Partial Order Pruning: for Best Speed/Accuracy Trade-off in Neural Architecture Search

Xin Li[1]   Yiming Zhou[1,3]   Zheng Pan[1]   Jiashi Feng[2]

[1] UISEE Technology Inc.

[2] Department of ECE, National University of Singapore

[3] National Key Lab. of Communications, UESTC

{xin.li, yiming.zhou, zheng.pan}@uisee.com   elefjia@nus.edu.sg

Abstract

Achieving good speed and accuracy trade-off on a target platform is very important in deploying deep neural networks in real world scenarios. However, most existing automatic architecture search approaches only concentrate on high performance. In this work, we propose an algorithm that can offer better speed/accuracy trade-off of searched networks, which is termed "Partial Order Pruning". It prunes the architecture search space with a partial order assumption to automatically search for the architectures with the best speed and accuracy trade-off. Our algorithm explicitly takes profile information about the inference speed on the target platform into consideration. With the proposed algorithm, we present several Dongfeng (DF) networks that provide high accuracy and fast inference speed on various application GPU platforms. By further searching decoder architectures, our DF-Seg real-time segmentation networks yield state-of-the-art speed/accuracy trade-off on both the target embedded device and the high-end GPU.

1. Introduction

Deploying deep convolutional neural networks (CNNs) on real-world embedded devices is attracting increasing research interest. Different from high-end GPUs, these devices usually offer rather limited computation capacity, leading to low efficiency when deploying popular high accuracy CNN models [12, 4] on them. Despite the considerable efforts on accelerating inference of CNNs such as pruning [11], quantization [29] and factorization [17], fast inference speed[1] is usually achieved at the cost of degraded performance [22, 15]. In this paper, we address such a practical problem: Given a target platform, what is the best speed/accuracy trade-off boundary curve by varying CNN architecture? Or more specifically, we aim to answer two
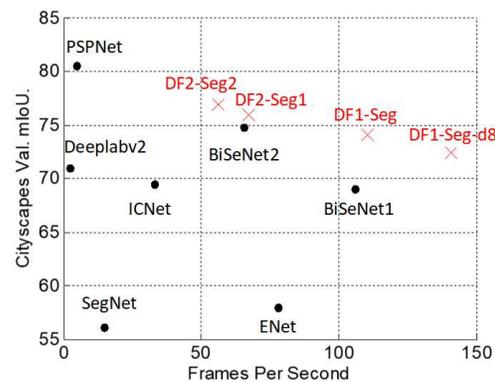

Figure 1. Speed (frames per second) and segmentation accuracy (mIoU$_{class}$) comparison on the Cityscapes [6] validataion set. DF-Seg networks yield state-of-the-art speed/accuracy trade-off. The compared models include PSPNet [34], Deeplabv2 [4], ENet [22], SegNet [1], ICNet [33], BiSeNet [31] and our DF-Seg networks.

questions: 1) Given the maximum acceptable latency, what is the best accuracy one can get? 2) To meet certain accuracy requirements, what is the lowest inference latency one can expect?

Some existing works manually design high accuracy network architectures [33, 31, 13, 25]. They usually adopt an indirect metric, *i.e.* FLOP, to estimate the network complexity, but the FLOP count does not truly reveal the actual inference speed. For example, for a 3×3 convolution on Nvidia GPUs which is highly optimized in terms of both hardware and software design [16], one can assume it is 9 times slower than a 1×1 convolution on GPUs since it has 9 times more FLOPs, which is not true actually. Besides, another important factor that affects the inference speed, the memory access, is not covered by measuring FLOPs. Considering the diversities of hardware and software, it is almost impossible to find one single architecture that is optimal for all the platforms.

Some other works attempt to automatically search for the optimal network architecture [36, 23, 19], but they also rely

---

[1] Inference speed is measured by inference latency, which is defined as inference time with batch size 1 in a CNN.

on FLOPs to estimate the network complexity and do not take into account the discrepancy of this metric with the actual inference speed and also the target platforms. Despite a few works [8, 2, 28] consider the actual inference speed on target platforms, they search the architecture in each individual building block and keep fixed the overall architecture, *i.e.* depth and width.

