上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

学士学位论文

THESIS OF BACHELOR

论文题目：高效神经网络模型研究

学生姓名：唐昊天

学生学号：516030910534

专　　业：计算机科学与技术 (IEEE)

指导教师：卢宏涛　教授

学院 (系)：计算机科学与工程系

上海交通大学

本科生毕业设计（论文）任务书

课题名称：高效神经网络模型研究

执行时间：2020 年 01 月 至 2020 年 06 月

教师姓名：卢宏涛　　职称：教授

学生姓名：唐昊天　　学号：516030910534

专业名称：计算机科学与技术(IEEE)

学院(系)：电子信息与电气工程学院

毕业设计（论文）基本内容和要求：

课题简介：

随着卷积神经网络为代表的人工智能技术发展，越来越复杂的卷积神经网络模型被用来处理各类计算机视觉问题。对卷积神经网络的要求也不仅仅是使用更深的神经网络达到更高的精度，而是希望现有的神经网络能够在手机、可穿戴设备等计算/存储资源有限的边缘设备上部署。这就需要深度神经网络的压缩与轻量化网络设计技术。

比如,尽管基于深度学习的语义分割在各数据集上取得了很高的精度，但由于目前的高精度语义分割方法在计算量、 模型大小（参数量）以及延迟方面都难以满足实时和资源受限条件的任务需求。

而现在的轻量级网络设计正在逐步从过去的单纯剪枝、量化现有模型转向基于神经网络架构搜索（Neural Architecture Search , NAS）的自动设计方法。但目前的 NAS 主要面向的是分类问题设计，而很少结合目标检测/语义分割等实际问题直接设计。而且，NAS 往往对计算资源有极高的要求，这无疑也给实际应用带来挑战。

本课题研究拟基于 NAS 技术搜索模型架构与量化方法等要素，改进现有算法的性能,以尽量低的网络搜索代价实现性能更高效的语义分割算法。

课题要求：

首先学习神经网络架构搜索的（Neural Architecture Search , NAS）

基本理论。深入研究近年来提出的 Genetic CNN、Single Path One-Shot Neural Architecture Search with Uniform Sampling、FairNAS 等面向轻量化神经网络的 NAS 论文。并学习 PyTorch 等深度学习平台和 TensorRT 等高效推理工具，实现至少两个目前流行的语义分割方法。并在通用 GPU 和计算资源受限环境运行、分析及比较目前流行算法在语义分割中的性能。

提出基于 NAS 的轻量化网络设计和高效语义分割的一种实现方案，进行实验验证并与现有方法的计算量、参数量、参数量化效果、内存占用、速度和精度进行比较。实现一个可以运行于各种平台的(CPU/GPU/mobile)语义高效分割的 Demo 程序。

课题工作量要求：

阅读相关的论文 15 篇以上。收集或自己实现至少 2 种分割方法，并进行性能比较。提出自己的基于轻量化网络设计和基于 NAS 的高效语义分割框架并进行编程实现，实现一个可运行的 Demo 程序。全时工作 4 个月。

毕业设计（论文）进度安排：

序号	毕业设计（论文）各阶段内容	时间安排	备 注
1	搜集、阅读文献	2020.01-2020.02	
2	确定算法，初步实现程序模块，进行实验、比较结果	2020.03-2020.04	
3	改进算法、实现 Demo 系统，撰写论文、答辩	2020.05-2020.06	

课题信息：

课题性质 ：设计☐　　论文 √

课题来源*：国家级 √　　省部级☐　　校级☐　　横向☐　　预研☐

项目编号 61702230

其他

指导教师签名：卢宏涛

2020 年 1 月 18 日

学院（系）意见：

审核通过

院长（系主任）签名：唐飞龙

2020 年 1 月 31 日

学生签名：唐昊天

2020 年 1 月 18 日

上海交通大学
毕业设计（论文）原创性声明

本人郑重声明：所呈交的毕业设计（论文），是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名： 

日期：2020 年 06 月 20 日

上海交通大学
毕业设计（论文）版权使用授权书

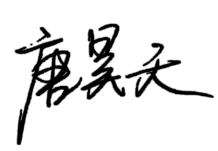
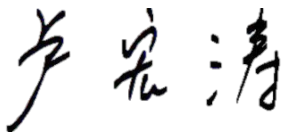
本毕业设计（论文）作者完全了解学校有关保留、使用毕业设计（论文）的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本毕业设计（论文）的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本毕业设计（论文）。

保　密 □，在 _____ 年解密后适用本授权书。

本学位论文属于

不保密 ☑。

（请在以上方框内打“✓”）

学位论文作者签名：  　指导教师签名： 

日期：2020 年 06 月 20 日　　日期：2020 年 06 月 20 日

高效神经网络模型研究

摘　要

随着卷积神经网络为代表的人工智能技术发展，越来越复杂的卷积神经网络模型被用来处理各类计算机视觉问题，例如图像分类，场景理解中很重要的语义分割任务，以及实例分割。尽管取得了相对优秀的精确度，卷积神经网络往往具有极大的计算量和参数量，这使得它们很难在移动端设备（例如手机、嵌入式设备或移动端 GPUs）上高效运行。于是，神经网络的加速，或者高效神经网络设计，成为了近年来的一个研究热点。

前人有关于高效神经网络设计的研究往往可以分为剪枝 (Pruning)、量化 (Quantization) 和神经网络结构搜索 (Neural Architecture Search) 三类。剪枝的目标是使用更少的参数来达到减小计算量的目的，而量化则通过减低每个参数所使用的位数来降低计算量和访存需求。神经网络结构搜索则从更复杂的优化空间去考虑神经网络速度优化，例如卷积核的尺寸、网络的深度，甚至网络的连接方式。而这些研究往往都集中在对图像分类模型的优化上，较少有工作专注于图像分割任务模型的优化。事实上，图像分类模型对计算资源的要求相对较低，容易在一般的 CPU 上达到实时，甚至可以在手机、单片机上运行。但图像分割模型，往往因为其输入数据分辨率很高，在桌面级 GPU（如英伟达 GTX1080Ti）上也很难达到实时处理，这使得加速图像分割模型的研究变得十分必要。

在本论文中，我们以图像语义分割模型为载体探讨高效神经网络模型设计问题。我们将利用一种创新的基于超网络权重共享、深度渐进收缩和硬件反馈直接约束的神经网络架构搜索实现多平台下的高效神经网络模型自动设计。我们将从硬件视角探讨不同平台下不同的神经网络设计结果，并从体系结构层面给出深入分析。我们的神经网络架构搜索算法可以在一次训练超网络的情况下方便地实现多个平台（论文中我们将使用三款英伟达公司生产的图形处理器，GTX 1060 / GTX 1080Ti / RTX 2080Ti / Titan RTX；以及一款 AMD 公司生产的桌面级中央处理器 Ryzen 7）下专用神经网络结构设计。我们同时确保网络架构搜索过程当中可以直接建模硬件延时，而无需参考浮点运算次数（FLOPs）或特征图尺寸（activation size）等不精确的代理指标。在所有平台上加速基线模型设计、提升模型性能，在 1080Ti GPU 上获得 1.1% 的 IoU 提升（与此同时 1.2 倍更快的推理速度），即便在 2016 年就已经十分流行的、常常装配在笔记本电脑上的 NVIDIA GTX 1060 GPU 上也可以达到 43 帧每秒的推理。这体现了我们提出的神经网络架构搜索算法的有效性。通过自动混合精度量化，我们的算法相比 8 位均匀量化减低 2.2× 计算量，维持精度不变。此外，我们也展示了类似于本文所提出的神经网络搜索技术在三维计算机视觉模型自动设计中的应用，亦取得了很好的效果：比此前最先进方法少 1.5× 计算的同时，达到 3.3% 的 IoU 提升。

关键词： 神经网络架构搜索 (NAS), 高效机器学习, 机器学习系统 (SysML), 图像分割, 深度学习

EFFICIENT NEURAL NETWORK ARCHITECTURE DESIGN

ABSTRACT

Deep neural networks have been widely applied in various visual recognition tasks such as image classification, semantic segmentation and instance segmentation thanks to the rapid development of machine learning and artificial intelligence. Achieving impressive accuracy on multiple challenging tasks, neural networks usually require huge amount of parameters and floating point operations (FLOPs), which makes it difficult to deploy them on mobile devices such as smartphones, embedded CPUs or mobile GPUs. Therefore, accelerating neural networks, or designing efficient neural network architectures becomes increasingly popular in recent years.

Previous research on efficient neural network design usually focus on pruning, which reduces the number of parameters so that computation can also be reduced; quantization, which reduces the number of bits required for each weight so that both computation and memory access are reduced; and finally neural architecture search (NAS), which explores a much larger design space for NN acceleration including kernel sizes, network depths and even network connections. However, previous work usually focuses on applying these techniques to design efficient image recognition models, leaving the optimization of image segmentation models understudied. Comparing with classifiers which require less computation, and runs fast on CPUs, mobile CPUs, or even microcontrollers; image segmentation models cannot run in realtime even on some high-end desktop GPUs, such as NVIDIA GTX 1080Ti. Such challenges make designing efficient deep neural nets for image segmentation important in practice.

In this work, we discuss the problem of efficient neural network architecture design in the specific domain of image semantic segmentation. We propose a novel neural architecture search (NAS) paradigm based on weight sharing, and use parameter quantization to achieve automatic neural network design on multiple platforms such as NVIDIA GPUs (1060，1080Ti, 2080Ti, Titan RTX), and CPUs (AMD Ryzen7). Our pipeline directly models hardware latency instead of referring to inaccurate proxys such as FLOPs / activation sizes. We give in-depth analysis on the behavior of NAS from the hardware / system perspective. On the Cityscapes dataset, we achieve uniform speedup on all five hardware platforms, with best results on GTX 1080Ti: 1.2× speedup with 1.1% improvement in IoU. Even on NVIDIA GTX 1060, which has been ubiquitous since 2016 and is equipped on many laptops, our automatically designed models can still achieve 43 FPS inference speed, which demonstrates the effectiveness of the proposed technique. With automatic mixed-precision quantization policy search, our quantized NAS networks reduce the BitOps of the 8-bit uniform quantization baseline by 2.2× without incurring loss of accuracy. We also present the transfer of the proposed NAS pipeline to 3D deep learning, which also achieves impressive results: reducing computation by 1.5× but achieving 3.3% IoU improvement comparing with previous state of the art.

Key words: Neural Architecture Search (NAS), Efficient Machine Learning, Systems for Machine Learning (SysML), Image Segmentation, Deep Learning

目 录

插图索引

表格索引

算法索引

第一章　简介

由于神经网络的过参数化（over-parameterization) 和巨大的计算量带来的在端设备上部署的巨大挑战，高效神经网络设计是近年来一个被广泛研究的课题。研究人员分辨提出了神经网络剪枝、量化和神经网络结构搜索等策略来设计高效的神经网络模型，也有一些工作提出专门的硬件加速器来加速神经网络。

在剪枝领域，斯坦福大学的韩松博士等人 [23] 首次在深度学习时代提出了神经网络剪枝的概念，实现了大幅减少神经网络的参数量（3 至 10 倍），并在后续工作“深度压缩”” [21] 中以可学习量化和哈夫曼编码等方式进一步提升了神经网络参数压缩的效果。但这两篇工作在剪枝后破坏了原有参数规则储存的模式，从而没有真正解决在通用硬件（CPU / GPU) 上大幅加速神经网络这一任务，而依赖于专门设计的 ASIC 加速器例如 EIE [22]。但 EIE 体系结构本质上是为全连接层（FC）设计，没有很好地为卷积作专门优化。此后, 何宜晖等人 [25] 和刘壮等人 [41] 提出了结构化剪枝的思想，对神经网络以通道为单位进行压缩，在取得不错的理论计算量减小的同时，实现了通用硬件上的真正加速，但通道为单位的压缩往往是粗粒度的，压缩率不如朴素的剪枝算法。此后，何宜晖等人 [26] 和旷视科技 [39] 又提出利用自动机器学习 (AutoML) 方法来搜索不均匀的剪枝策略 (non-uniform pruning policy)，取得了比均匀剪枝更好的效果。近期机器学习系统（SysML）领域的工作 [48] 指出，如果对神经网络模式化的、细粒度的剪枝，可以在利用稀疏表示特性的情况下在通用硬件上加速神经网络，这也为神经网络剪枝带来了新的洞见。

在量化领域，早期的工作 [35] 首次提到将神经网络的 32 位浮点数权重转为 8 位定点数，以实现在 CPU 或 FPGA 加速器上的快速推理。这利用了 CPU 指令集中的向量化特性或 FPGA 加速器善于高效地通过位移操作实现定点数运算。后期的工作如 PACT [11] 往往关注如何通过更好地设计量化阈值最大程度上避免精度损失，亦有类似于前述非均匀剪枝策略的非均匀神经网络量化，例如王宽等人的工作 [72]，这类工作往往也利用了自动机器学习（AutoML）和强化学习（Reinforcement Learning, RL）。

不同于前述工作利用剪枝，量化技术从一个高度冗余的大神经网络当中裁剪得到小模型，加州大学伯克利分校的刘壮等人 [42] 又指出，有时直接设计一个参数量小、计算量低的小模型也可以达到从大模型中剪枝得到的小模型的相同效果。受此启发，麻省理工学院的蔡涵等人 [3, 4], 谷歌大脑 Quoc V. Le 团队 [27, 66, 67], Facebook AI Research [77] 利用 Neural Architecture Search (NAS [91]) 技术直接设计神经网络模型，在无需显式地考虑剪枝的情况下也得到了在多种设备 (CPU, GPU, mCPU, mGPU) 上非常高效的模型，同时他们提出的方法可以考虑比网络剪枝更广的搜索空间，例如神经网络的深度，以及所用卷积核的大小，甚至是网络的连接形式。神经网络架构搜索也可以考虑到量化本身，例如旷视科技近年的工作 [20].

过去的的高效神经网络设计工作的往往考虑图像分类这一任务，而图像分割则很少被涉及 [36, 10]。此外，鲜有工作在自动机器学习（AutoML）的场景下同时考虑神经网络设计、网络压缩和网络量化。这使得我们利用神经网络自动设计技术达到高效图像分割的研究具有很大意义。

在本文中，我们研究多平台下高效的图像分割模型设计问题。受前人工作启发，我们直接用神经网络架构搜索（NAS）来替代剪枝，在所有 GPU 硬件平台上达到实时推理，最低

也能在 GTX 1060 上达到 43 FPS 的图像分割处理速度（而 GTX 1060 如今已经是笔记本级的 GPU）。此外，神经网络结构搜索为模型带来了可观的准确率提升，我们在 Cityscapes [13] 数据集上，四种硬件平台下取得了最高 1.1% 的 mIoU 提升，同时反而相比基线模型提高了 1.2 倍的推理速度。我们的工作亦能通过延迟表技术做到为任何硬件平台进行硬件延迟的直接建模，而不需要考虑像浮点运算量（FLOPs），特征图尺寸或激活大小（activation size）或参数量这些建模模型高效性的不准确指标。此外，不同于谷歌公司 [66, 67, 27] 或 Facebook 公司 [77] 动辄需要 10^4 GPU Hours 的神经网络搜索算法，我们的算法得益于超网络训练与架构搜索分离的设计范式，即便在 GTX 1080Ti 单卡下也可以达到约 72 小时的超网络训练时间，且对于不同平台网络的部署，超网络只需要训练一次。这使得我们的神经网络搜索算法在探索相同大小的搜索空间的同时，比工业界的竞品具有二至三个数量级的训练代价优势，因而十分适合在学术界计算资源不足的情况下作为一个很好的研究基线工作。我们的代码将随论文一起提交，以强化我们研究工作的可复现性，并为低年级同学的研究提供帮助。我在代码实现中定义的接口高度模块化，亦可以十分容易地扩展到许多本文中不曾提及的任务当中，例如图像分类、图像实例分割，以及物体检测等。本文中所提出方法的主体亦在三维深度学习领域取得成功，例如我近期参与的工作 [68] 是第一个用神经网络搜索设计高效三维深度学习网络结构的工作，这足以证明本文中提出的方法在不同领域是通用、可泛化的。

在论文的其余部分中，我们将在第二章介绍相关研究工作，第三章介绍朴素的 One-Shot 神经网络搜索算法和我们在其基础上做出的多项改进，第四章介绍实现细节和实验结果，第五章进行总结。

第二章　相关工作

我们将从三方面介绍为本论文带来启发的相关工作：在我们的任务图像语义分割中比较常用的方法，基于软/硬件的神经网络通用加速策略，以及神经网络结构搜索。

2.1　图像语义分割

2.1.1　传统方法

在 2012 年 Krizhevsky 等人 [33] 提出深度学习以前，图像分割领域的算法往往基于传统的机器学习算法。典型的方法通常利用 Boosting [61, 71]，或支持向量机 [17] 或随机森林 [61]。更早的方法则基于图模型，以能量最小化为目标，将图像分割任务等效转化为图最小割问题 [60]。此类方法可通过条件随机场进一步优化 [32]，其本质是考虑了相邻像素的预测类标的相容性。然而，传统方法往往并不能实现端到端的大规模训练、难以达到很高的精确度，基于条件随机场的方法在推断方面过于缓慢，也限制了此类方法的实用性。

2.1.2　高表现深度学习模型

在深度学习被广泛应用于计算机视觉任务之后，图像分割领域也出现一系列以高表现为目标的、基于深度学习的方法。早期工作 FCN [43] 在图像分类的神经网络之后加上上采样操作，得到与输入尺寸相同的特征图，并通过分类器输出预测结果。在 FCN 中，最后一个上采样层往往需要将特征图的尺寸扩大 4× 至 16×，这导致 FCN 的输出结果往往缺乏细节建模。为改善这一情况，一系列编码器-解码器类的方法被提出。U-Net [58] 最早用于医学图像分割，通过一个渐进上采样的网络逐渐恢复特征图的尺寸，并提出利用跨层残差连接 [24] 来加快收敛。DeepLab [6], DeepLabV3 [8], DeepLabV3+ [7] 系列工作依次提出使用扩张卷积（dilated convolution），空间金字塔池化（Spatial Pyramid Pooling，亦见于 PSPNet [86]）技术改进了编码器-解码器范式的性能；而 AutoDeepLab [36] 利用可微分神经网络架构搜索寻找最优的分辨率变化策略，在不需要 ImageNet [15] 预训练的情况下进一步提高了 DeepLab 系列模型的性能。此外，还有一些工作关注保持图像中的高分辨率信息，如 HRNet [65] 始终保持最高分辨率的信息，但通过减小通道数的方法降低计算；PointRend [30] 在全景分割网络 SemanticFPN [31] 的基础上参考了三维视觉领域的 PointNet [53], 用二维点云的表示形式为矩阵特征图补充高频信息。

2.1.3　轻量化深度学习模型

由于图像语义分割模型常常需要在自动驾驶、虚拟现实等领域部署，模型的运行效率变得十分重要。上一类图像语义分割模型往往可以在诸多常见数据集上达到最先进的性能（state of the art, SOTA），但模型的运行速度往往不尽如人意（在桌面级 GPU 如 NVIDIA GTX1080Ti 上小于 5 帧每秒）。通常，这是因为此类模型当中含有分辨率直接上调至 1024 × 2048 尺度的 ImageNet 模型所导致。受此观察启发，轻量化语义分割模型设计往往有两种思路。其一是采用轻量化的 ImageNet 模型做基线网络，并简化语义分割的专门分支（如空间金字塔池化），相关工作包括 MobileNetV3+DeepLab [27]。另一种思路则更改基线网络的性

