南京大学

研究生毕业论文
(申请硕士学位)

论文题目 面向卷积神经网络的模型加速研究

作者姓名 赵加成

学科、专业名称 计算机科学与技术系

研究方向 人工智能

指导教师 申富饶 教授

2021 年 5 月 29 日

学　　号：MG1833090

论文答辩日期：2021 年 5 月 24 日

指 导 教 师：　　（签字）

Model Acceleration based on Convolutional Neural Network

by

ZHAO Jia-Cheng

Supervised by

Professor SHEN Fu-Rao

A dissertation submitted to
the graduate school of Nanjing University
in partial fulfilment of the requirements for the degree of
MASTER
in
Computer Science and Technology

Computer Science and Technology
Nanjing University

May 24, 2021

南京大学研究生毕业论文中文摘要首页用纸

毕业论文题目：面向卷积神经网络的模型加速研究

计算机科学与技术系 专业 2018 级硕士生姓名：赵加成

指导教师(姓名、职称)：申富饶 教授

摘　要

近些年来，随着卷积神经网络 (Convolutional Neural Network，CNN) 在目标检测、图像识别等领域中大放异彩，越来越多的研究者们关注到了这一技术，并成功将其应用到了相关领域上。随着各项研究的深入开展，研究者们发现神经网络层数的增加会随之带来精度的提升。因而神经网络的计算量以及参数量开始了爆炸性的增长，最终导致了网络模型的部署成本、推理时间大大上升。而在一些嵌入式设备或者边缘计算的场景下，运行大型复杂的神经网络模型又变得极为困难。此时便需要借助模型加速技术来使得该场景下，高精度神经网络的运行成为可能。模型加速是指通过模型压缩、部署优化等技术来实现在对模型精度不产生较大影响的情况下，复杂模型依然可以较好地运行在资源受限的设备上。

本文从模型加速两个方向——模型压缩以及推理优化，完成对卷积神经网络模型的加速优化，本文的主要内容如下：

1. 本文提出了一种新型的剪枝策略——“假剪枝”(Pseudo Pruning)。首先基于权值共享的神经架构搜索技术，在其搜索空间中增加了对每一个网络层最优剪枝比例的搜索。同时结合我们提出的“假剪枝”策略，也即将训练过程与剪枝过程进行解耦合，从而可以获得该网络在剪枝前后的精度差。最终重新设计激励函数，将精度差融合其中，完成对优异剪枝策略的引导搜索，以获得更好的精简模型。通过 CIFAR-10 的消融实验证明，我们所提出的各项改进均能让网络获得有效增益。最终我们的方法在保持模型精度性能的前提下，有效缩减了模型 60% 的计算量和参数量。

2. 本文同时从工程方向完成对压缩后模型的推理加速。主要基于 Tensorflow 计算平台，采用了算子重写的方法，对于模型当中的耗时模块进行了底层计算重构。在其中集成了高性能加速代码，并结合了一些加速手段，从算子优化以及图优化两方面来完成网络的推理加速。最终通过各个模块的耗时分析以及与 Tensorflow 原生框架运行速度的对比，针对性地对一些耗时模块进行了深度优化，进一步提高了模型的加速效果。相比于 Tensorflow 推理速度，本文提出的方法可以有效带来较大幅度的速度提升。
3. 本文从算法与工程两个角度提出的加速方法可以成功运用在现实环境中，对实际当中的鸟类识别系统带来较好的加速效果。

相关实验证明了本文提出的两个方法的有效性，并且相比于已有的模型加速方法，本文提出的方法压缩效果更为显著，同时结合了工程化加速的方法，将模型加速的潜能极大地挖掘出来。在实际系统当中的运用也展现了本文方法所蕴含的巨大价值。最后，本文提出的方法在未来也可以进行深入研究，获得更好的优化效果。

关键词： 计算机视觉, 模型加速, 网络剪枝

南京大学研究生毕业论文英文摘要首页用纸

THESIS: Model Acceleration based on Convolutional Neural Network

SPECIALIZATION: Computer Science and Technology

POSTGRADUATE: ZHAO Jia-Cheng

MENTOR: Professor SHEN Fu-Rao

ABSTRACT

In recent years, as Convolutional Neural Network (CNN) shine in the fields of objection detection, image recognition, etc., more and more researchers have paid attention to this technology and successfully applied it to related fields. With the in-depth development of various studies, researchers have found that the deepening of the neural network will increase the accuracy. As a result, the amount of calculation and parameters of the neural network began to increase explosively, which eventually led to a significant increase in the deployment cost and running time of the neural network. In some embedded devices or edge computing scenarios, it becomes extremely difficult to run large and complex neural network models. At this time, model acceleration technology is needed to make the operation of high-precision neural network possible in this scenario. Model acceleration refers to the use of technologies such as model compression and deployment optimization to achieve a complex model that can still run well on resource-constrained devices without a big loss in model accuracy.

This article completes the model acceleration of the convolutional neural network from the two directions of model acceleration-model compression and inference optimization. The main contents of this article are as follows:

1. This article proposes a new type of pruning strategy-"Pseudo Pruning". First, we add a search for the optimal pruning ratio of each network layer in the search space of neural architecture search technology. At the same time, combined with the "Pseudo Pruning" strategy we proposed, that is, the training process

and the pruning process are decoupled, so that we can get the accuracy difference caused by network pruning. Finally, the reward function is redesigned, and the accuracy difference is merged. So the guided search for more excellent pruning strategies is completed to obtain a better compression model. Through the ablation experiment on CIFAR-10, it is proved that all the improvements we propose can make the network get effective promotion. Finally, our method effectively reduces the computation and parameters of the model by 60% while maintaining the accuracy of the model.

2. This article also completes the acceleration of the compressed model from the engineering direction.Mainly based on the operator rewriting method of Tensorflow, we reconstruct the computation of the time-consuming module in the model. In the meantime, it integrates high-performance acceleration code and combines operator optimization and graph optimization to complete network inference acceleration. Finally, through the time-consuming analysis of each module and the comparison of the native running speed of Tensorflow, some time-consuming modules were specifically optimized and accelerated, thereby greatly improving the acceleration effect of the model. Compared with the running speed of Tensorflow, the method proposed in this paper can effectively bring a significant speed increase.
3. The two methods proposed in this paper can be used in the actual production environment and have a huge impact on the actual face recognition system.

Related experiments have effectively proved the effectiveness of the two methods proposed in this paper. Compared with the existing model acceleration methods, the compression effect of the method proposed in this paper is more significant. At the same time, it combines the engineering acceleration method to accelerate the model. The potential is greatly unearthed. At the same time, the application in the actual system also shows the great value contained in the method of this article. Finally, the ways provided by this article can also be studied in depth for better result.

KEYWORDS: Computer Vision, Model Acceleration, Network Pruning

目　　录

插图清单

附表清单

第一章　绪论

1.1　研究背景与意义

卷积神经网络的发展可以追溯至上世纪 60 年代，起源于 Hubel 和 Wiesel 一项有关于猫大脑中视觉系统的研究，他们通过一系列实验系统性地创建了一张有关视觉皮层的地图。而在 80 年代，日本科学家福岛邦彦提出了一个简易的神经网络，在其之中包含了卷积层和池化层。在此基础之上，Yann Lecun 于 1988 年创新性地将反向传播算法[8] 应用在神经网络的训练学习上。由此，卷积神经网络的雏形慢慢形成。

但起初卷积神经网络的效果并不好，在各个领域上都不能超越传统的机器学习算法，因而一度沉寂了 10 多年，直到 2012 年的 Imagenet[9] 图像识别大赛上，Hinton 将一个全新的深层结构和 dropout[10] 方法引入到了神经网络中，提出了 Alexnet 神经网络[11] 架构，大幅降低了 Imagenet 大赛的错误率，将图像识别领域引入到了一个全新的阶段。而后各种神经网络架构层出不穷，卷积神经网络在各个领域大放异彩，完完全全地超越了人类的能力。然而神经网络能力的提升却是建立在网络层数不断加深、计算复杂性不断提高的基础上，而这又会让神经网络的参数量和计算量成倍上涨，从而导致网络的部署与运行的成本大幅上扬，对于嵌入式设备或者边缘计算等资源受限的设备，大型且复杂的模型运行更是成为了不可能。

现代神经网络的研究越来越趋向于网络复杂化、深层化，无数的研究[12] 表明，更加复杂的神经网络往往会带来更好的学习能力，也即是更好的拟合能力，从而带来卓越的精度性能。但这样做往往会导致网络模型的复杂度呈指数级上升，这便为底层硬件以及存储设施提出了更高的要求。例如，VGG16[13] 网络含有 1.4 亿的参数量，如果每个参数以浮点数的形式存储在计算机中则会占用 500 兆的存储空间。而在执行图片的运算时，单张 224*224 大小的测试图片大约需要 3.13 亿次的浮点数运算，这样对运算的需求，即便是极为先进的运算设备也难以实现实时计算，而在一些边缘设备或者计算资源受限的设备上，更是无法

对这样复杂的网络进行部署。

同时，高性能运算设备通常成本较高，且体积大、运算代价高，这无形之中便为深度学习应用者、使用者带来了较大的困难，也为深度学习应用的落地带来了重重屏障。而随着深度学习技术的快速发展及广泛运用，越来越多的消费者将更加频繁地接触、使用这项技术。因而，如何在资源受限的场合下，如日常生活当中的手机、平板、甚至是嵌入式设备上较为流畅地运行神经网络，是深度学习迈向普及的关键步骤。

需要强调的是，神经网络的压缩不仅具有重要意义，同时也有着较大的可实施性。近些年来，越来越多的研究证明神经网络通常具有较大的冗余性[14]。这些冗余甚至还会为网络性能带来负收益，如果能够采取正确方法对网络进行剪裁，非但能达到大大减少网络复杂度的效果，而且还能进一步提升网络的精度性能。同时，在资源受限的情况下，如果能对神经网络模型进行正确有效的压缩剪裁，对于模型的部署、资源的合理分配具有重要作用。因而，如何正确有效地对神经网络进行压缩剪裁成为了网络加速一个重要的研究分支。

上述分析表明，对深度卷积神经网络的加速具有较大的必要性以及理论可行性，设计研究一种能够有效降低网络参数量及运算量的压缩方法，从而减少存储空间占用、计算资源消耗，并使之有效应用在现实场景下，对深度学习的进步发展具有强烈意义。

1.2 研究现状与难点

神经网络由于其复杂的模型结构、变化万千的连接架构导致模型压缩存在较大的困难性，同时目前对于神经网络的可解释性[15]还亟待研究者们解决，它在应用上的成功与否对我们来说依然是一个黑箱，这些无形之中又为神经网络的压缩加速带来阻碍。而在最终的部署环节，又有各式各样诸如 FPGA、CPU、GPU 等的终端设备，对于这些不同硬件，神经网络的加速方法又需要进行针对性设计，这些更需要做更加精细化的加速工作。因此，神经网络的压缩加速从算法设计到模型部署各个环节均具有较大的难度。

针对于网络压缩算法设计部分，现今有较多方法可以对网络进行压缩，他们从不同角度来对神经网络进行加速，其中主流的一些方法包括：参数剪裁[16]、

紧凑卷积核设计[1]、低秩分解[17]以及网络蒸馏[18]等。参数修剪主要针对神经网络当中的冗余部分，对其进行修剪从而得到较好的压缩效果；紧凑卷积核则是通过设计结构特殊的卷积核来达到去除网络冗余、压缩网络的效果，但该方法需要从头训练才能取得较好的效果；低秩分解通过矩阵运算来对卷积核进行分解，从而达到降低网络计算量的效果，实用简单、可以较为轻易地跟其他方法进行结合，但需要较强的专家经验；网络蒸馏不同于上述方法，它采取了一种全新的方法来对网络进行压缩，通过训练一个更加复杂但结构类似的神经网络，利用其学习结果对小型网络进行蒸馏学习，从而达到紧凑神经网络的目的，但该方法对网络结构较为敏感，因而与其他方法相比则不太常用。

参数修剪思想源于 Oracle pruning[19] 的方法，即挑选出网络中不重要的参数部分对其进行修剪，而对最终网络效果不会造成太大影响。而如何找到不重要的参数，便是参数修剪所要研究的核心问题。目前主流的做法花样百出，各有不同，但最终都能取得较好的压缩效果。在具体实现部分又分为结构化剪枝和非结构化剪枝，具体区别就在于他们所操作的粒度不同。结构化剪枝通常是针对于网络层级间的剪枝，这种剪枝方法压缩性能较为明显，但由于是较为粗暴地直接删减网络层，因而可能会对网络性能造成较大影响；非结构化剪枝则聚焦于更加细粒度的卷积核级别，它只会删减卷积核内部的一些参数神经元，因而这种方法往往可以取得更高精度的网络，但同时它也需要着特定硬件才能完成网络的加速，对部署平台要求较高。总而言之，在最终的压缩方法的选择上，要根据具体应用场景来选择合适的参数修剪方法来完成网络压缩任务。

紧凑卷积核的设计需要较为精巧的设计理念，需要研究者对神经网络具有较为深刻的理解，能够对神经网络存在的问题有着敏锐的观察力。从而进行针对性的设计，来大大简化神经网络结构，降低网络的计算量。近些年的相关工作大多在聚焦于使用分组卷积来达到降低网络计算量与复杂度的目的，但如何在保障网络精度性能的前提下，设计行之有效的分组策略，这是紧凑卷积核设计的一个核心研究问题。

低秩分解是利用数学方法将卷积运算进行分解简化，一个典型的卷积核是一个四维张量，通常在这其中会含有较大冗余的成分，使用 SVD 等方法可以有效地将它们去除，同时也能降低卷积核维度，达到降低网络计算量的目的。但这种方法实现较为困难，涉及到的分解操作计算成本较为高昂，同时也需要大量

训练来达到模型的收敛。

网络蒸馏遵循着一种“老师—学生”的学习范式来对待优化网络进行学习训练，该方法需要预先训练好一个网络结构较为类似，但却更加复杂、更加深层的大型网络，利用该网络的 Softmax 输出来获得更加精细的类分布输出供小网络学习。虽然这种方法并未对小型网络结构进行改变，训练前后小网络的结构复杂度也均未发生变化。但通过学习蕴含着更多信息的类分布输出，小网络可以更好地学习到目标任务的特征信息，从而有效提升小网络性能。在网络结构不变的情况下，也能近似拥有更加复杂网络的学习能力，也是另一个维度上的网络压缩。基于网络蒸馏的方法可以有效显著地降低网络计算成本，但这种学习模式假设较为严格，而且压缩效果有时还会逊于其他方法。

针对网络模型部署加速部分，在模型训练方面有诸如 Tensorflow[20]、Pytorch[21]、MXNet[22] 等众多学习训练框架，而在模型部署上有存在着 CPU、GPU、FPGA 等众多部署硬件，因而推理优化不仅需要灵活适配各式各样的训练框架，同时又要针对各种不同的硬件来做部署加速。总的来说，完整的推理优化流程包括了三个步骤：算子优化、图优化以及部署优化。算子优化就是优化单个算子的性能，通过利用内存布局调整等方法来对单个算子的运算进行加速；图优化则是聚焦于更高的纬度，通过子图变换、算子融合等方式，提高整个计算图的计算效率，更加合理统筹全局，合理地进行资源分配；最后的部署优化则需要根据底层具体硬件来设计加速方法，同时合理调度模型运行时的资源分配来进一步优化计算性能。此外，现今诸多厂商也聚焦于推理加速中的编译优化部分，设计研发了一套成熟的解决方案。

在上述众多的网络压缩的方法中，本文主要聚焦网络剪枝部分以及网络部署加速部分，从算法端和工程端两个不同角度同时对神经网络进行压缩加速，最终达到较好的压缩效果。

1.3 研究内容

本文在研究网络压缩算法的基础上，同时结合网络推理优化的成熟方法，基于神经网络架构搜索[23] 的网络剪枝方法以及算子重写的部署加速方法分别设计了两种可以有效针对神经网络模型进行加速的算法，并将算法成功应用在了实

际系统当中，通过实践证明算法的有效性。其中，本文的主要研究内容如下：

1. 本文提出了一种基于神经网络架构搜索的网络剪枝方法。首先在网络搜索空间中加入每层最优剪裁比例的搜索，同时针对性地设计了网络搜索的激励函数，旨在引导网络能够搜索到兼顾网络性能同时也可以大幅降低网络复杂度的压缩方法。在搜索过程中，本文创新性地提出了一种假剪枝的方法 (Pseudo Pruning)，该方法摒弃了传统剪枝与训练交替进行的方式，转而将这两种方式解耦合，即网络剪枝与网络训练同时进行，通过优化剪裁前后的精度差来提升网络的压缩效果。本文提出的方法经过在 CIFAR-10 数据集上的验证，可以达到只损伤 0.3% 精度的情况下，带来对模型 60% 的压缩效果。

2. 本文同时结合了推理优化的加速手段，通过模型算子的重写，在其中集成了相应的优化加速方法。具体来说，是通过 mkl 的矩阵乘法库来对模型当中的卷积操作进行加速计算，同时结合了 omp 的向量化加速方法，来大大提升底层计算资源的利用率。更进一步，根据网络计算架构针对性地调整了矩阵的内存布局，以此来省去类似矩阵转置等的一些耗时操作。最终，在对重写的所有算子进行耗时分析以及类比原生 Tensorflow 框架的运行速度，发现了一些仍然耗时较高的热点并做了针对性优化，进一步提高了该部分的运行速度。本文集成的工程化加速方法，在基于前文压缩算法得到的压缩模型之上，可以进一步为模型带来 1.63 倍的性能提升。

3. 本文从算法端以及工程端两个方向，提出了对模型行之有效的加速方法，同时经过该方法加速之后的模型可以成功运用在实际的生产环境当中。在鸟类识别的场景下，识别模型采用了基于本方法搜索得到的压缩模型，并在部署端通过高性能代码加速，大大提升了模型的推理速度。同时，该方法对于识别精度并无较大损伤。这些验证了在实际生产当中，本文提出的方法具有较强的应用价值。

1.4 论文纲要

本文主要基于卷积神经网络的加速课题进行研究，并从算法以及工程两个截然不同的角度提出了模型优化加速的算法。它们分别是基于神经网络架构搜索的假剪枝方法以及基于 Tensorflow 算子重写的推理优化方法，并且已经成功在现实环境当中应用。全文一共分为以下六章: 第一章是绪论，主要介绍了神经网络加速的研究背景及其意义，同时还详细介绍了模型加速的研究现状及难点; 第二章则介绍了目前针对神经网络加速的相关研究工作; 第三章主要介绍基于神经网络架构搜索的假剪枝方法; 第四章介绍基于 Tensorflow 算子重写的工程端加速方法; 第五章主要介绍本文提出的优化方法在鸟类识别系统中的应用; 第六章总结全文，同时对未来的工作进行了展望。

第二章　相关工作

自从 2012 年 Hinton 等人采用卷积神经网络的方法来进行 Imagenet 图像识别比赛，并一举领先传统方法十余个点精度，深度学习的方法便广泛运用在视觉领域。然而这些模型通常具有成千上万亿的参数量，与之而来模型所需要的运算量也远远超过传统方法。对于这样的模型，通常需要 GPU 等高速大规模并行计算等的硬件才能较好地执行计算，这无疑大大提升了计算成本，使计算机视觉的研究门槛提高。另一方面，在一些边缘设备或是终端设备等计算资源受限的硬件上，神经网络模型的运行更是成为了不可能，这又极大限制了深度学习的应用。因而，如何行之有效地压缩网络模型并同时尽可能保持模型精度性能，对于深度学习的发展具有重要意义。

近些年来，越来越多的研究表明深度学习模型中通常具有较大的冗余，这通常是由于深度学习模型的复杂度与其实际解决的问题不相匹配，造成了最终模型当中会有部分模块对于最终的任务没有帮助，甚至是负收益。所以如果能较好地找出这些冗余部分，同时结合一些合理手段将它们去除，不仅可以大大缩减网络复杂度，还可以进一步提升网络性能。总而言之，神经网络的压缩加速不仅具有较大的现实意义，同时理论与实践上均能证明压缩的合理性。而现今，经过众多科研人员的不懈研究，关于神经网络的压缩的方法已经较为成熟，同时这些方法自成体系，与此同时它们也都各有优缺。不同方向的压缩手段适合的场景也各不相同，需要研究者根据具体情况来使用。本章节首先会简单概述一下卷积神经网络的组成部分，然后会介绍一些有关于神经网络加速的概念以及当下主流的各个方向上的压缩方法，同时会对这些方法加以分析，总结指出他们的优势，为后文提出本文的压缩方法打下良好的基础。

2.1　卷积神经网络的组成结构

卷积神经网络的研究历史最早可以追溯到 1962 年 Hubel 和 Wiesel 有关于猫大脑视觉系统的研究，而后到了 20 世纪 60 年代，David Hubel 和 Torsten Wiesel 在他们的论文中提出了 Receptive fields 的概念。继而日本科学家福岛邦彦在前

人的基础上提出了一个包含了卷积层和池化层的神经网络结构。至此，卷积神经网络雏形初现。而到了 80 年代，Yann Lecun 首次将反向传播算法[24] 应用到神经网络的训练当中，极大地推动了卷积神经网络的发展。近些年到了 2012 年，在 Imagenet 图像识别大赛中，Hinton 组在卷积网络中创新性地引入了全新的深层结构以及 dropout 网络结构，提出了 Alexnet 网络结构，大大降低了大赛图像识别的识别率，一举颠覆了图像识别领域。从此卷积神经网络开始逐渐应用于视觉的各个领域，在目标检测[25]、图像分割[26]、关键点检测[27] 等领域，卷积网络的应用数见不鲜。卷积神经网络自此迈入了一个新的境界。

图2-1描述了卷积神经网络的组成，它包括了卷积层 (Convolutional Layer)、池化层 (Pooling Layer)、激活函数 (Activation Function) 以及最后的全连接层 (Fully Connection) 等四个部分。其中，卷积层以及全连接层占据了整个网络中绝大多数的计算量以及参数量，因而它们也是网络压缩中主要研究的对象。

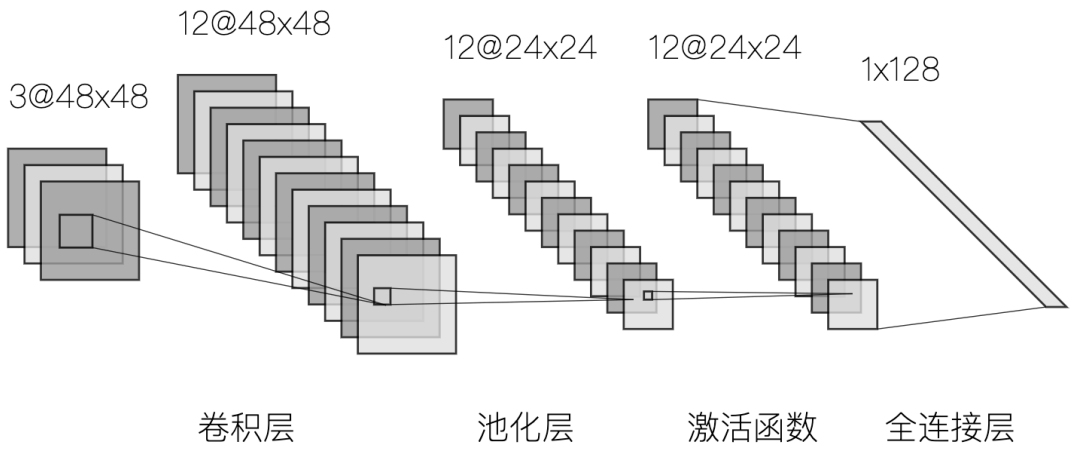


图 2-1: 卷积神经网络的组成部分

2.1.1 卷积层

卷积层通过一个 3x3 或者 5x5 或者其他更大尺度的卷积核在目标图像上进行动，滑动的同时会分别计算图像与卷积核对应位置的乘积，并对一次滑动所得到的所有乘积进行求和，作为当前滑动位置经过卷积计算得到的最终结果。通常情况下，卷积层的输入特征图、相应的输出特征图以及卷积核的尺寸维度都可以用 N，H，W，C 四个参数来代表。其中，N 是指样本数，H 是指特征图或者卷积核的高，W 指特征图的宽或是卷积核的宽，C 则是特征图或者卷积核的