In this paper, we develop an efficient architecture search algorithm that automatically selects the networks that offer better speed/accuracy trade-off on a target platform. The proposed algorithm is termed "Partial Order Pruning", with which some candidates that fail to give better speed/accuracy trade-off are filtered out at early stages of the architecture searching process based on a partial order assumption (see Section 3.3 for details). For example, a wider network cannot be more efficient than a narrower one with the same depth, thus accordingly some wider ones are discarded. By pruning the search space in this way, our algorithm is forced to concentrate on those architectures that are more likely to lift the boundary of speed/accuracy trade-off.

The proposed "Partial Order Pruning" algorithm differs from previous neural architecture search algorithms in three aspects. Firstly, it explicitly takes platform characteristics into consideration. Secondly, it balances the width and depth of the overall architecture, instead of searching for complicated building blocks. Thirdly, it employs a partial order assumption and a cutting plane algorithm to accelerate searching, instead of using reinforcement learning, evolutionary algorithms or gradient-based algorithms.

With the proposed algorithm, we are able to obtain a set of networks that provide better accuracy and faster inference speed on a target platform, which we call Dongfeng (DF) networks. We apply our algorithm to searching decoder architectures in semantic segmentation and gain a set of DF-Seg networks. Figure 1 shows a comparison of our DF-Seg networks and other methods. It can be seen that our segmentation networks achieve new state-of-the-art in real-time urban scene parsing tasks.

To sum up, we make following contributions to network architecture search:

- We are among the first to investigate the problem of balancing speed and accuracy of network architectures for network architecture search. By pruning the search space with a partial order assumption, our "Partial Order Pruning" algorithm can efficiently lift the boundary of speed/accuracy trade-off.

- We present several DF networks that provide both high accuracy and fast inference speed on target embedded device TX2. The accuracy of our DF1/DF2A networks exceeds ResNet18/50 on ImageNet validation set, but the inference latency is 43% and 39% lower, respectively.

- We apply the proposed algorithm to searching decoder architectures for a segmentation network. Together with DF backbone networks, we achieve new state-of-the-art in real-time segmentation on both high-end GPUs and target embedded device TX2. On GTX 1080Ti, our DF1-Seg network achieves 106.4 FPS at resolution $1024{\times}2048$ with mIoU$_\text{class}$ 74.1%. On TX2, our DF1-Seg network achieves 21.8 FPS at resolution 1280×720, *i.e.* 720p.

2. Related Work

Efficient Network Design   Group convolution plays a key role in current efficient CNN architecture design [20, 13, 25]. MobileNet_V2 [25] adopts an inverted residual module that uses group convolutions to reduce the FLOPs during inference. ShuffleNet [32] uses pointwise group convolution and channel shuffle operation to reduce FLOPs while maintaining accuracy. [20] points out that there is a discrepancy between indirect metric (FLOPs) and direct metric (inference speed), and proposes four guidelines for efficient network design. These works design a single architecture without considering the target platform while our algorithm explicitly takes platform characteristics into consideration.

Neural Architecture Search   Automatic network architecture search is often tackled with either reinforcement learning [36, 35] or evolutionary algorithms [23, 24]. They require huge computational resources, and the obtained networks are relatively slower than manually designed ones [20, 8], even with comparable FLOPs. More recently, several gradient-based algorithms [19, 28, 2, 9] are proposed to reduce the architecture search cost. Besides, a few works [8, 28, 2] also take platform-related objectives into consideration in architecture search. Although their goal is somewhat similar to ours, our work differs in that we pursue the balance of width and depth of a network, instead of searching the architecture in each individual block.