质，使之与 ImageNet 上的模型不同，往往是在网络早期做更为激进的下采样。此类代表工作有 B iSeNet [81], FastSCNN [52], ICNet [85] 等，它们往往可以在桌面级 GPU 如 NVIDIA GTX1080Ti 上不加优化地获得 30 帧每秒以上的执行速度。另有一类方法如 EVS [50] 不仅设计了轻量化的深度学习模型，同时利用数据集的视频特性，通过消除帧间冗余, 实现单帧模型难以达到的加速比。

2.2 基于规则的神经网络加速

语义分割之外，神经网络加速也是本文主要研究的课题。早期的工作往往采用基于规则（贪心）的神经网络加速策略来设计剪枝、量化算法，也有一系列工作直接从头手动设计高效的小模型，用于各类视觉识别任务。

2.2.1 神经网络剪枝

神经网络剪枝从对全连接层（FC）的加速开始。基于 FC 的矩阵乘法本质，SVD 被早期研究 [16] 用于加速 FC 层，但往往带来很大的识别精度损失。此后, 基于贪心算法的细粒度剪枝 [23] 被提出，其通过迭代式地剪除神经网络中较小的卷积核权值达到压缩神经网络参数量的目的。而深度压缩 [21] 则进一步引入了基于 k 近邻的权重量化以及哈夫曼编码，使得一些典型的卷积神经网络如 VGG [62] 能被压缩一个数量级以上。然而，如前所述，剪枝和深度压缩带来了稀疏化权重，这不利于通用处理器上的并行处理。因而，后续工作在压缩率方面退而求其次，提出卷积核的整通道修剪 [25, 41] 以获得规整的稀疏模式，并可以使剪枝后的模型容易地在通用处理器上被加速。近期亦有工作 PatDNN [48] 指出，即便是对卷积核权重做较细粒度的（但有固定模式的）修剪，也可以通过编译器层面的优化在通用处理器上获得加速。而由于 PatDNN 剪枝的细粒度特性，其往往可以获得介于朴素的剪枝 [23] 和通道修剪方法 [25, 41] 之间的压缩比。此外，Slimmable Networks [83, 82] 考虑了神经网络剪枝算法多次部署的问题。不同于通常的剪枝算法每设计一个复杂度的模型都要进行一次训练，SlimmableNets 只需一次训练便可以方便地支持各种剪枝比例，大大降低了部署的代价，也启发了后来的 One-Shot 神经网络架构搜索算法。

2.2.2 神经网络量化

神经网络剪枝通过减小参数量以达到减少计算和内存访问的效果，从而加速神经网络。而神经网络量化则探索通过为每个权重赋予更低的比特数，来减小访存，并利用处理器指令集中的向量化指令实现更低的计算延迟。神经网络量化最早的尝试出现在论文 “定点量化” [35] 中, 而之后的工作 DoReFaNet [88] 则提出直连估计器（Straight-Through Estimators）实现了可训练量化，而 PACT [11] 则在量化操作中设定了可学习的截断阈值，提升了 DoReFaNet 在视觉识别任务当中的精确度。此外，英特尔亦提出渐进式的量化方法 INQ [87]，并不一次性量化所有参数，而是逐渐将浮点表示转换成 8 位整数表示。近年来亦有工作关注无需重新训练的量化策略，例如高通研究院提出的无数据量化方法 DFQ [47]，可以直接将训练好的浮点模型在不用额外数据校准（calibrate）的情况下无损地转换成定点模型。此外，更具有挑战性的低比特量化，例如二值量化 [14] 或三值量化 [90]，也曾在过去的研究中被探讨。

2.2.3 高效小模型手动设计

此外，刘壮等人 [42] 指出，对冗余的大神经网络进行剪枝可能并不优于直接设计小模型。此前已有一些工作进行该方面探索，例如模型尺寸小于 500KB 的 SqueezeNet [29], 专为移动平台设计的、基于深度可分离卷积或分组卷积的工作 MobileNet [28], MobileNetV2 [59], ShuffleNets [44, 84] 等。其中 MobileNetV2 [59] 中提出的 Mobile Inverted Residual Block 在其他高效视觉任务变得尤为常用。

2.3 神经网络自动搜索 (AutoML)

基于规则的神经网络加速通常因为人类专家可探索的设计空间有限而无法获得最优的选择。因此，研究人员提出神经网络自动搜索（AutoML）方法来更探索更大的搜索空间，以期获得精度-速度权衡更优的模型。

2.3.1 暴力搜索方法

谷歌大脑团队首先开创了基于暴力搜索的 AutoML 方法 [91]。一个基于 LSTM 的超控制器（meta-controller）被用于采样神经网络结构，每个神经网络结构在目标数据集上被从头训练，所得到的评估精度被用于更新超控制器。后来谷歌在 [92] 提出使用强化学习算法提升采样效率并减小搜索空间，但搜索一个模型的代价仍在 10^4 GPU Hours 以上。此后，相同团队亦使用类似的暴力算法设计了能在移动设备上运行的 MNasNet [67] 和 EfficientNet [66] 等网络结构用于图像分类，以及 NAS-FPN [18] 用于物体检测。

2.3.2 基于超网络的搜索方法

由于暴力搜索算法对计算资源极高的要求，学术界开始思考其替代品——基于超网络的神经网络结构搜索。超网络通常指代神经网络设计空间中最大的网络，而所有子网络的权重可以从超网络中抽取。基于超网络的神经网络结构搜索往往先训练超网络，然后进行架构搜索，最后把搜索到的子网络重新训练后进行部署。而这类方法又主要分为可微分搜索（DARTS [38]）和 One-Shot 神经网络搜索 [2] 两种路线。

可微分搜索 [38] 在超网络的权重参数基础上引入一套新的架构参数来象征待选路径的重要性，在训练过程当中通过交替优化的方式对架构参数和权重参数进行调整。在搜索阶段，只有架构参数最大的待选路径才能被保留下来，其他路径皆被删除。后续工作如 Proxyless-NAS [3], P-DARTS [37], PC-DARTS [80] 分别解决了 DARTS 结果不稳定、内存占用极大的问题，逐渐把神经网络架构搜索的时间和空间代价都降低至与训练单个模型相同的水平。

不同于可微分搜索（其训练过程与搜索过程高度耦合），One-Shot 神经网络搜索 [2] 本质上解耦了超网络训练和架构搜索的过程。旷视 [20] 指出，可以在不需要架构参数的情况下通过各候选路径的均匀采样来训练带有一定权值共享的超网络，然后用一个基于遗传算法的搜索过程在资源限制下寻找最优的子网络。后续工作 [63, 4] 提出了更细粒度的权值共享，进一步降低了超网络训练的代价。其中蔡涵等人提出的 Once-for-All 更是在 ImageNet 将搜索网络和重新训练结果网络的代价降低至忽略不计。

2.3.3 神经网络自动搜索在剪枝、量化中的应用

神经网络自动搜索亦在网络压缩领域有着广泛的应用。例如剪枝领域，何宜晖和林己等人 [26] 提出 AMC，利用强化学习设计 ImageNet 上模型的剪枝策略，可以达到比人类专家 [21] 更高的压缩率，且在小模型如 MobileNet [28] 可以达到比均匀收缩通道数更优的加速效率和精确度。后来基于 One-Shot 的剪枝策略 Meta Pruning [39] 又解决了 AMC 无法高效处理残差连接剪枝的弱点，在具有残差连接的结构如 MobileNetV2 [59] 上也能取得加速。

在网络量化领域，亦有基于强化学习的方法 HAQ [72] 和基于 One-Shot 的方法 Single-Path One-Shot [20]。HAQ 将硬件模拟器给出的 FPGA 延时反馈作为强化学习智能体的激励（reward），并观察到强化学习算法会针对云/端的不同硬件给出截然不同的量化策略。Single-Path One-Shot 则无需强化学习，利用遗传算法直接在 BitOps 限制下搜索高效的量化网络。

2.4 本章总结

本章中，我们探讨了图像语义分割领域近年来的一些进展，包括传统方法、基于深度学习的高表现模型和轻量级模型。此后，我们综述了高效神经网络设计领域的主要思想，例如剪枝、量化和神经网络自动搜索。本章的讨论为后续方法的提出奠定了基础。

第三章　方法

在本章中，我们将介绍所提出的高效神经网络设计方法在图像语义分割中的应用。我们将首先简单描述在 Cityscapes 数据集 [13] 上的基线网络模型，其设计思路类似于 FastSCNN [52]。然后，我们将回顾朴素的 One-Shot 神经网络搜索方法，并讨论其局限性。根据对于朴素方案局限性的讨论，我们提出改进的 One-Shot 神经网络搜索方法，具有权重共享、深度渐进收缩、直接建模硬件延迟、支持网络量化等新特性。我们提出的改进方案在 NVIDIA GTX1080Ti 单卡上设计单个网络的代价仅为 4 天（训练单个网络的时间需要 2 天），且新增一个网络设计的边际代价仅为单卡 10 小时。我们自动设计的语义分割神经网络在 2016 年就已十分普遍的桌面级图形处理器 NVIDIA GTX1060 上可以达到实时处理（而常见的语义分割模型，如 DeepLabV3+，即便采用最轻量级的骨干网络，亦难在高端的 V100 GPU 上获得实时速度）。我们将在下一章中给出改进的 One-Shot 神经网络搜索方法在多平台上设计专用神经网络的详细结果并进一步分析。

3.1　语义分割基线模型

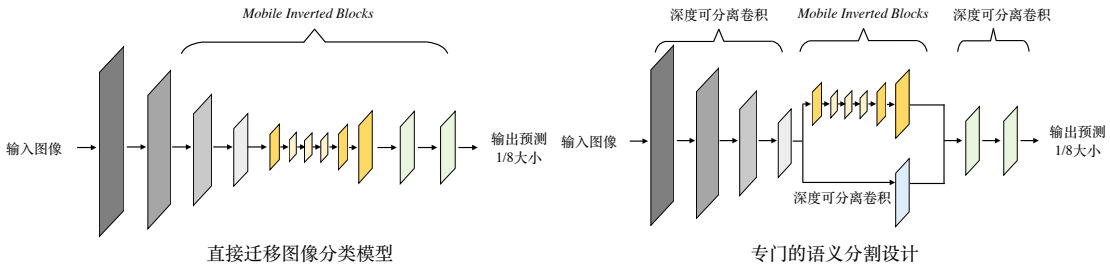


图 3–1 直接从分类模型迁移的 MobileNetV2 [59] 和为语义分割重新设计的 MobileNetV2。

方法	IoU	运行时间 (ms)
MobileNetV2 直接迁移	62.0	36.3
MobileNetV2 重新设计	64.4	**11.7**
MobileNetV2 重新设计 + 高分辨率残差连接	**68.3**	13.1

表 3–1 不同的语义分割模型设计之间的表现、高效性比较。可见，专为语义分割设计的 MobileNetV2 具有最优的表现和高效的运行时间。

我们在图 3–1中展示了所设计的语义分割基线模型和直接迁移 ImageNet 上的分类模型 MobileNetV2 [59] 之间的对比。由于我们的目标是设计可以在图形处理器上进行实时推理的语义分割神经网络，我们大量采用了 MobileNetV2 [59] 中提出的 Mobile Inverted Blocks。这是一种类似于 Bottleneck Residual Blocks [24] 的神经网络基本块，不同点在于普通卷积被替换为深度可分离卷积，且基本块的扩张率大于 1（而非 Bottleneck Residual Blocks 中的 $\frac{1}{4}$）。

语义分割模型的输入分辨率往往为 1024×2048，约为图像分类模型的 $4 \times -8\times$。这使得直接将 ImageNet [15] 上设计的图像分类模型迁移到图像分割任务上存在困难。一方面，

MobileNetV2 [59] 在初期降采样层使用了普通的卷积层，导致消耗的时间约为总运行时间的 30% 以上，上述降采样层直接迁移到语义分割任务后计算量会增加 32×，带来很高的、难以优化的延迟（单图可达 **10ms** 以上）。而且，如图 3–1左所示，ImageNet 上的分类模型均很早（$\frac{1}{4}$ 原图分辨率）开始进行主要模块（如 Mobile Inverted Blocks [59]）的堆叠计算。据统计，在 GTX1080Ti 上以批大小为 8 运行时，第一个扩张率（expand ratio）为 6 的基本块的运行时间便可以达到 **12.5ms**，这直接阻碍了整个模型达到很高的推理速度。另一方面，图像分类模型往往为较小的分辨率调优，并不是专门为高分辨率输入设计。我们在表 3–1中通过相同的训练参数分别训练左，右图的神经网络并对比了他们在 Cityscapes 数据集上的 IoU 性能。可以明显发现，在 ImageNet 上专门设计的图像分类模型即便在图像分割任务中使用更大计算量，也没有达到理想的分割准确率。

受上述观察启发，以及近期工作 FastSCNN [52] 的思考，我们对 ImageNet 上的分类模型做两点调整，以适应图像分割任务的特点。其一，我们将图像分类模型初期的降采样操作从 2 次调整至 3 次，并用深度可分离卷积代替不同卷积。这样，我们将降采样层的代价从 10 ms 降低到 **3 ms** (GTX 1080Ti)，同时保证了后续的基本块 Mobile Inverted Blocks 正好工作在接近 ImageNet 图像的分辨率上，这样做在降低计算的同时，也可以获得提升模型的表现（表 3–1）。其二，我们增加一个高分辨率残差连接分支为基本块分支补充高频信息，通过表 3–1中的消融实验，我们容易发现该残差连接极大地提升了图像分割模型的性能。此外，它仅引入了可忽略不计的运行时间代价（**1.5 ms**，GTX1080Ti）。

在得到高效且具有较高表现的基线模型后，我们将描述在此模型的基础上用神经网络架构搜索方法进行优化的思路。

3.2 朴素的 One-Shot 神经网络搜索方法

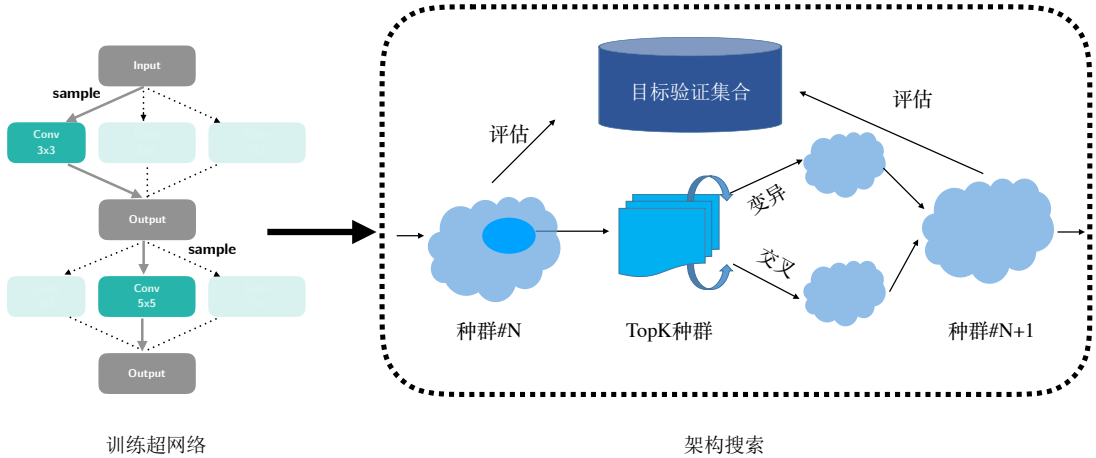
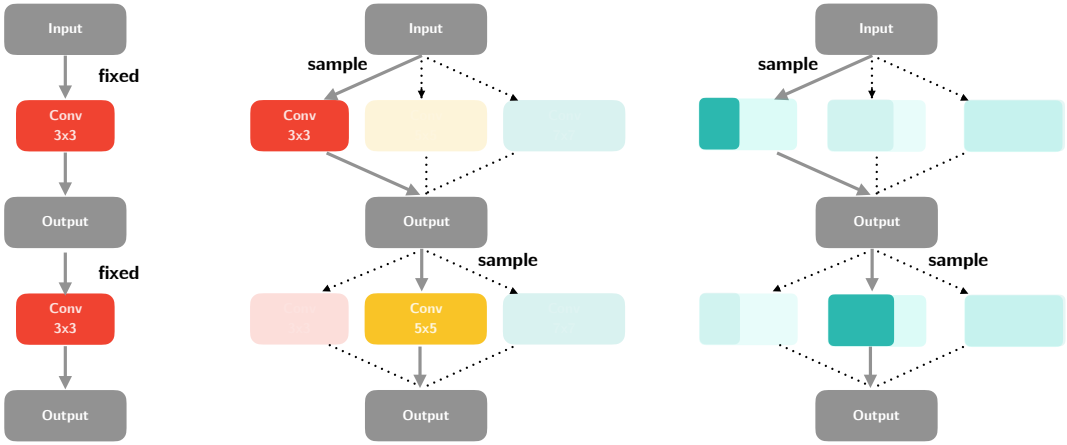


图 3–2 朴素的 One-Shot 神经网络搜索流程。

Bender 等人 [2] 在 2018 年提出了 One-Shot 神经网络搜索方法。朴素的 One-Shot 神经网络搜索的流程可由图 3–2概括。它被分为两段，首先一个包含搜索空间中所有子网络的超网络被训练，然后从超网络中搜索符合条件的最优的子网络。我们将在本节中简要介绍 One-Shot 神经网络搜索，并分析其问题，以便读者理解后文中对相应解决方案的描述。

3.2.1 超网络及其训练



单个网络　　超网络(卷积核尺寸可变)　　超网络(通道数可变)

图 3–3 超网络示意：包含了所有搜索空间中的子网络的（随机）神经网络结构。

超网络是 One-Shot 神经网络搜索 [2] 中提出的概念。我们在图 3–3中展示了超网络和单个网络的区别。通常，卷积神经网络的一层可以用

$$\boldsymbol{X}_{n+1} = \mathrm{ReLU}(\mathrm{BN}(\boldsymbol{K}_n * \boldsymbol{X}_n)). \tag{3–1}$$

表示，其中符号 $*$ 代表二维卷积操作，$\boldsymbol{K}_n$ 代表第 n 层对应的卷积核，其往往是 $C_{\text{out}}\times C_{\text{in}}\times k\times k$ 的 4D 张量，而卷积核大小 k 则往往取 3。而 $\boldsymbol{X}_n$ 代表第 n 层对应的特征图，亦为 4D 张量，具有 $B\times C_{\text{in}}\times H\times W$ 的空间尺寸，其中第一维是批大小。我们在以下所有表示中忽略空白填充（padding）。

不同于普通卷积神经网络，超网络中的一层往往可以如下表示：

$$\boldsymbol{X}_{n+1} = \sum_{i=1}^{m} f(\boldsymbol{\alpha}_{n,i}, i)\mathrm{ReLU}(\mathrm{BN}(\boldsymbol{K}_{n,i} * \boldsymbol{X}_n)). \tag{3–2}$$