通道数量。也就是说，特征图或者卷积核都可以看作是关于 N，H，W，C 的四维向量。

同时同一个卷积层会有多个卷积核，它们是由多个卷积平面堆叠形成的四维卷积核。卷积核的通道数是由当前卷积层输入图像的通道数所决定，而当前卷积层的卷积核个数又决定了当前层输出图像的通道数。因而，通过对于卷积核个数的压缩，一定程度上可以减少网络模型的计算量。而对于输出图像的长宽计算，则还需要考虑当前卷积层的补零个数以及步长参数，它们分别设为 p 和 s，输出图像的长宽计算见公式2-1和2-2。

$$W_{out} = \frac{W_{in} + 2 * p - W_{kernel}}{s} + 1 \tag{2-1}$$

$$H_{out} = \frac{H_{in} + 2 * p - H_{kernel}}{s} + 1 \tag{2-2}$$

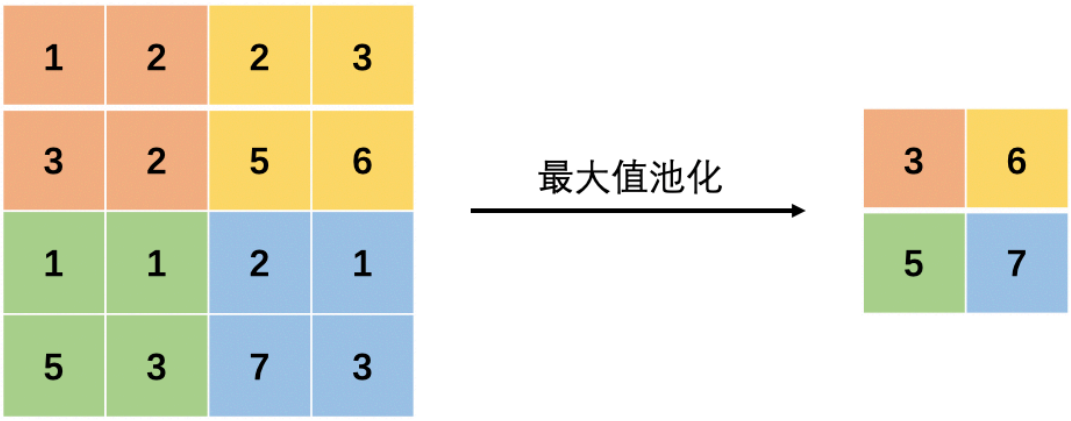


图 2-2: 最大值池化

2.1.2 池化层

池化即下采样，主要针对的是图像的长宽两个维度，一般会通过一个 2*2 的滑框，在图像上滑动，通过一定的运算方法来得到当前滑框位置经过池化后的结果。一般来说，池化的运算有四种方式。第一种方式是利用最大值来进行池化 (Max Pooling)，该方法取滑框内最大值作为池化结果，这也是最常用的池化方法，其过程如图2-2所示。第二种则以平均值为目标进行池化 (Mean Pooling)，它以滑框内平均值作为最终的池化结果。第三种是可训练池化方法，它会直接

学习一个池化函数作为池化的运算，但这种方法不太常用。最后一种则是高斯池化方法，它以一种类似高斯模糊的方法来进行池化运算，同样地这个方法也不太常用。

2.1.3 激活函数

激活函数是神经网络当中一个不可或缺的函数，他负责将神经元的输入映射到输出端，它通过对于函数的精巧设计来将非线性引入到神经网络当中，极大地提升了网络的学习拟合能力。而如果不采用激活函数的话，堆叠再多层的网络结构，最终也只是相当于一个原始的感知机。因而，在现代卷积神经网络的结构设计中，通常卷积层与激活函数都是结合出现。常用的激活函数有 relu 函数[28]，sigmoid 函数，tanh 函数等，图2-3分别展示了它们的几何图像。

1. sigmoid 函数是一种常用的激活函数，其数学公式见2-3。

$$Sigmoid(x) = \frac{1}{1+e^{-x}} \tag{2-3}$$

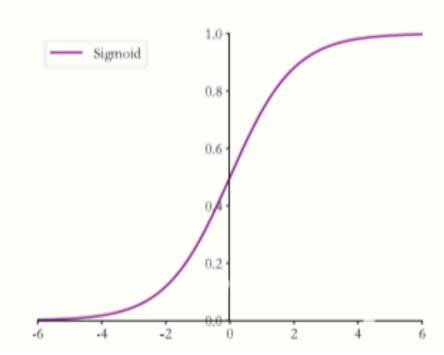
sigmoid 函数将输入非线性地映射到了 0 到 1 之间，当输入值较大时，映射的值会很接近 1；而当输入值较小时，映射的值会接近 0。由于 sigmoid 的函数特性，输入值较大或者较小的时候，函数相应位置的导数会趋近于 0，这便导致了基于反向传播算法训练的网络会在学习过程中出现梯度消失的问题。同时，sigmoid 函数所蕴含的幂运算也加大了函数的计算量，因而现在神经网络越来越少地采用 sigmoid 函数。

2. 相较于 sigmoid 激活函数的种种缺陷，relu 函数较好地解决了这些问题，其数学公式见2-4。

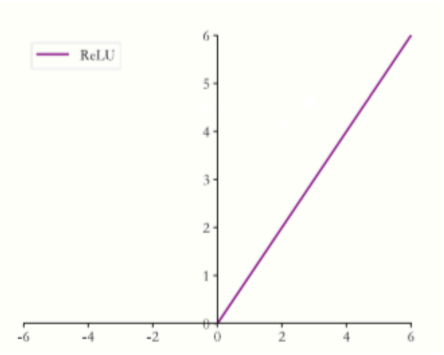
$$Relu(x) = \max(0, x) \tag{2-4}$$

relu 函数采取当输入值大于 0 时保持值不变，当输入值小于 0 时则取 0 的策略。在保持激活函数的非线性同时，可以极大地简化函数计算。同时收敛速度也远远快于一般的激活函数，因而比较常用。在 relu 的基础上，也演变出了 leakyrelu 等更加灵活的激活函数。

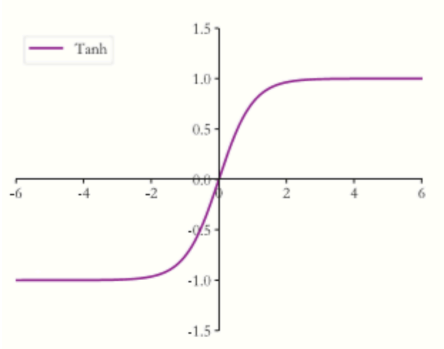
3. tanh 激活函数与 sigmoid 激活函数比较类似，同样存在着梯度消失以及



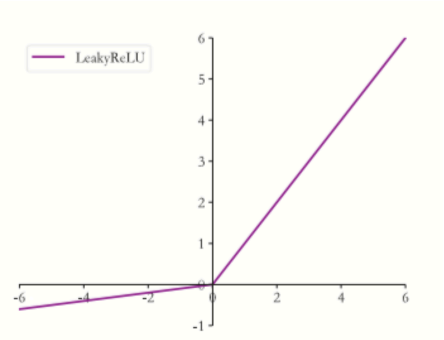
(a) Sigmoid



(b) Relu



(c) Tanh



(d) LeakyRelu

图 2-3: 几种常见的激活函数

幂运算耗时等问题，其数学公式见2-5。

$$Tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2-5}$$

2.1.4 全连接层

全连接层一般用在卷积网络的最后，全连接层利用矩阵乘积运算，将输入的多维向量展成一维，可以认为是二维图像到一维抽象之间的一座桥梁。因而全连接层可以跟卷积层相互变换，只需要将全连接层的所有参数填充到一个跟输入图像大小一致的大型卷积核中，即可认为是在进行卷积运算。由于全连接层的特点，其参数与输入图像的大小息息相关，因而在模型当中具有较多的参数量，同时也需要耗费较多的计算资源来进行运算。所以全连接层也是模型压缩过程中需要针对性关注的一个模块。

表 2-1: 网络压缩常用方法

压缩方法	方法简介	应用结构
网络剪枝	删除对网络性能影响不大的参数及结构	卷积层和全连接层
紧凑卷积核设计	设计精巧的卷积核来搭建网络	卷积层
知识蒸馏	搭建一个紧凑网络从大型网络中蒸馏学习	卷积层和全连接层
低秩分解	使用矩阵方法对参数进行分解估计	卷积层和全连接层

2.2 卷积神经网络加速方法

神经网络模型算法端的压缩可以分为网络剪枝、紧凑卷积核设计、知识蒸馏以及低秩分解四个类别。表2-1中展示了这四个压缩方法的简要概括及分析，网络剪枝主要思想是删除网络中的冗余部分，通常应用在卷积层以及全连接层等计算密集的网络层中。紧凑卷积核设计则从卷积核的精简化设计角度为模型进行压缩降维，因此它只能应用在卷积层当中。知识蒸馏采取了不同于前两种方法的压缩手段，它利用复杂模型的输出来对小模型进行学习优化，从而让小模型具有大模型的泛化能力，通常也是应用在卷积层和全连接层当中。最后的低秩分解采用矩阵方法对卷积核进行分解估计，达到降低计算复杂度的目的，同之前方法一样，也是只能应用在卷积层以及全连接层当中。现今比较常用的压缩方法主要是网络剪枝以及紧凑卷积核设计，本文提出的压缩方法也是在参数剪枝的方法上改良而来。接下来对于算法端的内容介绍将主要聚焦在网络剪枝、紧凑卷积核设计和知识蒸馏三个方法上。

神经网络模型工程端的加速主要是利用工业界一些成熟的加速手段，比如采用 tvm 进行算子以及部署加速，采用 tensorRT 进行部署加速，采用 CUDA 进行计算加速，本文中工程端的加速主要聚焦于 Tensorflow 的算子重写，并在其中集成加速方法，从而完成模型加速。本章节将简要概述现今工业界常用的加速方法，并对其进行分析总结。

2.3 网络压缩方法

2.3.1 网络结构轻量化

目前针对网络压缩的轻量化网络结构设计主要聚焦在卷积核的压缩加速上，核心思想在于使用更加精简的卷积核就能完成正常卷积网络可以学习的目标，从而可以达到降低网络参数量以及网络计算量的目的，针对通道进行随机混合[1-2]的方法便可以有效实现这样的目标。

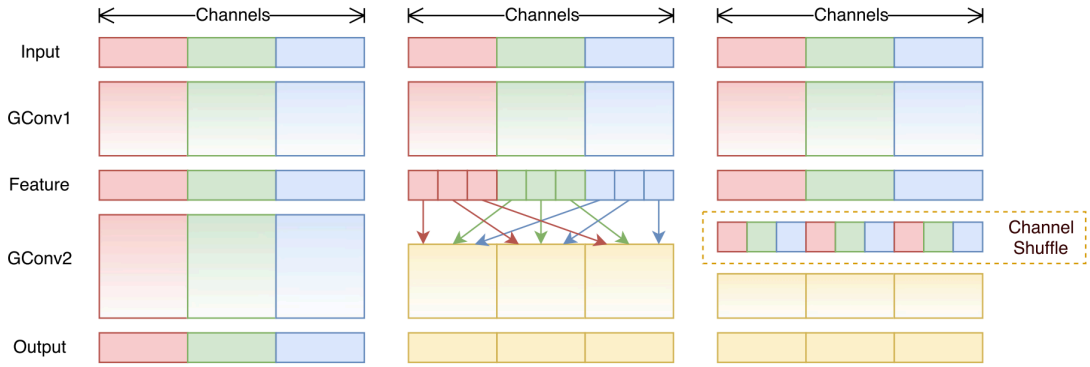


图 2-4: 通道随机混合[1-2]

通道随机混合的思想起源于组卷积操作，组卷积是指在传统卷积的基础上，添加分组的思想，按照通道维度对输入图像以及卷积核分成同样规模大小的组，以组为单位进行划分，图像以及卷积核进行一一对应的相乘，该过程在图2-4的左一中有所展示。可以看到，经过这样分组之后，卷积部分的计算量大大减少，极大提升网络速度，但这样的分组无疑是增加了组与组之间的隔阂，不利于网络信息流通以及知识学习。为了解决这样的缺陷，研究[1]提出了通道随机混合的思想，其具体的操作是在卷积之前，首先会对输入图像的通道进行随机打乱，也就是说按照原先的分组，打乱之后的通道会随机到一个新的组别中。之后的操作就是跟组卷积一样，根据打乱后的新分组，跟对应的卷积核通道执行卷积操作。这样做就可以解决组卷积带给组与组之间的隔阂问题，从而更快更好地帮助网络进行信息的学习。图2-4中的后两个部分即是在展示通道随机混合的过程，可以从中看出，经过通道随机混合的操作，不同组之间可以较好地进行信息交流与融合，组卷积固有的组与组之间的闭塞问题得到较好地解决。

2.3.2 知识蒸馏

在生物界，昆虫一般被分为幼虫期和成虫期。在幼虫期，昆虫主要负责汲取养分，而到了成虫，则开始繁衍后代。与昆虫的两个周期类似，机器学习应用也可以分为训练阶段和部署阶段，训练阶段由于其特殊性，并不对机器学习的训练做时间速度要求，因而这个模型可以通常做的很大很深，也可以通过多个复杂的模型堆叠搭建构成。而当这个模型训练完成，便可以采取不一样的训练方式，也即本节所讲的蒸馏方法，来将复杂模型学习到的知识传授给一个更小更适合实际部署的模型上去。相较于前文所讲的轻量化网络结构设计，知识蒸馏从另一个角度提出了网络加速的方法。

对于正常的机器学习模型来说，它们的目标通常是最大化正确答案的平均 log 概率，这样的做法的缺陷在于它对于其他错误答案都给予了一样的学习目标，即 0 值。这样便不能反映出所有答案之间的关系，真实的学习目标信息被掩盖了很多。举个例子，假设现在要学习的正确目标是猫，同时另外在错误目标中还存在这老虎跟汽车，但对于机器学习模型来讲，它被给予的信息只是猫的标签为 1，而老虎跟汽车的标签都是 0。这样无差别地将老虎和汽车拉齐，较为粗暴地忽视了猫跟老虎之间的关系。但是模型经过大量数据训练学习之后，会隐性地给予老虎更大的正样本概率，这种能力就是模型的泛化能力，而机器学习训练的核心正是在于泛化能力的获得。

知识蒸馏之所以可以达到让小模型能够具备大模型的学习能力关键就在于泛化能力的传递。一个常用的做法便是利用大模型产生的分类概率作为学习目标来训练小模型。因为相比于较为硬性的非 0 即 1 的学习目标，大模型产生的分类概率天然就包含了复杂模型在经过多轮迭代训练之后所习得的泛化能力。通过模型生成的分类概率这样传递方式，小模型可以更加容易学到复杂模型的泛化能力。而从信息论的角度，分类概率这样的软目标通常具有更高的熵值，相比于硬目标，每个样本能提供更多的信息，同时训练样本之间也会有更小的梯度方差，帮助小模型更好更快地收敛。所以小模型经常采用较小的数据集进行训练，因为学习信息更加丰富的缘故，在训练过程中一般会采取较大的学习率。

知识蒸馏的方法比较简单，核心就在于提取大模型对于训练集的分类概率，也就是只需要将数据集的数据输入到大模型中，在最后的 softmax 层提取出模型

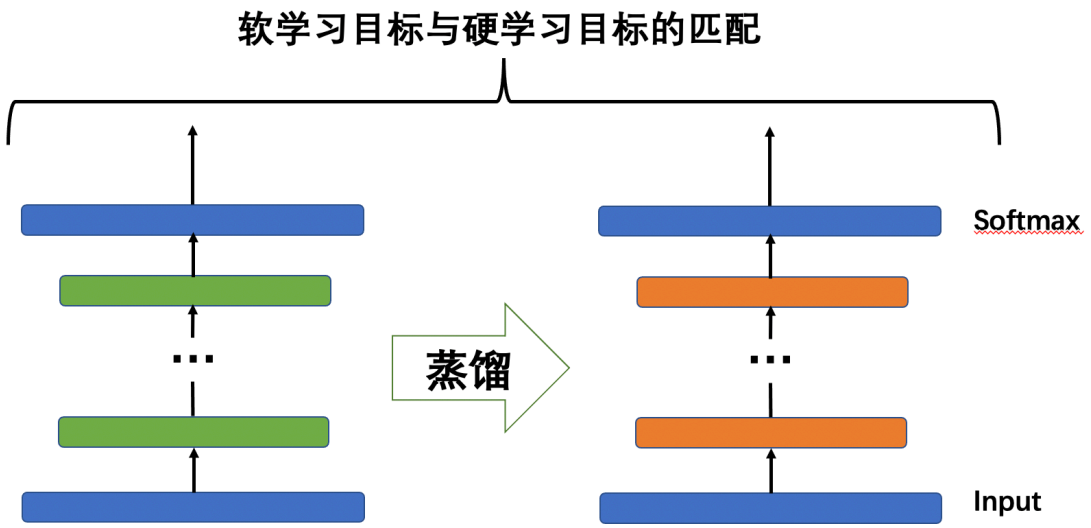


图 2-5: 网络蒸馏过程

运算的结果，并将其作为模型的标签输入来辅助小模型迭代优化。正如图中2-5所展示的，知识蒸馏只需要大模型的概率提取以及小模型的数据输入即可完成。由于通常 softmax 的结果具有负样本概率值方差较小的性质，数据之间的特性分布不够显著，因而在蒸馏的过程中也需要进一步调整。蒸馏过程中的 softmax 公式见2-6。

$$q_i = \frac{\exp(\frac{z_i}{T})}{\sum_j \exp(\frac{z_j}{T})} \tag{2-6}$$

其中，T 是蒸馏过程中的温度参数，T 越大便可以得到更加分散的分类概率，也即使得负样本之间的差异拉大，便于小模型学习。所以在蒸馏过程中，对于大模型，一般会采取较大的温度参数，来获取到相对较软的学习目标。而对于小模型的训练，采取跟平常训练时一致的做法，即温度参数设为 1。同时在蒸馏的过程中，用于训练小模型的数据集可以包含没有标签的数据，因为即使没有标签，依然可以通过训练好的复杂模型来获取软目标帮助小模型进行学习，这样一来在训练数据集的挑选上便有了更多的选择，这也是知识蒸馏的一大优势。

2.3.3 网络剪枝

网络剪枝是针对深度学习模型目前普遍存在的过参数化问题提出的方法，过参数化指的是模型通常具有大量的参数，而越来越多的研究表明，这些参数中存在着大量无用的参数，也就是所谓的冗余参数。如果能够有效将这些参数

裁剪，不仅能极大提升网络的计算速度，还能一定程度上提升网络精度性能，这便是网络剪枝的主要目标。过去认为，网络剪枝的本质思想在于将网络中冗余参数找到并且将他们剪裁，而最近几年，一些有趣的研究对于网络剪裁的本质思想这个问题提出了新的看法。其中,[29] 研究具有一定的代表性。传统观念认为训练一个大的过参数化的模型很重要，同时网络剪裁的最终目的在于得到剪裁之后剩下的网络参数权重，是这些权重对于整个剪裁更有意义。这些思想都在研究[29] 中得到批驳，它经过大量实验分析认为，网络剪裁的意义在于剪裁后剩下的网络架构，而不是参数权重。因而它认为网络剪裁本质上就是一个最优网络架构搜索的过程，本文提出的网络压缩算法也即是基于这样的思想，结合当下热门的网络架构搜索方法，创新性地设计了一套自动化网络模型剪枝的算法，通过不断地调整优化，最终取得了较好的压缩效果。

网络剪枝从剪枝的粒度来划分可以分为结构化剪枝和非结构化剪枝，以往的剪枝方法都是采取非结构化剪枝，以神经元为基本单位剪裁网络，最终得到的都是稀疏的卷积核。而这样的网络需要底层特定硬件的支持才能起到加速效果，因而现在越来越多的剪裁方法都采取结构化剪枝，即聚焦于卷积核以及卷积层这样的级别进行剪裁。而不论是结构化剪枝还是非结构化剪枝，他们都遵循着相同的剪枝流程。即首先会对网络中的所有神经元进行一个重要性判定，然后对重要程度较低的一批神经元进行剪裁，最后将剪裁后的网络进行再训练微调。重复上述流程，直到达到期望的速度与精度剪裁才会停止。

结构化剪枝以整个卷积核甚至是整个卷积层为剪裁的对象，算法会衡量出网络中所有卷积核或卷积层的重要性，基于此来对网络进行剪裁压缩。而对于网络结构重要性的判断诞生了很多优秀的结构化剪枝方法。例如，研究[30] 提出了以参数权重绝对值大小为判定重要性的标准，计算卷积核内所有参数权重绝对值之和作为这个卷积核的重要程度，按照预先设定的剪裁比例来选择需要裁剪的卷积核，最终反复迭代直到模型达到速度要求。这种方法虽然简单易懂，但大量实验证明了其有效性，也为其他方法的提出奠定了基础。相较于这样手动对卷积核进行排序再删减的方法，研究[3] 提出了一种更加自动化的剪裁方式。它认为可以直接在 BN 层的缩放因子上添加一个 L1 的正则化，以此来推动缩放因子趋近于 0。这种方法使得网络能够自动鉴别筛选出那些不重要的卷积层并将其筛除，而且正则化的方法对网络性能比较友好，一定程度上还可以帮助网络

拥有更高的泛化性能。图2-6 展示了该研究提出的剪裁方法，通过对缩放因子添加正则化，在网络完成训练之后，再将那些较小的缩放因子对应的卷积核删去，即可完成网络的剪裁压缩。

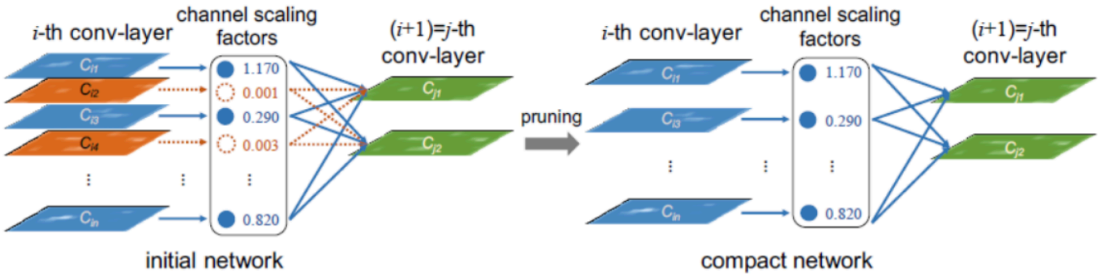


图 2-6: 基于缩放因子的网络剪枝方法[3]

