝率较高的网络结构，提出贪心训练策略对超网络进行二次训练，二次训练结束后可直接从精度队列和压缩池中获得较优的网络结构，不需要任何搜索算法，节约了大量的计算资源。

实验表明：在 3 个评价指标准确率、参数量剪枝率和计算量剪枝率上，GreedyPruner 可以在准确率几乎不变的情况下大幅降低模型的参数量计算量。与以往的算法相比，本文的主要贡献为：

1) 设计三种共享通道块用来构建超网络，通过平等训练策略对超网络进行预训练，预训练完成后，仅通过一次前向传播就可以预测给定网络结构的性能。

2) 提出贪心训练策略对超网络进行二次训练，二次训练完成后，可以直接从精度队列和压缩池中得到较优的网络结构，避免了使用智能搜索算法搜索全体解空间。

3) 提出的算法对多种网络模型都有效，包括具有短接结构的网络结构(ResNet，MobileNet)，相比于 SOTA (state of the art)方法，本文的算法可以在网络性能损失很少的情况下，达到更高的参数量和计算量的剪枝率。

2. 相关工作

为了更好地介绍 GreedyPruner 算法，本章将描述剪枝空间和共享通道块。首先介绍如何对解空间进行压缩，然后介绍针对不同的网络结构设计的三种共享通道块。

2.1. 剪枝空间的限制

给定一个卷积神经网络 $M\left(c_1,c_2,\cdots,c_l;W\right)$，我们定义 $C=\left(c_1,c_2,\cdots,c_l\right)$ 为 M 的网络结构，其中：c_i 为第 i 层卷积核的通道数，W 是卷积核权重。对于网络 M，剪枝结构共有 $\prod_{i=1}^{l}c_i$ 种可能组合，在超网络二次训练过程中，维护如此大的剪枝空间是极耗资源的。受 Mingbao Lin 等[15]压缩搜索空间的启发，本文对剪枝空间做了限制。

本文将每一层卷积核可保留通道数限制在 αc_i，其中 $\alpha\in\{10\%,20\%,\cdots,100\%\}$，修剪后的网络结构为 $C'=\left(\alpha_1c_1,\alpha_2c_2,\cdots,\alpha_lc_l\right)$，这表明每层都只有 10 个可行解，每个解中都只有 α_ic_i 个通道被保留，这样剪枝空间便从 $\prod_{i=1}^{l}c_i$ 降低到 $\prod_{i=1}^{l}10$。假设一个神经网络共 10 层，每层的卷积核通道数为 32，则剪枝空间就由原始的 32^{10} 变为 10^{10}，这极大的增加了算法的运行效率。

2.2. 共享通道块的设计

为了使 GreedyPruner 可以用在各种常见的网络结构中，本文设计了三种不同的通道块。如图 1 所示，红色字体标记的代表需要修剪的部分。通道块(a)由 Conv，BN 层以及激活函数 ReLu 组成，适用在 LeNet，VGG 和 MobileNet 这类没有短接结构的网络中。通道块(a)的最大输入通道数和输出通道数是固定不变的，真实的输入随着上一卷积层的输出通道数改变而改变，可以通过卷积核的切片很容易实现这个功能。

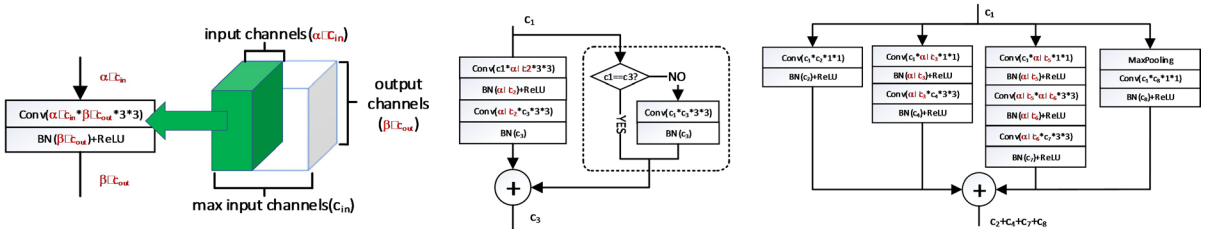

Figure 1. Three types of shared channel blocks
图 1. 三种共享通道块

通道块(b)适用在 ResNet，UNet 等具有短接的网络结构中。我们规定通道块(b)的输入通道数和输出通道数固定不变，只有中间层的通道数会发生改变，因此，卷积核的短接结构并不会发生改变。通道块(c)适用于 GoogLenet 这类网络结构，只有两侧的 1*1 卷积核不会被修剪，其余部分的卷积核通道数均会发生改变。在使用 GreedyPruner 对网络模型进行剪枝时，我们可以根据网络结构，选择合适的通道块构建超网络。通过设计三种通道块，本文提出的 GreedyPruner 算法可以用于剪枝各种常见的网络模型。