Real-time Semantic Segmentation   Most semantic segmentation methods [34, 3, 5] aim at high performance but with relatively slow inference speed. For fast semantic segmentation, early works [22, 1] employ relatively shallower backbone networks and lower image resolution, offering fast inference speed but poorer accuracy. More recently, ICNet [33] uses the image cascade to speed up inference, in which pre-trained deep CNNs are only applied to the images with lowest resolution. BiSeNet [31] employs a context path to obtain a sufficient receptive field, and an additional spatial path with a small stride to preserve spatial information. None of them attempts to accelerate inference by improving the backbone network, or considers the characteristics of target platforms. Comparatively, our algorithm explicitly takes platform characteristics into consideration, and aims at better speed/accuracy trade-off in both backbone network and decoder network.
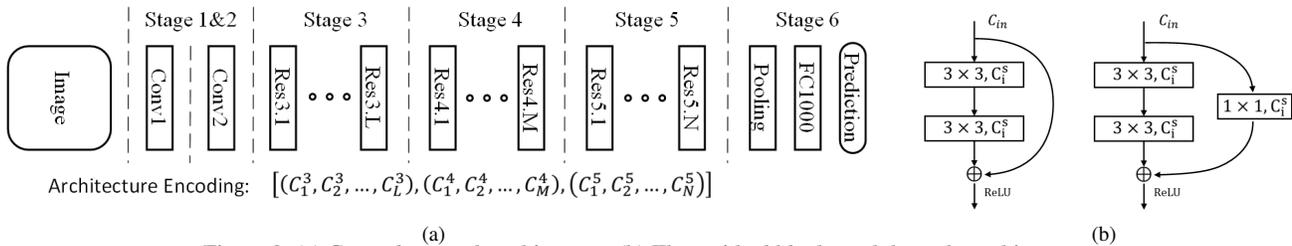
Figure 2. (a) General network architecture. (b) The residual block used throughout this paper.

Model Acceleration Some researchers try to accelerate inference of a pre-trained network via quantization [29], pruning [11], factorization [17], *etc.* For example, NetAdapt [30] automatically adapts a pre-trained CNN to a mobile platform given a resource budget. Compared with them, we try to balance the width and depth of the overall architecture.

3. Partial Order Pruning

3.1. Search Space

We provide a general network architecture in our search space, as shown in Figure 2(a). It consists of 6 stages to perform classification from input images. Stages 1∼5 down-sample the spatial resolution of the input tensor with a stride of 2, and stage 6 produces the final prediction with a global average pooling and a fully connected layer. Stages 1&2 extract common low-level features on large tensor size, which brings heavy computation burden. In pursuit of an efficient network, we only use one convolution layer in stage 1&2, *i.e.* $Conv1$ and $Conv2$. We empirically find this is enough for achieving good accuracy. For stages 3, 4, 5, each consists of L, M, N residual blocks, where L, M ,N are integers, *i.e.* $L, M, N \in \mathbb{N}$. Different settings of L/M/N lead to different network depths. The width (number of channels) of the i-th residual block in stage s is denoted as C_i^s. Therefore, an architecture can be encoded as shown in Figure 2(a). In practice, we restrict $C_i^s \in \{64, 128, 256, 512, 1024\}$. We empirically restrict the width of a block to be no narrower than its preceding blocks. Throughout this paper, we use the basic residual block proposed in [12] if not mentioned otherwise. As shown in Figure 2(b), the building block consists of two convolution layers and a shortcut connection. An additional projection layer is added if the size of input does not match the output tensor. All convolutional layers are followed with a batch normalization [14] layer and ReLU nonlinearity.

3.2. Latency Estimation

The set of all possible architectures, with different depths (number of blocks) and widths (number of channels per block), is denoted as $\mathbb{S}$ and usually referred to as the search space in neural architecture search [19, 36]. The latency of architectures in $\mathbb{S}$ can vary from very small to positive infinity. But we only care about architectures in a subspace $\widehat{\mathbb{S}} \subset \mathbb{S}$, which provide latency in the range $[T_{min}, T_{max}]$.
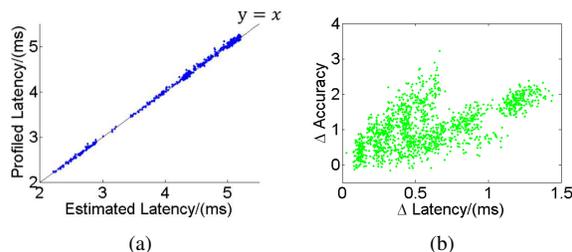

Figure 3. Preliminary experiments. (a) Our latency estimation is highly close to the actual profiled latency. (b) We experimentally verified that partial order assumption is generally true for efficient architectures of our concern.