相比于结构化剪枝，非结构化剪枝以一种更加细粒度的剪枝方式对网络进行压缩剪裁，通常他们锚定的是卷积核里面的每个参数，以此作为剪裁目标。剪裁过程与结构化剪枝类似，也是首先对剪裁目标进行重要性排序，接着按照重要程度对网络进行剪裁，最终反复迭代以达到好的剪裁效果。由于是一种更加精细化的剪枝，所以压缩效果会比结构化剪枝更好。但由于其非结构性的特点，在网络部署的过程中，通常会需要一些特定的硬件才能做网络计算上的加速，因而方法灵活性会比结构化剪枝更差一些。研究[31] 提出了一种常用的非结构化剪枝的方法，即对原始网络精度进行保留的方式来对网络进行剪裁。首先在初始网络训练的过程中，需要删除所有低于预设阈值的网络连接，此时原先的完全连接图便转化成了稀疏图。然后在这一阶段的学习中，得到了网络中重要程度比较低的网络连接并将它们删除。最后通过训练该稀疏网络，来尽可能地补偿因为裁剪而损失的网络精度。上述流程可以反复迭代进行，直到剪裁网络达到预期的速度性能。在此基础上，研究[32] 同时结合了参数量化以及霍夫曼编码的方式，来进一步提升网络剪裁性能。其中，量化是指将剪裁剩余的权重进行聚类，使用低位编码来表示这些共享权值以便达到压缩参数权值的目的；霍夫曼编码则使用短码来表示频率较高的权重，长码表示频率较低的权重。

2.4 推理优化方法

前文所述的模型压缩主要聚焦于算法层面，通过对于模型结构的研究以及参数权重等对最终精度的变化影响研究，形成一系列对于神经网络模型有效的

压缩方法，这些都集中在模型的训练端。而模型推理优化则聚焦于工程加速，通过对于特定硬件、特定深度学习框架甚至是特定模型，设计了一套行之有效的加速方案，也即模型推理优化注重在部署端进行加速，这相比于模型压缩通常具有更大的优化意义。通常来说，深度学习的训练只是面向一小群研究者，几张训练显卡或者一台工作站即可完成模型训练。而模型的推理应用则往往面对的是一个大规模的群体，这样需求对应的往往是大规模分布式架构以及大量的深度学习计算资源。因而，推理端的模型加速带来的意义通常要高于算法端的模型压缩。

推理优化即是在保持现有神经网络结构不变的情况下，通过高性能计算等方式大大提高底层硬件利用率，从而带来对模型推理速度的提升，使得网络模型能在资源受限的设备上高效运行。它要适配主流的各种深度学习框架，将他们训练得到的模型进行分析表示，然后通过图优化以及算子优化两个层面进行优化加速，最终结合具体部署平台做针对性优化即可完成整个模型的优化工作。模型推理优化通常包含了算子优化，图优化，模型压缩和部署优化等四个方面，接下来主要针对算子优化以及图优化做简单的介绍。

2.4.1 图优化

图优化的主要方法有子图变换和算子融合两种方式，它们希望能够实现减少计算量或者其他诸如访存开销等的系统开销，从而达到性能优化的目的。图优化同时基于不对模型的数值特性做较大改动的思想，通过图变换达到简化网络计算、降低计算资源开销，提升推理性能的目的，因而图优化的方法较为灵活有效，也是性能优化时的首选方法之一。子图变换主要基于常数折叠，公共子表达式折叠等数学变换的方法对网络子图进行变换折叠，达到减少计算量、精简网络模型的目的。下图2-7展示了卷积层结合批归一化层这样典型网络结构的子图变换方法，也是图优化当中一个常见的例子。它将批归一化算子中关于 α 参数以及 μ 参数的加法乘法计算部分直接融合到上一层的卷积运算当中，具体来说，就是将批归一化算子的 α 参数融合到上一个卷积层当中，卷积参数以及偏移量参数与 α 进行一一相乘。而批归一化算子的 μ 参数则与卷积层当中的偏移量进行融合，与 α 参数相乘之后的偏移量参数需要与 μ 参数进行减法结合。这些都要首先经过常量参数预计算，完成融合之后的新参数计算，也即是原本的

卷积层加上批归一化层经过图优化之后，只相当于一个新的卷积层，原本的批归一化算子的计算过程被有效省略，这较大幅度地降低了模型推理过程中的计算量。需要强调的是，这种方式只能针对已经训练完成的深度学习模型，而且通常是在部署应用端进行加速优化，对于算法模型训练端，图优化并不能发挥作用。

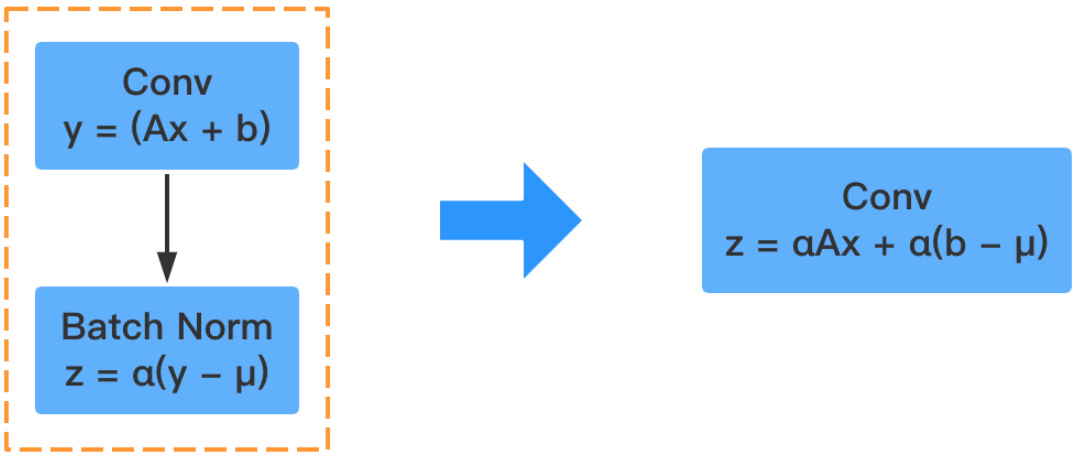


图 2-7: Conv+BN 结构的折叠

2.4.2 算子优化

算子融合的思想来源于深度学习各种拓扑结构模式的观察，深度学习模型是由各式各样的算子堆叠而来，它们的连接架构、算子类型千差万别。所以针对它们进行优化首先需要了解各种算子，需要掌握模型结构之间的连接思想。通常，深度学习算子按其对资源的需求可以分为两类：

1. 计算密集型算子，这些算子的时间绝大部分花在计算操作上，相较之下用于访存的时间可以忽略不计，诸如卷积操作、全连接操作等都是典型的计算密集型算子。而针对这类算子的优化主要从减少计算复杂度、提高硬件计算效率等方面入手。
2. 访存密集型算子，这些算子的时间绝大部分花在访存上，在这些算子里，计算的部分并不是很多或者计算操作的复杂度较低，他们大部分是求和算子、激活算子等。这类算子的优化主要从减少访存次数、提高 CPU 缓存命中率等角度进行思考。

图2-8展示的是一个残差模块里面较为常见的模型结构. 该结构首先是两个并行的卷积操作，然后它们的结果通过矩阵加法操作融合到一起，并对其执行一个激活函数算子的操作来实现非线性化。可以看出，除了卷积操作，该结构下其他算子均是访存密集型算子，因而优化主要沿着降低访存次数的思路进行。具体的优化方法就是将其中一个卷积算子、求和算子以及激活算子进行融合，将加法与激活算子两个操作在缓存甚至是寄存器中直接进行计算，而不用与内存进行反复交互，这有效减少了算子产生的中间结果对于内存的访问次数，大大提升网络运行效率。

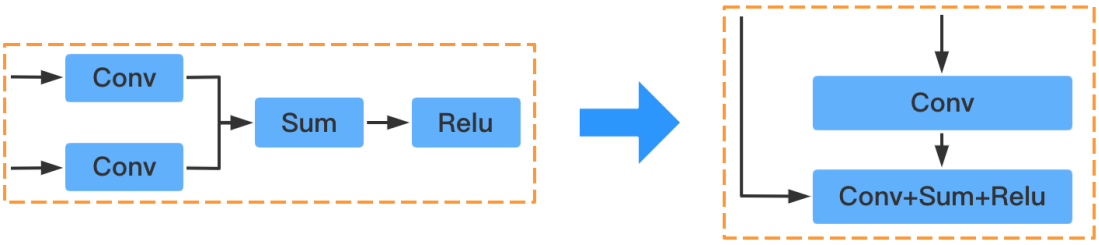


图 2-8: 残差模块常见结构的融合

2.5 本章小结

本章主要介绍了神经网络压缩的一些常用做法以及在工程端一些典型的模型推理优化方法，并对它们的应用场景做了较为详尽的分析。通过本章节的分析介绍，可以看出目前关于深度学习模型的加速优化方法层出不穷，但同时它们也各有侧重，需要结合具体情况采取合适的方法才能有效加速模型。受制于篇幅限制，本章节只简要介绍了一些当下主流的加速算法，但同时也是本文提出的方法的基石，这也为后文介绍本文方法打下良好基础。

第三章　基于假剪枝与网络搜索的压缩方法

本章提出了一种基于神经网络架构搜索的自动网络剪枝方法，它同时结合了本文提出的假剪枝方法（Pseudo Pruning），能够在保证网络精度性能不变的前提下，大幅提升网络的压缩效率。具体来说，本方法从三个角度提出了新的网络压缩方法：1、在神经网络架构搜索的搜索空间上添加了对各个网络层的剪裁比例，更加灵活地搜索出适合每个网络层的剪裁比例。2、重新修改网络搜索的激励函数，通过参数调节剪裁比例与网络精度性能的平衡。3、提出了假剪枝的方法，将网络训练与网络剪枝进行解耦，进一步提升了网络压缩效率。相关数据集上的消融实验以及关键参数的分析，强有力地证明了本文提出方法的有效性与合理性。

3.1　网络架构搜索

近些年来，深度学习的快速发展颠覆了很多传统行业，也极大地提升了人们的生产效率。而神经网络模型的架构设计是其中重要的一门课题，但网络模型设计需要大量的人工经验，而且设计的模型性能也存在着较大的不确定性。因而近些年，越来越多的研究者开始研究新型的网络设计方法，即神经网络架构搜索方法 (简称 NAS[23])。如图4-1 所示，经典的 NAS 方法采用 RNN 作为控制器，通过一定的概率模型采样出一系列的网络架构，同时会对采样的网络进行学习训练，以帮助得到较好的网络精度性能，最终根据精度变化得到更新梯度来对作为控制器的 RNN 进行优化。重复此过程，直到搜索到一个比较好的子网络。需要强调的是，子网络的性能精度并不可导，不能直接用于更新控制器的参数，但通过强化学习的方法，就能较好地实现这一目标。也即，网络架构搜索的过程本质上是搭建了一个强化学习的环境，通过在这个环境中对网络模型进行采样，同时通过采样网络的精度性能不断更新优化采样策略，来最终生成一个性能优异的网络。

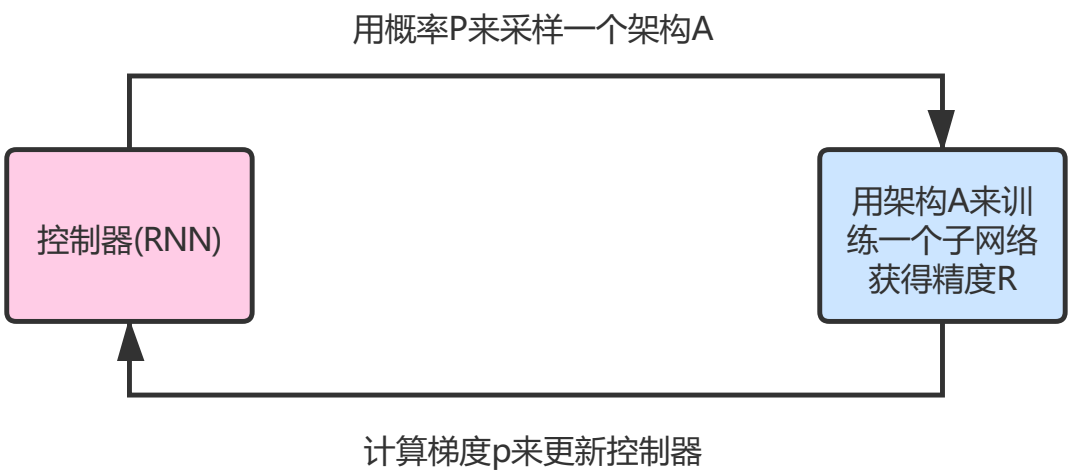


图 3-1: NAS 方法执行流程

3.1.1 ENAS 方法介绍

神经网络架构搜索的方法已经成功运用到图像分类以及语言建模模型等领域上，尽管它展现了强大的模型设计能力，但其强大能力的背后是对庞大计算资源的需求，比如研究[23] 中便使用了 450 块 GPU 耗时 3 到 4 天才完成网络搜索，这是普通研究者所难以企及的计算成本。而 ENAS[4] 正是为了解决传统 NAS 高成本的问题而提出的。其核心思想是所有搜索得到的最终网络图都可以看作是一个大图的子图，因而其搜索空间可以用一个单向的有向无环图来表示。图3-2即是 ENAS 搜索空间的一个 DAG 展示图，这个 DAG 图可以看作 ENAS 所有搜索结果模型的叠加，图中各个节点代表局部算子，连线则代表着网络图中信息的流动。同时，每个节点所代表的局部算子都有其自身的参数，但它们仅在节点被激活也即节点被某个搜索得到的子图选中的时候才会被训练学习。所以在搜索空间中，ENAS 允许所有参数节点在不同的子图之间共享。通过参数共享的方式，ENAS 可以避免传统 NAS 方法需要多次从头重新训练子图的缺点，从而大幅降低网络搜索的计算量，极大地解决了传统 NAS 计算成本较高的问题。需要说明的是，相比于传统的 NAS 方法，ENAS 可以保持相同的网络搜索性能，而同时只需要极少量的 GPU 参与训练优化，它的计算成本仅是前者的千分之一。在图像识别以及语言建模两个领域的多个数据集上的实验结果均强有力地证明了 ENAS 方法的有效性。ENAS 具有计算成本低廉以及网络搜索性能卓越两大特性，本文提出的方法便是结合 ENAS 方法对搜索得到的网络模型进行压缩。

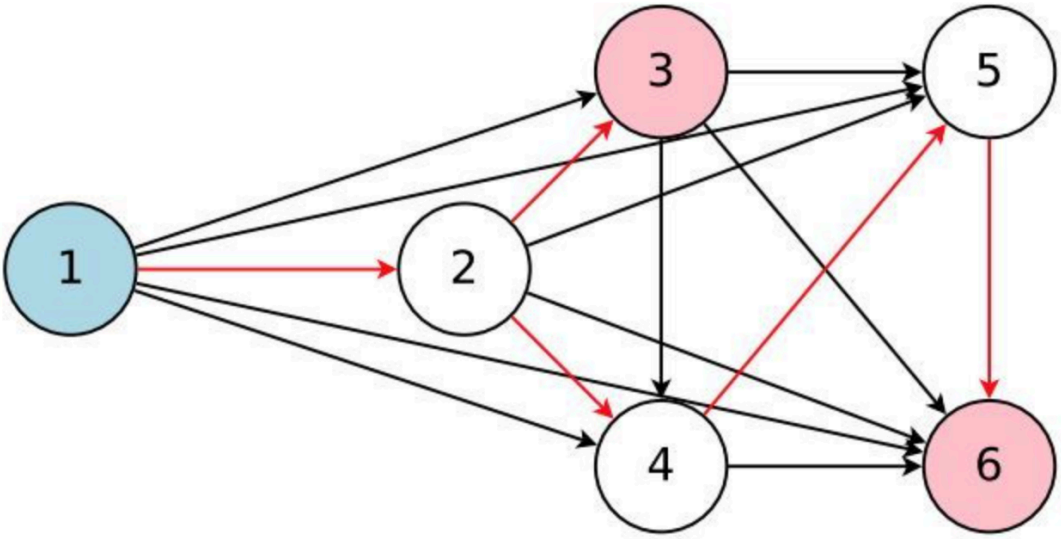


图 3-2: ENAS 搜索空间的 DAG 展示图[4]

3.1.2 基于网络搜索的压缩方法

目前针对深度学习模型压缩的手段都是基于手工压缩模型的方法，但手工压缩模型的方法存在这诸多弊端。例如，手工压缩的方法不仅需要较强的专家经验，而且也需要在模型运行速度以及模型性能之间进行权衡，所以手工压缩的方法通常难以得到最优的压缩效果。而随着网络模型搜索方法的逐渐发展，越来越多的模型压缩研究者开始将目光投向网络搜索。与手工压缩模型的方法相比，基于模型架构搜索的方法通常能够得到更好的压缩效果。研究[5] 针对 VGG-16 的压缩，可以达到在模型计算量减少 4 倍的情况下，相比于人工压缩的效果能够提升 2.7% 的精度。在研究[5] 中，作者发现压缩模型的性能对于每一层的稀疏度都较为敏感，这种敏感性对压缩方法提出了更高的要求。所以在实验过程中，该方法提出了一个连续压缩比的控制策略，基于强化学习 DDPG[33] 的方法去学习适合每一个网络层的压缩比例。在策略搜索完成之后，探索得到的最优模型还需要进一步的微调训练才能得到最终结果。图3-3展示了该方法的过程，左侧代表 AMC 替代人工对模型进行压缩，右侧则是 AMC 利用 DDPG 算法对模型每一层进行搜索采样，获得合适的剪裁比例。在之后的模型部署环节，该方法还进一步根据实际场景来做针对性压缩训练。对于速度主导的应用场景下，该方法通过限制每一层的压缩比例，在资源预算的限制下尽可能最优化网络性能。而在性能主导的应用场景下，该方法定义了一个关于网络性能和硬件资源的奖

励函数，在网络精度得到保证的前提下，尽可能地探索网络速度的边界。这两种压缩策略的设置可以灵活有效地帮助模型进行最终的部署。

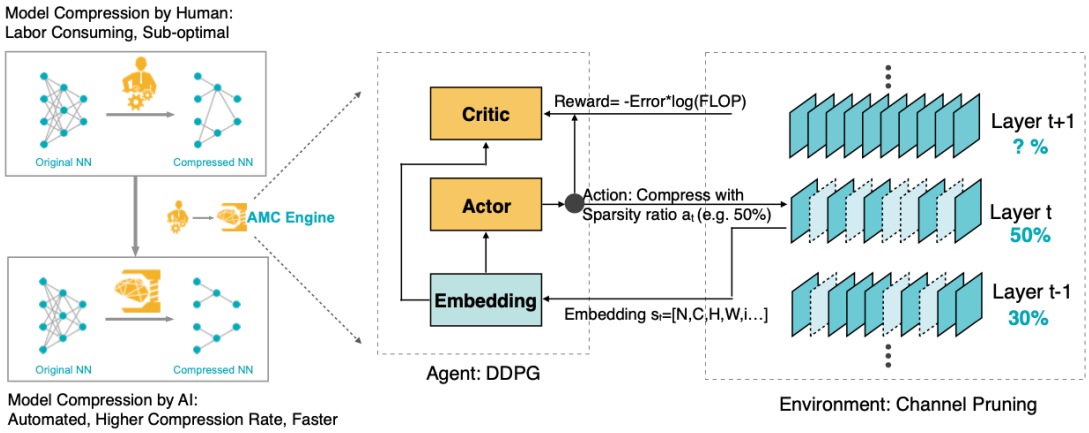


图 3-3: AMC 算法[5]

3.2 假剪枝方法

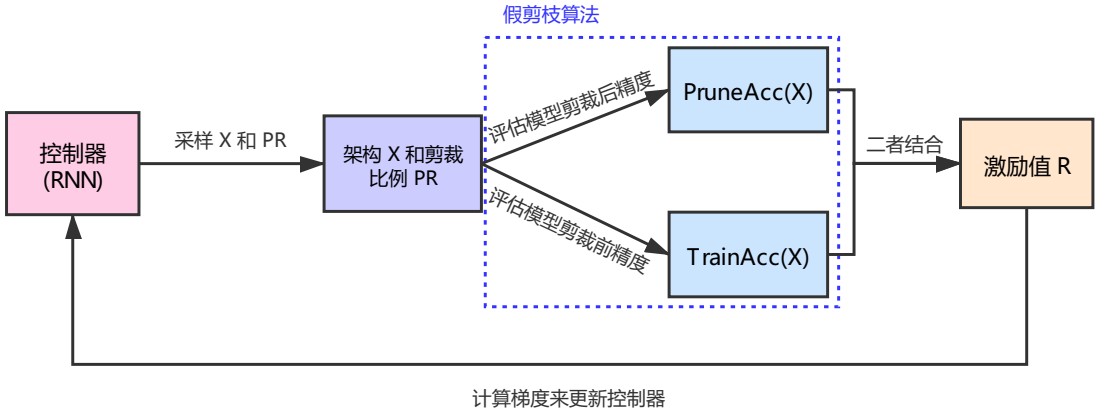


图 3-4: 本文算法流程图

传统的网络压缩方法总是将网络剪枝与网络训练交替着进行，每一轮网络剪枝之后总是会伴随着将网络模型进行重新训练，以获得当下剪枝策略的压缩效果，并以此来对剪枝策略进行调整。过去的诸多压缩方法也都是采取这样的策略，并且都取得了良好的压缩性能。但研究[29]认为，剪枝后的网络不需要进行重新训练也能获得相同的压缩效果，有时甚至可以达到更加卓越的性能。它认为是剪枝之后剩余的网络架构更加重要，而不是剪枝获得的参数权重更有意义，所以该研究认为剪枝的本质上就是一个最优网络架构搜索的过程。基于这

样的思路，本文提出的压缩方法结合了网络架构搜索的方法，即是前文提到的 ENAS 算法，意在对较优的网络架构进行搜寻。同时针对传统压缩方法的弊端，创新性地提出了假剪枝的方法。具体来说，它将网络剪枝以及网络训练进行解耦，摒弃了以往剪枝与训练交替进行的压缩方式，转而将两个过程按照并行的方式进行执行，而且互不干扰，只是利用不同剪枝策略下得到的模型精度差来引导剪枝方法的优化。最终通过一系列的对比实验以及消融实验，我们方法的有效性与合理性得到了有力的证明。