3. GreedyPruner

假设待剪枝的深度卷积神经网络共有 l 层，本文将模型压缩问题定义为公式(1)：

$$\begin{cases} \left(c_1,c_2,\cdots,c_l\right)^* = argmax\ acc\left(M\left(c_1,c_2,\cdots,c_l;W\right);T_{test}\right), \\ c_i \in \left\{\alpha c_i \mid \alpha \in \left\{10\%,20\%,\cdots,100\%\right\}\right\}, \\ s.t\ Cost\left(c_1,c_2,\cdots,c_l\right) < constraint. \end{cases} \tag{1}$$

其中，M 是通道数为 $\left(c_1,c_2,\cdots,c_l\right)$ 的网络模型，W 是卷积核权重，*constraint* 硬件限制。网络剪枝的目的是在硬件限制为 *constraint* 的情况下，获得较优的网络模型。然而，直接对公式(1)求解是困难的，且容易陷入局部最优解。为了解决这个问题，本文提出了 GreedyPruner，该算法可以将公式(1)的优化问题巧妙地转化为 SuperNet 的训练。训练后，可以直接从精度队列和压缩池中获得较优的网络结构，而无需任何搜索算法。整个过程简单、高效、可以获得较高的剪枝率。

3.1. SuperNet 的结构

为了对公式(1)进行求解，本文构造了一个 SuperNet。共享通道块是构建 SuperNet 的基本结构，图 1 给出了不同网络结构的共享通道块构建方式。如 2.1 节所述，本文对解空间做了限制。在设计 SuperNet 时为卷积层的每一个可行解都设计一个共享通道块，共 10 个。分别代表保留原卷积核通道数的 10%，20%，……，100%。在同一时间内每个卷积层只能有一个共享通道块被激活。如图 2 所示，对 SuperNet 的每一个卷积层都进行一次通道块采样便可得到一条路径。图 2 中所有的红色通道块组成一条路径，所有的黄色通道块组成另一条路径。每条路径代表解空间中的一个网络结构。对于一个 L 层的网络模型，通过共享通道块，我们仅仅需要使用 10 L 个通道块，就可以表示全体解空间。

网络构建完成后，SuperNet 的输入为代表网络结构的编码向量，通过一次推理就可以得到该网络结构在验证集上的准确率，如公式(2)所示。因为不需要重训练，所以可以通过 SuperNet 快速预测任一网络结构对应的性能。