We employ the profiler provided by TensorRT library to obtain layer-wise latency of a network. We empirically find that a block with a specific configuration (*i.e.* input/output tensor size) always consumes the same latency. Thus we can construct a look-up table $Latency(c_i, h_i, w_i, c_o, h_o, w_o)$ providing latency of each block configuration, where c_i/c_o is the number of channels in input/output tensor, and $h_i/w_i/h_o/w_o$ is the corresponding spatial size. For example, $Latency(32, 112, 112, 64, 56, 56) = 0.143$ms on TX2. By simply summing up the latency of all blocks, we can efficiently estimate the latency $Lat(x)$ of an architecture $x \in \mathbb{S}$. In Figure 3(a), we compare the estimated latency with the profiled latency. It shows our latency estimation is highly close to the actual profiled latency. All architectures with latency ranging $[T_{min}, T_{max}]$ form the subspace $\widehat{\mathbb{S}}$. This subspace construction significantly narrows down our search space, and hence accelerates the architecture selection.

3.3. Partial Order Assumption

A partial order is a binary relation defined over a set. It means that for certain pairs of elements (x, y) in the set, one of the elements x precedes the other y in the ordering, denoted with $x \prec y$. Here "partial" indicates that not every pair of elements needs to be comparable.

We find that there is a partial order relation among architectures in our search space. In Figure 4, we follow the architecture encoding in Figure 2(a), and illustrate the partial order relation among architectures. As explained in Section 3.2, $\widehat{\mathbb{S}}$ is a set that contains all architectures in which we are interested. Let $x, y \in \widehat{\mathbb{S}}$ denote two elements in the set $\widehat{\mathbb{S}}$. If x is shallower than y but they are with the same
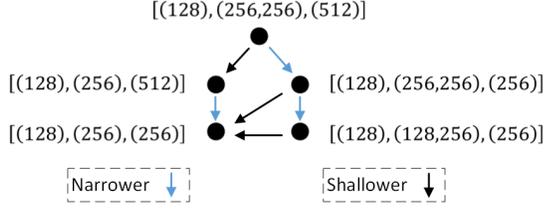
Figure 4. Partial order relations among architectures. An architecture (*e.g.* $[(128),(256),(256)]$) may be narrower than another one with same depth (*e.g.* $[(128),(256),(512)]$), or shallower than another with same width (*e.g.* $[(128),(256,256),(256)]$).

width, or narrower than y with same depth, we can borrow the concept from the order theory, and say that x precedes y in the ordering, denoted as $x \prec y$. In the rest of this paper, we also call x a *precedent* of y if $x \prec y$. Let $Acc(x)$ and $Lat(x)$ denote the accuracy and latency of the architecture x. Then the partial order assumption of architectures can be summarized as

$$Lat(x) \leq Lat(y), Acc(x) \leq Acc(y), \qquad (1)$$

where $\forall x, y \in \widehat{\mathbb{S}}, x \prec y$. Formula (1) assumes that the latency and accuracy of an architecture are both higher than those of its precedents. This assumption may not hold for very deep networks that contain hundreds of layers [12], but it is generally true for the efficient architectures of our concern, *i.e.* $\widehat{\mathbb{S}}$, which is experimentally verified in this work. We find all comparable architecture pairs $(x, y), x \prec y$ in our trained architectures (Section 4.2), and compute the latency difference $\Delta Lat = Lat(y) - Lat(x)$ and accuracy difference $\Delta Acc = Acc(y) - Acc(x)$ in each pair. As shown in Figure 3(b), most points locate in the first quartile. This means the accuracy of the precedent x is lower, for almost all comparable pairs. We also notice that a few points locate in the second quartile, but the lower limit of ΔAcc. is -0.1%, which is negligible considering the randomness during training. The above experimental results validate the reasonableness of our partial order assumption. This assumption can be utilized to prune the architecture search space, and speed up the search process significantly.