图3-4展示了我们方法的执行流程，首先我们的方法会利用搜索框架下的控制器对模型结构以及各个网络层的剪裁比例进行采样，继而就会进入到本文提出的假剪枝过程。假剪枝的方式会将网络训练以及网络剪枝看作是两个独立的步骤，同时网络剪枝也并没有真正执行，因而在整个模型搜索过程中，网络层的通道数都保持不变。在每一轮搜索结束的时候，我们会利用搜索得到的完整网络结构来获得该模型在测试集上的 $TrainAcc(X)$ 值，同时根据当前采样得到的各个网络层的压缩比例，将各层冗余的通道进行一个伪删除的过程，就是根据压缩比例屏蔽掉冗余通道的信息传递，只将剩余有效通道的信息向后面的网络层进行传播，利用这部分有效通道的网络参数计算得到得到测试集上的 $PruneAcc(X)$ 值。在分别获得剪裁前后的模型精度之后，我们会计算这两个值的差值，也即是网络模型因为剪枝操作而导致的精度差。根据得到的精度差来计算出激励函数的值，以此来获得更新的梯度而后对控制器进行反馈优化，帮助整个搜索架构搜索出更好的剪枝策略——更不易于损伤模型精度的压缩比例。在我们的实验过程中，我们发现这两个值之间通常存在较大的差距，也即 $PruneAcc(X)$ 值会远远小于 $TrainAcc(X)$ 值。这就意味着，搜索得到的剪枝策略通常会对网络精度有较大的消极影响，这也是本方法所竭力要解决的一大难题。后文通过激励函数的巧妙设置，这些问题都得到了较好的解决。

3.3 基于 ENAS 的通道剪枝

基于前文对于网络压缩的介绍以及相关神经网络架构搜索方法的简述，本文创新性地将他们进行结合。具体来说，首先是在网络搜索空间中加入了对于每个网络层剪裁比例的搜索，以获得每一层合适的剪裁比例。同时将所有层的

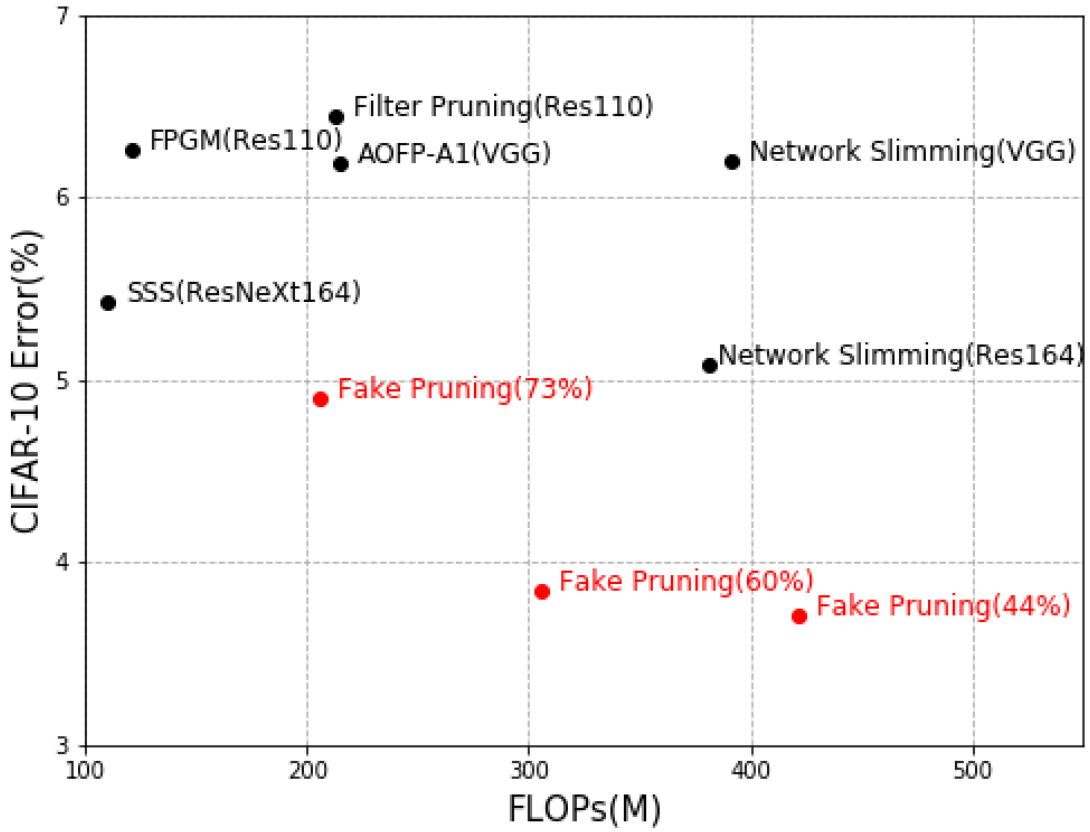
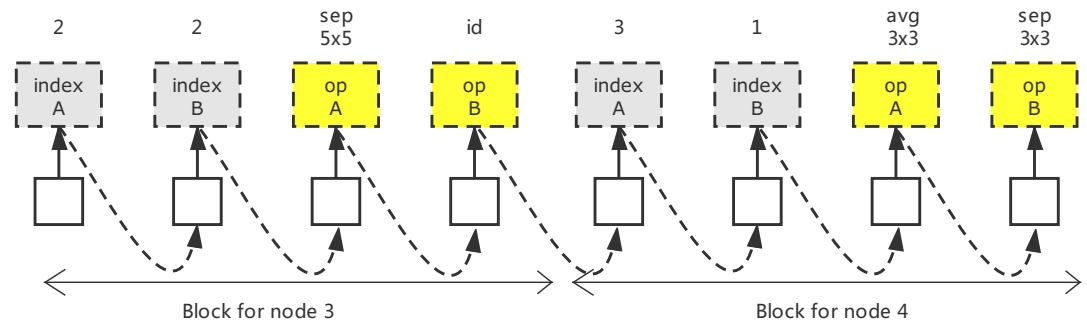


图 3-5: 本文方法与传统压缩方法对比图

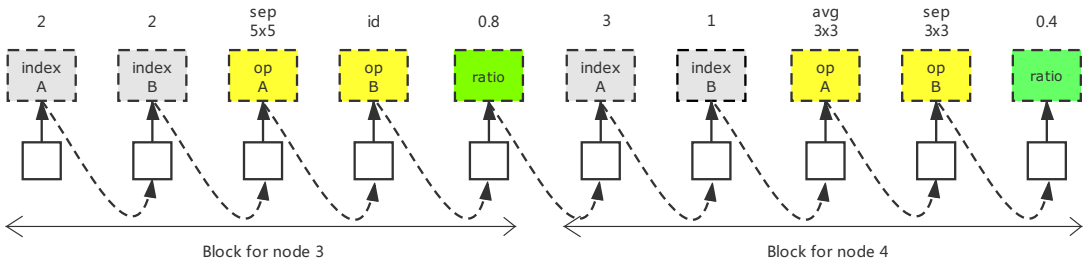
剪裁比例进行求和并融入到神经网络搜索的激励函数中，通过参数权重来对网络精度以及速度进行权衡。然后结合本文提出的假剪枝方法，也就是将网络训练和网络剪枝解耦，进一步提升方法压缩的效率。最终通过 CIFAR-10 消融实验以及相关参数的深入分析，强有力地证明了方法的合理性与有效性。图3-5 展示了本文方法与现今主流的一些手工压缩方法的对比图，红色部分即是本文方法设置的三种不同的压缩配置，分别是损失一定的精度来获得更好的压缩性能(73%)、损失一定的速度来获得更加优异的模型效果 (44%) 以及二者的合理权衡(60%)。图中它们相比于传统方法均处于坐标轴的左下角方位，这代表了精度或者速度等同的情况下，本文方法设置的三种模型的另一个指标均有效超越了其他方法，这佐证了本文方法的有效性。详尽的实验分析在后文的实验部分展开，在此便不做赘述。

3.3.1 搜索空间

图3-6 (a) 描述了 ENAS 方法中关于一个模块的搜索空间。其中，index A 和 index B 代表了该模块的两个输入节点索引。两个 op 操作则代表了这两个输入节点所对应的操作，具体操作在 3x3 卷积或者深度可分离卷积模块、5x5 卷积或者深度可分离卷积模块、平均值池化操作、最大值池化操作以及恒等变换操作中进行选择。最终在完成一个模块结构的搜索后，便是对相连的下一个模块进行搜索，这两个模块的重复堆叠即可完成神经网络的搭建。本文提出的方法与 ENAS 方法的不同点在于我们不仅对于一个模块的索引节点以及后续操作进行搜索，还会针对每个网络层进行剪裁比例的搜索，剪裁比例针对的便是模块中对应的两个索引节点。如图3-6 (b) 所示，紧接着在 op B 节点之后插入的 ratio 节点代表了当前模块一个适合的剪裁比例值，这便是前文假剪枝讨论当中提到的剪裁比例，它代表着一个伪删除的比例，并不会对网络模型做真实意义上的删除。需要说明的是，这里的剪裁比例是针对于其中特征图的通道维度而言，它代表剪裁的通道与总通道的比值。所以更大的剪裁比例意味着该特征图存在着较多的冗余通道需要进行删除，而更低的剪裁比例则意味着该特征图的冗余通道相对更少一些，因而需要进行剪裁的通道并不多。所以为了更加显著的压缩效果，我们会希望总的剪裁比例可以尽可能更大一些。针对每个网络层的两个节点，我们都会使用相同的剪裁比例，具体的剪裁手段主要基于该层的卷积操作，即是按照剪裁比例对卷积核的通道进行删除。剪裁比例对于卷积操作之外的例如池化操作以及恒等变换操作并没有意义，因此在遇到模块操作是非卷积操作类型时，剪裁比例都默认会被忽略掉，不会对该模块有任何影响。在剪裁完卷积通道之后，需要将两个节点的最终结果进行融合. 由于本文方法针对同一个模块下的卷积操作以及非卷积操作会得到通道数不等的两个结果，所以这里采取了通道补 0 的方式，将通道数较少的结果补充至与另一个结果通道数一致，然后采取矩阵求和的方式进行融合计算。在搜索空间中，索引以及操作的搜索均与 ENAS 保持一致，对于剪裁比例经过多次实验研究，最终设置了 $\{0, 0.2, 0.4, 0.6, 0.8\}$ 这样的搜索空间，便于方法更好地进行搜索。同时为了保证剪裁的过程不会对精度造成较大的损失，剪裁的方法采取了按照通道绝对值总和进行排序，根据剪裁比例将总和排序靠后的通道所对应的卷积核进行删除，较好地保证了网络的



(a) ENAS



(b) 本文方法

图 3-6: ENAS 与本文方法的搜索空间对比

性能。对于网络层 l，我们定义该层的剪裁比例为 $Ratio(l)$。通过这些精细化的设置，本方法能够更加针对性地搜索到每层最优的剪裁比例，为方法的可靠性打下坚实的基础。

3.3.2 激励函数设计

近些年的神经架构搜索方法都是基于强化学习的方式作为搜索策略，而激励函数则是强化学习中重要的一环，它能够引导控制器搜索得到优异的网络架构。通常情况下，神经架构搜索的最终目标就是得到一个性能卓越的神经网络，因而网络的精度即是该激励函数的最终优化目标。这一过程可以用公式 3-1来表示：

$$original\_reward = TrainAcc(X) \tag{3-1}$$

在本文方法中，不仅网络精度是激励函数需要进行优化的重要方面，同时我们的压缩方法也希望能够尽可能多地对网络模型进行压缩。因而我们也将整个网络的压缩比例总和融入到了激励函数之中，通过参数调节来在网络精度与网络速度之间取得一个较好的平衡。我们希望能够在尽可能不损伤网络性能的前提下提升网络的运行速度，因此我们将所有网络层的剪裁比例进行求和，希望激

励函数能够引导算法来最大化该值，该值的数学公示见公式3-2：

$$Ratio = \Sigma_{l \in L} ratio(l) \tag{3-2}$$

针对本文提出的假剪枝的方法，我们定义了 $TrainAcc(X)$ 以及 $PruneAcc(X)$ 两个变量，它们分别代表了网络剪枝前以及剪枝后网络模型在验证集上的精度。因为经历了网络剪枝，所以 $PruneAcc(X)$ 的值会比 $TrainAcc(X)$ 低一些，而我们所希望的是 $PruneAcc(X)$ 的值能够尽可能地接近 $TrainAcc(X)$ 的值。所以我们同时定义了 Gap 变量作为两个值的差值，同时希望该 Gap 值能够更大一些：

$$Gap = PruneAcc(X) - TrainAcc(X) \tag{3-3}$$

同时我们也引入了 β 值来调节 Gap 变量对激励函数的影响。当我们在优化 Gap 值的时候，我们可以容忍一定程度的负值，即我们允许 $TrainAcc(X)$ 的值比 $PruneAcc(X)$ 略大一些，因而在最终的激励函数计算中，我们将 $\beta * Gap + 1$ 的计算结果带入公式中。在最终的实验过程中，我们发现当 $TrainAcc(X)$ 值远大于 $PruneAcc(X)$ 值时，$\beta * Gap + 1$ 的值便会变成负数，这会对激励函数值造成较大的消极影响。从而促使激励函数趋向于去搜索能使 $PruneAcc(X)$ 值略小于甚至是大于 $TrainAcc(X)$ 值的压缩策略。结合前文所述的关于剪裁比例的公式 $Ratio$，我们将这二者进行求积结合，同时引入 α 值作为这一部分的参数调节：

$$pruning\_reward = (1 + \beta * Gap) * Ratio * \alpha \tag{3-4}$$

结合强化学习中原有的激励函数公式，最终我们的激励函数见公式3-5：

$$\begin{aligned} final\_reward =& (1 + \beta * (PruneAcc(X) - TrainAcc(X))) \\ & * \alpha * \Sigma_{l \in L} ratio(l) + TrainAcc(X) \end{aligned} \tag{3-5}$$

上述公式在模型精度搜索目标的基础上，加入了模型总的剪裁比例以及剪裁前后精度差的优化，较好地保证了方法能够搜索到对模型精度损伤小甚至还有增益的压缩策略。这也一定程度上提升了算法的压缩效率，在保证模型精度较高的前提下，极大地提升了模型速度性能。

3.4 训练流程

Algorithm 3.1 基于 ENAS 以及假剪枝策略的网络压缩流程

1: 初始化网络参数
2: **repeat**
3: 利用优化器来逐步训练整个计算大图
4: 利用控制器来采样网络 X 以及各层剪裁比例 PR
5: 计算未经剪裁的网络 X 的测试精度以及根据 PR 剪裁的网络精度来分别得到 $TrainAcc(X)$ 和 $PruneAcc(X)$ 的值
6: 根据公式 (3-5)以及获得的各个变量值来计算得到这一步骤下面的激励函数值
7: 根据激励函数计算参数更新梯度来优化控制器
8: **until** 网络搜索过程完成
9: 选取采样得到的最高精度的网络模型 X 并且根据相应的 PR 对其进行剪裁
10: **return** 最终的剪裁模型 X

我们的算法采取 ENAS 的训练框架，同时针对模型压缩的任务，对该框架进行了一定的修改，以帮助算法更好地对模型进行压缩。在算法执行过程中，首先需要对模型中所有参数进行初始化，接着就需要对这些参数进行优化训练，需要强调的是，这里涉及到的训练参数仅是被 ENAS 选择的网络模块中的参数，其他没有被选中的模块参数并不会参与训练。然后算法当中的控制器便会对网络组件以及各个网络层的剪裁比例进行采样，完成结果网络的搭建。在每个搜索轮次的最后，算法还会训练搜索得到的网络，获得该网络在测试集上的精度 $TrainAcc(X)$。该步骤也会利用搜索得到的剪裁比例对小网络进行压缩剪裁，并且训练该剪裁网络，以获得 $PruneAcc(X)$ 的值。得到这些结果之后，算法会根据公式 (3-5) 计算当前采样结果的激励函数值，通过强化学习的相关学习策略计算更新梯度来优化控制器，以引导网络学习到更加精简更加优异的小网络。重复上述流程，直到完成了预设的训练轮次。最终算法会选取网络搜索过程中采样得到所有网络里性能最优异的小网络，并且根据相应的剪裁比例完成对该网络模型的压缩剪裁。最终通过重新训练，得到精简模型。

3.5 实验与分析

为了验证本文算法的有效性，我们在 CIFAR-10 上做了一系列的实验分析。CIFAR-10 数据集由 10 个类的 60000 张 32x32 的彩色图像组成，其中每个类均有

表 3-1: 算法产生的更高精度的模型与其他主流压缩方法的对比

Method	Error(%)	FLOPs(10^8)	Params(10^8)
Network Slimming(VGG)[3]	6.2	391	2.3
Network Slimming(ResNet164)	5.08	381	1.44
Filter Pruning(ResNet110)[30]	6.45	213	1.68
SSS(ResNet164)[34]	4.83	248	1.65
ENAS + micro search space[4]	3.54	751	4.6
Pseudo Pruning(60%)	**3.85**	**306**	**1.7**
ENAS + micro search space + CutOut	2.89	751	4.6
Pseudo Pruning(60%) + CutOut	**3.27**	**306**	**1.7**

6000 张图片。50000 张图片用作训练，剩下 10000 张作为验证集用作测试。我们使用标准的数据预处理以及数据增强的手段，降低网络过拟合的概率。我们的实验环境是基于一张英伟达 1080Ti 显卡，完成网络的搜索以及最终训练。关于超参数的预设上，我们详细调整了 ENAS 方法的参数以帮助算法适应网络压缩的任务，因而诸多超参均与原论文当中的设置有所不同。在 adam 优化算法上，我们设置学习率为 0.0035，这是原论文当中设置值的十倍。针对搜索轮次数目的设置上，我们采取原论文的一半值，即 150 作为我们算法的搜索轮次。根据我们采用的硬件设备，我们将 batchsize 设置为 160，以最大化提升硬件利用率。对于我们提出的改进中涉及的超参数，我们也进行了精细的调整。我们将 Gap 值的参数 β 设置为 20，剪裁比例搜索空间设置为 $\{0, 0.2, 0.4, 0.6, 0.8\}$。考虑到实际环境需求的多样性，我们在后续实验中探索了两种不同压缩性能的模型，这里主要通过前文所述的 α 值进行调节。当我们希望搜索到的模型更加注重精度，以一部分的速度性能为代价时，α 值设置为 0.02；当我们追求更加轻量的模型，可以允许模型性能略有损耗时，α 值设置为 0.5。

3.5.1 CIFAR-10 的实验结果

我们在 CIFAR-10 数据集上进行了一系列的扩展实验来证明我们算法的有效性，我们遵循了算法3.1中的执行流程来运行我们的算法。我们针对不同的压缩比例目标设置了两套不同的参数，其中具有较高精度的模型可以达到 60% 计算量与参数量的压缩，而另一个追求更高速度的模型可以压缩掉模型 73% 的计

算量以及参数量。表3-1展示了我们方法与其他主流压缩方法在 CIFAR-10 上的分类精度以及模型计算量和参数量的实验对比。在本表格中，我们设置了 α 值为 0.02 来获得一个具有更高精度以及较低压缩比率的模型结果。其中，第一个部分展示了近些年主流压缩方法的结果，第二个部分展示了 ENAS 方法本身搜索得到的模型结果，这也是本文方法的基线模型。同时 Pseudo Pruning 即是 ENAS 搜索框架融合本文提出的压缩方法之后产生的精简模型结果，最后一个部分则是二者加了 cutout[35] 数据增强之后的对比结果。表格中可以看出，相比于经过 Network Slimming 方法压缩过的 VGGNet 和 ResNet164 两个模型，我们算法产生的模型在拥有更低的计算量以及参数量的前提下，还具有更高的模型精度，相比于这两个模型分别有 2.3% 和 1.2% 的精度提升。同时我们方法产生的模型与经过 Filter Pruning 以及 SSS 两种方法压缩后的模型具有相同的参数量，但却有一个点左右的精度提升。表格中同时也列出 ENAS 方法本身产生的模型结果，经过我们提出的一系列改进，最终算法产生的模型在牺牲 0.3 个点的精度情况下，可以有效裁剪 60% 的模型计算量以及 63% 的模型参数量。这一结果在 cutout 数据增强的使用与否下均保持一致，cutout 的数据增强还会进一步提升我们方法产生的模型精度，增强方法的竞争力。这充分表明了 ENAS 本身搜索产生的模型具有较大的网络冗余，而我们的方法在几乎不损伤模型精度的前提下可以有效地将这些冗余进行裁剪，提升模型计算速度。同时相比于其他主流压缩方法，我们提出的算法压缩效率更加优异。

为了进一步研究算法的有效性，在表3-2中，我们采取了较高的 α 值来获得一个具有较低精度以及较高压缩比率的模型结果。其中，第一部分展示当下主流压缩方法的结果，第二部分则是我们方法的模型效果。这一设置使得算法产生的模型更加精简，计算量与参数量分别为原算法产生模型的 27.4% 和 23.9%。同样的，经过 SSS 算法压缩过后的 ResNeXt164 与我们方法产生模型具有相似的参数量，但它却有更高的精度误差。相比较于 AFOP 和 SFP 两种方法优化之后的模型，我们的模型在保持一致的计算量前提下，具有更高的模型精度。虽然与 FPGM 和 GAL 两种方法相比，我们模型具有略高一些的计算量，但模型的精度的差距却是很显著——我们的方法有近一个点左右的提升。与这些近些年的热门方法相比，我们的算法在压缩效果以及模型精度上都有显著优势。更重要的是，传统方法大多基于手工剪裁的方式，这种方式费时费力，还需要较多的专家

表 3-2: 算法产生的更高速度的模型与其他主流压缩方法的对比

Method	Error(%)	FLOPs(M)	Params(M)
SSS(ResNeXt164)	5.43	110	1.1
AOFP-A1(VGG)[19]	6.19	215	/
SFP(ResNet110)[36]	6.17	216	/
FPGM(ResNet110)[37]	6.26	121	/
GAL-0.01(Densenet40)[38]	5.39	182.9	0.67
Pseudo Pruning(73%)	**4.9**	**206**	**1.1**
Pseudo Pruning(73%) + CutOut	**4.31**	**206**	**1.1**

经验，因而整个压缩工作比较繁琐困难。我们的方法是基于神经网络架构搜索的自动化方法，因而在压缩工作上相比于传统方法会有较大的效率提升。同时，算法也会自适应地去寻找每一个网络层最合适的压缩比例，从而进一步提升压缩的效率，这一点也是传统手工压缩方法不能实现的。

3.5.2 消融实验分析

本文在 ENAS 网络搜索的基础上提出了诸多算法改进，为了进一步验证这些改进的有效性，我们同时在 CIFAR-10 上做了相关的消融实验。表3-3是该实验的分析结果，其中 PR 是指搜索空间中加入每一层的剪裁比例的改进，FP 则是指本文提出的将算法训练与网络剪枝相解耦的假剪枝算法，α 和 β 是前文所述的两个超参数，当 β 等于 0 时，就意味着算法屏蔽掉了 Gap 值对最终的激励函数的影响。消融实验中，我们采用了 ENAS 产生的原有模型作为基准模型。当我们首先在搜索空间中加入对每层剪裁比例的搜索，同时结合假剪枝将算法训练与网络剪枝相解耦的思想，我们可以得到计算量参数量仅为基准模型一半的模型，此时网络精度下降了近一个点。需要说明的是，由于此时 β 值设置为 0，所以 Gap 值并未对激励函数有任何影响，也即算法并未被显式引导去搜索那些对精度损伤较小的压缩策略。而当我们将 β 值设置为 20 的时候，不仅网络计算量以及参数量得到了轻微的下降，网络精度也得到了较为明显的提升，相比于前者提升了 0.5 个百分点。此时的结果相比于基准模型仅有轻微的精度损失，但模型复杂度却得到了极大的降低。增大 α 值会令模型得到更大幅度的精简，但