$$acc = SuperNet\left(c_1,c_2,\cdots,c_l;T_{val}\right) \tag{2}$$

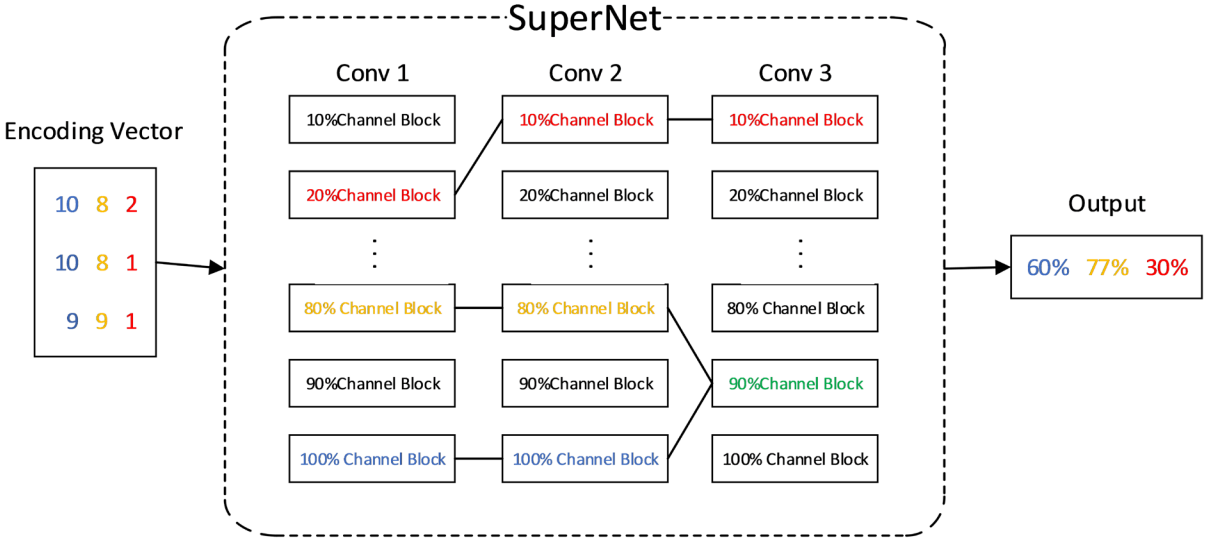

Figure 2. The network structure of SuperNet
图 2. SuperNet 的网络结构

3.2. SuperNet 的预训练

本文希望通过 SuperNet 预测的网络性能与真实的网络性能差距越小越好，这意味着 SuperNet 中通道块的权值应尽可能的接近真实的权值。在训练过程中，SuperNet 中的每一个通道块都被平等的训练。因此本文提出了路径平等策略预训练 SuperNet。

在前向传播中，随机生成一个编码向量(表示网络结构的每层通道数)作为 SuperNet 的输入，与编码向量对应的通道块被激活，其他通道块处于失活状态，每训练一个 batchsize，编码向量更新一次。在反向传播中，SuperNet 仅更新所选路径上通道块的权值。其它通道块的权值并不发生改变。通过这种方法，迭代一定次数后，所有的通道块都会得到平等的训练。即使某条路径没有被训练过，但是构成这条路径的通道块也一定在以往的迭代中被训练。

预训练完成后，此时的 SuperNet 不在是传统意义上的网络，而是在特定解空间下的网络性能评估器。SuperNet 就相当于一个有向无环图，图中的每一条路径代表压缩后的网络结构，在验证集上对网络性能进行评估时，只需要一次前向传播，就可以大致预测网络性能，省去了传统方法的重训练。

3.3. SuperNet 的二次训练

为了使 SuperNet 能够对解空间中的所有路径进行粗略的评估，本文基于通道块平等策略提出随机采样路径的方法对 SuperNet 预训练。但是，网络剪枝的目的是满足一定约束条件下，在解空间中找到性能最优的网络结构。因为 SuperNet 是基于共享通道块的，对性能较差路径上的通道块进行训练可能会反过来影响性能较好路径上的通道块。因此本文又提出对 SuperNet 进行二次训练，阻止性能较差路径的训练，贪婪的训练性能较好的路径。本文在路径评估时引入平衡因子，在模型剪枝率和准确率上做了平衡，防止准确率很高但压缩率很低这一现象的出现。二次训练结束后，可以直接从候选队列和压缩池中获得最优的网络结构，不需要使用智能优化算法进行搜索。

在二次训练中，网络结构的评估函数为：

$$pre = \mu * acc + (1-\mu) * ratio \tag{3}$$

其中：*ratio* 为修剪后网络结构的压缩率。在公式(3)中引入参数 μ 作为平衡因子，为算法提供两种压缩模式；

1) 高精度模式($\mu \geq 0.5$)：压缩后网络结构的性能以精度为主，允许较小压缩率的网络结构出现。

2) 高压缩率模式($\mu < 0.5$)：压缩后网络结构的性能以剪枝率为主，允许有较大的精度损失。

假设解空间中所有网络结构的性能已知，对所有网络性能进行排序，并定义前 k 条路径为好路径(good path)，剩下的路径为差路径(bad path)。进行二次训练的目的是对 SuperNet 中的好路径进行训练，拒绝差路径的训练需求。但是，并不知道哪些路径属于好路径，使用公式(2)对超网络的每一条路径都求解显然是不可接受的。因此本文创建精度队列，通过衰减的 ϵ -greedy 算法从 SuperNet 中对路径进行采样，将每次采样中的前 k 条路径放入精度队列中，使精度队列可以近似代表好路径。