3.4. Partial Order Pruning

Formally, the goal of our architecture searching algorithm is to obtain an architecture with highest accuracy within every small latency range $[T, T + \delta t]$:

$$\max_{x \in \mathbb{S}} Acc(x), s.t. Lat(x) \in [T, T + \delta t] \qquad (2)$$

where δt is a short time period such as 0.1ms. Instead of searching at every small latency range, we optimize within the entire latency range $[T_{min}, T_{max}]$. With our "Partial Order Pruning" algorithm, architecture searching at higher latency helps reduce the searching space at lower latency, and hence speeds up the overall searching process.

Algorithm 1 Partial Order Pruning

Initialize trained architecture set $D = \varnothing$
Initialize pruned architecture set $P = \varnothing$.
repeat
    Random select an architecture $x \in \widehat{\mathbb{S}} \setminus P$.
    Train x and obtain its Acc(x).
    $D \leftarrow D \cup \{x\}$.
    for all $w \in D$ **do**
        $y_w = \arg \min_{y \in D} Lat(y), s.t. Acc(y) \geq Acc(w)$
        $\triangle P_w = \{m \in \widehat{S} | m \prec w, Lat(m) \geq Lat(y_w)\}$
        $P \leftarrow P \cup (\cup_w \triangle P_w)$
    end for
    $B(D) = \{x \in D | \forall m \in D,$
        $Lat(m) \geq Lat(x) \text{ or } Acc(m) \leq Acc(x)\}$
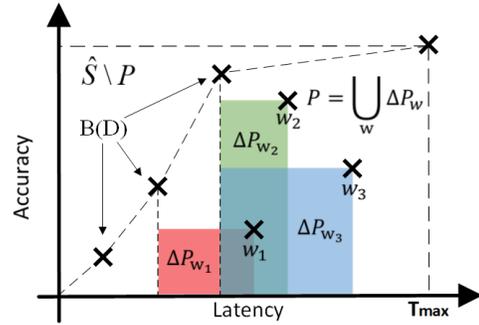**until** No change to B(D) for several iterations.

Figure 5. We construct pruned search space P with partial order assumption, and prune the search space to be $\widehat{S} \setminus P$. Architectures in $B(D)$ form the boundary for speed/accuracy trade-off we have achieved. (Best viewed in color).

We use a cutting plane algorithm to optimize the combinational optimization problem in Formula (2). Algorithm 1 summarizes the pipeline of our algorithm. D is a set containing all trained architectures, and is initialized as empty. P denotes the search space pruned from $\widehat{\mathbb{S}}$. Each time we train a new architecture $x \in \widehat{\mathbb{S}} \setminus P$ and obtain its accuracy $Acc(x)$, we are able to update the pruned search space P. Figure 5 shows how to construct P with the aforementioned partial order assumption. For each trained architecture $w \in D$, we find the fastest architecture $y_w \in D$ that provides better accuracy:

$$y_w \leftarrow \arg \min_{y \in D} Lat(y), s.t. Acc(y) \geq Acc(w). \qquad (3)$$

If no y_w is found that satisfies the condition, we continue to process the next w. Let $\triangle P_w$ denote the precedents of w with latency higher than y_w, *i.e.*

$$\triangle P_w = \{m \in \widehat{S} | m \prec w, Lat(m) \geq Lat(y_w)\}. \qquad (4)$$

Based on the partial order assumption, a precedent m has lower latency and accuracy, *i.e.* $Acc(m) \leq Acc(w)$. Therefore, even though we do not actually train m, we can assume

$$\forall m \in \triangle P_w, Acc(m) \leq Acc(y_w). \qquad (5)$$

In Figure 5, for all $m \in \triangle P(w_i)$, $i \in \{1, 2, 3\}$, the $(Lat(m), Acc(m))$ shall locate in the corresponding shadow area. These architectures in $\triangle P_w$ are very unlikely to provide better speed/accuracy trade-off, and thus get pruned from the search space to avoid unnecessary training cost.