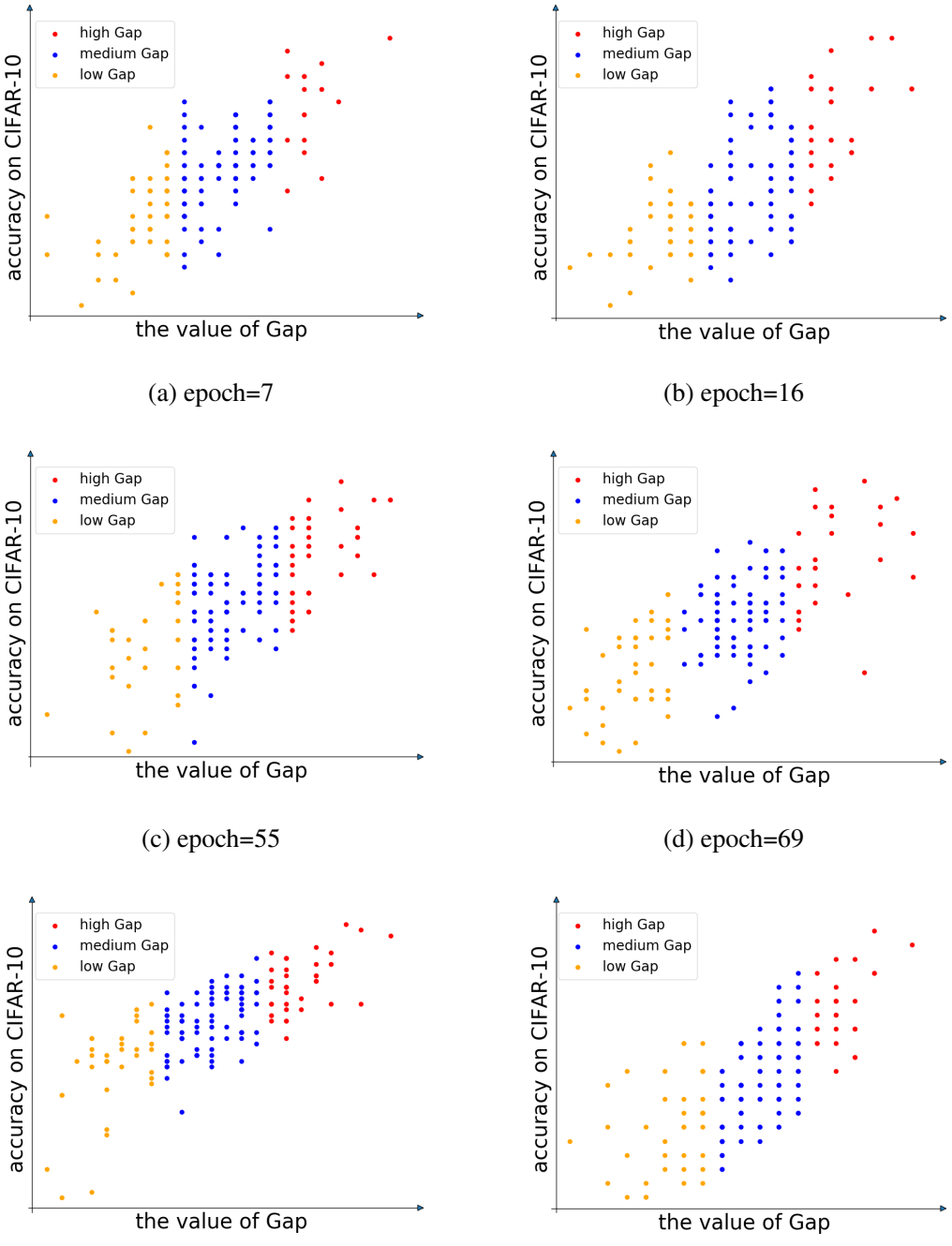
表 3-3: 基于 CIFAR-10 的消融实验。

Method	Error(%)	FLOPs(M)	Params(M)
Baseline(ENAS)	3.54	751	4.6
PR + FP(α = 0.02 β = 0)	4.3	321	1.8
PR + FP(α = 0.02 β = 20)	**3.85**	**306**	**1.7**
PR + FP(α = 0.5 β = 20)	4.9	**206**	**1.1**
PR + FP(α = 0.005 β = 5)	**3.71**	422	2.4
PR + FP(α = 0.005 β = 5) + CutOut	**3.11**	422	2.4

这需要损失一定的模型性能，采用与否要根据实际情况来选择合适的参数设置。表3-3中也展示了其他参数设置下得到的模型结果，通过对于 α 和 β 的灵活设置，模型性能有望得到进一步的提升。这些实验结果较好地证明了我们对于原方法的改进是卓有成效的，在不损失较多性能的前提下，能够较大程度地去除模型的冗余。同时本文算法灵活度较高，可以根据需要来自由调节模型压缩程度，更加便于模型部署与应用。

3.5.3 Gap 值分析

本章节继续对前文所述的 *Gap* 进行深入分析，为了探究其对最终模型压缩性能的影响，我们随机选取了搜索过程中的 5 个搜索轮次。同时为了避免算法优化对于实验分析的干扰，我们特地选取了控制器未被优化的一个训练轮次。我们以这 6 个搜索轮次作为我们的研究对象，在每一轮搜索的最后阶段，算法总是会利用当下的控制器参数来随机采样出 150 个网络模型，同时计算出它们相应的 *Gap* 值以及测试集上的模型精度。因此最终我们会得到 900 个不同训练程度下模型的 *Gap* 值以及对应的测试精度。在图3-7中，前五张图片是来自于整个搜索过程的 5 个轮次，最后一张图则是训练器未被优化的一个搜索轮次。图中橙色数据点代表着较低的一批 *Gap* 值，蓝色数据点代表着中等的一批 *Gap* 值，红色数据点则是较高的一批 *Gap* 值。可以明显地从图中观察到，*Gap* 值和测试精度之间存在着正相关的数据关系。具体来讲，当某个采样模型具有较低的 *Gap* 值，假设它小于 0，或许由于我们采样出来的压缩策略导致了该模型的压缩后精度小于压缩前的模型精度，此时这个模型经过当前的压缩策略最终会有较大可



(a) epoch=7 (b) epoch=16

(c) epoch=55 (d) epoch=69

(e) epoch=113 (f) no controller training

图 3-7: *Gap* 值以及对应模型精度的相关性分析

能得到一个较低的精度结果。而当某个采样模型具有中等的 *Gap* 值，此时该模型的压缩前精度或许近似于模型的压缩后精度，表明此时我们搜索出来的压缩策略对于模型精度损伤并不太大，该模型最终经过策略压缩之后也只有较大可能是中等的精度性能。如果某个采样模型具有较高的 *Gap* 值，即该模型的压缩精度略大于训练精度，此时表明我们的算法甚至搜索到了一些对于模型精度有

增益的压缩策略，那么该采样模型经过该策略剪裁之后最终较大的概率会得到较好的网络性能。需要强调的是，这样的现象不仅贯穿整个搜索过程，甚至于控制器从未被训练的搜索轮次下也出现了相似的现象。这就表明这样的现象是一个恒定特征，它不随着网络的训练学习而改变。这样的发现也与前文的消融实验中观察到的结果相呼应，即当 *Gap* 对激励函数产生影响时，算法搜索到的模型从性能以及速度上均有显著提升。这也有力地证明了，我们对于激励函数的修改是有益有帮助的。

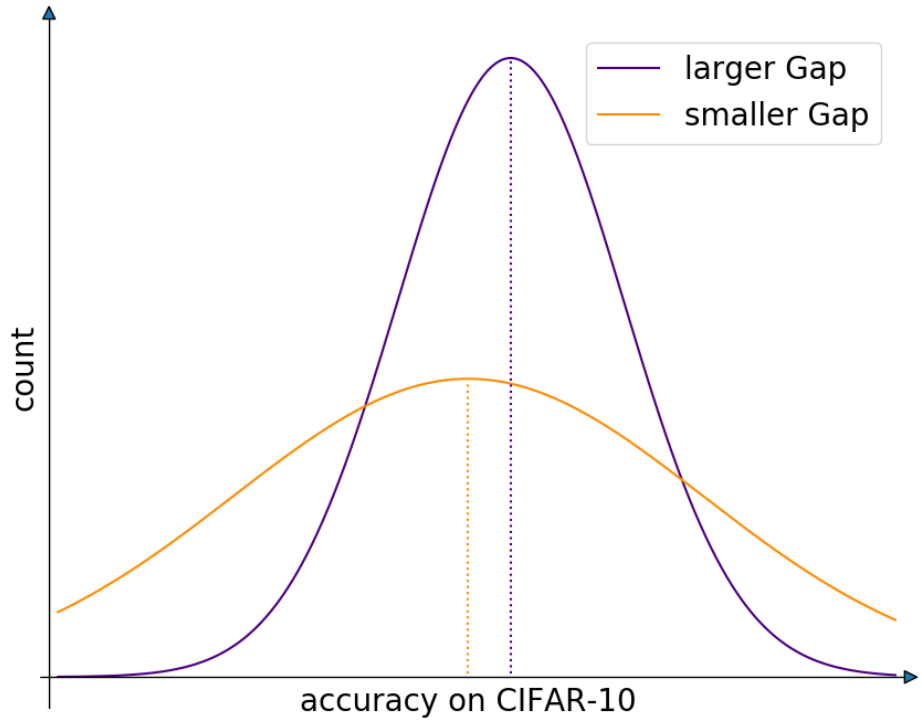
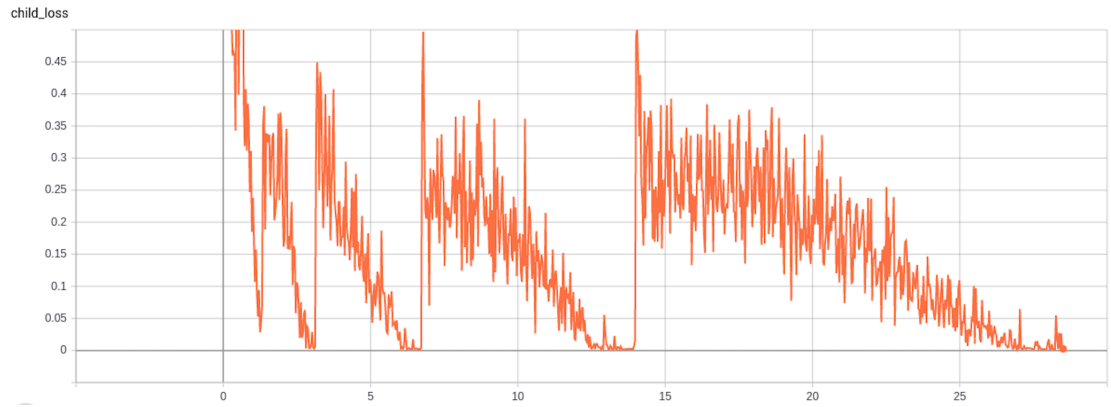
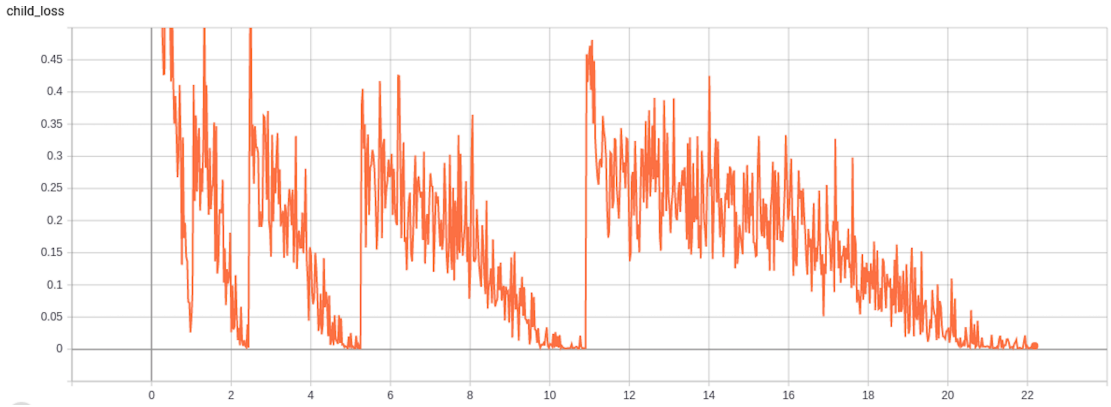


图 3-8: 高低 *Gap* 值关于对应网络精度的正态分布函数。

此外，为了进一步探究那些具有较大 *Gap* 值模型对于拥有较小 *Gap* 值模型的优势，我们在整个搜索过程中的最后 20 个搜索轮次下，每个轮次利用当前控制器的参数随机采样了 150 个网络模型，一共是 3000 个网络模型，依次计算出它们的模型精度以及对应的 *Gap* 值。然后我们按照 *Gap* 值大小将这些模型分成两个部分，一类是具有更高 *Gap* 值的模型，另一类则是具有相对较低 *Gap* 值的模型。最终根据它们的 *Gap* 值以及模型精度绘制了相应的正态分布函数曲线，该函数图像如图3-8所示。从图中可以清晰地看到，紫色曲线代表着更大 *Gap* 值



(a) baseline



(b) fakePruning

图 3-9: 我们算法模型与基线模型收敛性分析

的数据部分，它在模型精度的维度上拥有着更低的方差以及更高的均值。这就意味着，拥有更高 *Gap* 值的模型往往有更大的概率来获得较高的模型精度，这正是我们算法所追求的目标。而橙色部分的低 *Gap* 值曲线所对应的网络模型精度往往较为发散，它们的精度均值也有较大概率低于前者，这也佐证了前文所述的那些拥有较低 *Gap* 值的网络模型最终的精度效果会相对较差一些的结论。所以我们可以基于 *Gap* 这样的特质来改进我们的算法，将其引入激励函数当中并作为其的一个优化目标，让算法尽可能去挖掘那些能够最大化 *Gap* 值的压缩策略，达到在压缩比率恒定的前提下还较好地提升网络模型精度的效果，从而支撑起我们算法较好的压缩效果。上述两个分析实验都强有力地证明了我们引入的 *Gap* 参数对于模型架构搜索以及网络压缩具有较大的意义，我们基于此对网络搜索以及压缩的改进也是卓有成效。

3.5.4 收敛性分析

通常网络压缩不仅使模型在部署阶段受益，加快它们的推理速度，还能提升模型训练的收敛速度，大大缩减模型的训练时间。基于此，我们也做了模型的收敛性分析，我们以 ENAS 算法搜索到的最好模型作为基线模型，同时将我们算法在 α 值设置为 0.02 的配置下搜索得到的模型作为对比模型。该配置下会获得一个精度较高，但速度相对略低。相比于基线模型能够剪裁 60% 的模型计算量和参数量，但会损失 0.3% 分类精度。我们基于这两个模型进行实验，有力地证明了我们算法为模型的训练带来了收敛加速，实验结果如图3-9所示，图中纵坐标为模型训练过程中的分类损失值，横坐标为模型的训练时间。需要强调的是，我们在对于两个模型的优化训练中，采取了相同的参数配置，同时为了模型更好地训练，都针对学习率的变化采用了余弦退火算法 (SGDR 方法[39])，这会导致学习率的大小出现周期性变化，从而相应的分类损失值也呈现出如图所示的周期性变化。图中上方是我们基线模型的分类损失变化，下方是我们方法改进之后得到模型的分类损失变化。每个模型都训练了 630 轮，每轮训练大约 350 ~ 400 个 steps，并且按照余弦退火算法的设置变化，学习率分别在第 10、30、70、150、310、630 轮达到周期变化的最小值。相应的，分类损失值理论上会在第 3.5k、10.5k、24.5k、52.5k、108.5k、220.5k 左右的 steps 上取得极小值，将这些 steps 映射成相应的时间之后，便与我们在图中所观察到的一致。但我们可以清晰地从图中发现，在每个极点的附近，我们算法产生的模型在训练过程中会更早地到达分类损失的极小值位置。具体来说，在抵达六个极小值的耗时上，基线模型分别用了 0.4h、1.2h、2.8h、6h、12.5h、26.6h，而在经过我们方法改进之后，这些用时分别变成了 0.3h、1h、2.2h、4.5h、9.7h、20.2h。从这些数据中可以看出，我们的算法对于模型进行了有效的精简加速，在每个学习率的变化周期内都能较大幅度地缩减模型的训练时间，整体训练时间由基线模型的 28.5h 缩减到优化模型的 22.2h，降低了五分之一的训练时间消耗，而且可以较好地保证模型精度性能。这充分证明了我们对于原模型的压缩操作是合理有效的，它成功地帮助了模型删减掉了冗余复杂的部分，帮助模型的优化空间更加平滑，对于分类知识的学习更加简单，较好地提升了模型的泛化能力。收敛性分析实验强有力地证明了我们的方法对于模型训练的改善是卓有成效的，它较好地提升了

模型训练的速度，同时也能改善模型自身结构，帮助模型优化训练地更快更好。

3.6 本章总结

本章基于 ENAS 的算法提出了一系列改进，使得新算法能够较好地对网络模型进行压缩，同时结合了本文提出的假剪枝算法以及重新设计了激励函数，极大地提升了算法效率。相比于传统方法耗时耗力以及对于研究者提出了较高要求的缺点，本章的方法能够自适应地找到适合每个网络层的压缩比例，并对其进行合理压缩。在提升网络压缩便捷性的同时，也极大提升了对于网络模型压缩的效率。本章的后半部分通过在 CIFAR-10 数据集上的充分实验，有力地证明了本文方法的有效性。最终通过对本文方法进行的消融实验以及关键变量的详细分析，更加有效地证明了本文提出的一系列改进的合理性。同时本文方法也针对具体场景需求设置了两套不同的搜索参数，以便算法能够更加灵活地应用于实际场景。

第四章　基于算子重写的工程推理加速

随着人工智能技术在各行各业的广泛应用，深度学习模型在云端的大规模部署对诸如 GPU、CPU 等的计算资源产生了越来越大的依赖。而目前 GPU 成本高昂和计算资源紧缺以及 CPU 资源利用率较低等的现状，为深度学习模型的部署带来了较大的挑战。同时当下深度学习环境中存在着各式各样的训练框架，各个框架训练得到的模型又千差万别，这无形之中又为模型的部署工作带来了较大的工作量。此外，面对庞大的用户群体，服务运行方也对模型部署提出了更高的低延时高吞吐的要求。而随着深度学习的快速发展，模型迭代速度在日益增长，这同时又与提出的更高要求产生了较为尖锐的矛盾。在这样的大背景下，关于深度学习模型的工程推理优化研究显得愈加重要，如何加快提高深度学习模型的部署效率同时大幅减少模型服务部署成本，帮助业务团队快速落地算法应用，辅助模型又好又快地在云端运行，这便是模型推理优化的目标。

图 4-1: 深度学习对部署平台的挑战

深度学习模型的推理优化通常是针对那些需要部署到云端的深度模型，业界也有一些常用的深度学习推理服务框架，他们包括了谷歌的 Tensorflow serving[40]、英伟达的 Tensor RT inference server，亚马逊的 Elastic Inference 等。推理优化首先需要确认最终的优化目标，通常对于服务化后的模型，吞吐量以及网络延时是需要关注的两个方面，这需要针对不同的模型类别选择适当的加速方法进行优化。同时也需要对模型的速度瓶颈进行性能分析，继而做针对性的优化加速。最终在目标硬件的部署上，也可以进行深入优化。通常推理优化的方法可以分为系统级、应用级以及算法级三个层次。系统级别的优化主要基于硬件的角度进行加速，包括了使用 OMP 的数学库进行并行计算等；应用级主要从服务的角度进行流水以及并发的优化，包括数据处理以及网络请求的响应等操作；算法级的优化则主要从模型角度进行优化，主要指前文所述的参数修剪、量化加速等方法。本章节主要从系统级加速的角度对基于 CPU 运算的深度模型进行加速，主要采用了基于 Tensorflow 自定义算子的方式，对于模型中的耗时算子进行高性能重写，最终达到较好的模型加速效果。

4.1 Tensorflow 算子重写

TensorFlow 是一个基于数据流编程的的符号数学系统，被广泛应用于各种机器学习算法的编程实现。在 Tensorflow 中，使用图的概念来表示计算任务，但其仅用来搭建网络架构图以及表示图中的所有计算，并不参与计算。它的计算都被放在称为会话的上下文中执行，计算当中会涉及各式各样的数据流动，这些数据都被张量的概念进行表示。零阶张量称为标量，通常代表一个数；一阶张量为向量，由数组组成；二阶张量为矩阵，由二维数组组成。计算图中的每个节点都代表着图中的某个计算操作，张量形式的数据在这些节点间流动参与计算，最终构成整个图的计算过程。通常网络的耗时部分均在计算操作上，所以针对 Tensorflow 框架产生的模型进行加速优化本质上就是针对图中的各个计算节点进行计算加速。除了 Tensorflow 预先定义的计算操作，它也支持个性化的计算操作定义，这便为模型加速提供了一定的便利性。本章节基于这样的方式，首先通过对于模型的耗时分析得到热点算子，在算子重写的过程中集成一些高性能计算方法，提升硬件利用率，最终完成模型部署加速。

为了适应研究者更加多样的需求，Tensorflow 支持研究者自定义他们想要的算子，但重写的规范要符合 Tensorflow 的要求。为了使定制算子能够跟原框架兼容，研究者们需要按照相应流程进行算子重写。首先需要利用 C++ 语言来对算子进行注册定义，在其中不必涉及算子实现部分，而仅需要指定算子名称以及输入输出格式等信息。接着便是对这些定义的算子进行实现，在这其中完成研究者想要实现的操作，需要说明的是，操作的实现需要在 Tensorflow 定义的框架下进行，各个算子相应的输入输出标准都要符合 Tensorflow 的规范。在注册算子时，会自动生成一个默认的包装器，它是创建算子的一个公开的接口，当然也可以自定义一个新的包装器。根据实际需要来实现该算子的梯度计算以及可选地进行一个形状推断函数的实现，使得该函数能够从算子中推断出输入输出数据的形状。最后便是进行重写算子的测试，要保证该算子可以在 Tensorflow 的框架下正确地被调用以及最终的计算结果要与正确结果保持一致。在完成自定义算子的编写与测试之后，就可以进行下一步骤，即算子的替换流程。

对于前一步骤实现完毕的自定义算子方法，需要首先利用 $g++$ 将其生成出对应的动态链接库，然后需要将原模型中待转化的模块输入输出节点进行提取，融合转化成一个新的节点，此时该节点的操作名称需要指定为我们自定义编写的算子名称，结合 Tensorflow 提供的 tf.load_op_library 函数接口完成动态库的加载，最终 Tensorflow 会根据节点操作名称以及自定义算子注册名称来进行匹配，完成算子的替换。图4-2展示了算子替换前后网络图的变化，对于该网络模型，我们将其中两个网络模块进行融合，它们之中包含了大量的计算操作，我们通过对这些操作进行融合重写，并结合了相应的加速方法，使得转化之后算子计算效率大大提升。而替换算子之外的网络结构均不发生变化，这也使得通过网络算子替换来对模型进行加速的方式更加灵活高效。