其中第 n 层的架构参数 $\boldsymbol{\alpha}_n = [\boldsymbol{\alpha}_{n,1}, ..., \boldsymbol{\alpha}_{n,m}]$ 是一个概率分布，其代表第 n 层的 m 种潜在选择（例如图 3–3中所展示的 $3\times 3, 5\times 5, 7\times 7$ 卷积，或不同的通道数[1]）在训练中被选中的概率。在本文中，我们考虑一个简单的情形 [20]，即均匀分布：

$$\forall i \le m, \boldsymbol{\alpha}_{n,i} = \frac{1}{m}. \tag{3–3}$$

而函数 f 根据架构参数的值和当前选项本身决定各选项的权重。这里，存在两种定义方式：

$$f(\boldsymbol{\alpha}_{n,i}, i) = \boldsymbol{\alpha}_{n,i}. \tag{3–4}$$

被早期的工作采用，其优势是超网络的前向传播完全连续，于是在反向传播时具有可导的性质。但其最大的劣势时在反向传播时需要在内存中保留所有选择的激活值，这往往导致

[1]此处通道数指代 Mobile Inverted Blocks [59] 中的扩张率。

训练超网络的内存占用是单个网络的 m 倍（不妨假设每层的选择数目都是 m）。于是，f 也可以被简化为：

$$f(\boldsymbol{\alpha}_{n,i}, i) = \mathbf{1}_{\text{选项 i 在运行时被选中}}. \tag{3–5}$$

如此，在训练时，任何时刻的反向传播只要求在内存中保留一条被选中的路径；亦即，训练超网络所需要的内存占用与单个网络是相同的。当然，这样的 f 函数带来的代价是架构参数 $\boldsymbol{\alpha}$ 很难通过基于梯度的方法进行优化。蔡涵等人 [3] 提出使用 REINFORCE 对架构参数进行优化，但强化学习的训练对于超参选择极为敏感，因此，我们也可以将此 f 函数与一固定分布（如前述均匀分布）结合，这样就无需优化架构参数。

我们在算法 3–1中描述 One-Shot 神经网络搜索算法中超网络的训练过程。在算法中，我们已经假设架构参数不会被更新。在前向传播时，每一优化步中每层随机取样一种操作，而在反向传播中，只有该条路径的参数被更新。

算法 3–1 超网络训练

1: $\boldsymbol{X}_0 =$ 输入.
2: **for** $n = 1; n \leq N; i{+}{+}$ **do**
3: 对第 n 层，以架构参数 $\boldsymbol{\alpha}_n$ 为分布采样选项 i，记录 $\text{choice}_n = i$。
4: 按照 $\boldsymbol{X}_n = \sum_{i=1}^{m} f(\boldsymbol{\alpha}_{n,i}, i)\text{ReLU}(\text{BN}(\boldsymbol{K}_{n,i} * \boldsymbol{X}_{n-1}))$ 计算第 n 层的输出。
5: **end for**
6: **for** $n = N; n \geq 1; i{-}{-}$ **do**
7: 更新第 n 层第 choice_n 种操作的网络参数。
8: **end for**

3.2.2 神经网络结构搜索

在超网络训练完成后，图 3–2的右侧描述了从超网络中搜索得到符合要求的子网络的过程。Bender 等人的原始工作 [2] 假设子网络被随机地从超网络中取出并评估，最终在验证集上准确率最高的子网络将被保留。然而，随机搜索具有较低的采样效率，故旷视科技 [20] 提出利用遗传算法来替代随机搜索。具体地，我们首先初始化第 0 代种群 $\mathcal{G}_0$，它包含 P 个随机的子网络。此后，$\frac{P}{2}$ 个子网络通过 $\mathcal{G}_0$ 中验证集准确率最高的 $\frac{P}{2}$ 做自身变异得到，另 $\frac{P}{2}$ 个子网络则通过 $\mathcal{G}_0$ 中验证集准确率最高的 $\frac{P}{2}$ 个子网络做二元交叉得到。这里，变异操作指随机地改变某一层选择的操作，而二元交叉指在每一层抑或选第一个网络的操作，抑或选第二个网络的操作。得到的 P 个网络结构组成了下一代种群 $\mathcal{G}_1$。我们迭代式地通过上述方法产生 $\mathcal{G}_k, k \geq 2$：以 $\mathcal{G}_{k-1}$ 中最好的 $\frac{P}{2}$ 个网络作为样本池，分别通过变异和二元交叉各生成 $\frac{P}{2}$ 个网络，并将它们组合起来得到 $\mathcal{G}_k$。当然，我们也可以在生成子网络时限制它们满足一些限制条件，如 FLOPs、运行时间等，但这并未在朴素的 One-Shot 神经网络结构搜索中提到，而我们将在后文详述。

3.2.3 存在的问题

One-Shot 神经网络搜索具有超网络搜索、神经网络搜索过程解耦合的特征，这带来的最大优势便是超网络训练只需执行一次，而针对不同任务的部署只需要进行多次遗传算法搜索即可。而搜索过程本质是进行推理，其代价远低于神经网络训练。但朴素的 One-Shot 方法亦存在一些困难，我们将在本节中进行探讨。

搜索空间难以扩大。 较大的搜索空间意味着 $m = |\alpha_n|$ 较大。此时，假设我们采用前述定义的 α_n 符合均匀分布，每条路径在每次迭代中被采样到的概率 $\frac{1}{m}$ 则会降低。在实际操作当中，典型的 m 的值往往取 9（对应 Mobile Inverted Block 为主的搜索空间，则是三种卷积核尺寸和三种扩张率）；当 $m = 9$，这意味着每种选择等效的被训练时间是总时间的 $\frac{1}{9}$，亦可以理解为每一个子网络被等效训练的时间不多于总时间的 $\frac{1}{9}$。此时，在 One-Shot 方法搜索阶段对子网络进行评估时，往往得到的准确率与子网络从头训练 $\frac{1}{9}$ 总时间所得的准确率接近。在很多任务当中（例如语义分割），神经网络在训练前期的准确率并不能很好地反映其最终可达到的准确度。当 m 取更大的值时，这一问题会变得更加严重。为解决这一问题，我们在后文参考 [4, 63] 设计了权重共享策略，以保证每个子网络的等效训练时间等同于超网络训练时间。这样 m 便可以取足够大的值（如有必要）。

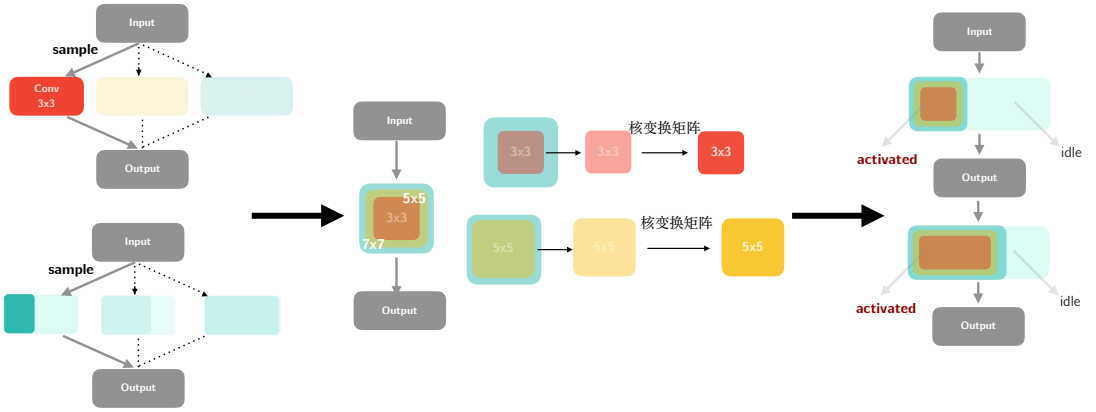
对可变网络深度的支持不好。 One-Shot 神经网络搜索在超网络训练阶段并未很好支持可变深度，原作者采用 DropBlock 策略，随机地从网络的每个阶段删除一些基本块，以达到改变深度的目的，但这使得训练超网络变得十分困难（因为对于基本块 n，原本其只需考虑块 $n-1$ 的输出作为其输入的情况，现在其将需要考虑块 $n-1, n-2$, 甚至直至块 1 的输出作为其输入的情况）。类似的做法还包括 [3] 在训练阶段引入了无操作路径作为每个基本块的一种可能选择，但这没有从本质上解决训练不稳定的问题。此外，在选择单元中引入无操作选项亦导致最大深度的网络被采样到的概率仅为 $\frac{1}{2^N}$（其中 N 为层数）。由于深度更大的网络往往具有更好的性能，这限制了网络搜索探索高表现模型的能力。为解决这一问题，我们将在后文描述深度渐进收缩策略。这一策略首先确保对于任何一个基本块 n，其永远只需考虑基本块 $n-1$ 作为输入的情形；其次保证了最大深度的网络被训练到的次数严格不小于一倍的标准训练周期。

搜索阶段没有显式的约束。 朴素的 One-Shot 神经网络搜索 [2] 仅仅考虑了在搜索阶段使用随机搜索，故不能做到根据显式约束来筛选种群的效果，而显式约束往往在实际任务中是极为重要的。例如，设计在手机上运行的模型往往需要考虑模型的运行时间，在单片机上运行的模型往往需要考虑单片机有限的片上内存、闪存资源等。在旷视 [20] 提出的改进中，也仅仅是使用计算量（FLOPs）作为一个代理的约束，并不能直接建模相应的子网络在对应硬件上的实际执行延时。事实上，在搜索过程当中直接测子网络在硬件上的延时是困难的，例如 GPU 存在热身期，这导致我们往往需要至少 50-100 次前向传播才能得到一个子网络的执行延迟；CPU 上网络的执行时间则受到操作系统调度、中断等因素的影响，更难以直接获得；对于手机等设备，则会有发热效应，多次运行子网络的前向传播后推理速度将减慢。基于这些原因，即便是旷视 [20] 提出的改进也很难把硬件直接的反馈融入到网络搜索的过程中。为解决这一问题，我们将在后文描述基于硬件延迟表的策略，可以直接把在硬件平台上运行子网络前向传播的代价由 $\mathcal{O}(T)$ 降低到 $\mathcal{O}(1)$，其中 T 是搜索的网络个数。

3.3 改进的 One-Shot 神经网络搜索

我们基于对于朴素的 One-Shot 神经网络搜索存在的限制的理解设计改进后的 One-Shot 神经网络搜索。如前所述，我们分别用权重共享策略、网络深度渐进收缩、直接建模硬件延迟和硬件延迟约束的神经网络结构搜索解决上一节提到的三个问题。并在此基础上，加入了对神经网络量化的支持。下文将描述相关细节。

3.3.1 权重共享策略



超网络(路径独立)　　超网络(卷积核参数共享)　　超网络(卷积核参数/通道数共享)

图 3–4 权重共享机制示意图。

如前所述，朴素的 One-Shot 方法中没有权值共享机制，于是在每层 3 种卷积核尺寸、3 种通道数的情况下只能保证每个子网络获得最多 $\frac{1}{9}$ 的总训练时间。受此观察影响，我们提出在卷积核、通道数两个层面分别实现权值共享（如图 3–4所示），以确保每个子网络均被完整训练。欲描述权值共享机制，我们先将卷积操作 $\boldsymbol{K} * \boldsymbol{X}^{(n)}$ 展开为下式：

$$\boldsymbol{X}_{uvw}^{(n+1)} = \sum_{i=1}^{R}\sum_{j=1}^{R}\sum_{k=1}^{C} \boldsymbol{K}_{ijk}^{w} \boldsymbol{X}_{t(u-1)-p+i,t(v-1)-p+j,k}^{(n)} + \boldsymbol{b}^{w}. \tag{3–6}$$

其中 $\boldsymbol{R}$ 为卷积核尺寸，$\boldsymbol{C}$ 为通道数，t 为步长（stride），p 为空白填充（padding），$\boldsymbol{b}$ 为偏置（bias）。

卷积核权值共享。 我们分别用 $R=5, R=7$ 对式子 3–6做实例化如下：

$$\boldsymbol{X}_{uvw}^{(n+1)} = \sum_{i=1}^{5}\sum_{j=1}^{5}\sum_{k=1}^{C} \boldsymbol{K}_{ijk}^{w} \boldsymbol{X}_{t(u-1)-2+i,t(v-1)-2+j,k}^{(n)} + \boldsymbol{b}^{w}. \tag{3–7}$$

$$\boldsymbol{Y}_{uvw}^{(n+1)} = \sum_{i=1}^{7}\sum_{j=1}^{7}\sum_{k=1}^{C} \boldsymbol{L}_{ijk}^{w} \boldsymbol{Y}_{t(u-1)-3+i,t(v-1)-3+j,k}^{(n)} + \boldsymbol{b}^{w}. \tag{3–8}$$

其中 $\boldsymbol{X}, \boldsymbol{Y}$ 表示特征图，$\boldsymbol{K}, \boldsymbol{L}$ 表示卷积核。我们不妨从式子 3–8中取出 $(i,j) \in [2,6] \times [2,6]$ 的部分，则又可以写为：

$$\boldsymbol{Y}_{uvw}^{(n+1)} = \sum_{i=2}^{6}\sum_{j=2}^{6}\sum_{k=1}^{C} \boldsymbol{L}_{ijk}^{w} \boldsymbol{Y}_{t(u-1)-2+(i-1),t(v-1)-3+(j-1),k}^{(n)} + \boldsymbol{b}^{w}. \tag{3–9}$$

做简单的代数代换得到下式：

$$\boldsymbol{Y}_{uvw}^{(n+1)} = \sum_{i=1}^{5}\sum_{j=1}^{5}\sum_{k=1}^{C} \boldsymbol{L}_{i+1,j+1,k}^{w} \boldsymbol{Y}_{t(u-1)-2+i,t(v-1)-3+j,k}^{(n)} + \boldsymbol{b}^{w}. \tag{3–10}$$

注意到，式子 3–10与式子 3–7在形式上几乎完全相同，若有 $\boldsymbol{X}^{(n)} = \boldsymbol{Y}^{(n)}$ 且 $\boldsymbol{L}_{i+1,j+1,k} = \boldsymbol{K}_{ijk}$，则两式输出完全相同。这等价于说明，在卷积的定义中，一个 7×7 卷积核的**中间** 5×5 部分与一个 5×5 的卷积核功能等效。类似地，我们亦可以证明一个 7×7 卷积核的中心 3×3 部分与一个 3×3 的卷积核功能等效。实际当中，我们修改 $\boldsymbol{L}_{i+1,j+1,k} = \boldsymbol{K}_{ijk}$ 之要求为：

$$\boldsymbol{K}_k = \boldsymbol{T}\boldsymbol{L}_k \tag{3–11}$$

其中 $\boldsymbol{T}$ 称为核变换矩阵（其参数可以通过梯度优化）。这是因为我们希望在 $\boldsymbol{X}^{(n)} = \boldsymbol{Y}^{(n)}$ 的条件下保持 $\boldsymbol{X}^{(n+1)} \approx \boldsymbol{Y}^{(n+1)}$，那么 $\boldsymbol{L}_{i+1,j+1,k} = \boldsymbol{K}_{ijk}$ 会使得卷积核的外层不得不为 0，这显然是次优的。而引入核变换矩阵后，卷积核的非中心 5×5 部分则不必为 0。

图 3–4中间部分给出了关于卷积核权值共享的直观表示。值得一提的是，核变换矩阵仅仅引入了极少的参数，例如从 7×7 矩阵中截取出中间的 5×5 部分，则只需要 $25\times25 = 625$ 个核变换参数。

通道数权值共享。 实现卷积核参数共享极大地加速了超网络的收敛，然而并没有从根本上解决朴素的 One-Shot 神经网络搜索中搜索空间难以扩大的问题，因为卷积层的输出通道数仍没有进行权值共享。欲做到这一点，我们仍回到式子 3–6，并做两组实例化（输出通道数满足 2 倍关系）如下。

$$\forall w \le C_{\text{out}}, \boldsymbol{X}^{(n+1)}_{uvw} = \sum_{i=1}^{R}\sum_{j=1}^{R}\sum_{k=1}^{C_{\text{in}}} \boldsymbol{K}^{w}_{ijk}\boldsymbol{X}^{(n)}_{t(u-1)-p+i,t(v-1)-p+j,k} + \boldsymbol{b}^{w}. \tag{3–12}$$

$$\forall w \le 2C_{\text{out}}, \boldsymbol{Y}^{(n+1)}_{uvw} = \sum_{i=1}^{R}\sum_{j=1}^{R}\sum_{k=1}^{C_{\text{in}}} \boldsymbol{L}^{w}_{ijk}\boldsymbol{Y}^{(n)}_{t(u-1)-p+i,t(v-1)-p+j,k} + \boldsymbol{b}^{w}. \tag{3–13}$$

不难发现，如果我们保证 $\boldsymbol{X}^{(n)} = \boldsymbol{Y}^{(n)}, \forall w \le C_{\text{out}}, \forall k \le C_{\text{in}}, \boldsymbol{K}^{(n)}_{ijk} = \boldsymbol{L}^{(n)}_{ijk}$，我们就将有

$$\forall w \le C_{\text{out}}, \boldsymbol{Y}^{(n+1)}_{uvw} = \boldsymbol{X}^{(n+1)}_{uvw}. \tag{3–14}$$

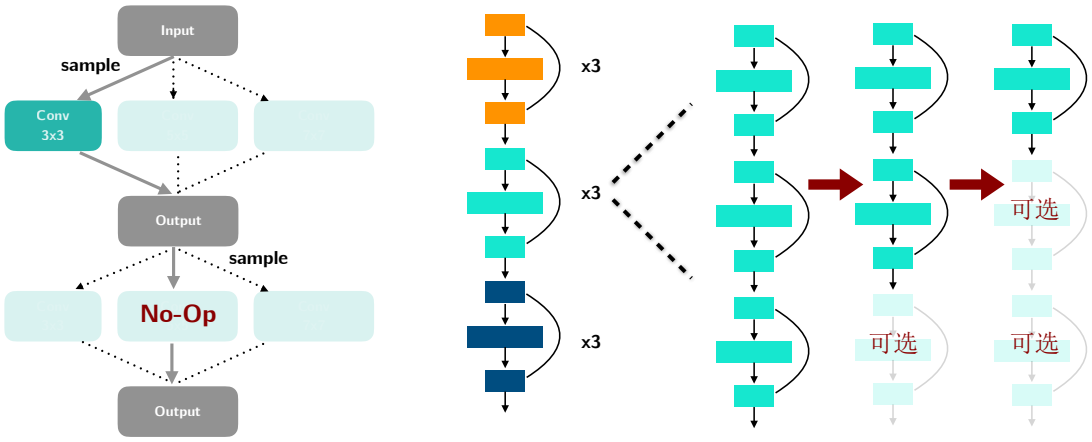
这就说明，可以将输出通道 C 的情况理解为输出通道 $2C$ 中的前 C 个通道，这与单独定义两条独立的路径在功能上可以达到等价。图 3–4右侧直观地描述了通道数共享的思路。我们总是取“左侧”的目标通道数。与此同时，我们也保留前述的卷积核权值共享。至此，我们便在深度不可变的网络中实现了完全的参数共享。对于每一种（卷积核尺寸，通道数）选择，其等效的训练时间均等同于整个超网络的训练时间。这极大地改进了超网络的精确度。我们在表 3–2中展示了相关结果（随机采样 100 个子网络得到的平均验证集 IoU）。可见，在相同的超网络训练时间下，权值共享将子网络平均准确率提升了 **25%**，并使之达到与单个网络从头训练（$\ge 60\%$) 接近的水平。这使得搜索阶段的遗传算法可以获得更有意义的反馈来筛选种群。