初始化时，精度队列为空，从 SuperNet 中随机采样 m 条路径，对这 m 条路径进行评估，根据评估结果进行排序，将前 k 条路径当作好路径放入候选池中。评估时从测试集中固定选取 10%样本，对 m 条路径进行测试，因为只选取测试集样本的 1/10，所以基本不会对训练速度造成影响。从第二轮迭代开始，精度队列中已经存在路径，我们认为精度队列的路径都是好路径。但为了保证可以对 SuperNet 中的路径进行充分探索，本文以 ϵ 的概率从 SuperNet 中抽取路径，以 $(1-\epsilon)$ 的概率从精度队列中抽取路径，ϵ 的值随着训练的进行动态改变。初始时候选队列为空，$\epsilon = 1$，随着训练的进行，精度队列不断向好路径空间靠拢，此时，减小 ϵ 的值，贪婪的训练精度队列中的好路径，当 ϵ 减少到一定程度时，ϵ 的值不再发生改变。随着训练的进行，我们就会将主要的训练从全体搜索空间贪婪地转移到候选队列中。

考虑到硬件的要求，精度队列并不是无限大的。在迭代的过程中将 k 条路径加入精度队列，会出现两种情况：1) 精度队列未满，这时我们取 k 条路径中不在精度队列里出现的路径，加入精度队列；对原先已经存在精度队列中路径的评估值进行更新；2) 若候选队列已满，对精度队列中性能不佳(评估值较低)的后 k 条路径进行更新，删除原有的路径加入新的路径。

在 GreedyPruner 中，允许高压缩率但精度稍微降低的网络架构的存在，为了更好地探索较高压缩率的网络架构，我们设置压缩池，每轮训练时，将 m 条路径中压缩率最高的两条路径放入压缩池中，每隔 n 轮对压缩池中的路径进行训练。这样可以保证高压缩率的路径也会得到充分训练。与以往的剪枝算法不同，二次训练结束后，只需要从候选队列和压缩池中取性能最好的网络结构进行权值微调，便可得到最终剪枝的网络模型，不在需要使用智能搜索算法。GreedyPruner 整个过程，简单高效，不在需要人类专家的干预，得到的修剪模型也是较优的。

3.4. GreedyPruner 的整体流程

GreedyPruner 的算法流程如下所示：

1、根据要剪枝的网络模型，搭建 SuperNet；

2、对 SuperNet 进行预训练；

3、对 SuperNet 进行二次训练；

4、从精度队列和压缩池最性能最优的网络结构，进行权值微调得到最终的修剪模型。

更具体地说：

SuperNet 的预训练：

输入： T_{data} 为训练集， $C=\{c_1,c_2,\cdots,c_i\}$ 为网络结构的编码向量。
参数设置： W 为卷积层权值，batchsize 为批次大小，itr 为当前迭代次数，$itrs$ 为最大迭代次数。
初始化： $itr=0$，W 使用随机正态分布初始化。
过程：
1、**for** $itr < itrs$ **do**；

Continued

2、随机初始化网络结构 C，从 T_{data} 中随机抽 batchsize 个样本数据；
3、激活 C 对应的共享通道块，其它通道块处于失活状态；
4、计算 C 这条路径的损失函数；
5、使用 Adam 优化器更新权重参数：
6、**end for**

SuperNet 的二次训练：

输入： T_{data} 、T_{val} 为训练集、验证集，$C=\{c_1,c_2,\cdots,c_l\}$ 为网络结构的编码向量。
参数设置： W 为卷积层权值，batchsize 为批次大小，itr 为当前迭代次数，$itrs$ 为最大迭代次数，m 为每次随机采样路径的数量，k 为每次好路径的数量，n 为每隔 n 轮训练压缩池，ϵ 为从解空间选取路径的概率。
过程：
1、**for** $itr < itrs$ **do**；
2、从解空间中随机选取 $m\cdot\epsilon$ 条路径，从精度队列中随机选取 $m\cdot(1-\epsilon)$ 条路径；
3、根据公式(3)对 m 条路径进行评估，取前 k 条路径加入精度队列，取剪枝率最高的路径加入压缩池，丢弃剩余的坏路径；
4、在本轮训练中，对这 k 条路径对应的网络结构均匀训练，训练结束后加入 k 条路径加入精度队列；
5、如果 $itr\%n=0$，从压缩池中取 m 条路径训练；
6、更新 ϵ；
7、**end for**