4.2 模型分析

本章节实验的对象是基于上一章节提出的假剪枝算法产生的模型，也即是网络搜索方法经过我们算法优化剪枝之后得到的模型。而由于该模型是由多个结构相同的子模块堆叠而成，它们之间的优化存在共通之处。为了突出本章节的优化工作而减少不必要的重复繁琐工作，我们对于前述算法产生的模型做了

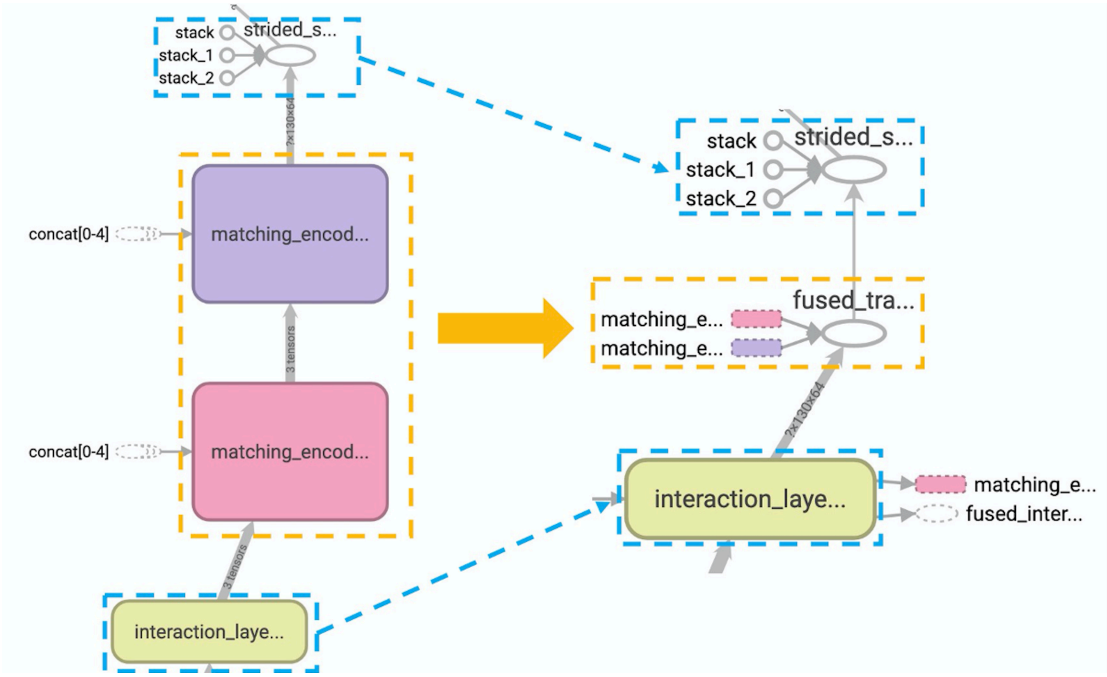


图 4-2: 算子替换前后的计算图变化

一定的简化工作。我们将原定算法中设置的 6 个网络层设置为 1 个，每个网络层中原先包含了 5 个子模块，在我们最新的设置当中修改为 1 个子模块，同时原先设置的 5 个分支也降低为 2 个分支。需要说明的是，这样的简化会导致网络性能大幅下降，但因为本章重点在于介绍推理优化的工作，而推理优化的过程并不会损伤网络精度，所以这样的方法是可以在保证网络精度的前提下，大幅提升网络运行速度。对于复杂网络的优化只是提升了工作量而已，本章节的内容也是基于简化之后留存的核心操作进行推理加速。

4.2.1 模型结构组成

如图4-3所示，我们的网络模型由 stem_conv、layer_0、relu、mean、fc 几个模块组成，而经过我们的耗时分析，relu、mean、fc 三个部分的运行耗时基本可以忽略不计，主要的运行延时都在前两个部分上，因此我们的优化也主要是集中在这两个模块上进行。

其中，stem_conv 模块由一个卷积操作以及一个批归一化操作组合，而对于 layer_0 模块，它的构成比较复杂。首先在 calibrate 模块中，是两个卷积和 bn 操作的组合，而后延伸出两个分支。其中一个分支经过 layer_base 模块中卷积 bn 组合的信息提取，送入到 cell_0 模块当中，cell_0 是两个深度可分离卷积的组合，

它们的结果最终通过矩阵加法进行结合。而另一个分支会进行池化卷积等操作，最终结合 cell_0 模块得到的结果，采取了 concat 的组合方式，将两个结果融合到一块。最终完成了本层网络信息的处理。

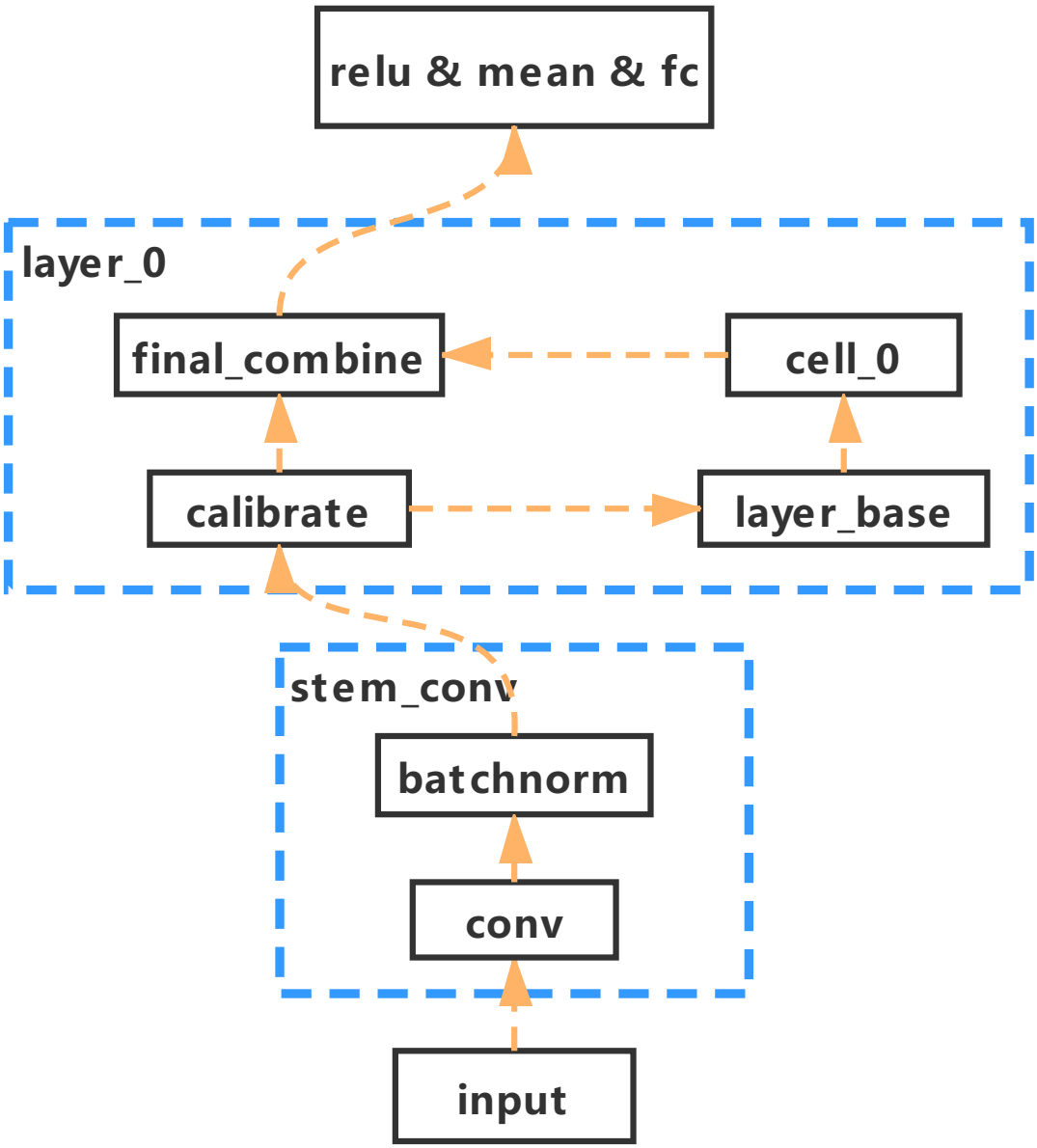


图 4-3: 网络结构图

4.2.2 算子热点分析

在分析完模型结构之后，推理优化的下一步即是对网络结构中的算子进行耗时分析，这里我们采取的方法是直接采用 Tensorflow 提供的 run 方法。当我们需要对某个模块进行耗时分析的时候，便对该模块的输入输出节点分别运行 run

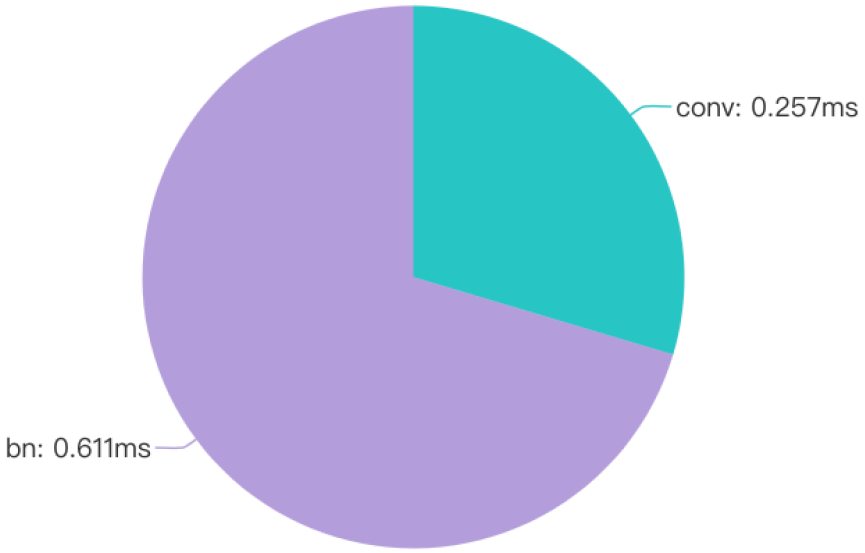


图 4-4: stem_conv 耗时分析

方法，在 run 方法的开始以及运行结束利用 time 库来获得节点的运行时间，最后将输入输出节点的运行耗时做差即可得到该模块内部的计算耗时。需要说明的是，为了保证实验测速环境的稳定性，我们统一设置了 Tensorflow 多线程环境下算子间以及算子内的线程数均为 1，omp 线程数也同样为 1，以便公平对比速度。按照上述测速方法以及相关参数的设定，我们对整个待优化模型进行了逐个模块、逐个算子的性能分析，最终的测速结果如图4-4以及图 4-5所示。其中，图4-4是有关 stem_conv 模块的耗时分析结果，而图4-5则是 layer_0 模块的耗时分析结果。从这两张图中可以分析得到，在 stem_conv 这个模块中，仅包含了一个卷积操作以及一个 bn 操作，从分析上看卷积操作占了该模块较少的计算时间，批归一化占用较多的时间，这便为我们后续的优化加速提供了方向性的指导。而在 layer_0 这个模块中，各个子模块耗时比较均等，同时每个子模块中均有各式各样的卷积操作。进一步的耗时分析又表明，在这些子模块中，卷积操作也是其中相对较为耗时的算子。因为整个计算图中流动的矩阵并不是很大，所以涉及到相关的访存操作并不会占用太多运行时间。因而在整个模型的推理优化过程中，最核心的要点便是针对卷积算子的加速优化。在我们的优化实践过程中，卷积操作的高性能重写是众多算子当中较为困难但同时加速效果却是比较明显的一个模块。

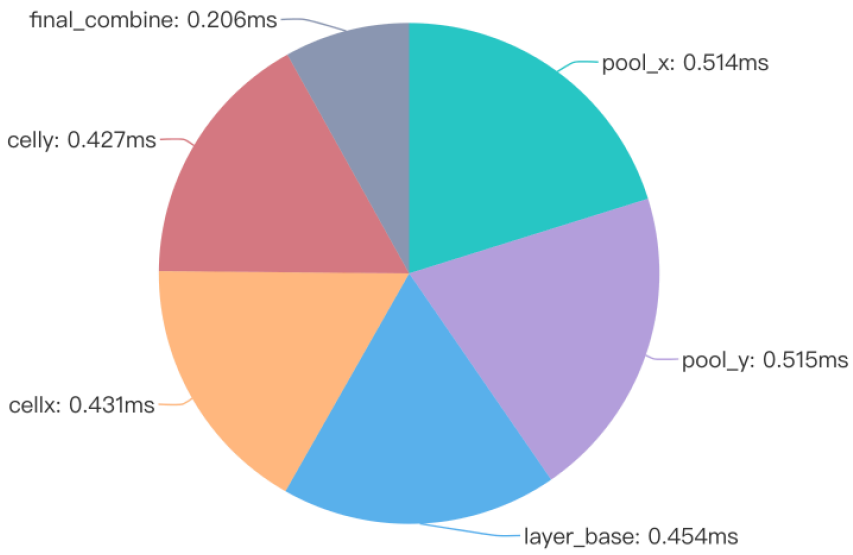


图 4-5: layer_0 耗时分析

4.3 算子高性能重写

本文模型高性能重写采用了上述 Tensorflow 算子重写的方式，将化简后的模型基于 c++ 实现的方式，对其中涉及到的所有算子进行重写，同时根据模型特性灵活调整数据结构，结合 omp、mkl 等加速手段完成算子的推理优化加速。重写代码需要经过 g++ 编译生成动态链接库，同时原模型的图结构需要将待替换的模块进行合并，在模型运行过程中通过调用动态库来实现算子替换，完成整个模型的加速。为了实现过程当中数据的统一性，本文设计了一套类似 tensor 的数据结构类 matrix，具体的数据结构成员如表4-1所示。不考虑 batch 维度下的特征图通常是三维结构，在 matrix 结构表示中需要将它们转化成二维，第一维对应了特征图的长宽，第二维对应特征图的通道维度。四维的卷积核也采用同样的方式，将卷积核的长宽以及通道数展平放置到第一维上，卷积核的数量维度也即输出特征图的通道数维度作为第二维。剩余涉及到归一化中的伽马和贝塔参数虽然只有一个维度，但也采用 matrix 的数据结构，它们只利用了 matrix 结构当中的第二维。同时为了提高数据的复用性，减少不必要的空间开销，该数据结构也支持同数据结构之间的复制以及部分引用。此外，我们也基于 chrono 高精度时钟库实现了计时器，以提供更加精准的计时测速功能。

经过之前的模型分析，我们将模型拆解成了两大方面进行推理优化的工作。

表 4-1: matrix 类成员变量

变量名	类型	说明
shadow_	布尔	该类对象是否是其他对象的引用
rows_	整型	该类对象的行数
cols_	整型	该类对象的列数
stride_	整型	该类对象的访问步长
data_	指针类型	该类对象指向的数据位置
alloc_size_	整型	该类对象申请的数据空间大小

一个是 stem_conv 模块的优化，它仅包含了 conv 操作以及批归一化操作，优化简单；另一个则是 layer_0 模块的优化，它不仅包含了深度可分离卷积这样非常规的卷积操作，同时网络架构上也比较复杂，算子模块之间的连接纵横交错，优化的工作相对更加复杂一些。在这两个模块之中，最为重要的就是关于卷积操作的重写加速，卷积由于其操作的复杂性，我们想要将其转化成矩阵乘法的操作以进一步集成高性能方法，而这就要依赖于 im2col 算法。

4.3.1 im2col 算法

im2col 算法就是将传统的卷积乘法转换成矩阵乘法的一种方法，它的核心就是将卷积核对应的图像部分转化成按行或者列进行存储，来帮助减少访存时间，从而提升计算效率。图4-6详细展示了 im2col 的算法过程，上方三张图代表了维度为 3x3x3 的特征图内容，每一个小图都代表着一张特征图，下方的大图则是该图像经过 im2col 算法之后得到的结果。每一行代表了每次卷积核滑动所对应的图像区域，将它们展平横向放置到结果矩阵中。这样做的用意在于，通常计算机程序一次性读取相近的内存是最快的，而卷积操作的特性导致了每次卷积过程中所取的图像数据都是不连续的，从而会带来较多的访存负担。而 im2col 算法正是为了减缓这样的访存负担所以被提出，同时也能将卷积操作转换成矩阵乘法，便于进行更多的并行加速计算。

本文自行实现了 im2col 算法，首先是实现图像的补零操作，以实现卷积中的 pad 功能，接着便是实现 im2col 的过程，将补零后的图像按照卷积核滑动的顺序来提取对应的数据到结果矩阵当中，这一过程需要保证结果矩阵的每一行

就对应卷积核一次滑动。这两个操作过程均由多个循环构成，在这里我们采用 omp 的方法来对它们进行加速。具体来说，在循环的外层采用 omp 的并行化加速，开启多个线程同时执行循环内部操作，而对于循环的内层则是采取了 omp 的向量化加速，基于单指令多数据流的方式完成循环内部操作的向量化运行。通过 im2col 算法的高性能重写，极大地提高了卷积的计算效率。下一节的实验分析有力地验证了这一点。

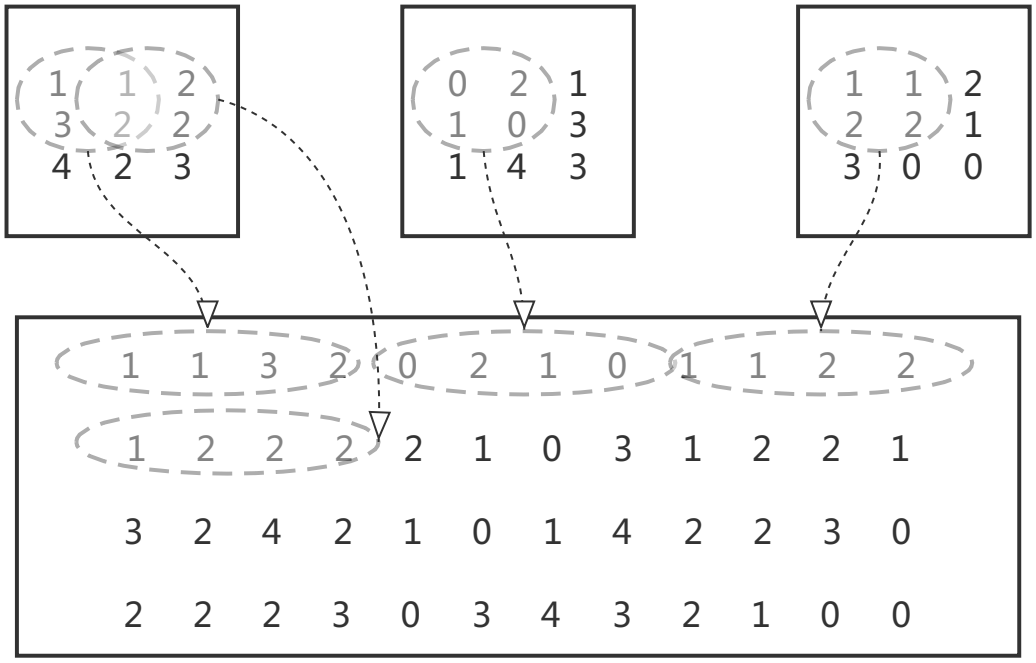


图 4-6: im2col 算法示意图

4.3.2 stem_conv 模块优化

如图4-3所示，stem_conv 模块仅由卷积操作以及批归一化操作组成，卷积的优化首先利用上述 im2col 算法将图像展开，然后结合 mkl 提供的 sgemm 高性能矩阵乘法函数完成后续的矩阵乘法操作，批归一化的重写中集成了常量预计算的方式降低计算量，同时利用 omp 对循环操作进行加速。优化后的耗时分析如表4-2上半部分所示，最终优化之后的运行耗时为 0.953ms，比基于 Tensorflow 引擎运行的 0.868ms 耗时还要略慢一些。经过细致分析，重写的卷积操作以及批归一化操作相比于原生框架的计算时间均有不俗的提升。耗时的原因还是在于批归一化操作前后的矩阵转置操作，而这产生的原因在于批归一化通常是针

表 4-2: stem_conv 模块优化分析

tensorflow 框架下运行耗时					
conv			bn		total
0.257ms			0.611ms		0.868ms
stem_conv 模块初步优化耗时					
im2col	sgemm	transpose	bn	transpose	total
0.107ms	0.082ms	0.342ms	0.276ms	0.146ms	0.953ms
stem_conv 模块最终优化耗时					
im2col	sgemm	transpose	bn	transpose	total
0.096ms	0.081ms	——	0.23ms	——	0.407ms

对矩阵最后一维进行操作，按照批归一化的常规写法是需要对矩阵进行转置的，转置之后便可以转为对第一维直接进行操作，但与此同时也引入了转置的代价。从表格中可以看出，批归一化的操作引入了前后两次矩阵转置，它们占用了整个模块近一半的运行时间。针对该问题，本文创新性地修改了批归一化的实现，使之能够不提升计算量的前提下可以灵活地针对最后一维展开操作，同时也可省去批归一化前后引入的矩阵转置的费用。最终优化结果如表4-2下半部分所示，重写批归一化省去了两次转置操作，最终带来了相比于归一化重写前有一倍性能的提升。需要说明的是，多次测速的结果有一定波动，因而在表格的展示中同一个模块的两次测速结果会有细微的差距。与图4-4中 Tensorflow 框架的运行速度相比，我们优化后的卷积操作耗时仅为 0.177ms，性能提升了 30% 左右，优化后的归一化操作仅为 0.23ms，耗时接近原框架运行的 1/3。整体性能提升一倍，优化性能较为显著。

4.3.3 layer_0 模块优化

layer_0 模块的结构较为复杂，简化结构图如图4-7 所示。该模块的输入首先经过 calibrate 模块中的两个卷积与归一化的组合，延伸出两个分支。一个分支经过 layer_base 中的计算操作将结果送入 cell_0 模块中，该模块包含了两个子模块，每个子模块都是由两个深度可分离卷积堆叠而成，两个子模块最终通过矩阵加法将结果进行融合。另一个分支送入到了 final_combine 模块的子模块