Given trained architectures D, $B(D)$ denotes the architectures that provide best speed/accuracy trade-off in trained architectures:

$$
\begin{aligned}
B(D) \quad = \quad & \{x \in D | \forall w \in D, Lat(w) \geq Lat(x); \\
& or \; Acc(w) \leq Acc(x)\}. \quad (6)
\end{aligned}
$$

Architectures in $B(D)$ form the boundary for speed/accuracy trade-off we can achieve on the target platform. Figure 5 shows $B(D)$ and the corresponding speed/accuracy trade-off boundary. Intuitively, no architecture in $D \setminus B(D)$ could obtain higher accuracy with lower latency. By pruning P from the search space $\widehat{\mathbb{S}}$, our algorithm speeds up the architecture search process, and lifts the boundary of speed/accuracy trade-off. We stop the search process if no change to the $B(D)$ happens for several iterations.

3.5. Decoder Design

With the proposed Algorithm 1, we are able to find backbone architectures that provide best speed/accuracy trade-off on the target platform. Given a backbone network, we build semantic segmentation networks as shown in Figure 6(a). Each stage in the backbone network down-samples the resolution by 2. The resolution of tensors in stage 5 is thus $1/32$ of the input image. We append a pyramid pooling module [34] after the output tensor of stage 5 to improve segmentation performance. These tensors are then processed by the decoder to produce final prediction.

We append a 1×1 convolution layer after stage 3/4/5 as a "Channel Controller" (CC). The channel controllers reduce the number of channels in the corresponding stage without changing its spatial resolution. The decoder fuses the tensors in different stages through the fusion nodes. The architecture of the fusion node is shown in Figure 6(b). A fusion node first projects a low resolution tensor from C_ℓ channels to C_h channels with a 1×1 convolution layer, and then upsamples it by 2. We concatenate the up-sampled tensor with a higher resolution tensor, and then process it with a 3×3 convolution layer, to fuse the expressive power of different backbone stages. We fuse the features from stage 3/4/5 and produce a $1/8$ resolution score map. The score map is then up-sampled by 8 to produce final per-pixel semantic segmentation prediction.

Let C^s, $s = 3, 4, 5$ denote the width of each CC. We heuristically set $C \in \{K, 32, 64, 128, 256, 512\}$, where K is the number of classes. Given a backbone network, different settings of channel controllers, *i.e.* $[C^3, C^4, C^5]$, lead
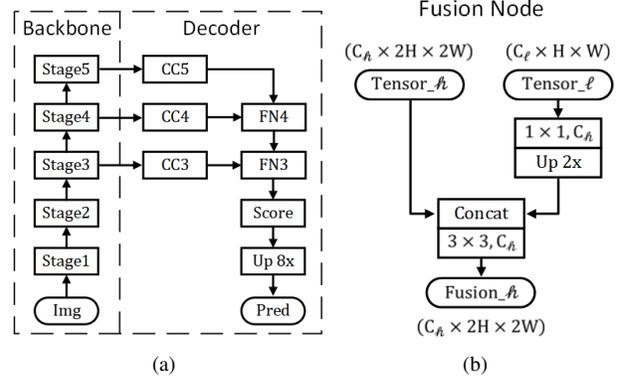

Figure 6. (a) Overall architecture of segmentation network. (b) Detailed architecture of the fusion node.

to different decoder architectures. All the possible decoder CC settings form the search space of the decoder architecture. Similar to backbone network architectures, we also apply a partial order assumption over the CC settings. That is, a narrower decoder is always more efficient and less accurate than a wider one. Therefore we can also employ the "Partial Order Pruning" algorithm to lift the speed/accuracy trade-off boundary in the decoder architecture search.

4. Experiment

4.1. Experimental Settings

We adopt two typical kinds of hardware that provide different computational power.

- Embedded device: We use Nvidia Jetson TX2 with an integrated 256-core Pascal GPU as the target embedded device. It provides considerable computational power with limited electrical power consumption.

- High-end GPU: We use Nvidia Geforce GTX 1080Ti that provides enormous computing power. We also use GTX Titan X (Maxwell) for fair comparison with previous methods.