3.3.2 网络深度渐进收缩

朴素的 One-Shot 神经网络结构搜索另一问题在于处理网络深度可变的情况时会引起训练不稳定。如图 3–5所示，原始的 One-Shot 神经网络搜索算法通过在每一层加入一个无操作选项来达到跳过当前层的效果。按照前述，这种做法导致最深网络被采样到的概率仅有 $\frac{1}{2^N}$，其中 N 为网络基本块数量；同时训练极不稳定，因为对于块 n，原本只需考虑块 $n-1$ 为输

方法	IoU
One-Shot 超网络（无权重共享）	32.8
One-Shot 超网络（有权重共享）	57.5
One-Shot 超网络（有权重共享，深度渐进收缩）	**57.6**

表 3–2 从超网络中直接抽取的子网络性能期望与是否使用权重共享的关系。



朴素支持可变深度网络　　深度渐进收缩

图 3–5 提出的深度渐进收缩技术。

入的情况，而现在则对于块 $i, \forall i \leq n-1$ 都需要考虑，显著增加了训练难度。为解决这一问题，我们将超网络的训练由一个阶段更改为 K 个阶段（假设每组基本块重复 K 次）。具体地，在第 $k(k \geq 1)$ 个阶段，如图 3–5所示，我们允许超网络中基本块组中保留的**最小**基本块数目为 $K-k+1$ 个，且如果保留的基本块数目为 l，必定是当前基本块组的前 l 个基本块被保留。网络深度渐进收缩的训练方式同时解决了最深网络被采样到概率极低的问题和训练不稳定的问题。对于最深的网络，它被采样到的次数至少为一整个训练阶段的总时间，而不是该时间的 $\frac{1}{2^N}$。对于第 k 个基本块，它只要在训练中被选中，一定只需考虑第 $k-1$ 个基本块的输出。在表 3–2中我们展示了网络深度渐进收缩对超网络中随机抽取的子网络的精确度影响。可以看出，有渐进收缩策略存在时，可变的网络深度并不会对超网络中抽取的子网络的表现有负面的影响（而通常，在使用朴素的策略时，负面影响是存在的）。这使得我们可以在搜索阶段探索深度方面变化很大（实际为 2×）的不同网络结构。

在实现上，网络深度渐进收缩等同于对预训练好的，支持最小深度为 K 的网络做 $K-1$ 次调优（finetune）。值得注意的是，在训练的每个阶段，我们均允许卷积核的尺寸和通道数在权值共享的前提下任意选择。

3.3.3　直接建模硬件延迟

至此，我们基本解决了超网络训练阶段朴素的 One-Shot 神经网络搜索所面临的困难。接下来的两个小节我们将考虑在搜索阶段设计与硬件相关的直接约束。对于给定的硬件平台，神经网络在其中运行的延迟（latency）是最好的反应网络高效性（efficiency）的指标。平台无关的指标例如计算量（FLOPs）或激活尺寸（activation size）也可以描述神经网络的

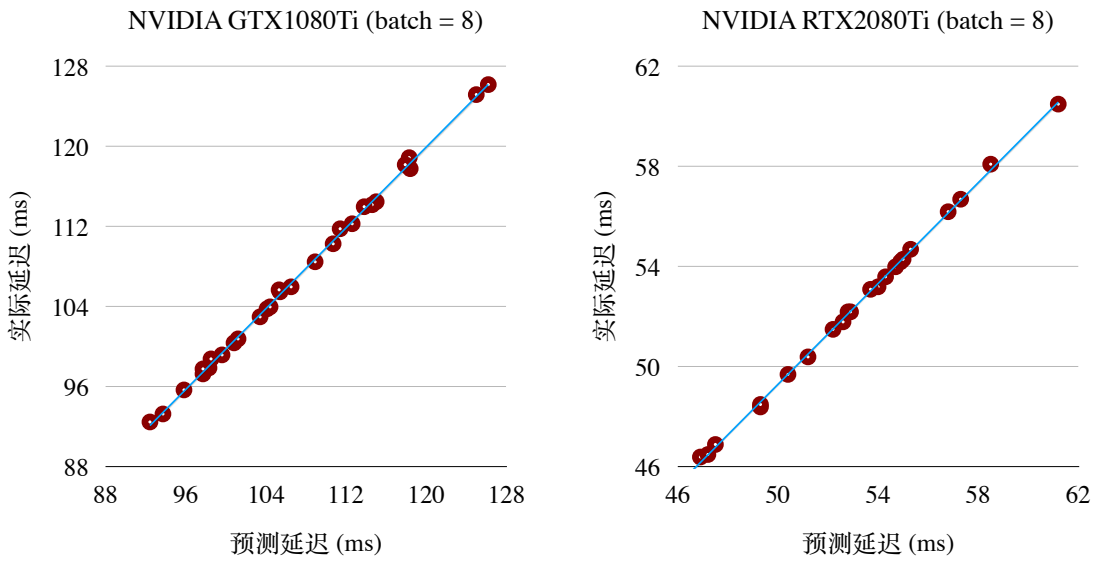


图 3–6 用层延迟累加估计整个网络延迟表的合理性：估计延迟与真实延迟几乎满足 $y = x$ 关系。

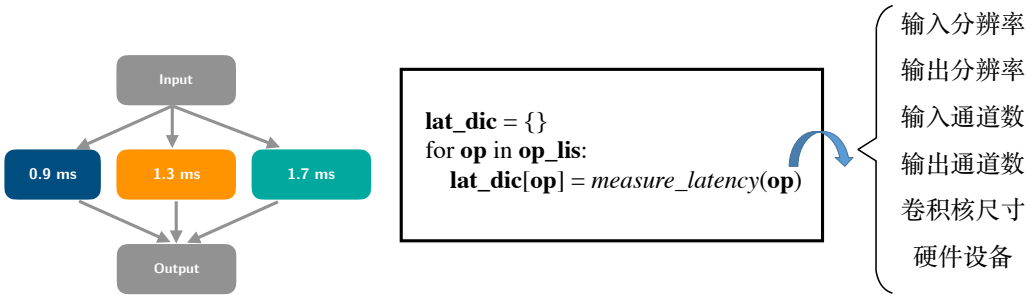


图 3–7 从超网络中获得延迟表的流程图示。

高效性，但忽略了许多与体系结构相关的因素，例如并行度，核函数调用代价，访存代价等重要指标。而通过对各种体系结构相关因素做白盒建模以得到精确的、解析的延迟预测值在 FPGA 或部分 ASIC 上可行，在内部结构更为复杂的 CPU 和 GPU 上缺乏可操作性。而如果在运行时测量神经网络的延迟，则面临需要多次测量（往往 ≥ 50）的困境，这样会使得测量效率极低。为解决这一问题，我们选择使用延迟表技术（图 3–7）来对硬件延迟做黑盒分析。具体地，对于神经网络 $F = \{O_1, O_2, ..., O_N\}$，我们假设：

$$\text{latency}(F) = \text{latency}(\{O_1, O_2, ..., O_N\}) = \sum_{i=1}^{N} \text{latency}(O_i). \tag{3–15}$$

上述建模对于 GPU，CPU 上未经优化的深度学习框架是可行的，因为主流框架如 PyTorch [49], Tensorflow 皆以顺序执行计算图为主，并不会进行调度优化 [9]，对多个算子并行执行。我们在图 3–6展示了两种硬件平台（GTX1080Ti, RTX2080Ti）下，利用式子 3–15预测网络的延时和直接测量网络延时所得结果的区别。可见，预测得到的延时与实际硬件上执行延时基本符合线性关系。于是，获得神经网络 F 的延时被等价转换为获得 $\forall i \leq N, O_i$ 的延迟。

在超网络中，O_i 往往是一个离散集合。如图 3–7所示，可以用六元组（输入分辨率，输

出分辨率，输入通道数，输出通道数，卷积核尺寸，硬件设备）来表征 O_i 的特性。显然，其中的每一个维度都是离散的，故我们可以通过枚举六元组对应的所有可能性来建立层 O_i 所有可能操作的延迟信息并尺寸在一个字典结构中。在测试时，我们通过一次运行网络直接得到每一层的输入输出分辨率，通过随机采样得到每一层的输入输出通道数和卷积核尺寸，而硬件设备人为指定。这样一来，我们得到确定的六元组，便可以从字典结构中召回事先测试的操作延迟。

如前所述，延迟表技术可以显著地把测试时延迟测量的代价从 $\mathcal{O}(N)$ 降低到 $\mathcal{O}(1)$，也可以有效规避在 CPU 设备上运行时测试延迟不稳定的问题。

3.3.4 资源限制条件下的神经网络结构搜索

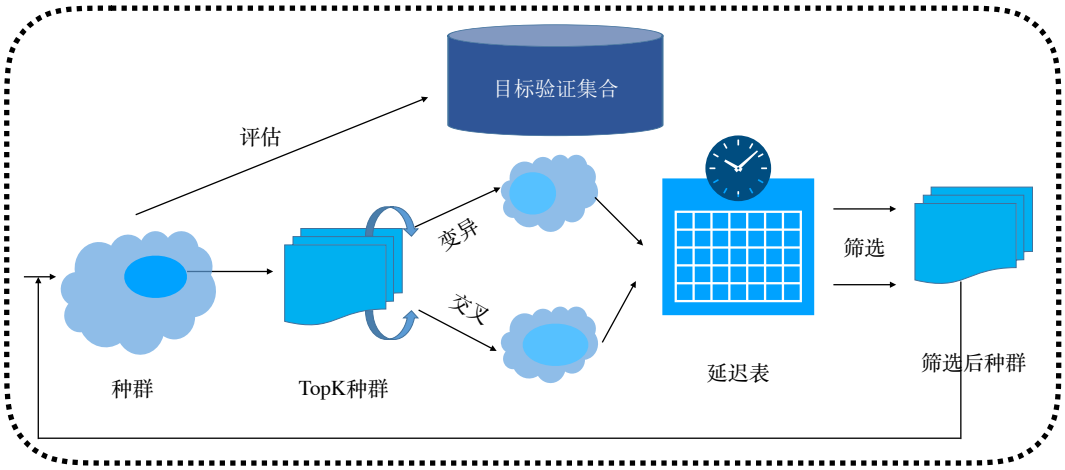


图 3–8 在延迟表约束下进行神经网络结构搜索。

由于精确的延迟表的存在（图 3–6），我们可以在遗传算法搜索阶段直接引入给定硬件平台上的延迟限制作为筛选种群的条件。具体地，我们将图 3–2中的搜索部分替换为图 3–8。具体地，我们通过以下方式产生 $\mathcal{G}_k, k \geq 2$：以 $\mathcal{G}_{k-1}$ 中最好的 $\frac{P}{2}$ 个，**在给定平台延迟不大于给定限制的**网络作为样本池，分别通过变异和二元交叉各生成 $\frac{P}{2}$ 个**在给定平台延迟不大于给定限制的**网络，并将它们组合起来得到 $\mathcal{G}_k$。此外，在搜索过程中，由于权重共享的存在，不同于朴素的 One-Shot 神经网络搜索算法在评估阶段可以直接使用超网络中直接抽取的权重在验证集上运行，我们提出的改进方案必须对批归一化参数 (BatchNorm statistics) 做重新校准。欲理解这一操作的必要性，我们先考察批归一化层的定义如下。

$$\hat{\boldsymbol{x}_{\text{train}}} = \gamma \frac{\boldsymbol{x}_{\text{train}} - \mathbb{E}[\boldsymbol{x}_{\text{train}}]}{\sqrt{\text{Var}(\boldsymbol{x}_{\text{train}}) + \epsilon}} \tag{3–16}$$

上式 3–16指代在训练过程中，每个时间步下批归一化操作的行为。与此同时，批归一化层还在训练时维护两个全局量 μ 和 σ^2，分别表示训练数据集上移动平均的 $\boldsymbol{x}$ 的期望和方差。其中

$$\mu_t = \lambda \mu_{t-1} + (1-\lambda)\mathbb{E}[\boldsymbol{x}_{\text{train}}]; \sigma_t^2 = \lambda \sigma_{t-1}^2 + (1-\lambda)\text{Var}[\boldsymbol{x}_{\text{train}}]. \tag{3–17}$$

在测试时，我们使用

$$\hat{\boldsymbol{x}_{\text{test}}} = \gamma \frac{\boldsymbol{x}_{\text{test}} - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{3–18}$$

来进行前向推理。若欲使得式子 3–18的行为尽可能与式子 3–16接近，我们希望测试集上的 $\boldsymbol{x}_{\text{test}}$（即输入分布）和训练集合上的 $\boldsymbol{x}_{\text{train}}$ 尽量接近，且测试集上的 $\mathbb{E}[\boldsymbol{x}_{\text{test}}]$ 和 μ 接近，$\text{Var}(\boldsymbol{x}_{\text{test}})$ 和 σ^2 接近。

其中，$\boldsymbol{x}_{\text{test}}$ 和 $\boldsymbol{x}_{\text{train}}$ 分布之间的接近程度完全由数据集决定。但 $\mathbb{E}[\boldsymbol{x}_{\text{test}}]$ 和 μ 的关系在权重共享的超网络中并不是仅由数据集决定。注意到，超网络中的 μ 计算涉及到 $\mathbb{E}[\boldsymbol{x}_{\text{train}}^{(\text{supernet})}]$ 的移动平均，其不同于原始的 $\mathbb{E}[\boldsymbol{x}_{\text{train}}^{(\text{singlenet})}]$ 的移动平均。对于 $\boldsymbol{x}_{\text{train}}^{(\text{supernet})}$ 的第 c 个通道，其可以是来自当前超网络基本块中任何一个输出通道数 $\geq c$ 的选择。于是，$\mathbb{E}[\boldsymbol{x}_{\text{train}}^{(\text{supernet})}]$ 的值在不同时刻是从不同宽度的子网络输出得到。而对于 $\mathbb{E}[\boldsymbol{x}_{\text{train}}^{(\text{singlenet})}]$，它总是从相同确定宽度的网络的输出中得到。因此，两者的移动平均不同分布。这会直接导致式子 3–18的行为与式子 3–16完全不同，从而影响超网络中直接抽取的子网络在验证集上的准确率。实际当中，如果直接从超网络中不加改变地抽取出子网络并在验证集上评估，得到的 IoU 范围将在 2.5% **到** 53.4% 的范围内变动。这与我们在表 3–2中所汇报的表现相去甚远。为得到表 3–2中的结果，我们需要重新计算批归一化层的全局统计量。

具体地，给定超网络当中确定的子网络选项，我们先使用递归调用的**确定化**（determinize）函数（算法 3–2）从超网络中截取出子网络，然后清除子网络的批归一化统计量（亦即令 $\mu = 0, \sigma = 1$），并用训练集的一个子集计算式子 3–17。通过这样做，我们得到的批归一化统计量与直接用单个网络在训练集上做统计别无二致，于是消除了 μ 和 σ^2 的计算定义对 3–18与式子 3–16的行为差别，使其仅受到（不可抗力的）训练、测试集差异影响。

算法 3–2 “确定化” 函数：从超网络中抽取子网络

```
新的网络 s = 超网络 S 的拷贝.
将新的网络 s 转为队列 q.
while q 为非空队列 do
    操作 x = q.popleft();
    for x 中的每一个子操作 m do
        if m 是 NAS 基本块 then
            m = 确定化 (m);
            将 m 加入队列 q.
        end if
    end for
end while
从队列 q 重新构造模型 s.
返回模型 s.
```

3.3.5 支持量化的训练过程

到目前为止，我们基本解决了利用改进的 One-Shot 神经网络搜索高效地在目标数据集上、硬件平台延迟直接约束下，自动设计进行浮点运算的语义分割网络的目标。然而，由于语义分割模型本身计算量较大，在一些设备如 CPU 上，完全浮点的模型具有很高的运行时间（例如单图 350 ms），此时如果能通过 8 位参数量化以及相应的向量化指令进行推断，则可以在理论上将计算量减小 4×，实际上也可获得一定加速。此外，在一些其他硬件设备如 FPGA 上，进行浮点运算的单元数量有限，因此往往用定点数进行相关运算是更优的选择。鉴于此，我们在本节描述如何在神经网络结构搜索的框架中引入对参数量化、激活特征图量化的支持。

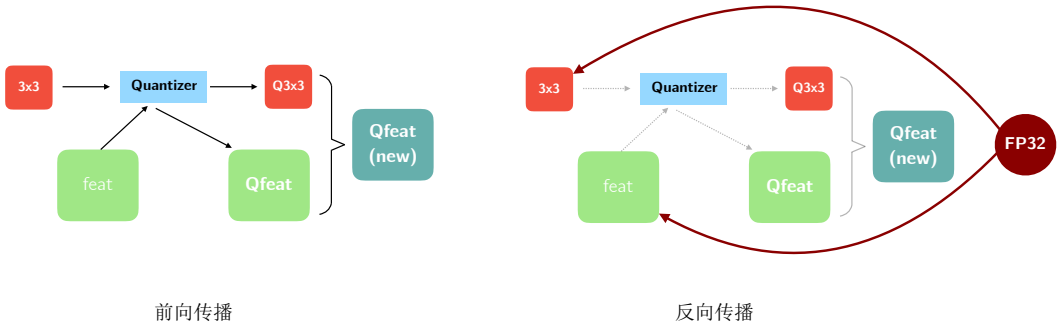


图 3–9 支持量化的超网络训练流程。

欲理解支持量化的训练过程，我们需对神经网络量化操作先作介绍。我们以 k 比特量化为例。k 比特量化假设神经网络的权值/激活值可以用 2^k 个数来表示。对激活值的量化可以用下式来表示：

$$\boldsymbol{X}_{\text{clip}} = \frac{1}{2}(|\boldsymbol{X}| - |\boldsymbol{x} - \alpha| + \alpha). \tag{3–19}$$

其中 α 为一可学习的截断阈值。上式表明，当 $\boldsymbol{X}$ 中元素的值小于等于 0 时，其被截断为 0（这与 ReLU 激活函数的效果类似），否则，$\boldsymbol{X}$ 中元素的值被截断至给定阈值 α。此后，可以通过下式对激活值进行量化：

$$\boldsymbol{X}_{\text{quant}} = \text{round}(\boldsymbol{X}_{\text{clip}} \frac{2^k - 1}{\alpha}) \frac{\alpha}{2^k - 1}. \tag{3–20}$$

注意到，亦可以对权重 $\boldsymbol{K}$ 做类似的操作：

$$\boldsymbol{K}_{\text{quant}} = \text{round}(\boldsymbol{K}_{\text{clip}} \frac{2^k - 1}{\beta}) \frac{\beta}{2^k - 1}. \tag{3–21}$$

而 β 亦是截断阈值。此后，原始的卷积操作被重新实现为 $\boldsymbol{X}_{\text{quant}} * \boldsymbol{K}_{\text{quant}}$。前向传播中的量化是容易理解的，但量化操作为反向传播带来困难，因为函数 round 是不可导的。为解决这一问题，我们使用直连估计器（Straight-Through Estimators, STE）来近似 $\boldsymbol{w}$ 的梯度。具体地，我们**人为定义**

$$\frac{\partial \text{round}(\boldsymbol{w}_{\text{clip}})}{\partial \boldsymbol{w}} = 1 \tag{3–22}$$

由于 STE 的存在，我们无需直接更新 $\boldsymbol{w}_{\text{quant}}$，而可以选择一直保留 $\boldsymbol{w}$，通过对 $\boldsymbol{w}$ 的更新来达到利用梯度优化的效果。实验证明，STE 对于 $\boldsymbol{w}$ 的梯度是很好的估计，可以尽可能地恢复原始 FP32 模型的精确度。