4. 实验与分析

本文在 mnist 和 cifar10 数据集上对 lenet5，vgg，resnet32 和 resnet56 进行实验来验证 GreedyPruner 的有效性。在这一章节里，首先详细描述了实验中 SuperNet 的参数设置；然后，将本文得到的结果与 SOTA 方法进行了比较；最后，探索了平衡因子对较优网络结构的影响。

4.1. 实验设置

本文的实验资源是 NVIDIA Tesla P40 GPU，使用 pytorch 来实现 GreedyPruner。为了保证剪枝后的网络具有泛化性，作者对 mnist 和 cifar10 的训练集进行重新划分，取原本训练集的 90%作为新的训练集，剩余 10%作为验证集。使用验证集对 SuperNet 采样得到的路径进行性能评估。

对于 SuperNet 的预训练，本文设置 batchsize 为 256，初始学习率为 0.1，学习率衰减率为 0.1，动量大小为 0.9 不设置衰减。在 mnist 数据集上，预训练共迭代 300 轮，每 50 个回合学习率衰减一次，150 回合后学习率大小保持不变。在 cifar10 数据集上，预训练共迭代 1000 轮，每迭代 50 个回合学习率衰减一次，600 回合后学习率大小保持不变。

在 SuperNet 的二次训练过程中，主要通过精度队列和压缩池对好路径进行保存。本文设置精度队列大小为 300，压缩池大小为 150，ϵ 随着训练的进行呈线性衰减，衰减到 0.2 时保持不变。每轮训练时，共选取 30 条路径进行评估，取评估值较高的前 10 条路径当作好路径，加入精度队列。压缩池 20 轮迭代训练一次。在后面的实验中，我们默认选择平衡因子为 0.5，关于不同的平衡因子对剪枝后模型的影响，我们将在 4.3 节详细讨论。

4.2. 实验结果

本文首先使用 GreedyPruner 在 mnist 数据集上对 LeNet5 进行剪枝，结果如表 1 所示。GreedyPruner 在 LeNet5 上可以在精度损失仅为 0.80%的情况下，移除 82.39%的卷积核通道数、63.98%的参数量和 92.67%的计算量。

为了更全面地分析 GreedyPruner 的有效性，本文在 cifar10 数据集上，使用 GreedyPruner 对三个广泛使用的网络(VGG16、ResNet34 和 ResNet56)进行了网络剪枝，修剪结果已在表 1 中展示。

Table 1. Pruning results using GreedyPruner on LeNet5, VGG16, ResNet34 and ResNet56
表 1. GreedyPruner 在 LeNet5、VGG16、ResNet34 和 ResNet56 上的剪枝结果

模型	TOP-1 准确率/+微调	通道数/通道数剪枝率	计算量/计算量剪枝率	参数量/参数量剪枝率
原始 LeNet5	99.20%	142	61706	150,110
GreedyPruner-0.5	97.25%/98.40%	25/82.39%	4526/92.67%	54,068/63.98%
原始 VGG16	93.56%	4224	317.44 M	15.25 M
GreedyPruner-0.5	88.90%/92.33%	1378/67.38%	71.68 M/77.42%	1.46 M/90.13%
原始 ResNet34	94.10%	8512	1187.84 M	21.28 M
GreedyPruner-0.5	93.79%/94.26%	5651/33.61%	491.52 M/68.10%	6.15 M/73.85%
原始 ResNet56	94.40%	26,560	1341.44 M	23.52 M
GreedyPruner-0.5	91.83%/93.60%	21,300/19.80%	471.04 M/64.89%	6.15 M/73.85%

Table 2. Comparison result of GreedyPruner and SOTA method on VGG16
表 2. GreedyPruner 与 SOTA 方法在 VGG16 上的比较结果

	方法	TOP-1 准确率/+微调	计算量/计算量剪枝率	参数量/参数量剪枝率
原始 VGG16		93.56%	0	0
传统的手工剪枝	L1 [3]	93.40%	35.11%	64.59%
	Sliming [4]	93.66%	50.94%	84.92%
基于 AutoML 的自动剪枝	ABCpruner-80% [15]	93.08%	73.91%	89.04%
	GAL-0.1 [12]	90.78%/93.42%	45.85%	82.49%
	GreedyPruner-0.5	88.90%/92.33%	77.42%	90.13%