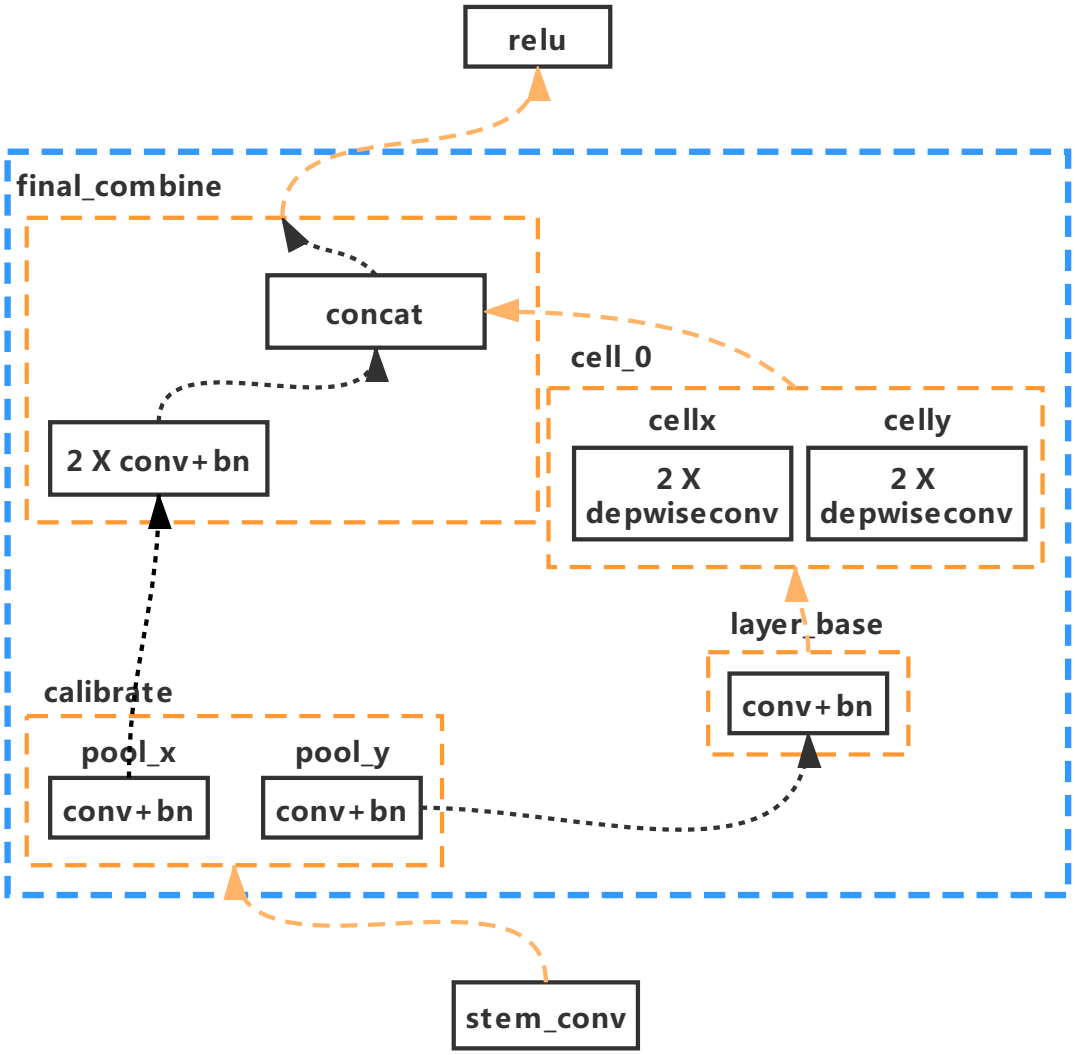
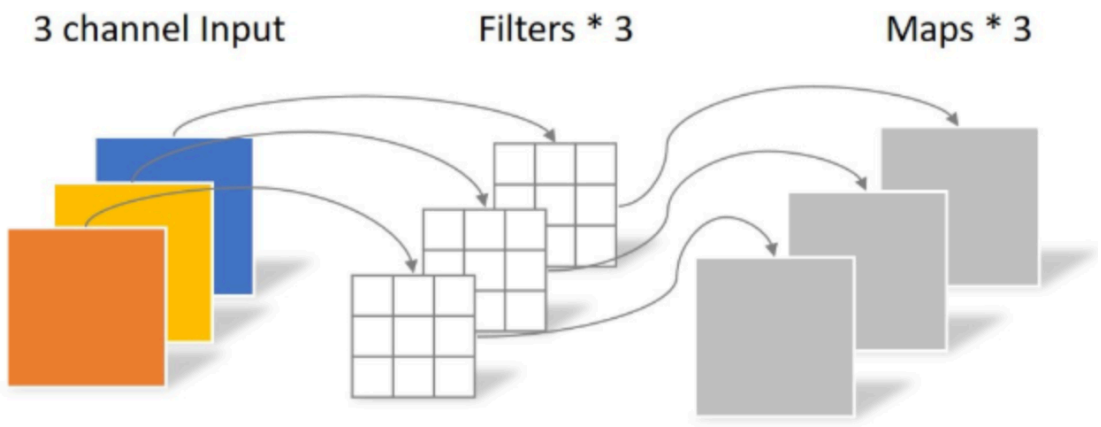
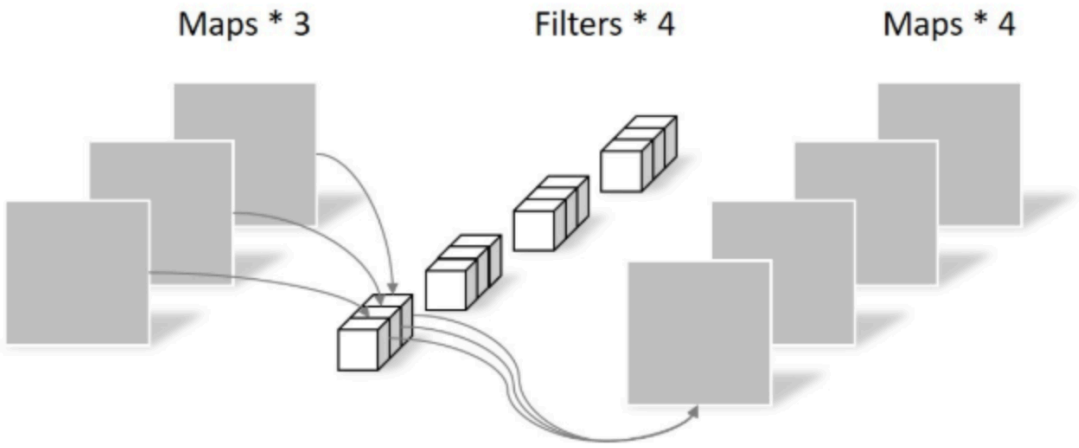


图 4-7: layer_0 模块结构图

中，该子模块结果由两个并行的卷积结果相连产生。最终子模块的结果与 cell_0 的结果横向拼接，完成了整个 layer_0 模块的计算。其中的卷积操作与批归一化操作的优化均可复用 stem_conv 当中的优化。除此之外，还需实现矩阵加法与乘法、relu 激活函数以及深度可分离卷积操作等算子。矩阵的加法乘法与 relu 激活函数结构比较类似，都是访存密集型算子，影响其计算速度在于访问效率而非计算速度。因而对于这些算子的重写，除了要按照计算机底层存储顺序对数据进行读取之外，还结合了 omp 指令进行加速。这些操作较好地保证了底层硬件资源的利用率，大大地提升了计算效率。



(a) 逐通道卷积



(b) 逐点卷积

图 4-8: 深度可分离卷积的两种卷积操作[6-7]

4.3.4 深度可分离卷积的优化

深度可分离卷积[6-7]相比于正常卷积会多一次计算流程，它分为逐通道卷积以及逐点卷积，逐通道卷积是指一个卷积核只用来负责一个通道，摒弃了原先一个卷积核需要对应所有输入图像的通道，因而逐通道卷积下的卷积核仅有一个通道，如图4-8(a) 所示。同时，卷积核的数量与输入图像的通道数一致，所以一个五通道的输入图像经过逐通道卷积之后得到的输出图像通道依然是五。逐通道卷积的这种设置导致了逐通道卷积下的卷积核没法进行灵活扩展，同时缺乏通道间的信息传递，它使得各个通道间的信息闭塞了起来。因而便需要深度可分离卷积下的第二个卷积方式，即逐点卷积。图4-8(b) 展示了逐点卷积的过程，该过程与逐通道卷积正好相反，逐点卷积采取了每个卷积用来负责所有输

入图像通道的传统卷积方式，同时逐点卷积下的卷积核尺寸都为 1x1，卷积核的个数最终决定了该深度可分离卷积过程下输出图像的通道数。所以，相比于逐通道卷积在图像层面进行卷积计算，逐点卷积采取的是在所有通道层面进行卷积计算。需要说明的是，逐通道卷积以及逐点卷积由于它们自身设计的优势，两种卷积方式带来的计算量都很小，而不同维度下的卷积又保证了卷积过程中信息的传递与学习都能较好的进行。因而，深度可分离卷积通过卷积核精巧的重新设计，有效地带来了网络模型速度性能上的提升。所以针对深度可分离卷积的重写实现主要分为两个部分，对于逐点卷积操作的实现，只需要对于单个逐点卷积核将其展平成一个竖列，多个逐点卷积核再横向拼接。即对 n 个 1x1xm 的卷积核进行拼接，最终得到的矩阵形状为 mxn。将该结果矩阵与逐通道卷积之后得到的特征图进行矩阵乘法操作，最终即可完成逐点卷积。因而深度可分离卷积的重写关键就在于逐通道卷积的实现，它也相应具有一定的难度。逐通道卷积的实现主要分为逐通道卷积下 im2col 的实现以及逐通道卷积的矩阵乘法操作。

与正常卷积下的 im2col 实现算法不同，本文为逐通道卷积设计实现了一套更加适合的 im2col 算法。本文实现的正常卷积下的 im2col 算法的输出矩阵以卷积特征图的面积大小作为矩阵的长，以卷积核长宽高之积作为矩阵的宽。与以卷积核长宽高之积作为长、卷积核个数作为宽的卷积核参数矩阵进行相乘之后，便可得到结果特征图矩阵，它以特征图的面积作为长，特征通道作为宽，刚好符合本文设计的 matrix 形状定义。考虑到逐通道卷积的特性，本文设计的 im2col 算法的输出结果矩阵以特征图的通道数为长，图像面积与卷积核长宽积的乘积作为宽。卷积核参数矩阵则以卷积核数量也即是特征图通道数作为长，卷积核长宽之积作为宽。逐通道卷积的过程如图4-9所示，图中表示的是一个 3x3 大小的特征图，共有两个特征通道。相应的与之进行乘积操作的卷积核通道也为 2，卷积核大小为 3x3。逐通道的卷积首先需要对特征图进行补零的 pad 操作，之后便是对该结果进行 im2col 算法，该算法得到的结果矩阵每一行代表了一个特征图通道转换之后的结果，最终还需要与卷积核进行矩阵乘法操作。具体来讲，该过程会对结果矩阵的长进行枚举，结果矩阵按照卷积核长宽之积作为步长与卷积核参数矩阵相应的行进行两两相乘，每一行的乘积结果都是特征通道为 1 的特征图，最终将所有行乘积得到的特征图进行纵向拼接，并对该矩阵进行转置

就完成了逐通道卷积的过程。

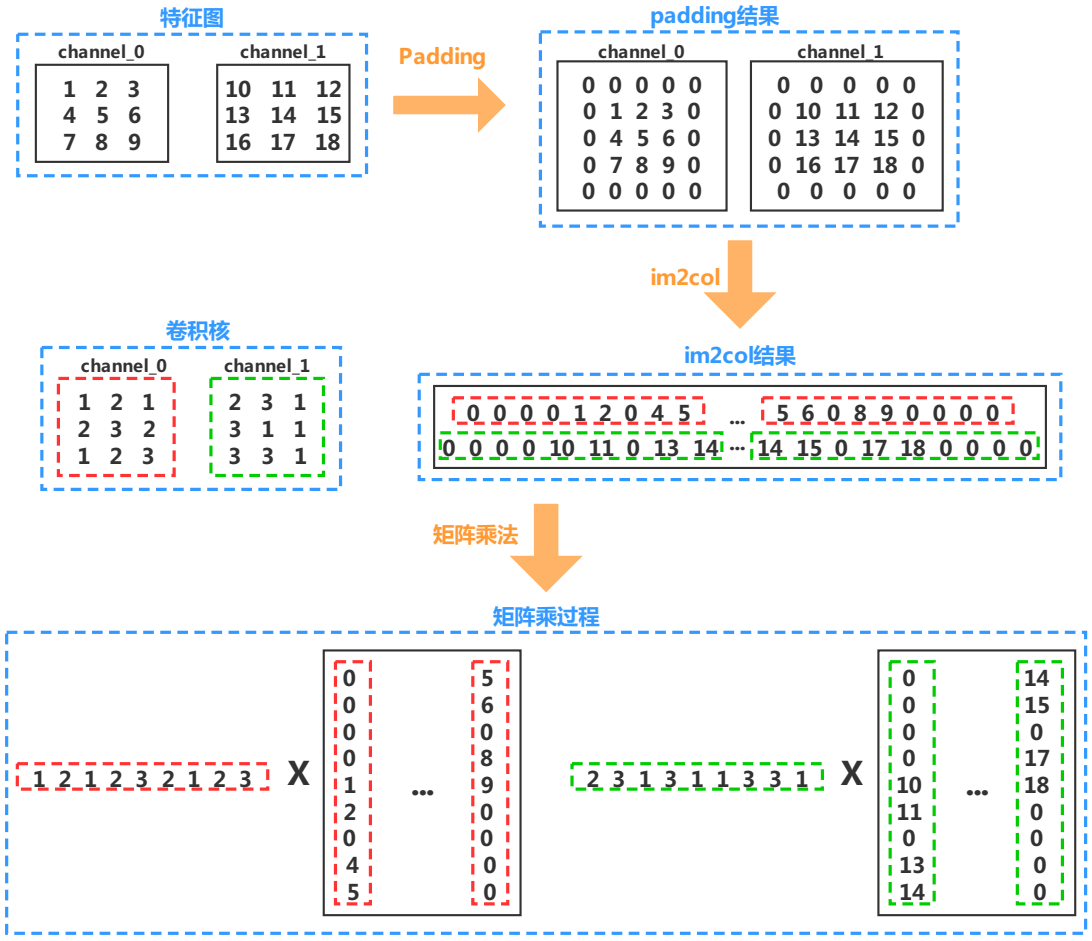


图 4-9: 逐通道卷积过程

4.4 实验加速效果

在完成 layer_0 模块所有算子的重写之后，我们对该模块下各个自模块的运行耗时进行了实验分析，分析结果如表4-3所示。表中第一部分展示了该模块各个部分在 Tensorflow 框架下的运行速度，可以看到这些子模块各自耗时相对均等，所以总体的优化策略并不会特别侧重哪个子模块。表中第二个部分则是 layer_0 模块的一个初步优化结果，可以从表格中分析发现，除了 cellx 与 celly 两个子模块，其他模块的优化效果都相对较为显著。而 cellx 与 celly 两个子模块优化之后的运行速度相比于原框架慢了 3 倍，经过细致分析，这两个模块均包含了深度可分离卷积操作，而其中重写的 im2col 也正占了绝大多数的计算时间，因而该算法还有较大的提升空间。

表 4-3: layer_0 模块优化分析

Tensorflow 框架下运行耗时						
pool_x	pool_y	layer_base	cellx	celly	final_combine	total
0.514ms	0.515ms	0.454ms	0.431ms	0.427ms	0.206	2.94ms
layer_0 模块初步优化耗时						
pool_x	pool_y	layer_base	cellx	celly	final_combine	total
0.435ms	0.294ms	0.329ms	1.328ms	1.284ms	0.131ms	3.801ms
layer_0 模块最终优化耗时						
pool_x	pool_y	layer_base	cellx	celly	final_combine	total
0.391ms	0.295ms	0.290ms	0.394ms	0.332ms	0.128ms	1.829ms

我们对于 im2col 的改进起源于两个地方的观察，一个是由于循环结构设计的不合理，在结果矩阵以及输入特征图矩阵的内存访问上，存在着访问了某些不连续的内存问题；二是在于对于结果矩阵的赋值操作，还存在着更快的优化方式，通过将原本基于单点内存的访问赋值转化成同时对整块连续内存进行访问赋值可以更快提升计算速度。第一点的改进主要通过调整循环结构，尽可能同时处理连续内存上的数据，第二点的改进则是利用 c++ 当中的 memcpy 函数，将单点赋值操作转换成连续内存赋值操作，这样可以显著降低 im2col 计算复杂度一个数量级。表中第三部分的数据验证了我们的猜想，我们对于 im2col 的进一步优化使得 cellx 和 celly 的耗时相比于初步优化时优化了 3 倍多，同时也比 Tensorflow 框架运行下的速度更高，说明我们对于算法的迭代优化卓有成效。需要说明的是，在 stem_conv 模块中的 im2col 之所以没有出现这样耗时极高的问题，是因为 stem_conv 模块中的 im2col 处理的数据维度远远小于本模块处理的数据维度，因而对于单点内存操作优化的不彻底并没有给 stem_conv 模块带来较大的影响。

结合前文对 stem_conv 模块的最终优化，我们将两个模块重写的算子联合到一起对模型进行算子替换以及优化加速，整个模型在我们替换算子的方式下的运行速度为 2.35ms，相比于 Tensorflow 运行速度的 3.83ms，我们的方法有效提升了 1.63 倍的速度。同时，我们对于算子的重写可以保证相比于 Tensorflow 执行的结果精度误差在 0.1% 以内，较好地保证了算子重写过程的精确度，而要进一步缩小误差则需要以部分优化速度为代价，这则是一个精度与速度的权衡了。

4.5 本章总结

本章基于 Tensorflow 算子重写的方式，首先对模型结构进行分析，获取其中耗时较多的热点算子，然后针对性地设计了优化方案，结合内存布局转换、算子优化等手段重写模型当中的所有算子，极大地降低了它们的运行延时。具体来讲，本文自行实现了 im2col 算法、卷积等算子，结合 omp、mkl 等手段对 relu 激活函数、批归一化算子进行并行化加速。同时，通过深入分析逐通道卷积下 im2col 算法延时较高的问题，创新性地提出了一套行之有效的解决方案，成功地解决了该缺陷。最终，我们对于整个模型的优化相比于 Tensorflow 框架下的运行速度提升了 1.63 倍的性能，而且所有的算子重写保证了最终优化后的模型输出结果与原结果的精度误差在 1e-3 到 1e-4 的范围内。这些都有力地证明了我们模型推理优化工作的有效性。

第五章　网络加速在鸟类识别系统中的应用

本章节结合前两个章节分别从算法端以及工程端两个角度对深度学习模型进行优化加速的内容，对鸟类识别系统进行了改进。利用我们算法端产生的模型完成识别模型的替换，同时基于我们工程端优化的经验方法对该模型完成部署加速。最终系统的优化效果强有力地证明了我们算法的有效性，同时也佐证了我们方法较大的实用价值。

5.1　鸟类识别背景

在自然生态保护区中，对于鸟类的观察以及数量统计一直是一项重要课题。通过对于鸟群数量以及鸟类的出现、停留时间的统计，可以有效掌握鸟类们的健康状态，从而获取保护区生态平衡等重要信息，方便相关人员开展环境保护工作。但是对于鸟类信息的获取统计大多采取传统方法，通过摄像头对鸟群进行长时间拍摄得到视频数据，然后在较强的专家经验帮助下对于这些视频进行信息筛选，这样通常耗时耗力，给研究人员带来了较大的工作负担。而随着深度学习的广泛应用，专家经验可以较好地迁移到神经网络当中，以往繁琐的工作就可以解放出来交由计算机完成。尽管鸟类识别并不是一个实时性要求很高的任务，成百上千的视频量还是为模型以及底层计算资源带来了较大的压力，尤其是当模型需要部署在边缘设备上进行运算时，这样的问题显得愈加尖锐。如何尽可能提高网络运行速度，大幅降低网络对于计算资源的需求，这也对于鸟类识别乃至自然保护区的维护具有重要帮助。而本文提出的网络加速的两个算法可以较好解决该问题。

需要说明的是，鸟类识别系统一般包括了目标检测以及图像识别两个任务，而本章节的优化主要聚焦于后者，也即是对于目标检测获取到的鸟类图像进行识别。通过对于检测得到待分类的鸟类图像进行快速识别，我们提出的方法较好地为鸟类识别系统缓解了压力。

5.2 鸟类识别系统

鸟类识别首先需要对摄像头采集得到的图片进行目标检测，获取到鸟类图像，然后将其送入分类网络进行进一步的鸟类判断，从而可以统计得到鸟类的作息数据。因而鸟类识别项目主要包含了检测以及识别两个重要的模型，这两个模型可以部署在云端也可以部署在边缘设备上。但由于在云端部署需要将鸟类视频传输回后方服务器进行处理，大量视频将为传输以及存储带来巨大的压力。如果模型部署在边缘侧，就可以将摄像头当下采集到的视频立即送入神经网络处理，只需要将具体鸟类出现的时间地点传输回服务器便可完成鸟类作息监控，这样的方案无疑更加经济也更有效率，但也为模型的部署带来了较大的挑战。本章便基于前文提出的两个加速算法从算法和工程的角度，针对鸟的识别分类模型较好地完成了对其在边缘计算设备上的部署。

5.2.1 系统需求

对于在野外环境下运行的鸟类识别系统，我们将其核心需求归纳总结成三个方面：

1. 较强的边缘计算能力：考虑到野外环境的恶劣以及成本压缩的需求，通常这类环境下搭建的计算平台都是基于CPU等相对较为廉价的硬件。而对于鸟类的生活作息追踪需要拍摄大量的视频，这些视频如果全部传送回服务器进行处理，不仅耗费大量的传输资源，也难以较为实时地对鸟类进行追踪。所以本系统下的网络模型应当有较高的边缘计算能力。
2. 较高的鸟类识别精度：系统最终的目标是要可以准确追踪到鸟类信息，对于鸟的分类便显得尤为重要，否则采取深度学习方法替换人力便显得毫无意义。相较之下，检测得到的鸟类框精准度并不太关键，只要可以保证不影响后续的鸟类识别即可。
3. 较为高效的模型迭代能力：在自然保护区搭建计算平台的工作是比较繁琐的，而如果模型的每次迭代都会给平台带来比较大的改动的话，无疑会使得整个系统的运行效率大大降低。所以一个良好的系统一经搭建，应该能够保持较长时间的环境稳定，只做软件上的远程迭代更新，这样

才能保证系统的高效运转。这便为系统的模型迭代提出了较高的要求。

经过上述三点需求的判断，为了能够提供较高的边缘计算能力，复杂且网络层数太深的模型应当不予考虑，同时为了尽可能减少模型的迭代工作，也不应采用过多的深度学习模型。所以鸟类识别的目标检测部分一般都采用基于传统的边缘检测等的方法，其引入的较多脏数据可以通过后续的分类算法进行筛除。而传统算法所固有的无法得到较为精准目标框的缺点也可以通过对于识别模型的针对性训练来获得弥补。所以整个系统的关键便在于识别模型，计算的压力也主要来源于识别模型。考虑到保护区下的鸟的多样性较低，相应的深度学习网络也便不需要太过于复杂，由此推断该场景下模型精度对于网络层数的剪裁相对不太敏感。所以在精度与速度的抉择上，该系统应当更加注重速度的提升，可以一定程度上加大对于网络模型的剪裁，换取更高的运行速度。

基于上述的分析判断，本系统聚焦于分类算法，采用了深度学习的方式来对鸟类图像进行分类。具体来讲，该算法采取了本文提出的假剪枝算法来生成性能更加优异的压缩网络，而后进一步通过算子重写的方式来在模型部署的时候提升资源利用率，最终提升运算速度。