我们在图 3–9中更直观地描述了支持量化的训练过程。正如前文数学表达式所描述，在支持量化训练的前向传播阶段，卷积核参数和激活特征图均被不可微的量化器转化为离散表达。然后两者进行卷积运算输出结果。在反向传播阶段，来自上游的梯度直接绕过量化器传到权重和浮点特征图上，从而可以用梯度下降直接优化。

3.4 本章总结

在本章中，我们首先分析了常见的迁移自图像分类模型的语义分割神经网络的问题，并给出了手动设计更快、更好的语义分割神经网络的方法。进而在更好的语义分割基线网络的基础上探讨神经网络结构的自动设计问题。我们分析了朴素的 One-Shot 神经网络架构搜索方法存在的主要问题：搜索空间难以扩大、对可变神经网络深度支持不好、无法直接建模

硬件特性，并对应地提出权值共享、深度渐进收缩策略和延迟表引导的架构搜索技术。我们还提出支持量化的训练过程解决为 FPGA，ASIC 等特殊硬件设计低精度神经网络的问题。

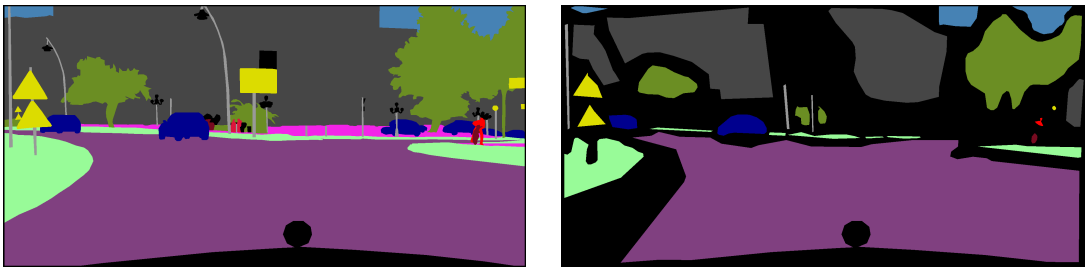
第四章　实验

本章节中，我们将介绍上一章节中的改进的 One-Shot 神经网络搜索算法在 Cityscapes [13] 数据集上的表现。我们将在多种硬件平台：NVIDIA GTX1060, NVIDIA GTX1080Ti, NVIDIA RTX2080Ti, NVIDIA TitanRTX, AMD Ryzen 7 上部署我们的模型，且都以更低的硬件延迟达到比基线模型更高的精确度。我们将系统分析神经网络搜索算法的决策，并给出模型在数据集上预测的可视化结果。

4.1　实验设置

4.1.1　Cityscapes [13] 数据集

图 4–1 Cityscapes [13] 数据集中的场景示例。



细粒度标注　　粗粒度标注

图 4–2 Cityscpaes [13] 数据集上的标注示例。

Cityscapes [13] 数据集是一个室外驾驶场景语义分割数据集。该数据集包含来自 50 个欧洲城市（如亚琛、汉堡、汉诺威等）的一组高分辨率（1024 × 2048）的图像，其中的一些示例如图 4–1所示。数据集包含 5000 张具有精细标注的图像，其中 2975 张图像用作训练

集，500 张用作验证集，1525 张用作测试集。在我们的实验中，我们不使用测试集当中的图像，而把官方的验证集作为测试集。在所有单个网络的训练中（亦包括神经网络结构搜索得到的最优模型重新训练），我们使用全部的 2975 张训练图像作为训练数据优化模型。在所有神经网络结构搜索中，我们从训练集中剔除德国亚琛的全部数据（174 张图像），并用剩下的 2801 张子训练集图像作为训练超网络的训练集。我们将德国亚琛的 174 张图像作为神经网络结构搜索的验证集，以验证集 IoU 为指标选择最优模型。此外，数据集提供了 20000 张具有粗粒度标注的图像。粗细粒度标注的区别可参见图 4–2。在我们的实验中，为控制实验的时间，我们不使用具有粗粒度标注的 20000 张图像。前人的工作表明使用粗粒度标注的图像做预训练可以进一步提升模型的性能。

Cityscapes 数据集的像素级别标注涵盖 30 个分类，其中 19 个分类被计入最终的评价指标。数据集的最主要评价指标是平均 Intersection-over-Union (mean IoU / mIoU)。mIoU 的是 19 个类 IoU 的简单代数平均，而单个类的 IoU 计算方式如下：

$$\mathrm{IoU}_c = \frac{\mathrm{correct}_c}{\mathrm{seen}_c + \mathrm{predict}_c - \mathrm{correct}_c}. \tag{4–1}$$

其中 $\mathrm{correct}_c$ 代表类 c 中被正确预测的像素数，seen_c 表示整个数据集中见到的，属于 c 类别的像素数，而 $\mathrm{predict}_c$ 表示模型预测的属于 c 类别的像素数。相比像素准确率（Pixel Accuracy），IoU 受到类别不均衡的影响更小，可以更好地反映模型在少数样本类上的表现。

4.1.2 基线模型实现细节

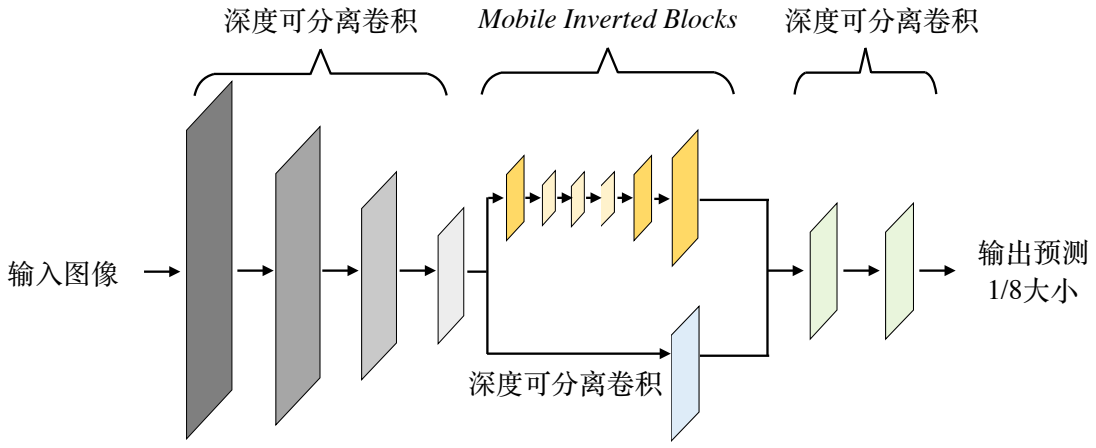


图 4–3 提出的为语义分割专门设计的 MobileNetV2 结构。

我们在图 3–1中介绍了为语义分割专门设计的网络结构，此处我们将原图的右半部分复制于图 4–3以方便读者阅读。本节中，我们给出详细的模型定义及其训练细节。我们分别用输出通道数为 32，卷积核尺寸为 3 的普通卷积、输出通道数为 48，卷积核尺寸为 3 的深度可分离卷积和输出通道数为 64，卷积核尺寸为 3 的深度可分离卷积对输入尺寸为 1024×2048 的三通道图像进行三次下采样，输出分辨率为 128×256。在图 4–3中的 Mobile Inverted Blocks 分支，我们使用三组 Mobile Inverted Blocks 基本块，输出分辨率分别为 128×256, 64×128, 32×64。三组基本块每组包含 3 个基本块，均采用扩张率为 6、卷积核尺寸为 3，每个 Mobile Inverted Blocks 在三组中的输入/输出通道数依次为 64、96、128。随后，三组基本块后输出的特征图被上采样回 128×256，与下方的通过一个输出为 128 通道深度可分离卷积的高分

辨率信息分支融合。最后，基本块分支和高分辨率分支融合后，再经过两层输出通道数分别为 128, 128 的 3×3 深度可分离卷积后，输出在原图 $\frac{1}{8}$ 分辨率上的预测。该预测直接通过双线性插值到原图分辨率，最终输出。

基线模型的训练遵循图像分割领域常见的数据增广，例如多尺度训练，以及 HSV 颜色空间上的颜色增广。训练所用的损失函数为交叉熵损失（cross entropy loss，CE-loss），模型以 0.045 的初始学习率、Poly 学习率调整策略（式子 4–2，其中 t 为当前迭代步，p 一般取 0.9。），批大小 12 被训练 1000 轮（1000 epochs，或 24.7 万次迭代）。我们在普通的交叉熵损失外对模型加深度监督，分别在三次降采样结束后和 Mobile Inverted Blocks 基本块与高分辨率分支融合之后。此二处输出的特征图亦在上采样后通过一个深度可分离卷积层达到与原图相同的分辨率，并通过计算 CE-loss。深度监督部分的 CE-loss 以权重 0.4 与最终的 CE-loss 加权，是为训练的优化目标。

$$\gamma_t = (1 - \frac{t}{t_{\max}})^p \tag{4–2}$$

按照上述训练细节描述，我们将在 Cityscapes 数据集上得到表 3–1中的结果。

4.1.3 神经网络架构搜索实现细节

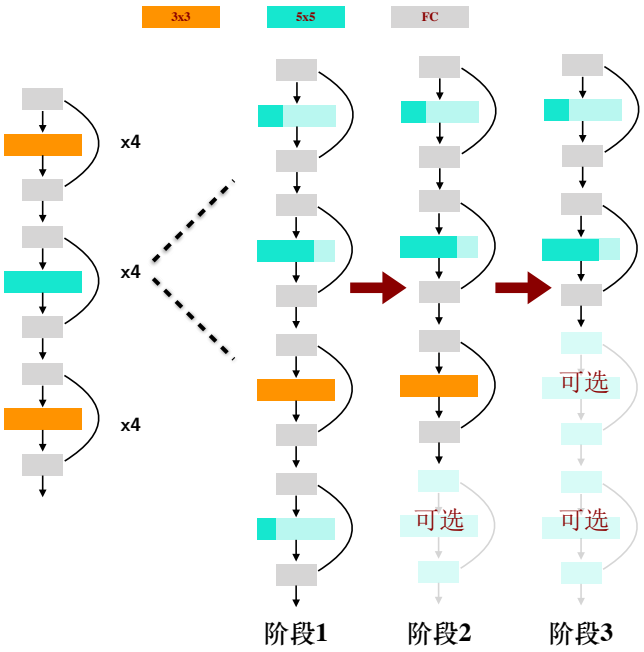


图 4–4 超网络的训练流程：先支持可变的卷积核尺寸与通道数，再支持可变的深度。

如前所述，我们使用改进后的 One-Shot 神经网络搜索方法作为我们设计高效神经网络的方法，我们将从超网络训练和神经网络结构搜索两部分介绍实现细节。

在超网络中，本文主要考虑对基线设计中三个分辨率段、每个分辨率段 3 个的 Mobile Inverted Blocks 基本块做搜索。我们允许每个分辨率段的基本块数量在 2，3，4 之间选择，每个基本块的卷积核尺寸在 3，5 之间选择、扩张率在 3，4，6 之间选择，这引入了极大的设计空间。如图 4–4所示，超网络的训练根据第三章节提出的深度渐进收缩被分成三个阶段。

超网络训练的第一阶段，我们要求三个分辨率段的基本块数固定为 4 个。而每个基本块的卷积核尺寸和扩张率不受限制地选择，但要求在上一章节提出的权重共享机制下采样权重。第一阶段采用和基线模型完全相同的训练流程（即训练 1000 轮）。训练结束后，我们保留第一阶段的权重，放开每个分辨率段的基本块数量为 3 个或 4 个。此时我们用起始学习率 2.5×10^{-3}，训练 250 轮。最后，我们再度保留第二阶段的权重，放开每个分辨率段的基本块数量为 2 个至 4 个。此时用起始学习率 8×10^{-3}，训练 750 轮。因此，超网络的总共训练时间是 2000 轮，大约是单网络训练轮数 2 倍左右。由于超网络的最后阶段探索的绝大多数网络速度比基线网络更快，超网络的实际训练时间在单 GTX 1080Ti 图形卡上约 72 小时，是单网络训练时间的 1.5 倍左右。

超网络训练完成后，我们按上一章所述进行硬件延迟直接约束的神经网络结构搜索，用到的搜索算法为遗传算法。我们初始化 100 个（如非特殊说明，本章中均指符合硬件延迟约束的）子网络，并用它们中最好的 50 个产生下一代种群。在下一代种群中需用二元交叉和变异产生之后的种群，其中我们定义二元交叉运算将等概率地选择两个亲代网络中的操作，而变异操作在每一层发生的概率是 20%。我们一共产生 10 代种群，总共 1000 个子网络，并将最后一代中验证集 IoU 最高的子网络作为最终的搜索结果。

我们将得到的搜索结果用上一章算法 3–2保留权重地抽取出来，用训练单个网络的流程在完整的 2975 张训练图像上（上一节所述）训练 1000 轮，并汇报网络在官方 500 张图像的验证集上的测试结果。

4.2 实验结果

下面我们将介绍提出的改进的 One-Shot 神经网络搜索算法在 Cityscapes 数据集上的结果，以及和基线网络结构的对比。

4.2.1 和目标实时的高效语义分割网络的对比

方法	IoU	运行时间 (ms)	FPS	硬件平台	使用额外数据预训练
SegNAS-1080Ti (Ours)	69.4	**11.2**	**89.6**	GTX1080Ti	否
MobileNetV2-DeepLabV3 [59]	62.0	36.3	27.5	GTX1080Ti	否
MobileNetV2-DeepLabV3-IN [59]	70.4	36.3	27.5	GTX1080Ti	使用 ImageNet
MobileNetV3-DeepLabV3 [27]	64.2	33.3	30.0	GTX1080Ti	否
IC-Net [85]	69.5	33.0	30.3	Titan X	否
ERFNet [57]	68.0	89.3	11.2	Titan X	否
ContextNet [51]	65.9	23.9	41.9	Titan X	否

表 4–1 所提出的改进的 One-Shot 神经网络结构搜索算法（SegNAS）和其他目标实时的高效语义分割网络的对比。

我们在表 4–1中比较了我们的方法和其他一些目标实时的高效语义分割模型。可见，我们的模型在 Cityscapes 验证集上取得最高的速度的同时，IoU 仍超越了大部分竞争对手，甚至与在 ImageNet [15] 上预训练过的 MobileNetV2-DeepLab [59] 十分接近，然而，我们的模型要比其快 3× 以上。值得一提的是，虽然 ICNet [85], ERFNet [57] 和 ContextNet [51] 的延

迟是在不同硬件设备上测得，GTX1080Ti 和 Titan X 图形处理器的性能差异并不会改变我们设计的网络要远比竞争对手更为高效的事实。

4.2.2 多硬件平台下部署结果

硬件平台	方法	IoU	FLOPs (G)	参数量 (M)	运行时间 (ms)	FPS
GTX1060	MobileNetV2Seg	68.3	7.44	1.21	27.1	36.9
	SegNAS (Ours)	**68.7**	**5.61**	**0.86**	**23.7**	**43.3**
GTX1080Ti	MobileNetV2Seg	68.3	7.44	1.21	13.1	76.3
	SegNAS (Ours)	**69.4**	**5.54**	**0.85**	**11.2**	**89.3**
RTX 2080Ti	MobileNetV2Seg	68.3	7.44	1.21	6.6	151.5
	SegNAS (Ours)	**68.6**	**5.52**	**0.80**	**5.9**	**169.5**
Titan RTX	MobileNetV2Seg	68.3	7.44	1.21	6.1	163.9
	SegNAS (Ours)	**68.6**	**5.41**	**0.79**	**5.4**	**185.2**
AMD Ryzen7	MobileNetV2Seg	68.3	7.44	1.21	358.0	2.8
	SegNAS (Ours)	**69.1**	**5.68**	**0.87**	**328.9**	**3.0**

表 4–2 所提出的改进的 One-Shot 神经网络结构搜索算法（SegNAS）和基线网络在不同硬件平台上的性能比较。

我们亦在表 4–2中展示了我们为多硬件平台设计不同的神经网络结构的结果。可见，我们的神经网络结构搜索算法在任何一个平台上均设计出了表现比基线模型明显更优的模型：不仅 IoU 取得了 0.3% 至 1.1% 不等的提升，同时我们的模型将基线模型的 FPS 分别从 36.9 提升到 43.3（GTX1060），76.3 提升到 89.3（GTX1080Ti），151.5 提升到 169.5（RTX2080Ti），163.9 提升到 185.2（TitanRTX，注意到我们在略差的计算设备 RTX2080Ti 上就可以达到基线模型在 TitanRTX 上的速度和精度），2.8 提升到 3.0（AMD Ryzen7 CPU）。此外，我们最大将基线模型的参数量减小 **1.5**×，浮点运算量 FLOPs 减小 **1.4**×，这也体现了我们的方法在进行神经网络结构搜索的同时隐式的达到了剪枝的效果，验证了刘壮等人的论文 [42] 提出的网络剪枝本质上是神经网络结构搜索特例的观点。在下文中，我们还将仔细分析为不同硬件平台设计的神经网络结构，从计算机系统的角度分析神经网络结构搜索的选择合理性。

4.2.3 预测结果可视化

我们在图 4–5中展示了我们为 GTX1080Ti 专门设计的神经网络模型在 Cityscapes [13] 官方验证集上的结果。可以发现，我们的方法尽管运行时间高达接近 90 FPS，仍然能给出极为精确的预测。

4.3 实验结果分析

4.3.1 网络结构可视化

我们在图 4–6中展示了我们提出的改进的 One-Shot 神经网络搜索方法在 Cityscapes 数据集上分别为 NVIDIA GTX1060（图4–6b），NVIDIA GTX1080Ti（图4–6c，NVIDIA RTX2080Ti（图4–6d），NVIDIA TitanRTX（图4–6e），AMD Ryzen7(图 4–6f) 专门设计的神经网络结构。