使用本文提出的算法对原始的 VGG16 网络(13 个卷积层，3 个全连接层)进行剪枝，剪枝后的网络相比于原网络减少了 67.38%的卷积核通道，77.42%的计算量和 90.13%的参数量，但是在测试集上的 Top-1 精度仍可以达到 88.90%。相比于参数量和计算量的大幅降低，我们认为 4.5%的精度损失是可以接受的。为了弥补剪枝后网络结构的精度损失，对修剪后的网络进行权值微调。通过微调可以将精度提高到 92.33%，相比于原始 VGG 只降低了 1.23%。同时，本文也对 GreedyPruner 在 VGG16 上每一层的剪枝率进行了分析，结果如图 3 所示。图 3 显示，GreedyPruner 对 VGG16 的前几层保留了较多的通道和参数量，从 Conv8 开始参数量修剪率明显提高。这是因为 VGG16 的前几层主要用于特征提取，保留较多的通道和参数有助于压缩后的模型维持高准确率，而网络的后几层在设计时本身就包含大量的冗余通道，删除这些冗余不仅可以有效地防止过拟合，同时还能加快网络训练。

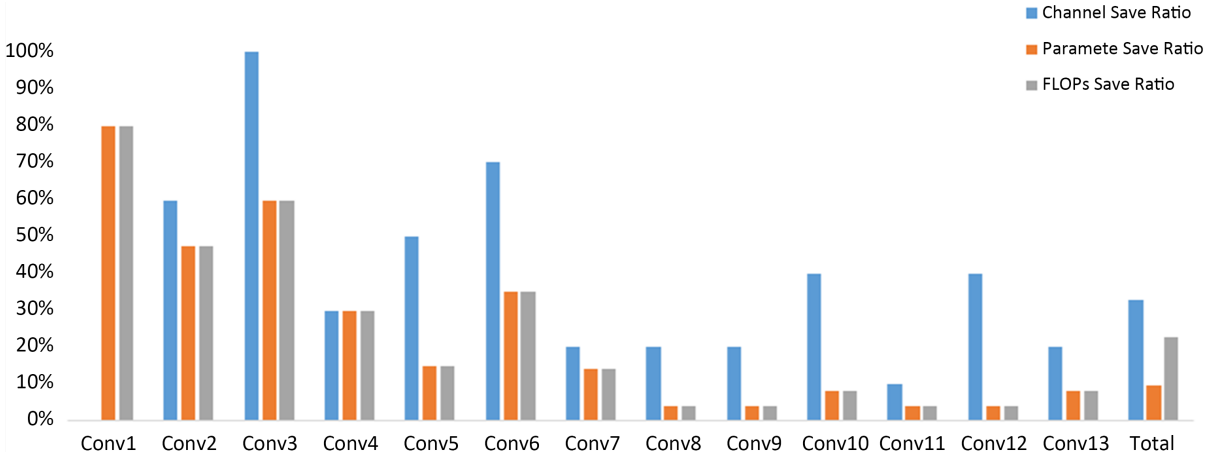

Figure 3. The pruning rate of each layer of VGG16 by GreedyPruner
图 3. GreedyPruner 在 VGG16 上每一层的剪枝率

Table 3. Comparison result of GreedyPruner and SOTA method on ResNet56
表 3. GreedyPruner 与 SOTA 方法在 ResNet56 上的比较结果

	方法	TOP-1 准确率/+微调	计算量/计算量剪枝率	参数量/参数量剪枝率
原始 ResNet56		94.40%	0	0
传统的手工剪枝	L1	94.20%	27.56%	14.12%
	ThiNet50 [5]	93.56%	66.11%	73.62%
基于 AutoML 的自动剪枝	ABCpruner-70%	93.55%	56.61%	57.14%
	MetaPruning-0.5 [13]	93.08%	56.72%	60.28%
	GAL-0.5	92.92%/93.10%	48.52%	44.76%
	GreedyPruner-0.5	91.83%/93.60%	64.89%	73.85%