5.2.2 系统架构

整个系统的的运行分为了三个部分，首先通过检测算法获取鸟类图像信息，继而通过分类算法得到该鸟的种类，最后是对于鸟类信息的整合。本系统更加关注第二部分，也即是唯一涉及深度学习算法的一部分，整个系统的执行流程如图 5-1所示。首先系统摄像头会实时运作，拍摄下监控场景的视频。然后我们的系统服务会利用 ffmpeg 对视频进行截帧操作，同时将截取下来的每一帧都调用检测算法来检测图中是否有鸟类图像。如果不存在鸟类图像，那么该图像会被直接丢弃，否则就会将检测到的鸟类图像部分传入到后续的识别模块中。识别模块首先会对该图像做灰化、归一化等图像预处理操作，然后调用分类模型对处理后的图像进行识别。这里分类模型即采用我们算法搜索得到的精简模型，同时在部署的时候采用了本文提出的推理优化方法对模型的运行进行进一步加速。完成识别任务之后，对于未成功识别到鸟类的图像，系统也会进行丢弃。而对于成功识别的图像，系统会将图像以及识别信息一并传输回服务器，并调用

前端界面进行相关信息的展示，最终辅助研究人员进行信息的整理和统计，以便更好地掌握鸟类信息，维护生态平衡。

在整个鸟类识别系统当中，图像预处理部分以及鸟类类别判断的网络部分代码均由 python 编写，分类网络主要利用本文基于 ENAS 提出的假剪枝算法，搜索出较为精简的网络模型，而后针对我们自然场景下的鸟类图像数据对该模型进行迁移训练，较好地保证了该分类模型的识别性能。在完成模型训练之后便是需要对模型进行线上部署，这里采用了 C++ 语言对整个模型当中的所有算子进行高性能重写，经过 g++ 对重写代码进行编译生成动态链接库，供 Tensorflow 调用以替换算子完成部署加速。在最终传输信息给服务器的环节上，系统基于了一种高性能、开源同时也更加通用的 RPC 框架——gRPC 框架，通过 protobuf 序列化协议来定义函数接口，它能够保证数据传输的过程更加高效也更加灵活。边缘计算平台每当检测并识别到了鸟类图像，便会采取远程调用的方式在服务器端 (也即用户端) 调用起接收以及展示程序，将识别到的鸟类图像传送回服务器端，并以前端页面展示的形式将其显示出来。这里前端主要基于 PyQt5 框架完成搭建，以提供良好的可视化界面。

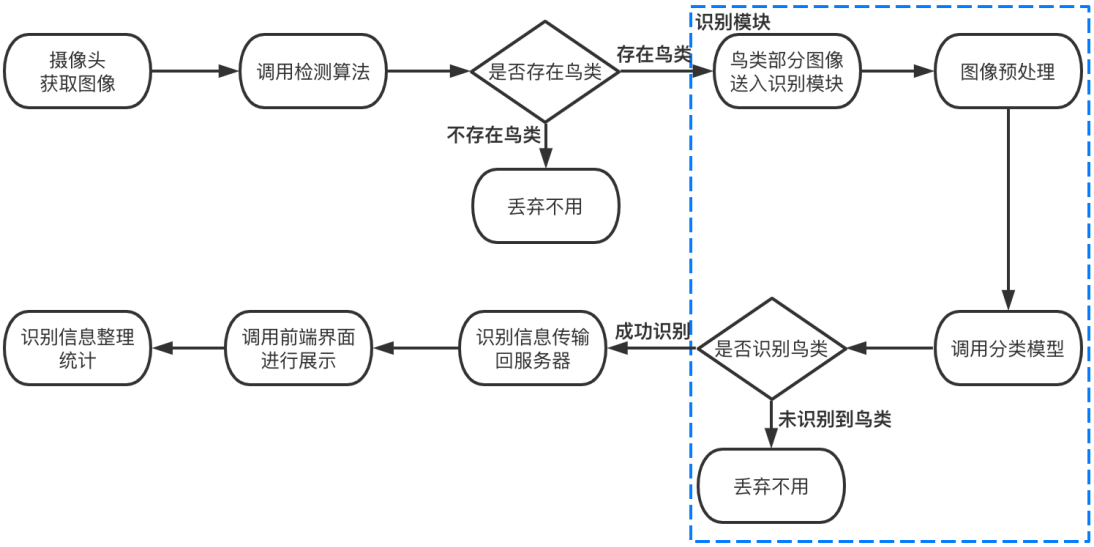


图 5-1: 鸟类识别系统流程图

5.2.3 系统效果

本系统专注于解决鸟类识别模型的优化提速，同时为用户提供了一个可视化界面，方便研究人员更好地掌握鸟类信息。本系统经过多次实测，模型的性能

与实时性方面均能较好地达到边缘计算的要求，系统的应用价值较高，极大地提升了工作效率。

5.2.3.1 前端展示

本系统在执行识别任务的同时会将识别到的鸟类信息存储在本地，系统设定当累计识别达到了 20 张鸟类的时候，会开启远程调用，同时将所有鸟类图像以及它们的类别、出现时间等信息传送回服务端，并在服务端开启一个如图5-2所示的前端界面。该界面左上角展示了识别对应的鸟类图像，其下方对应着该鸟的类别以及出现时间信息。右侧还提供了查看下一张图像，将鸟类信息以表格的形式导出，以及关闭系统等的功能。

图 5-2: 鸟类识别系统前端展示

表 5-1: 本文提出的加速方法速度实验

模型方案	1000 次前向传播平均耗时
ENAS 搜索得到的模型 (baseline)	7.566ms
假剪枝方法 (我们提出的算法端优化)	3.808ms
假剪枝方法 + 模型推理优化 (算法与工程的结合优化)	2.320ms

5.2.3.2 加速效果

本文采取了 CUB200-2011 数据集进行实验验证。CUB200-2011 数据集是加州理工学院在 2010 年提出的关于鸟类的一个细粒度数据集，也是目前关于鸟类细粒度分类识别研究的基准图像数据集。它由 11788 张鸟类图像组成，同时还有 200 类鸟类子类，每张图像均提供了相应的类别标签，可以用来进行识别任务的研究，也可以为我们的实验结果提供强有力的支持。对于模型的测速，本文基于了 Intel(R) Core(TM) i5-7500 单核进行实验，模型的输入采用了一张 32*32 的彩色图像，以模型前向传播 1000 次的平均时间作为模型的运算耗时。同时为了保证测速过程的公平性，我们设置了 Tensorflow 算子间以及算子内线程并发数为 1，omp 并行线程数也为 1。

我们将本文提出的基于算法与工程两个角度的加速方法运用在了本系统当中，系统的识别模型采用了我们算法端搜索得到的优异模型，进而结合我们工程端提出的加速方法，在该模型部署在边缘计算设备上时，对其进行所有模块的高性能重写，完成模型在前向传播过程中的加速。表5-1展示了我们提出的每一个方法经过实际验证均为模型的实际部署带来了不俗的影响。在实验过程中，我们压缩算法生成的模型相比原搜索框架在同样参数下搜索得到的模型，计算量以及参数量显著减少了近 3 倍，但在该鸟类数据集上的实验表明，这样的缩减并未严重损伤网络精度。而对模型较大程度的缩减带来的是最终模型的运行速度上升了两倍，ENAS 搜索得到的网络模型平均耗时为 7.566ms，我们方法搜索得到的模型平均耗时为 3.808ms，在我们的测速平台上有效提升了近 2 倍的速度。经过我们提出的推理优化方案部署模型之后，我们的模型线上运行平均耗时为 2.320ms，相比于最开始模型的 7.566ms 运算耗时，我们提出的优化方案最终帮助模型提升了 3.26 倍的计算速度。

5.3 本章总结

本章主要介绍了基于本文提出的模型加速方法在一个实际系统当中的应用—基于卷积网络的鸟类识别系统，通过深度学习来帮助科研人员进行更加高效的鸟类信息捕捉，从而完成生态环境的维护。基于本文提出的方法，识别模型才能满足边缘计算当中的实时性要求，也能进一步降低模型部署成本。这些都强有力地证明了本文方法的有效性，同时也证实了模型加速有着巨大的应用以及研究价值。

第六章　总结与展望

自从 2012 年 Hinton 课题组将卷积神经网络首次应用在 Imagenet 图像识别大赛上，并一举夺得冠军之后，卷积神经网络开始越来越多地吸引了研究者的关注。在此后的研究当中，他们发现网络层数越多、模型越复杂越容易帮助模型带来更高的精度，因而现今深度学习模型的发展越来越趋向于深度化。但模型更加精准的同时，也带来了模型训练更加耗时、部署更加依赖大量计算硬件的问题。这无疑是为模型的研究带来了较高的成本，也为深度学习的应用带来了极大的阻碍。

为了解决这样的问题，针对模型进行的加速方法应运而生，模型蒸馏、模型剪枝各式各样的方法均已成功应用在模型压缩的领域当中，能够为模型带来不菲的优化收益。但传统的压缩方法过于依赖人工经验，且对于模型的精简并没有优化到极致。所以本文提出了基于神经架构搜索的网络剪裁方法，在传统 NAS 的基础上添加了对每一层网络压缩比例的搜索，同时创新性地提出了假剪枝的方法，摒弃了传统剪枝与训练交替进行的压缩流程，转而将这两个部分解耦合，以最大化压缩前后的精度差 (压缩后的模型精度减压缩前的模型精度) 作为优化目标，旨在搜索到对模型精度损失更小的压缩方法。通过在相关数据集上的实验分析，以及对于我们引入的精度差的深入讨论，强有力地证明了我们提出的算法的有效性。

本文也结合了工程端的推理优化方法，针对算法搜索生成的网络模型，我们采用了 Tensorflow 自定义算子的方式，对该模型内的所有算子模块进行高性能重写。具体来说，我们自行实现了 im2col、卷积、relu 激活函数、批归一化等算子，针对逐通道卷积下的 im2col 算法计算延时较高的问题，深入分析并提出了一套有效的解决方案，成功降低了它的延时。最终将这些重写算子编译生成出相应的动态链接库，在加载模型的时候完成算子替换，达到了模型计算加速的效果，多次实验详实地验证了我们方法的有效性，它为模型带来了较大的速度提升。同时我们也成功地将我们提出的方法应用在实际系统当中—鸟类识别系统，在保证模型识别精度的前提下，本文针对模型加速的两个算法均有效提

升了模型的计算速度，满足了实际生产环境下的需求，这也佐证了我们算法的实用性以及较大的应用价值。

按照本文的工作路线，可以继续开展相关研究工作。首先，本文提出的压缩方法不仅适用于图像识别，也同时可以应用在目标检测、实例分割等视觉任务当中，进一步的推广可以为这些领域带来更好的帮助。其次，本文的压缩仅采用了网络剪枝的方法，在未来也可以进一步结合网络蒸馏、模型量化等方法更好地对模型进行优化。最后，模型部署目前比较依赖于算子重写，这也是比较耗费人力的工作，如果能将其改进成为一种自动化的方式，便可以极大地提高模型部署效率，降低部署成本，为模型加速带来更好的收益。

参考文献

[1] ZHANG X, ZHOU X, LIN M, et al. Shufflenet: An extremely efficient convolutional neural network for mobile devices[J/OL]. CoRR, 2017, abs/1707.01083. http://arxiv.org/abs/1707.01083.

[2] MA N, ZHANG X, ZHENG H T, et al. Shufflenet v2: Practical guidelines for efficient cnn architecture design[C]//The European Conference on Computer Vision (ECCV). [S.l.: s.n.], 2018.

[3] LIU Z, LI J, SHEN Z, et al. Learning efficient convolutional networks through network slimming[J/OL]. CoRR, 2017, abs/1708.06519. http://arxiv.org/abs/1708.06519.

[4] PHAM H, GUAN M Y, ZOPH B, et al. Efficient neural architecture search via parameter sharing[J/OL]. CoRR, 2018, abs/1802.03268. http://arxiv.org/abs/1802.03268.

[5] HE Y, LIN J, LIU Z, et al. Amc: Automl for model compression and acceleration on mobile devices[C]//The European Conference on Computer Vision (ECCV). [S.l.: s.n.], 2018.

[6] HOWARD A G, ZHU M, CHEN B, et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications[J/OL]. CoRR, 2017, abs/1704.04861. http://arxiv.org/abs/1704.04861.

[7] SANDLER M, HOWARD A G, ZHU M, et al. Mobilenetv2: Inverted residuals and linear bottlenecks[C/OL]//2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018. IEEE Computer Society, 2018: 4510-4520. http://openaccess.thecvf.com/content$_{c}$v pr$_{2}$018/html/Sandler$_{M}$obileNetV2$_{I}$nverted$_{R}$esiduals$_{C}$VPR$_{2}$018$_{p}$aper.html. DOI: 10.1109/CVPR.2018.00474.

[8] RUMELHART D E, HINTON G E, WILLIAMS R J. Learning Representations by Back-propagating Errors[J/OL]. Nature, 1986, 323(6088):533-536. http://www.nature.com/articles/323533a0. DOI: 10.1038/323533a0.

[9] DENG J, DONG W, SOCHER R, et al. Imagenet: A large-scale hierarchical image database[C]//Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on. [S.l.]: Ieee, 2009: 248-255.

[10] SRIVASTAVA N, HINTON G E, KRIZHEVSKY A, et al. Dropout: a simple way to prevent neural networks from overfitting[J/OL]. J. Mach. Learn. Res., 2014, 15 (1):1929-1958. http://dl.acm.org/citation.cfm?id=2670313.

[11] KRIZHEVSKY A, SUTSKEVER I, HINTON G E. Imagenet classification with deep convolutional neural networks[J/OL]. Commun. ACM, 2017, 60(6):84-90. http://doi.acm.org/10.1145/3065386.

[12] BA L J, CARUANA R. Do deep nets really need to be deep?[J/OL]. CoRR, 2013, abs/1312.6184. http://arxiv.org/abs/1312.6184.

[13] SIMONYAN K, ZISSERMAN A. Very deep convolutional networks for large-scale image recognition[C/OL]//BENGIO Y, LECUN Y. 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings. 2015. http://arxiv.org/abs/1409.1556.

[14] DENIL M, SHAKIBI B, DINH L, et al. Predicting parameters in deep learning [C/OL]//BURGES C J C, BOTTOU L, GHAHRAMANI Z, et al. Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States. 2013: 2148-2156. https://proceedings.neurips.cc/paper/2013/hash/7fec306d1e665bc9c748b5d2b99a6e97-Abstract.html.

[15] ZHOU B, KHOSLA A, LAPEDRIZA À, et al. Learning deep features for discriminative localization[J/OL]. CoRR, 2015, abs/1512.04150. http://arxiv.org/abs/1512.04150.

[16] ANWAR S, HWANG K, SUNG W. Structured pruning of deep convolutional neural networks[J/OL]. CoRR, 2015, abs/1512.08571. http://arxiv.org/abs/1512.08571.

[17] JADERBERG M, VEDALDI A, ZISSERMAN A. Speeding up convolutional neural networks with low rank expansions[J/OL]. CoRR, 2014, abs/1405.3866. http://arxiv.org/abs/1405.3866.

[18] HINTON G E, VINYALS O, DEAN J. Distilling the knowledge in a neural network[J/OL]. CoRR, 2015, abs/1503.02531. http://arxiv.org/abs/1503.02531.

[19] DING X, DING G, GUO Y, et al. Approximated oracle filter pruning for destructive CNN width optimization[J/OL]. CoRR, 2019, abs/1905.04748. http://arxiv.org/abs/1905.04748.

[20] ABADI M, BARHAM P, CHEN J, et al. Tensorflow: A system for large-scale machine learning[C/OL]//KEETON K, ROSCOE T. 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016. USENIX Association, 2016: 265-283. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi.

[21] PASZKE A, GROSS S, MASSA F, et al. Pytorch: An imperative style, high-performance deep learning library[C/OL]//WALLACH H M, LAROCHELLE H, BEYGELZIMER A, et al. Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada. 2019: 8024-8035. https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.

[22] CHEN T, LI M, LI Y, et al. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems[J/OL]. CoRR, 2015, abs/1512.01274. http://arxiv.org/abs/1512.01274.

[23] ZOPH B, LE Q V. Neural architecture search with reinforcement learning[J/OL]. CoRR, 2016, abs/1611.01578. http://arxiv.org/abs/1611.01578.

[24] RUMELHART D E, HINTON G E, WILLIAMS R J. Learning representations by back-propagating errors[J/OL]. Nature, 1986, 323. https://doi.org/10.1038/323533a0.

[25] GIRSHICK R B, DONAHUE J, DARRELL T, et al. Rich feature hierarchies for accurate object detection and semantic segmentation[C/OL]//2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014. IEEE Computer Society, 2014: 580-587. https://doi.org/10.1109/CVPR.2014.81.

[26] LONG J, SHELHAMER E, DARRELL T. Fully convolutional networks for semantic segmentation[C/OL]//IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015. IEEE Computer Society, 2015: 3431-3440. https://doi.org/10.1109/CVPR.2015.7298965.

[27] PFISTER T, CHARLES J, ZISSERMAN A. Flowing convnets for human pose estimation in videos[C/OL]//2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015. IEEE Computer Society, 2015: 1913-1921. https://doi.org/10.1109/ICCV.2015.222.

[28] 机器学习[M/OL]. 清华大学出版社, 2016. https://books.google.com/books?id=j0G8nQAACAAJ.

[29] LIU Z, SUN M, ZHOU T, et al. Rethinking the value of network pruning[J/OL]. CoRR, 2018, abs/1810.05270. http://arxiv.org/abs/1810.05270.

[30] LI H, KADAV A, DURDANOVIC I, et al. Pruning filters for efficient convnets [J/OL]. CoRR, 2016, abs/1608.08710. http://arxiv.org/abs/1608.08710.

[31] HAN S, POOL J, TRAN J, et al. Learning both weights and connections for efficient neural network[C/OL]//CORTES C, LAWRENCE N D, LEE D D, et al. Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada. 2015: 1135-1143. https://proceedings.neurips.cc/paper/2015/hash/ae0eb3eed39d2bcef4622b2499a05fe6-Abstract.html.

[32] HAN S, MAO H, DALLY W J. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding[C/OL]//BENGIO Y, LECUN Y. 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings. 2016. http://arxiv.org/abs/1510.00149.

[33] SILVER D, LEVER G, HEESS N, et al. Deterministic policy gradient algorithms [C/OL]//JMLR Workshop and Conference Proceedings: volume 32 Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014. JMLR.org, 2014: 387-395. http://proceedings.mlr.press/v32/silver14.html.

[34] HUANG Z, WANG N. Data-driven sparse structure selection for deep neural networks[J/OL]. CoRR, 2017, abs/1707.01213. http://arxiv.org/abs/1707.01213.

[35] DEVRIES T, TAYLOR G W. Improved regularization of convolutional neural networks with cutout[J/OL]. CoRR, 2017, abs/1708.04552. http://arxiv.org/abs/1708.04552.

[36] HE Y, KANG G, DONG X, et al. Soft filter pruning for accelerating deep convolutional neural networks[J/OL]. CoRR, 2018, abs/1808.06866. http://arxiv.org/abs/1808.06866.

[37] HE Y, LIU P, WANG Z, et al. Pruning filter via geometric median for deep convolutional neural networks acceleration[J/OL]. CoRR, 2018, abs/1811.00250. http://arxiv.org/abs/1811.00250.

[38] LIN S, JI R, YAN C, et al. Towards optimal structured CNN pruning via generative adversarial learning[J/OL]. CoRR, 2019, abs/1903.09291. http://arxiv.org/abs/1903.09291.

[39] LOSHCHILOV I, HUTTER F. SGDR: stochastic gradient descent with warm restarts[C/OL]//5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. OpenReview.net, 2017. https://openreview.net/forum?id=Skq89Scxx.

[40] OLSTON C, FIEDEL N, GOROVOY K, et al. Tensorflow-serving: Flexible, high-performance ML serving[J/OL]. CoRR, 2017, abs/1712.06139. http://arxiv.org/abs/1712.06139.

简历与科研成果

基本信息

赵加成，男，汉族，1996 年 3 月出生，江苏省沭阳人。

教育背景

2018 年 9 月 — 2021 年 6 月 南京大学计算机科学与技术系 硕士

2014 年 9 月 — 2018 年 6 月 吉林大学软件学院 本科

攻读硕士学位期间完成的学术成果

1. **Jiacheng Zhao**, Hongyan Hao, Baile Xu, Jian Zhao, Furao Shen, "Fake Pruning with AutoML for Model Compression", under review.

攻读硕士学位期间的专利成果

1. 申富饶，**赵加成**，于惠. "一种基于自编码器和 tSNE 降维的图片检索方法" (201910932207)

攻读硕士学位期间参与的科研课题

1. 国家自然科学基金"基于深度感知增量式联想记忆神经网络的信息融合系统研究"（课题年限 2019.01 ~ 2022.12），负责神经网络模型加速相关研究。

攻读硕士学位期间的比赛奖项

1. 韩峰，李雪健，**赵加成**，姜少魁. 2019 首届 IKCEST "一带一路"国际大数据竞赛，国际二等奖

致　　谢

光阴似箭，岁月如梭。转眼之间研究生三年的学习生涯已接近尾声，在这三年的科研时光中，我学会了很多，也成长了很多。结识了一群可爱的同学们，也感受到了老师学长们严谨的科研态度。在这之中，我需要向他们致以深深的谢意，是他们时刻伴随在我的身边，指引着我成长成材。

首先需要感谢的就是申富饶教授，申老师在科研上对我们严谨要求，对我们的科研动态也十分关心。我的每一步科研在申老师的把控下一丝不苟地进行，同时申老师也强调独立思考的重要性，引导我们要从问题的本质出发，不能人云亦云。生活上，申老师也非常乐意同我们进行交流，悉心开导我们遇到的困难与不快，帮助我们更加轻松愉快地度过研究生涯的三年。

同时也要感谢电子学院的赵健老师，赵健老师会在我们的组会上不定期分享论文的写作方法，同时也乐于线下为我们的写作进行详细指导，为我们的科研工作提供了极大的帮助。

此外，我也要感谢 426 朝夕相处的小伙伴们，大家在组会上对自己科研工作的展示令我大开眼界，生活上我们也一起玩耍、一起吃饭，这些都无不组成了日后的宝贵回忆。

最后还要感谢我的父母，是他们不辞劳苦的养育才有了我的今天，他们是我成长路上的坚实后盾，更是我回报社会的初心起点。

《学位论文出版授权书》

本人完全同意《中国优秀博硕士学位论文全文数据库出版章程》（以下简称“章程”），愿意将本人的学位论文提交“中国学术期刊（光盘版）电子杂志社”在《中国博士学位论文全文数据库》、《中国优秀硕士学位论文全文数据库》中全文发表。《中国博士学位论文全文数据库》、《中国优秀硕士学位论文全文数据库》可以以电子、网络及其他数字媒体形式公开出版，并同意编入《中国知识资源总库》，在《中国博硕士学位论文评价数据库》中使用和在互联网上传播，同意按“章程”规定享受相关权益。

作者签名：_____

2021 年 5 月 29 日

论文题名	面向卷积神经网络的模型加速研究				
研究生学号	MG1833090	所在院系	计算机科学与技术	学位年度	2021
论文级别	✔学术学位硕士 ☐专业学位硕士 ☐学术学位博士 ☐专业学位博士 （请在方框内画钩）				
作者 Email	793532302@qq.com				
导师姓名	申富饶教授				

论文涉密情况：

✔ 不保密

☐ 保密，保密期（_____年_____月_____日 至 _____年_____月_____日）

注：请将该授权书填写后装订在学位论文最后一页（南大封面）。