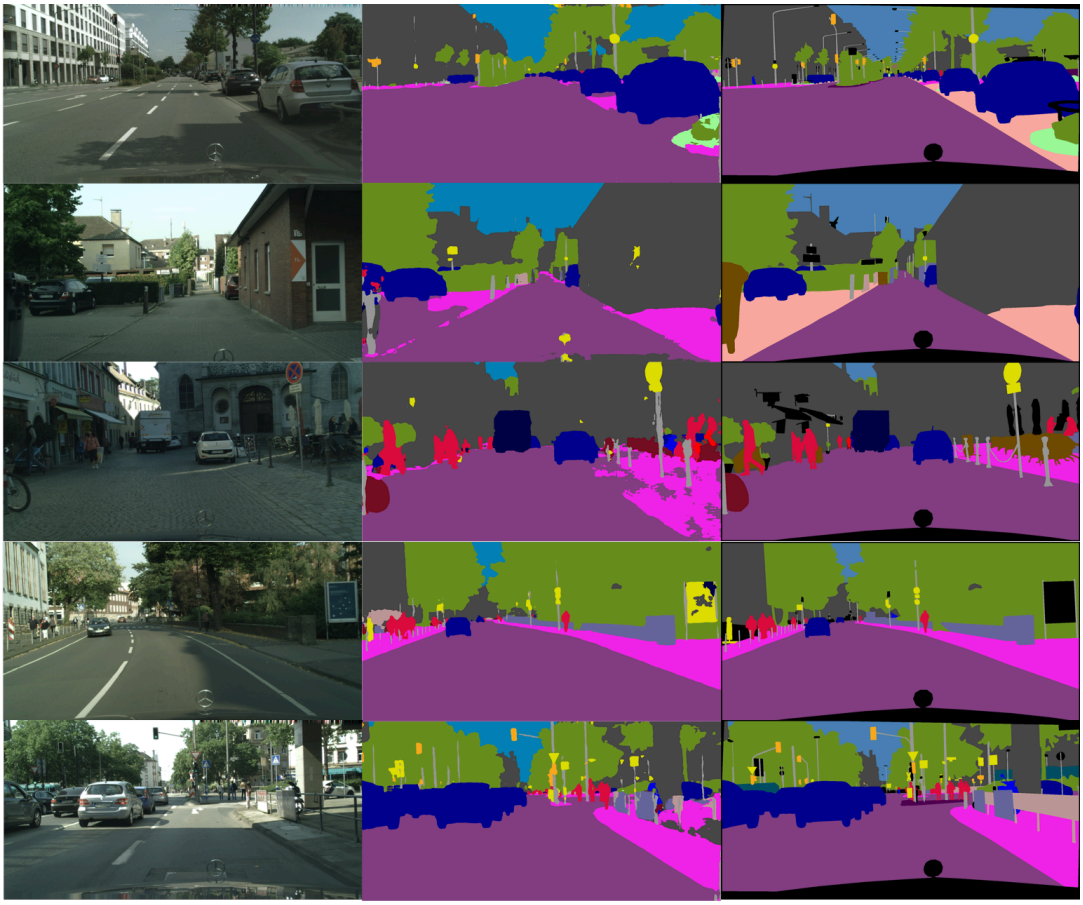


图 4–5 Cityscapes 数据集 [13] 上的定性结果。左边为输入场景，中间为我们自动搜索的神经网络的预测结果，右边为真实标注。

我们使用橙色的矩形表示 3 × 3 卷积核，青色的矩形表示 5 × 5 卷积核，而矩形的高度则表示相应的 Mobile Inverted Block 基本块的扩张率。图中仅展示目前的算法进行搜索的部分，即图 4–3中的 Mobile Inverted Block 分支。我们亦可视化了基线网络模型在该分支的行为（图4–6a），即全部的基本块都选择使用 3×3，扩张率为 6 的选项。接下来，我们针对多种平台上搜索的结果给出一些深入分析。

4.3.2 专用结构分析

大卷积核的使用。 自 VGG [62] 被提出以来，小卷积核几乎成为神经网络设计的标准。从 FLOPs 的角度看，具有相同感受野的两个 3 × 3 卷积和一个 5t × 5 卷积相比仅使用了 72%($\frac{2\times3\times3}{5\times5}$) 的计算，但带来了更多的非线性，这被认为有助于提升网络的表达能力。之后的一些常规神经网络设计如 ResNet [24]、MobileNetV2 [59] 也遵循了只使用 3 × 3 卷积核的先例。但事实上，对于 Mobile Inverted Block 基本块来说，“两个 3×3 卷积比一个 5 × 5 卷积更高效” 的论述并不成立。假设基本块的输出通道数为 C，扩张率为 e，卷积核尺寸为 k，则其计算量可以用 $2C^2e + k^2eC$ 表示。于是，两个 3×3 基本块与一个 5×5 卷积核的计算比例为

$$\frac{4C^2e + 18eC}{2C^2e + 25eC} \tag{4–3}$$

注意到往往有 $C >> e$，故该式的输出通常接近 2。此外，我们也可以从并行计算的角度进行分析。在 CUDA（NVIDIA Compute-Unified Device Architecture）计算架构中，基本操作往

往需要通过调用核函数（kernel function）执行。核函数执行前需要启动线程、为线程分配必要的空间，并进行逻辑线程与物理的流处理器（stream multiprocessor, SM）之间的映射。这一整套流程具有一定的代价。一般来说，单个核函数的调用代价至少在 $10\mu s$ 到 $100\mu s$ 之间，而比较快速的深度可分离卷积实现的运行时间一般也在几百 μs。而利用更多的基本块将造成更多的核函数调用，引入更多的额外代价。此外，由于 NVIDIA GPU 大量的流处理器单元的存在，单个卷积操作往往不能充分利用处理器全部的计算资源，则运行时间的增长可能随计算量增长而亚线性增长。这样一来，使用大卷积核就显得更具有优势了。

我们不难观察到，在图 4–6中几乎所有的硬件平台上，在有延迟限制的情况下，神经网络搜索都倾向于选择使用 5×5 卷积核的基本块。在 GPU 上比较统一的观察是，往往计算能力更强的 GPUs（如 NVIDIA RTX2080Ti 和 NVIDIA TitanRTX）会更少使用 3×3 卷积核（往往 1 次，2 次）；而计算能力稍差的 GPUs（如 NVIDIA GTX1060 和 NVIDIA GTX1080Ti）则倾向于稍多使用 3×3 卷积核，但也不会以之为主体。有趣的是，虽然 CPU 的计算能力比 GPU 更差，但其却没有经常选择小卷积核。通过分析延时表，我们发现一个 5×5 的卷积并不会在执行时相较一个 3×3 的卷积有显著更大的延时（例如在分辨率 32×64 时，即便在计算能力最差的 GPU GTX1060/AMD CPU 上，其并行能力都使得 5×5 的卷积只比 3×3 的卷积慢 25% 左右），而其具有更强的表达能力。另外，一个有趣的观察是，3×3 卷积常常在每个分辨率段的第一个基本块使用（也就是下采样基本块），而且一般配合比较低的通道数。

网络宽度、深度与硬件平台的关系。 我们继续观察图 4–6中自动设计的神经网络宽度、深度和硬件平台的关系。在 GPU 平台上，一个通用的观察是往往具有更优算力的设备 RTX2080Ti 会选择**减低网络深度、增加网络宽度**来提高并行性，减小核函数调用的代价，从而优化模型的执行效率。而对于算力较弱的设备如 GTX1060，则更青睐较窄、较深的模型。从 FLOPs 计算式 $2C^2e + k^2eC$ 看，增加 e 对于 FLOPs 的增加是一次关系。注意到对于算力较弱的设备，在通道数较大时单层可能会超过设备的并行能力限度，这使得上一节所提到的亚线性运行时间增长会转化为与 FLOPs 相同的线性时间增长，因此对于 GTX1060 这样的设备，选择更大的通道数不利于高效推断。此外，对于几乎所有的平台，我们的搜索算法均选择用大卷积核来交换网络宽度（扩张率为 6 的基本块从未被使用过）。

另一个有趣的观察是，表 4–2中展示的表现较高的三个自动设计的模型（GTX1060，GTX1080Ti，AMD Ryzen7）均在最小的分辨率阶段采用了更大的深度，但总深度与原始基线模型保持一致。显然，在低分辨率处堆叠更多卷积层可以更高效地扩大感受野，而图像语义分割相关的文献 [6, 8] 早在四年前便开始通过低分辨率处使用扩张卷积（dilated convolutions）快速提高模型的感受野。这样看，尽管我们的模型并未使用扩张卷积，却也利用可变网络深度达到了类似的效果。

类似的硬件平台，类似的网络设计。 我们在实验中恰好有两组硬件具有相似的特性，它们分别是 NVIDIA Pascal 架构下的 GTX1060 与 GTX1080Ti；及 NVIDIA Turing 架构下的 RTX2080Ti 与 Titan RTX。通过图 4–6的可视化，我们不难发现，对相似的硬件专门设计的神经网络结构是相似的。例如两款 Pascal 架构的 GPU 上设计的模型均采用 “2-3-4” 深度配比，且网络的最高分辨率阶段只使用 3×3，扩张率较低的基本块减小计算；而两款 Turing 架构的 GPU 设计的模型则反其道而行之，在模型早期更青睐使用大卷积核，而整个网络的深度更低。这体现了我们的网络搜索算法潜在地具有泛化能力。事实上只需要在一种硬件结构上做模型设计，便可以在类似的硬件结构上部署，也达到接近最优的高效性。

	GTX1060	RTX2080Ti	AMD Ryzen7
GTX1060	**23.7**	24.4	25.0
RTX2080Ti	6.0	**5.9**	6.0
AMD Ryzen7	331.3	332.6	**328.9**

表 4–3 所提出的改进的 One-Shot 神经网络结构搜索算法（SegNAS）为不同硬件平台专门设计的网络结构在该硬件平台上是最优的。

专用化与通用化的区别。 我们亦在表4–3中展示了专门为某一硬件平台设计模型和将其他硬件平台上设计的模型迁移到当前硬件平台的区别。可以发现，在三种类型的设备上（NVIDIA Pascal 架构、NVIDIA Turing 架构和 AMD CPU 架构），分别为本设备设计的最优模型在其他设备上并没有表现出最优的延时。这体现出了为不同硬件平台设计专门的模型是必要的。此外，我们将表4–3中的硬件延迟和表 4–2中基线模型的硬件延迟做对比，发现即便设计的平台与部署的平台不同，我们模型的延迟仍显著优于人工设计的网络，这也体现了神经网络结构搜索的优势。

方法	量化策略	IoU	BitOps (G)
MobileNetV2-Q	均匀 8 位激活 8 位权重	66.2	432
SegQNAS	4,6,8 位混合量化	**67.0**	**240**
MobileNetV2-Q	均匀 6 位激活 6 位权重	65.4	243
SegQNAS	4,6,8 位混合量化	**66.2**	**200**

表 4–4 提出的 SegQNAS 混合精度神经网络自动量化算法大幅减少了均匀量化基线模型的计算量，并提升性能。

混合精度神经网络量化实验结果。 我们亦在 Cityscapes 数据集上尝试了混合精度神经网络量化的自动设计。在我们的设计空间中，我们允许所有基本块的权重、激活在 4, 6, 8 位之间选择，同时开始的快速下采样阶段和输出分类器阶段也允许进行类似空间上的搜索。我们同时允许与 SegNAS 相同的网络卷积核、通道数、深度搜索。值得注意的是，我们不量化第一层的输入激活，因为输入图像本质上是无符号 INT8 的，转为 FP32 后再线性量化为 INT8 会丢失信息。在混合精度量化搜索中，我们还将所有上采样层由双线性插值改为直接映射。这样做会带来大约 0.1 的 IoU 损失，但却获得了更好的硬件映射性：因为双线性插值是浮点操作，需要专门的硬件单元来实现，而直接映射则只需要普通的内存拷贝就能容易地实现了。

我们在表 4–4中展示我们自动设计混合精度神经网络量化的实验结果。我们分别对比我们的方法与均匀的 8 位量化、均匀的 6 位量化。可以发现，我们在**节省 1.8×** 计算量时，可以取得比均匀 8 位量化高 0.8 的 IoU；而在节省 1.2× 计算时，可以获得比均匀 6 位量化高 0.6 的 IoU。与均匀 8 位量化表现相同时，我们自动设计的 SegQNAS 具有 **2.2× 的计算量节省**。通过分析自动设计的量化策略，我们发现 SegQNAS 倾向于在基本块处选择更低的位数，因为我们的实验发现量化位数的降低比起量化本身带来的误差更小（量化位数从 8 位降低到 6 位，误差增加 **0.8%**；但从 FP32 到 INT8，误差增加 **1.9%**）。另外，SegQNAS 继承了 SegNAS

搜索网络本身拓扑的优秀性能，在网络深度和卷积核尺寸上做出了不同于手动设计的选择，例如使用大量 5×5 卷积核，在低分辨率处使用更大的网络深度以快速扩大感受野等。以上因素的综合造就了 SegQNAS 的成功。

4.4 主体方法在其他任务中的应用

我们已经展示了所提出的方法在图像语义分割领域可以在多硬件平台上设计出效率更高且表现更好的模型。事实上，所提出的改进的 One-Shot 神经网络结构搜索算法在更具挑战性的任务，如三维点云语义分割中也能取得很好的效果。本节中，我将介绍我和刘志健等人 [68] 提出的首个利用神经网络结构搜索自动设计三维深度学习结构的工作 SPVNAS 及其相关结果。

4.4.1 三维计算机视觉简介

有赖于现实世界中重要的诸如自动驾驶的应用场景，三维计算机视觉在近年来获得广泛的关注。三维视觉感知往往设计对点云数据类型的处理。点云是一种不规则的数据表示形式，其利用三维坐标和特征来表征一个点:$\boldsymbol{p}=[x,y,z,\boldsymbol{f}]$。早年的研究 [5, 46, 55, 76, 89] 注意到三维数据与二维数据本身的相似性，用扩展的像素（pixel）表示——栅格（voxel）来重新建模点云，并用普通的三维卷积神经网络对体素进行分析，以完成三维点云分类、语义分割、检测等任务。然而，由于三维数据本质稀疏，研究人员也意识到稠密的栅格表示是本质上冗余且不高效的。并且，栅格表示类似于上一章节提到的量化，显然会损失高精度信息。于是，祁芮中台等人在 PointNet 中 [53] 提出直接在点云上利用对称函数学习三维表征。然而，PointNet 不具有卷积的性质，无法建模三维点云的局部信息。受此观察影响，后来的研究者分别提出了在三维点云几何邻域上的卷积操作 [34, 45, 54, 64, 69, 70, 78, 79] 或在语义邻域上的卷积操作 [75]。

然而，早年提出的三维点云上的卷积操作往往因为需要进行邻域寻址而不十分高效。一些工作开始研究高效的三维深度学习基本操作，如 Riegler 等人 [56]、王鹏帅等人 [74, 73] 提出使用 Octree，一种可变密度的栅格化表示形式来建模点云；刘志健和我等人 [40] 分析了早年三维深度学习方法的不规则访存行为和邻域计算行为，提出在规则的栅格域上卷积、在不规则的点云域上处理高频细节的 PVCNN。Graham 等人的 SSCN [19] 和 Choy 等人的 MinkowskiNet [12] 则提出稀疏卷积，在朴素的三维栅格卷积的基础上提出跳过非激活区域，以极大地减小计算量。

在我和刘志健等人最新的工作 [68] 中，我们根据三维语义分割任务中最有挑战的自动驾驶雷达点云场景（较大的场景尺度，大量的小物体，较大的物体尺度差异）设计了新的三维深度学习基本操作稀疏 Point-Voxel 卷积（Sparse Point-Voxel Convolution, SPVConv），如图 4–7所示：它利用一个可扩展性（scalability）良好的稀疏表示分支高效处理大规模场景，并用一个 MLP 点云分支处理高频信息，尤其是小物体。以 SPVConv 组成的神经网络 SPVCNN 在三维点云语义分割最大的数据集 SemanticKITTI [1] 上达到最先进的水平。

4.4.2 所提出的改进 One-Shot 神经网络结构搜索在三维视觉中的应用

我们将在此节中介绍本文中提出的改进 One-Shot 神经网络结构在三维雷达点云语义分割 [1] 的应用。我们在三维视觉任务中的设计空间与图像语义分割的设计空间略显不同。如

	# 参数 (M)	#MAdds (G)	GTX1080Ti 延迟 (ms)	Mean IoU
PointNet [53]	3.0	–	500	14.6
PointNet++ [54]	6.0	–	5900	20.1
PVCNN [40]	2.5	42.4	146	39.0
KPConv [70]	2.4	211.2	12500	58.8
MinkowskiNet [12]	21.7	114.0	294	63.1
SPVNAS (Ours)	2.6	15.0	110	63.7
	12.5	73.8	259	**66.4**

表 4–5 SPVNAS [68] 在 SemanticKITTI [1] 三维语义分割数据集上和主流的三维语义分割算法的比较：SPVNAS 以最高的速度取得最先进的性能。

	#Params (M)	#MAdds (G)	Latency (ms)	Mean IoU
DarkNet21Seg [1]	24.7	212.6	73	47.4
DarkNet53Seg [1]	50.4	376.3	102	49.9
SPVNAS (Ours)	1.1	8.9	89	**60.3**

表 4–6 SPVNAS [68] 在 SemanticKITTI [1] 三维语义分割数据集上的结果（和基于雷达二维投影的方法进行比较）。我们的方法以更快的速度取得了 10% 以上的 IoU 提升。

图 4–8所示，我们允许搜索基本 SPVConv 残差块的扩张率和输入、输出通道数，以及每个阶段的 SPVConv 残差块数量。不同于图像语义分割任务，我们并不搜索卷积核尺寸。类似于图像语义分割任务，我们定义一个超网络以包含所有可能的子网络，并将神经网络结构搜索建模成超网络训练和遗传算法搜索两阶段。在超网络训练阶段，类似地，我们用一个阶段使超网络支持可变的通道数，而另一阶段用于支持可变的网络深度。在搜索阶段，我们用计算量（FLOPs）而非特定硬件平台的延迟作约束筛选种群。

在表 4–5中, 和之前基于普通的稀疏卷积的三维深度学习方法、在多个数据集上取得最先进表现的 MinkowskiNet 相比，我们提出的方法 SPVNAS 在减小计算量 1.5× 的前提下获得 **3.3%** 的 IoU 提升。此外，我们的方法 SPVNAS 将 MinkowskiNet 的参数量减小 **1.7×**，运行时间降低 1.1×。我们又尝试在计算量约束条件下搜索高效的 SPVNAS 模型。具体地说，在计算量约束为 15G FLOPs 时，我们通过遗传算法搜索到的最优模型仍可以在 SemanticKITTI [1] 上获得 **63.7 IoU**，此结果仍然高于 MinkowskiNet 基线模型。但不同的是，我们的方法却有 7.6× **的计算量减小**，8.3× **的参数节省**，以及 2.7× **的实测加速**。

我们继续在极为高效的设定下（表 4–6）探索 SPVNAS 的潜能。本表中，我们将 SPVNAS 与基于球面投射和二维图像语义分割的雷达点云分割方法作对比。值得一提的是，通常主体是二维卷积神经网络的方法会远比三维视觉方法快速，因为二维卷积操作在各种深度学习库中都被良好优化，且二维卷积操作鲜少涉及三维深度学习方法中常见的、GPU 不擅长的不连续访存问题。然而，尽管直接基于三维的卷积方法对比二维卷积有天然的劣势，用 SPVNAS 设计的神经网络仍然可以在减小计算 **40 余倍**的同时，以 1.15 倍的加速获得 **10.4%** 更高的、具有本质级别差异的 IoU。即便是与 2019 年最先进的直接基于点云的深度学习方法 KPConv [70] 比较（表 4–5中展示了其详细结果），我们的方法亦可以在加速 **100 倍**以上的情况下，获得 **1.5%** 的 mIoU 提升。我们提出的 SPVNAS 也是三维计算机视觉领域不依赖

二维球面投影的、第一个可以达到实时的雷达点云语义分割模型[1]。

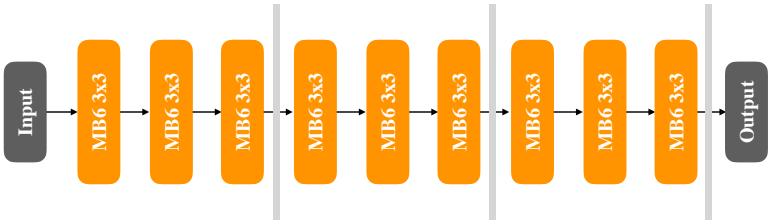
在表4–9中我们分别展示了计算量 (4–9a)、GPU 实测延迟 (4–9b) 与 SemanticKITTI [1] 上模型 IoU 的权衡。作为比较的基线方法分别是通过通道数均匀收缩手动设计的 SPVCNN 和 MinkowskiNets。我们发现在 FLOPs 12G 至 30G 的范围，或 GPU 延迟 110 到 150 毫秒之间的范围，SPVNAS 相对于手动设计的 SPVCNN 取得了 **3.6%** 至 **5.2%** 的提升。其原因在于手动设计的 SPVCNN 无法解决原始设计中编码器和解码器计算不平衡的问题，未能分配更多的计算给更重要的编码器。更为值得一提的是，与此前最先进的三维语义分割方法 MinkowskiNet 相比，我们方法在 110 毫秒附近的提升甚至达到了 **6%**。