将 GreedyPruner 的剪枝结果与其他方法相比，比较结果如表 2 所示。在 VGG16 上，GreedyPruner 相比于传统的剪枝算法 L1 和 sliming，可以获得更高的计算量剪枝率和参数量剪枝率。例如，L1 可以修剪 35.11%计算量和 64.59%参数量，但是 GreedyPruner 在精度只降低 1.23%的情况下，将计算量修剪率从 35.11%提高到 77.42%，将参数量修剪率从 64.59%提高到 90.13%。这是因为 GreedyPruner 可以自动寻找最优的网络结构，将人类专家与网络剪枝任务解耦合，降低了主观经验对压缩结果的干预。即使与最近的自动剪枝算法 GAL 相比，GreedyPruner 在精度降低只有 1.09%的情况下，将计算量剪枝率提高了 31.57%，参数压缩率提高了 7.64%，修剪后的模型可以更好地应用在移动端和嵌入式端。

对于短接网络，本文使用 GreedyPruner 修剪了 ResNet34 和 ResNet56 这两种不同深度的网络结构。表 1 展示了对这两种网络结构的修剪结果。对于 ResNet34，GreedyPruner 可以移除 33.61%的通道、68.10%的计算量和 79.89 的参数量。剪枝后的 ResNet34 在不进行微调的情况下可以达到 93.79%的 Top-1 精度，微调后精度相比于原始的 ResNet34 提高了 0.16%。但是，将 GreedyPruner 应用在 ResNet56 上，性能便有所降低。相比于 ResNet34，ResNet56 的计算量剪枝率降低 3.21%，参数量剪枝率降低 6.04%并且 Top-1 精度降低 1.77%。这是因为 ResNet56 有更深的网络结构和更多的参数量，判断冗余结构的难度大大增加。表 3 展式了 GreedyPruner 和其他算法在 ResNet56 上的比较结果。与传统的剪枝算法 L1 和 ThiNet50 相比，GreedyPruner 可以达到更高的计算量和参数量修剪率。与自动剪枝算法相比，GreedyPruner 可以获得更高的计算量和参数量修剪率，在权值微调后，Top-1 精度也会高于以往的自动剪枝算法。

通过上述分析可以得到，GreedyPruner 可以自动修剪网络模型，并且在精度损失很少的情况下达到更高的 FLOPs 和参数修剪率。相比于以往提出的算法，GreedyPruner 更具竞争力。

4.3 比例因子 μ 对压缩结果的影响

本文在 cifar10 上对 VGG16 进行修剪来探讨比例因子 μ 对网络剪枝结果的影响。修剪结果如图 4 所示。从图中可以明显的看出，随着 μ 的增大，模型的参数和计算量剪枝率降低，但模型在测试集上的 TOP-1 精度增高。这是容易理解的，随着 μ 的增加，越来越多的卷积核通道被保留。同时，剪枝率的降低会使模型更充分的提取图像特征，导致精度升高。为了在精度和压缩率上做出平衡，本文所有的实验都将 μ 设置为 50%。

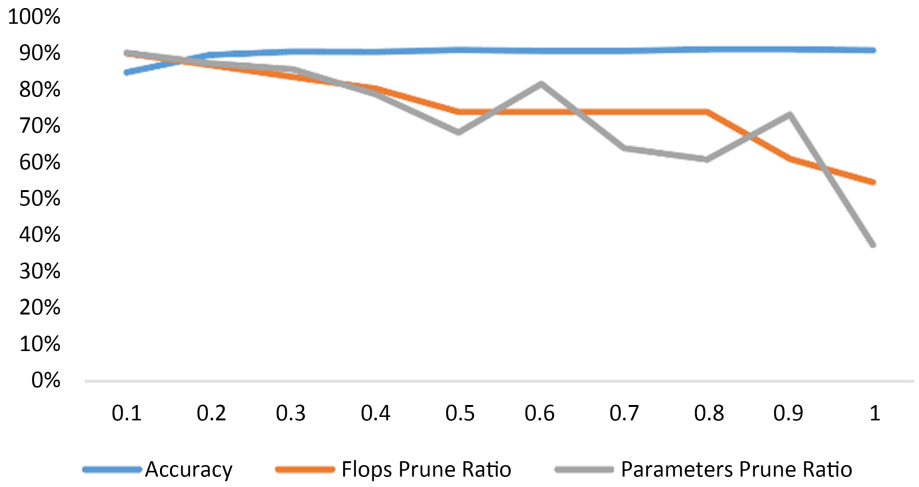

Figure 4. The effect of μ on pruning results

图 4. 平衡因子 μ 对修剪结果影响

5. 结束语

本文提出了无需搜索的自动剪枝算法 GreedyPruner。该算法可以自动探索每一卷积层的剪枝率，将人类专家从繁琐的规则设计中解放。对于 VGG16，GreedyPruner 可以移除 67.38%的卷积核通道和 90.13%的参数，而在 cifar10 上精度只下降了 1.23%；对于 ResNet56，GreedyPruner 可以移除 19.80%的通道，64.89%的参数量以及 73.85%的计算量，精度下降 0.80%。剪枝后的网络体积明显减小，运行速度加快，更适用于部署在移动设备和嵌入式设备上。