显然可见，本文中提出的改进 One-Shot 神经网络搜索算法在迁移到三维视觉任务时也取得了十分优秀的性能，足以证明本文中提出的框架是通用的、可泛化的。我们预期提出的方法在更多任务当中也能取得不错的效果，而这将是我之后的努力方向。

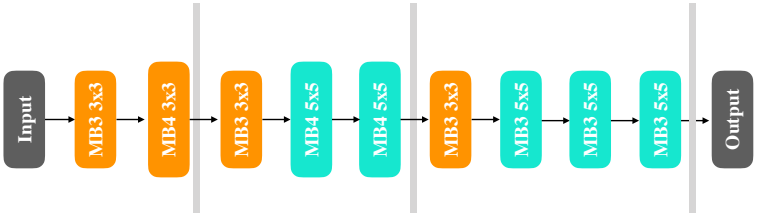
4.5 本章总结

本章中，我们展示了所提出的 SegNAS 方法在语义分割数据集 Cityscapes [13] 上的结果。我们的方法可以为五种硬件平台定制神经网络结构，最大获得 1.2× 加速和 1.1 IoU 提升，这在传统的压缩算法中是难以做到的。同时，我们还可视化了各硬件平台上的神经网络结构，发现算力更强的设备倾向大、宽、浅的模型，而算力更弱的设备则反之的现象，这与计算机系统中关于并行度的直觉是完全相符的。此外，我们利用 SegNAS 进行混合精度神经网络量化自动搜索，可以在降低均匀精度量化 2.2× 计算量的同时仍获得相同的准确率，这得益于量化策略和网络结构的共同搜索。最后，我们实验上证明了所提出的 SegNAS 方法的思想可以迁移到更具挑战性的三维视觉任务，在获得比此前最优秀模型 MinkowskiNet [12] 更高性能的同时，我们的方法 SPVNAS [68] 可节省其计算量 7.8×，在 GTX1080Ti 上获得 2.7× 实际加速。SPVNAS 还设计了历史上第一个纯三维的，达到 10FPS（雷达实时）的雷达点云语义分割模型，比此前基于二维球面投影的雷达点云语义分割模型具有更快的速度和 10% 以上的 IoU 提升。

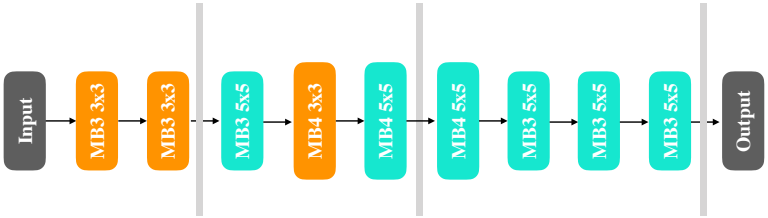
[1]注：雷达的采集频率往往是 10 帧每秒，所以达到 10 帧每秒的模型就是实时模型。



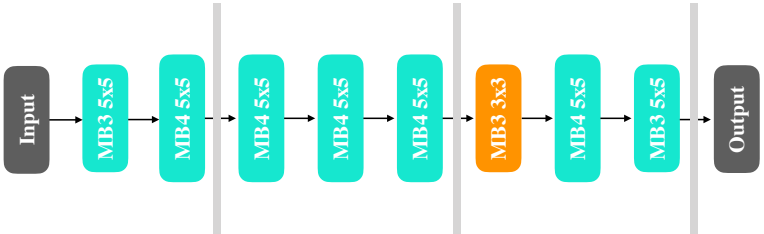
a) 基线网络结构



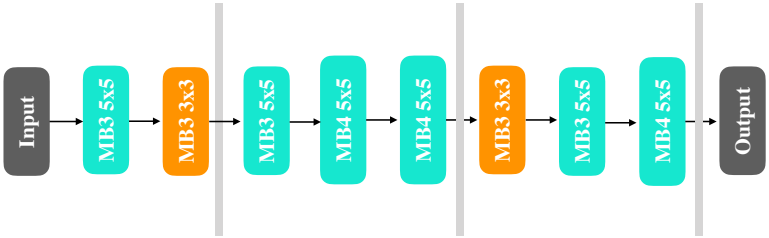
b) 为 NVIDIA GTX1060 专门设计的网络结构



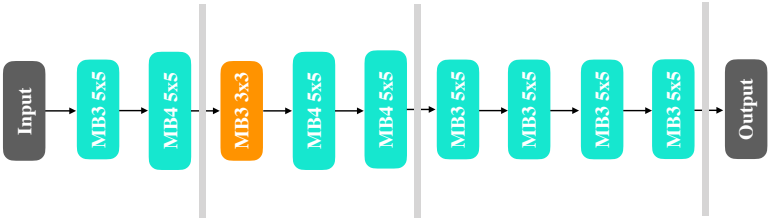
c) 为 NVIDIA GTX1080Ti 专门设计的网络结构



d) 为 NVIDIA RTX2080Ti 专门设计的网络结构



e) 为 NVIDIA Titan RTX 专门设计的网络结构



f) 为 AMD Ryzen7 (CPU) 专门设计的网络结构

图 4–6 我们提出的改进的 One-Shot 神经网络结构搜索方法，SegNAS 在五个硬件平台上专门设计的网络结构可视化。

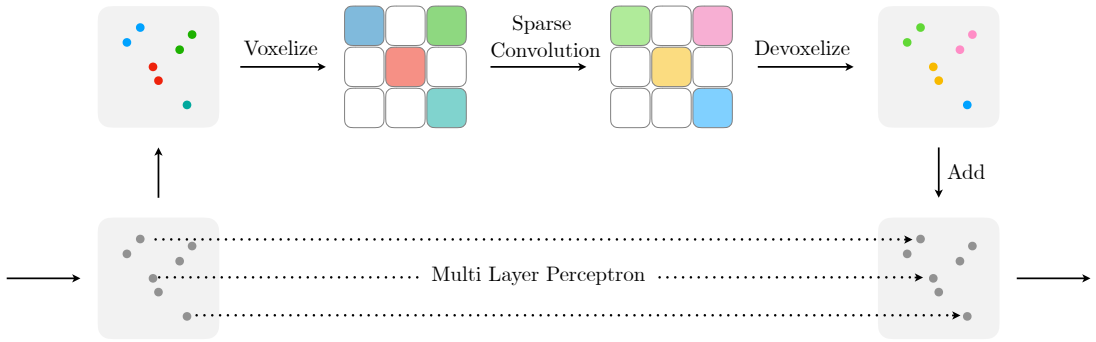


图 4–7 我们提出的三维视觉中的一种新的原子操作，稀疏 Point-Voxel 卷积（SPVConv [68]）。

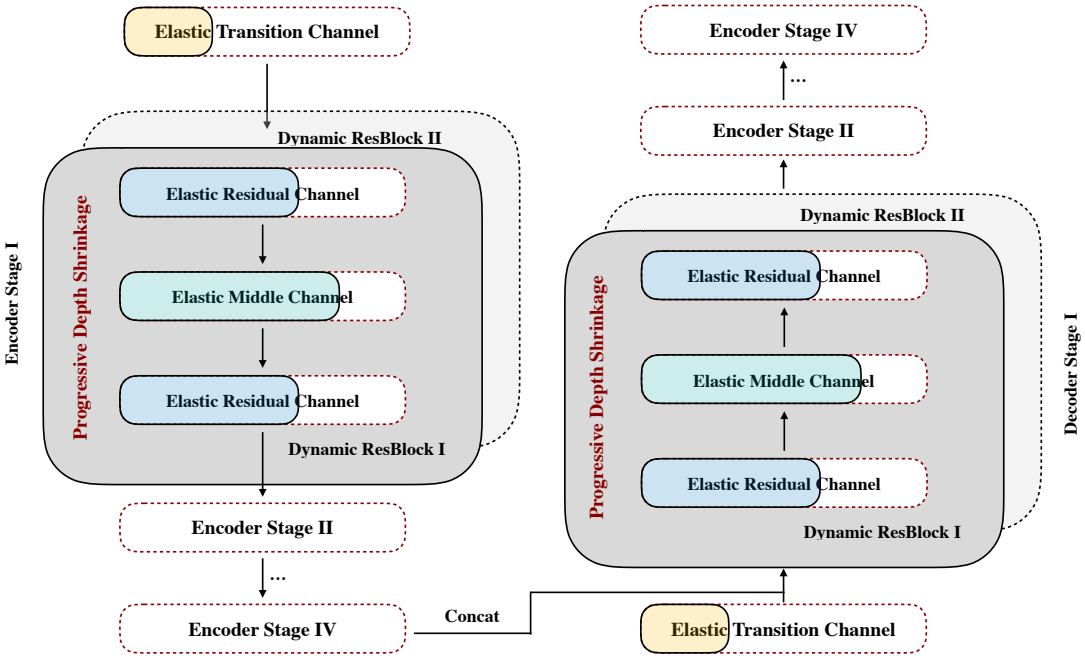
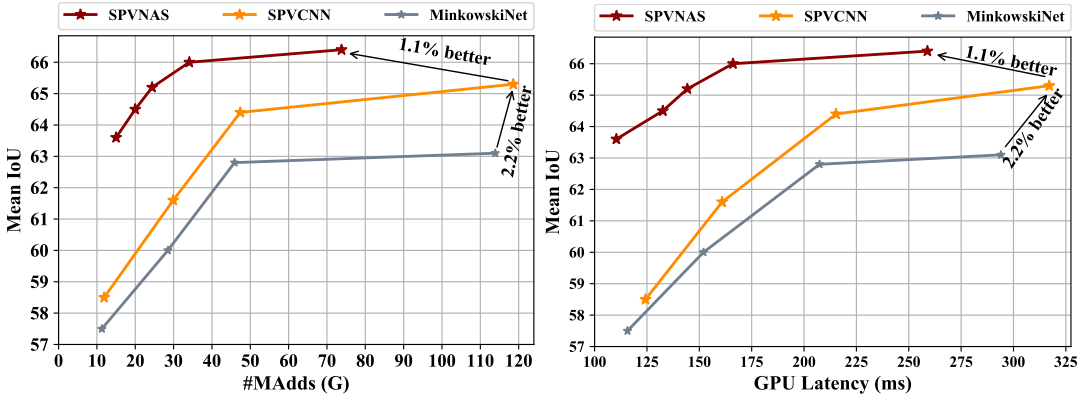


图 4–8 所提出的用于三维视觉的 SPVNAS [68] 算法的搜索空间。



a) FLOPs *v.s.* IoU 权衡曲线　　b) Latency *v.s.* IoU 权衡曲线

图 4–9 SPVNAS [68] 在各个延迟/计算量下均取得了远超手动设计的 SPVCNN [68] 和 MinkowskiNets [12] 的性能。

全文总结

本文中，我们研究了高效神经网络模型的设计问题，尤其是其在图像语义分割任务当中的应用。我们分析了常见图像语义分割模型难以达到实时执行和较高准确率的原因：高分辨率处的操作耗时过长，以及所使用的图像分割模型并不是为图像语义分割的分辨率所设计，并结合前人工作改进了直接从图像分类模型迁移的图像语义分割模型，该模型可以在 1024×2048 的极高分辨率下达到实时推理，且在精确度上大幅提升了之前直接迁移自图像分类模型的基线。

而后，我们提出一种改进的 One-Shot 神经网络搜索方法，SegNAS，来为多个硬件平台自动定制高效的神经网络模型。相较于朴素的 One-Shot 神经网络搜索，我们先后引入卷积权值共享、神经网络深度渐进收缩、硬件延迟直接约束的遗传算法搜索等改进，并亦在框架中加入对量化神经网络训练的支持。在上述框架下，我们在 GTX1060、GTX1080Ti、RTX2080Ti、Titan RTX 图形卡和 AMD Ryzen7 CPU 上定制语义分割模型，均取得了比改进的语义分割基线更好的 IoU 准确率和更快的推理速度（最好结果为 GTX1080Ti 定制的网络取得 **1.1% 的 IoU 提升，1.2 倍实测加速**）。我们在 2016 年就已十分寻常的 NVIDIA GTX1060 图形卡上达到**超过 43 帧每秒**的推理速度，这使得我们的模型可以高效地运行在现今大部分配置有英伟达图形卡的**笔记本电脑**上，而不需要在数据中心服务器上才能达到高效的推理。值得一提的是，我们的方法作为神经网络自动搜索算法，其难能可贵的特点是具有很低的网络设计代价。这得益于我们设计的超网络训练与神经网络结构搜索解耦合的范式，使得耗时极长的训练过程只需进行一次，此后多平台部署只需要执行多次以网络推断为主的搜索过程，而这要远比训练本身高效得多。令人印象深刻的是，我们的方法在一张 GTX1080Ti 图形卡上只需要 72 小时训练超网络、10 小时搜索就能设计一个新的、定制化的用于图像语义分割的神经网络结构，而此前谷歌公司提出的算法往往需要 40000 小时才能设计一个用于图像分类的网络模型。

我们亦展示了所提出的改进的 One-Shot 神经网络结构搜索方法，SegNAS，可以被迁移到三维计算机视觉的应用当中。我们介绍了我近期参与的工作 SPVNAS，利用十分类似于 SegNAS 的网络搜索流程设计了学术界第一个自动生成的三维深度学习模型，在极具挑战性的三维雷达点云语义分割任务上取得最先进的性能和最快的推理速度。这足以证明本文提出的方法具有极强的普适性和可泛化性。

参考文献

[1] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Juergen Gall. “SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences”. In: *ICCV*. 2019.

[2] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc V Le. “Understanding and Simplifying One-Shot Architecture Search”. In: *ICML*. 2018.

[3] Han Cai, Ligeng Zhu, and Song Han. “ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware”. In: *ICLR*. 2019.

[4] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. “Once for All: Train One Network and Specialize it for Efficient Deployment”. In: *ICLR*. 2020.

[5] Angel X. Chang et al. “ShapeNet: An Information-Rich 3D Model Repository”. In: *arXiv* (2015).

[6] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. “DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs.” In: *TPAMI* (2016).

[7] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. “Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation”. In: *ECCV*. 2018.

[8] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. “Rethinking Atrous Convolution for Semantic Image Segmentation”. In: *ICCV*. 2017.

[9] Tianqi Chen, Lianmin Zheng, Eddie Yan, Ziheng Jiang, Thierry Moreau, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. “Learning to Optimize Tensor Programs”. In: *NeurIPS*. 2018.

[10] Wuyang Chen, Xinyu Gong, Xianming Liu, Qian Zhang, Yuan Li, and Zhangyang Wang. “FasterSeg: Searching for Faster Real-Time Semantic Segmentation”. In: *ICLR*. 2020.

[11] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. “PACT: Parameterized Clipping Activation for Quantized Neural Networks”. In: *arXiv* (2018).

[12] Christopher Choy, JunYoung Gwak, and Silvio Savarese. “4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks”. In: *CVPR*. 2019.

[13] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. “The Cityscapes Dataset for Semantic Urban Scene Understanding”. In: *CVPR*. 2016.

[14] Matthiew Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. “Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1”. In: *arXiv* (2016).

[15] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. “ImageNet: A Large-Scale Hierarchical Image Database”. In: *CVPR*. 2009.

[16] Emily L. Denton, Wojciech Zaremba, Joan Bruna, Yann Lecun, and Rob Fergus. “Exploiting linear structure within convolutional networks for efficient evaluation”. In: *NeurIPS*. 2014.

[17] Brian Fulkerson, Andrea Vedaldi, and Stefano Soatto. “Class segmentation and object localization with superpixel neighborhoods”. In: *ICCV*. 2009.

[18] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V. Le. “NAS-FPN: Learning Scalable Feature Pyramid Architecture for Object Detection”. In: *CVPR*. 2019.

[19] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. “3D Semantic Segmentation With Submanifold Sparse Convolutional Networks”. In: *CVPR*. 2018.

[20] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. “Single Path One-Shot Neural Architecture Search with Uniform Sampling”. In: *arXiv* (2019).

[21] Song Han, Huizi Mao, and William J Dally. “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding”. In: *ICLR*. 2016.

[22] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. “EIE: Efficient Inference Engine on Compressed Deep Neural Network”. In: *ISCA*. 2016.

[23] Song Han, Jeff Pool, John Tran, and William J. Dally. “Learning both Weights and Connections for Efficient Neural Networks”. In: *NeurIPS*. 2015.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep Residual Learning for Image Recognition”. In: *CVPR*. 2016.

[25] Yihui He, Xiangyu Zhang, and Jian Sun. “Channel Pruning for Accelerating Very Deep Neural Networks”. In: *ICCV*. 2017.

[26] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. “AMC: AutoML for Model Compression and Acceleration on Mobile Devices”. In: *ECCV*. 2018.

[27] Andrew Howard et al. “Searching for MobileNetV3”. In: *ICCV*. 2019.

[28] Andrew G. Howard, Menglong Zhu, Bo Chen, Dimitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”. In: *arXiv* (2017).

[29] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. “SqueezeNet: AlexNet-Level Accuracy with 50x Fewer Parameters and < 0.5MB Model Size”. In: *arXiv* (2016).

[30] Alenxander Kirillov, Yuxin Wu, Kaiming He, and Ross Girshick. “PointRend: Image Segmentation as Rendering”. In: *CVPR*. 2020.

[31] Alexander Kirillov, Ross Girshick, Kaiming He, and Piotr Dollar. “Panoptic Feature Pyramid Networks”. In: *CVPR*. 2019.

[32] Philipp Krähenbühl and Vladlen Koltun. “Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials”. In: *NeurIPS*. 2011.

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *NeurIPS*. 2012.

[34] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. “PointCNN: Convolution on $\mathcal{X}$-Transformed Points”. In: *NeurIPS*. 2018.

[35] Darryl D. Lin, Sachin S. Talathi, and V.Sreekanth Annapureddy. “Fixed Point Quantization of Deep Convolutional Networks”. In: *ICLR*. 2016.

[36] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L. Yuille, and Li Fei-Fei. “Auto-DeepLab: Hierarchical Neural Architecture Search for Semantic Image Segmentation”. In: *CVPR*. 2019.

[37] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. “Progressive Neural Architecture Search”. In: *ECCV*. 2018.

[38] Haoxiao Liu, Karen Simonyan, and Yiming Yang. “DARTS: Differentiable Architecture Search”. In: *ICLR*. 2019.

[39] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. “MetaPruning: Meta Learning for Automatic Neural Network Channel Pruning”. In: *ICCV*. 2019.

[40] Zhijian Liu, Haotian Tang, Yujun Lin, and Song Han. “Point-Voxel CNN for Efficient 3D Deep Learning”. In: *NeurIPS*. 2019.

[41] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. “Learning Efficient Convolutional Networks through Network Slimming”. In: *ICCV*. 2017.

[42] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. “Rethinking the Value of Network Pruning”. In: *ICLR*. 2019.

[43] Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully convolutional networks for semantic segmentation”. In: *CVPR*. 2016.

[44] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. “ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design”. In: *ECCV*. 2018.

[45] Jiageng Mao, Xiaogang Wang, and Hongsheng Li. “Interpolated Convolutional Networks for 3D Point Cloud Understanding”. In: *ICCV*. 2019.

[46] Daniel Maturana and Sebastian Scherer. “VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition”. In: *IROS*. 2015.