基金项目

国网浙江省电力有限公司科技项目：基于边缘计算与小样本学习的人工智能算法模型研究(NO. 5211NB1900VN)。

参考文献

[1] Han, S., Pool, J., Tran, J., *et al*. (2015) Learning both Weights and Connections for Efficient Neural Networks. *NIPS*15: *Proceedings of the* 28*th International Conference on Neural Information Processing Systems*, Vol. 1, December 2015, 1135-1143.

[2] Han, S., Mao, H. and Dally, W.J. (2015) Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding.

[3] Li, H., Kadav, A., Durdanovic, I., *et al*. (2016) Pruning Filters for Efficient ConvNet. *International Conference on*

Learning Representations 2017, Toulon, France, 24-26 April 2017.

[4] Liu, Z., Li, J., Shen, Z., *et al*. (2017) Learning Efficient Convolutional Networks through Network Slimming. *Proceedings of the IEEE International Conference on Computer Vision*, Venice, 22-29 October 2017, 2736-2744. https://doi.org/10.1109/ICCV.2017.298

[5] Luo, J.H., Wu, J. and Lin, W. (2017) Thinet: A Filter Level Pruning Method for Deep Neural Network Compression. *Proceedings of the IEEE International Conference on Computer Vision*, Venice, 22-29 October 2017, 5058-5066. https://doi.org/10.1109/ICCV.2017.541

[6] He, Y., Zhang, X. and Sun, J. (2017) Channel Pruning for Accelerating Very Deep Neural Networks. *Proceedings of the IEEE International Conference on Computer Vision*, Venice, 22-29 October 2017, 1389-1397. https://doi.org/10.1109/ICCV.2017.155

[7] He, Y., Lin, J., Liu, Z., *et al*. (2018) AMC: Automl for Model Compression and Acceleration on Mobile Devices. *Proceedings of the European Conference on Computer Vision* (*ECCV*), Munich, 8-14 September 2018, 784-800. https://doi.org/10.1007/978-3-030-01234-2_48

[8] Liu, N., Ma, X., Xu, Z., *et al*. (2020) AutoCompress: An Automatic DNN Structured Pruning Framework for Ultra-High Compression Rates. *Proceedings of the AAAI Conference on Artificial Intelligence*, **34**, 4876-4883. https://doi.org/10.1609/aaai.v34i04.5924

[9] Boyd, S., Parikh, N., Chu, E., *et al*. (2011) Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. Now Foundations and Trends. https://doi.org/10.1561/9781601984616

[10] Zhang, T., Zhang, K., Ye, S., *et al*. (2018) ADAM-ADMM: A Unified, Systematic Framework of Structured Weight Pruning for DNNS.

[11] Lin, S., Ji, R., Yan, C., *et al*. (2019) Towards Optimal Structured CNN Pruning via Generative Adversarial Learning. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Long Beach, 15-20 June 2019, 2790-2799. https://doi.org/10.1109/CVPR.2019.00290

[12] Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., *et al*. (2014) Generative Adversarial Networks.

[13] Liu, Z., Mu, H., Zhang, X., *et al*. (2019) Metapruning: Meta Learning for Automatic Neural Network Channel Pruning. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, Seoul, 27-28 October 2019, 3296-3305. https://doi.org/10.1109/ICCV.2019.00339

[14] You, S., Huang, T., Yang, M., *et al*. (2020) Greedynas: Towards Fast One-Shot NAS with Greedy Supernet. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Seattle, 13-19 June 2020, 1999-2008. https://doi.org/10.1109/CVPR42600.2020.00207

[15] Lin, M., Ji, R., Zhang, Y., *et al*. (2020) Channel Pruning via Automatic Structure Search. *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, 673-679. https://doi.org/10.24963/ijcai.2020/94