[47] Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. “Data-Free Quantization Through Weight Equalization and Bias Correction”. In: *ICCV*. 2019.

[48] Wei Niu, Xiaolong Ma, Sheng Lin, Shihao Wang, Xuehai Qian, Xue Lin, Yanzhi Wang, and Bin Ren. “PatDNN: Achieving Real-Time DNN Execution on Mobile Devices with Pattern-based Weight Pruning”. In: *ASPLOS*. 2020.

[49] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *NeurIPS*. 2019.

[50] Matthieu Paul, Christoph Mayer, Luc Van Gool, and Radu Timofte. “Efficient Video Semantic Segmentation with Labels Propagation and Refinement”. In: *WACV*. 2020.

[51] Rudra P K Poudel, Ujwal Bonde, Stephan Liwicki, and Christopher Zach. “ContextNet: Exploring Context and Detail for Semantic Segmentation in Real-time”. In: *BMVC*. 2018.

[52] Rudra PK Poudel, Stephan Liwicki, and Roberto Cipolla. “FastSCNN: Fast Semantic Segmentation Network”. In: *arXiv* (2019).

[53] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation”. In: *CVPR*. 2017.

[54] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. “PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space”. In: *NeurIPS*. 2017.

[55] Charles Ruizhongtai Qi, Hao Su, Matthias Niessner, Angela Dai, Mengyuan Yan, and Leonidas J. Guibas. “Volumetric and Multi-View CNNs for Object Classification on 3D Data”. In: *CVPR*. 2016.

[56] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. “OctNet: Learning Deep 3D Representations at High Resolutions”. In: *CVPR*. 2017.

[57] Eduardo Romera, Jose M. Alvarez, Luis M. Bergasa, and Roberto Arroyo. “ERFNet: Efficient Residual Factorized ConvNet for Real-Time Semantic Segmentation”. In: *IEEE Transactions on Intelligent Transportation Systems* (2018).

[58] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *MICCAI*. 2015.

[59] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. “MobileNetV2: Inverted Residuals and Linear Bottlenecks”. In: *CVPR*. 2018.

[60] Jianbo Shi and Jitendra Malik. “Normalized Cuts and Image Segmentation”. In: *TPAMI* (2000).

[61] Jamie Shotton, Matthew Johnson, and Roberto Cipolla. “Semantic texton forests for image categorization and segmentation”. In: *CVPR*. 2008.

[62] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *ICLR*. 2015.

[63] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. “Single-Path NAS: Designing Hardware-Efficient ConvNets in less than 4 Hours”. In: *arXiv* (2019).

[64] Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, and Jan Kautz. “SPLATNet: Sparse Lattice Networks for Point Cloud Processing”. In: *CVPR*. 2018.

[65] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. “Deep High-Resolution Representation Learning for Human Pose Estimation”. In: *CVPR*. 2019.

[66] Mingxing Tan and Quoc V Le. “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”. In: *ICML*. 2019.

[67] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. “MnasNet: Platform-Aware Neural Architecture Search for Mobile”. In: *CVPR*. 2019.

[68] Haotian Tang, Zhijian Liu, Shengyu Zhao, Yujun Lin, Ji Lin, Hanrui Wang, and Song Han. “Searching Efficient 3D Architectures with Sparse Point-Voxel Convolution”. In: *arXiv* (2020).

[69] Maxim Tatarchenko, Jaesik Park, Vladlen Koltun, and Qian-Yi Zhou. “Tangent Convolutions for Dense Prediction in 3D”. In: *CVPR*. 2018.

[70] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. “KPConv: Flexible and Deformable Convolution for Point Clouds”. In: *ICCV*. 2019.

[71] Zhuowen Tu and Xiang Bai. “Auto-context and its application to highlevel vision tasks and 3d brain image segmentation”. In: *TPAMI* (2010).

[72] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. “HAQ: Hardware-Aware Automated Quantization with Mixed Precision”. In: *CVPR*. 2019.

[73] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. “Adaptive O-CNN: A Patch-based Deep Representation of 3D Shapes”. In: *SIGGRAPH Asia*. 2018.

[74] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. “O-CNN: Octree-based Convolutional Neural Networks for 3D Shape Analysis”. In: *SIGGRAPH*. 2017.

[75] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. “Dynamic Graph CNN for Learning on Point Clouds”. In: *SIGGRAPH*. 2019.

[76] Zongji Wang and Feng Lu. “VoxSegNet: Volumetric CNNs for Semantic Part Segmentation of 3D Shapes”. In: *TVCG* (2019).

[77] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. “Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search”. In: *CVPR*. 2019.

[78] Wenxuan Wu, Zhongang Qi, and Li Fuxin. “PointConv: Deep Convolutional Networks on 3D Point Clouds”. In: *CVPR*. 2019.

[79] Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. “SpiderCNN: Deep Learning on Point Sets with Parameterized Convolutional Filters”. In: *ECCV*. 2018.

[80] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. “Partial Channel Connections for Memory-Efficient Differentiable Architecture Search”. In: *ICLR*. 2020.

[81] Changqian Yu, Jingbo Wang, Chao Peng, Changxin Gao, Gang Yu, and Nong Sang. “BiSeNet: Bilateral Segmentation Network for Real-time Semantic Segmentation”. In: *ECCV*. 2018.

[82] Jiahui Yu and Thomas S. Huang. “Universally Slimmable Networks and Improved Training Techniques”. In: *ICCV*. 2019.

[83] Jiahui Yu, Lingjie Yang, Ning Xu, Jianchao Yang, and Thomas S. Huang. “Slimmable Neural Networks”. In: *ICLR*. 2019.

[84] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. “ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices”. In: *CVPR*. 2018.

[85] Hengshuang Zhao, Xiaojuan Qi, Xiaoyong Shen, Jianping Shi, and Jiaya Jia. “ICNet for Real-Time Semantic Segmentation on High-Resolution Images”. In: *ECCV*. 2018.

[86] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. “Pyramid Scene Parsing Network”. In: *CVPR*. 2017.

[87] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. “Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights”. In: *ICLR*. 2017.

[88] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. “DoReFaNet: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients”. In: *arXiv* (2016).

[89] Yin Zhou and Oncel Tuzel. “VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection”. In: *CVPR*. 2018.

[90] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. “Trained Ternary Quantization”. In: *ICLR*. 2017.

[91] Barret Zoph and Quoc V Le. “Neural Architecture Search with Reinforcement Learning”. In: *ICLR*. 2017.

[92] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. “Learning Transferable Architectures for Scalable Image Recognition”. In: *CVPR*. 2018.

致　谢

四年的本科生涯即将结束，我很荣幸在交大 120 周年校庆之时作为 “双甲子” 新生入校，并在四年之后并无遗憾地离开。在我四年的交大生活中，我有太多人和事需要感谢。

首先感谢我的三位本科室友。来自辽宁的温涵博，他是我们寝室的体育健将，在他的带领下我逐渐养成了每天夜跑的习惯，1000 米成绩较高中竟提升了 1 分钟。来自安徽的徐飞翔，他的为人处事之道使我获益良多，无数次遇到压力巨大、力不从心，都要感谢徐飞翔的开导，使我重振信心。来自北京的杨晨宇，四年以来一直鞭策着我不断前进。他踏实、沉稳的学习作风，充满创造力的科研思考，总让我感到大受启发。我很幸运在交大遇到最棒的室友，没有他们，就没有我今天的成绩。

我还要感谢在交大四年所有指导过我的老师和前辈。我的毕业论文导师卢宏涛教授，从 2017 年我大一时就开始指导我的研究。讨论班上，从最基础的机器学习知识传授，到前沿文献交流，卢宏涛教授指导我建立了最初始的科研世界观。他教会我如何关注最新的研究进展，教会我如何建立自己研究的品位，更重要的是他教会我如何从头到尾做一份完整的学术研究。我大三时请教过的老师卢策吾研究员，他是一名才华横溢的计算机视觉研究者，在三维计算机视觉领域有较深的造诣。感谢卢策吾老师，教会我跳出传统的思维去看问题，并敢于提出创造性的算法。我在腾讯公司的实习导师郜颖博士，感谢他带领我走进真正的工业界，学习如何将学术界的研究真正落地、解决实际问题。亦感谢他带领我体会到工业界一丝不苟、谨慎踏实的作风。我在麻省理工学院实习期间的导师韩松教授、带领我的学长刘志健博士，感谢他们带领我开展最高水平的研究，手把手地为我指明方向，甚至一点点纠正我刚开始实习时写得完全像实验报告的论文。在他们的帮助下，我第一次感受到不是追逐学术研究的前沿，而是自己就是学术研究的前沿的感觉，我也逐渐建立起了更高的学术追求和对自己更严格的要求。感谢韩老师和志健学长，使我的本科生涯经历真正的质变。

我要感谢所有和我合作学术论文以及一起参与讨论的前辈、同学，他们是刘志健博士（MIT），林宇鋆博士（MIT），赵晟宇同学（清华大学），王瀚锐博士（MIT），林己博士（MIT），赵一儒博士（上海交通大学），蔡涵博士（MIT）和朱力耕博士（MIT）；没有他们，我的任何一篇学术论文都不会取得成功，感谢他们。我还要感谢在我的学术论文投稿期间热心帮助我的其他研究者，他们是王鹏帅博士（微软亚洲研究院）、祁芮中台博士（Waymo 公司）、Hugues Thomas 博士（多伦多大学），Chris Choy 博士（英伟达公司）、史少帅博士（香港中文大学）。没有他们的帮助，我无法如此顺利地解决有关于他们论文的疑问。另外，感谢评阅过我的论文的匿名审稿人，是你们让我的每一篇论文都比初稿远远出色。

我亦要感谢在我的研究生申请当中帮助我的所有老师、学长。为我撰写推荐信的五位推荐人程帆教授、韩松教授、蒋力教授、卢宏涛教授、臧斌宇教授[1]，感谢你们不厌其烦地帮助我向各大学校强烈推荐录取我，使我最终收获自己满意的录取。我要感谢在我申请季中一直为我提供真知灼见的刘志健学长，他在我连续两个月没有拿到一封新的博士项目录取通知时不断鼓励我，使我走出低谷，最终结局圆满。

我还要感谢 IEEE 试点班这个项目和我优秀的同学们。感谢前项目主任王新兵教授从 2012 年以来一直为这个项目尽心尽力，对项目的每一位同学的发展关心备至，并在大一时就教会我们要出去看看、开拓视野。感谢在试点班中兴时期带领我们前进的现任项目主任

[1]按姓氏拼音排列。

程帆教授，我得以获得麻省理工学院的实习机会与他密切相关，我收获的博士录取的三分之二亦与程老师直接相关，十分感谢程老师对我一直以来的提携。感谢我的每一位同学，不论是和你们一起学习，甚至是和你们一起竞争，我都很快能受到启发、意识到自己的不足，常有“听君一席话，胜读十年书”之感。没有 IEEE 试点班就不会有今天的我，衷心希望这个项目在未来越办越好。

最后，最重要地，我要感谢我的家人。我要感谢我的父母一直以来对我的养育之恩。感谢我的母亲二十多年以来对我无微不至的照顾、关怀，建立我的三观，始终教会我在任何时候做一个正直诚实的人。感谢我的父亲为我提供的资源，并在我很小的时候就带我走进计算机的世界。也感谢一直爱我的爷爷奶奶、外公外婆，和其他的亲戚们，你们一直是我取得成功的最大动力。

攻读学士学位期间已发表或录用的论文

[1] Haotian Tang, Yiru Zhao and Hongtao Lu. Unsupervised Person Re-identification with Iterative Self-Supervised Domain Adaptation. In: *CVPRW*, 2019.

[2] Zhijian Liu*, Haotian Tang*, Yujun Lin and Song Han. Point-Voxel CNN for Efficient 3D Deep Learning. In: *NeurIPS*, 2019.

[3] Haotian Tang*, Zhijian Liu*, Shengyu Zhao, Yujun Lin, Ji Lin, Hanrui Wang and Song Han. Searching Efficient 3D Architectures with Sparse Point-Voxel Convolution. *In submission*, 2020.

EFFICIENT NEURAL NETWORK ARCHITECTURE DESIGN

Designing efficient neural network architectures is of great importance in modern applications such as autonomous driving, robotic navigation, and AR / VR. Previous research has made great effort on designing fast neural nets for image classification tasks through pruning, which zeros out weights with small magnitude to reduce memory consumption; quantization, which uses low-bit representation to store the weights such that both computation and memory occupancy will be reduced; and neural architecture search, which designs specialized neural networks for different tasks. However, comparing with image segmentation, image classification is a much easier task since it typically requires small input resolution, such as 224×224, and small models with less than 1 GFLOPs can easily run in real time on GPUs, CPUs, or even mobile devices. Differently, image semantic segmentation models often takes extremely high resolution input of 1024×2048, which makes computation an order of magnitude higher than classification models if the same architecture is applied. Therefore, it is usually very difficult for image segmentation models to run in real time on high-end GPUs such as NVIDIA GTX 1080Ti, not to mention on CPUs or even mobile devices. Unfortunately, image segmentation models are always deployed in latency-sensitive applications, such as autonomous driving. Such dilemma makes designing specialized, efficient neural network architectures for image segmentation extremely important in today's computer vision research.

In this research, we propose a novel pipeline to automatically design efficient neural networks for image semantic segmentation. Based on the observation on the suboptimality of directly transferring classification models to the segmentation tasks, we propose to modify the classifier backbone by downsampling more aggressively in the early stage and supplement low resolution feature map with high resolution bypass connection.

Furthermore, with such optimized image segmentation baseline, which already runs in real time and more than $2\times$ faster than the directly-transferred classifier counterpart, we design a novel, improved One-Shot Neural Architecture Search pipeline, named SegNAS, to automatically design neural network architectures for different hardware platforms. The SegNAS pipeline has significant improvement over the previous naive One-Shot NAS counterpart in both training and architecture searching. The naive One-Shot NAS pipeline goes through two phases to design a neural network architecture with highest performance: training a super network which contains all possible neural architectures in a predefined search space, and randomly searching within the super network to derive the best NN architecture. However, the original One-Shot NAS pipeline suffers from inscalability in the search space, incapability of imposing direct efficiency constraints and bad support for depth-variable networks, which makes it less practical to use naive One-Shot NAS to design hardware-efficient neural networks.

However, for SegNAS, we successfully solve all problems of naive One-Shot NAS without introducing any additional cost in both training and searching phase. For the training of the super network, our SegNAS differs from naive One-Shot NAS pipeline in terms of a novel weight-sharing mechanism and a progressive depth shrinkage technique to enable stable training of depth-variable

neural networks. Specifically, the weight-sharing mechanism assumes convolution kernels with different sizes, different output channels share the same set of weights, and such weights are dynamically cropped to satisfy given kernel size / output channel configurations in the runtime. Thanks to the weight-sharing strategy, all the sub networks within the super network now becomes equally well-trained, and therefore design spaces can be scaled up to include more operations. For progressive depth shrinkage, it is designed to solve the instability in naive One-Shot NAS training with stochastic depth shrinkage. Stochastic depth shrinkage makes each choice block in One-Shot NAS connecting to multiple different blocks, which leads to difficulty in optimization. Worse still, the optimization of the sub network with largest depth only goes through $\frac{1}{2^N}$ of the total training schedule, with N to be the number of choice blocks, which further makes the training of larger sub networks very challenging. Fortunately, our proposed progressive depth shrinkage strategy solves both problems: all the choice blocks are now connected to only its immediate predecessor, and the full-depth network is optimized for at least the entire training schedule of a single network. Thanks to progressive depth shrinkage, the super network with support for variable network depth can be trained to convergence stably and efficiently. We also support quantization-aware neural network training in the super network optimization phase, such that we can obtain quantized supernets which are helpful for architecture search on embedded devices or FPGA accelerators.

We also propose direct resource constraint in the searching phase of SegNAS, which solves the important drawback of naive One-Shot NAS that only random search is performed in the network discovery phase. Specifically, we propose to build a hardware latency lookup table to directly model the behavior of different hardware platforms including GPUs and CPUs. We also modify the original random architecture search mechanism to evolutionary architecture search, for the sake of better sample efficiency. The latency lookup table is built based on the assumption that latency of the entire network is the sum of latencies of its building blocks, which is verified through thorough experiments. Such hardware latency table successfully reduces the cost of latency measurement in the search phase from $\mathcal{O}(N)$ to $\mathcal{O}(1)$, which makes it very convenient to impose hardware latency constraint as a criterion to filter the population in evolutionary architecture search.

Thanks to the aforementioned (a) **optimized image segmentation baseline** from directly transferred classifiers (b) a **novel SegNAS pipeline** which optimizes both the super network training phase and the neural architecture search phase, and enables direct, hardware-in-the-loop neural architecture design, we automatically design five hardware-specialized efficient neural network models for image semantic segmentation on the Cityscapes benchmark. Thanks to SegNAS, consistent improvement is obtained over the already successful baseline architecture described in (a). Our specialized model improves GTX 1060 speed from 36.9 FPS to 43.3 FPS and IoU by 0.4%; improves GTX 1080Ti FPS from 76.3 to 89.3, and IoU by 1.1%; improves RTX 2080Ti speed from 151.5 FPS to 169.5 FPS, and IoU by 0.3%; improves Titan RTX speed from 163.9 FPS to 185.2 FPS, and IoU by 0.3%; improves latency on AMD Ryzen 7 from 358.0 ms to 328.9 ms, while also improves IoU by 0.8%. We also discover some interesting behavior of specialized neural networks on different platforms, such as the preference of using large convolution kernel size (like 5×5 instead of 3×3), and the **wider and shallower** designing trend on hardware devices with stronger computation capacity, such as the GPUs with NVIDIA Turing architecture. We also observe that specialized neural net architecture on one hardware platform is typically suboptimal on the other one, which further demonstrates the

importance and effectiveness of specializing neural network architectures for hardware platforms.

With automated mixed-precision quantization policy search, our quantized SegNAS (SegQNAS) outperforms the uniform 8-bit quantization baseline by **0.8% IoU** with **1.8× BitOps saving**. After further scaling down the BitOps constraint by 1.2×, we are still capable of keeping the same accuracy as the 8-bit quantization baseline, while achieving a BitOps saving of **2.2**×. We believe that it is possible to deploy SegQNAS models onto state-of-the-art mixed precision computer architecture, such as BISMO and Bit Fusion.

In this research, we also present the exciting transfer of SegNAS to automatically designing neural networks for 3D semantic segmentation tasks, which is far more challenging and understudied than its 2D counterpart. The proposed SPVNAS pipeline, who is the first ever work which explores neural architecture search for 3D, and whose neural architecture search part is mainly derived from SegNAS, achieveing **3.3%** better IoU comparing with previous state-of-the-art method with significantly improved efficiency. SPVNAS pipeline also succesfully designs the first ever 3D deep learning architecture which runs in real time for LiDAR point cloud segmentation without 2D spherical projection. While running faster than its spherical projection counterpart, the network found by SPVNAS achieves **10.4%** better IoU, which demonstrates the clear advantage of pure 3D methods in 3D modeling comparing with suboptimal 2D solutions.

In conclusion, in this paper, we propose a way to transfer and optimize image classification models for the target of real-time image semantic segmentation, and developed a novel SegNAS pipeline, which is a significant improvement upon the naive One-Shot NAS, to automatically design efficient neural networks under direct hardware feedback. The proposed method designs faster and better specialized neural networks on five hardware platforms, and is proved to be transferrable to more challenging 3D deep learning tasks.