We adopt two tools to measure inference speed. First, we employ the widely used high-performance CNN inference framework TensorRT-3.0.4. Second, for a fair comparison with ICNet [33], we use the time measure tool Caffe Time, and set the repeating number to 100 and take the average inference time for comparison. All experiments are performed under CUDA 9.0 and CUDNN V7.

We conduct experiments on two benchmark datasets. The ImageNet [7] is a large-scale image classification dataset, which contains over 1.2 million color images in the training set and 50k color images in the validation set. The Cityscapes [6] is a large benchmark dataset for urban scene parsing. It contains $5,000$ images with high quality pixel-level annotations, and is split to $2,975$ for training, 500 for validation, and $1,525$ for testing.
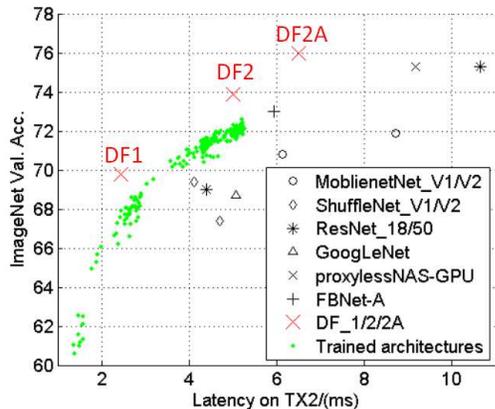
Figure 7. Comparison with other popular networks on TX2.

4.2. Backbone Architecture Search

In contrast to current architecture search algorithms that conduct architecture searching on small datasets, we directly conduct architecture searching on ImageNet. We use the SGD optimizer with the poly learning rate policy to train models. The power is set to 2, and the momentum is set to 0.9. We use a weight decay of 0.0001. The batch size is set to 2048. We employ random scaling and stretching for data augmentation to relieve overfitting. Following [10], we first train each network for 5 epochs with learning rate 0.1 as a warm up scheme, and then train for 80 epochs with an initial learning rate 0.8.

We conduct backbone architecture searching experiments on TX2 platform. During searching, we evaluate the single crop Top-1 accuracy on ImageNet validation set and the inference latency at resolution 224×224. We are interested in the efficient architectures with latency falling in the range $[1ms, 5ms]$, and construct the search space $\mathbb{S}$ accordingly (Section 3.2). We conduct architecture search with Algorithm 1, and stop the search process when no remarkable boundary update is found during the search. The resulting speed/accuracy trade-off boundary is considered to be nearly optimal in our search space $\mathbb{S}$ on the target platform TX2. We train $\sim$ 200 networks in total, as shown in Figure 7. With the training configuration kept unchanged during architecture search, we train two representative network architectures with additional supervision [26] and more epochs, to further improve their accuracy. The resulting models are referred to as DF1, DF2. We further replace some of the building blocks in DF2 from basic block in Figure 2(b) to bottleneck block [12]. The resulting network is denoted as DF2A. Figure 7 and Table 1 give a comparison of our DF networks and popular models[2] on the target platform TX2. Table 2 shows detailed architectures of these three DF networks. Training with more sophisticated methods, *e.g.* dropout or label smoothing, may produce higher accuracy, which however is not the focus of

---

[2]We report latency with our re-implementation.

Model	Top1 Acc.	Latency (ms)	FLOPs
ShuffleNet_V2 [20]	69.4%	4.1	146M
ResNet-18 [12]	69.0%	4.4	1.8G
ShuffleNet_V1 [32]	67.4%	4.7	**140M**
GoogLeNet [26]	68.7%	5.1	1.43G
MobileNet_V1 [13]	70.8%	6.1	569M
MobileNet_V2 [25]	71.9%	8.7	300M
ResNet-50 [12]	75.3%	10.6	3.8G
FBNet-A [28]	73.0%	5.9	249M
ProxylessNAS-GPU [2]	75.1%	9.3	-
NASNet-A [36]	74.0%	20.7	564M
PNASNET-5 [18]	74.2%	27.6	588M
DF1	69.8%	**2.5**	746M
DF2	73.9%	5.0	1.77G
DF2A	**76.0%**	6.5	1.97G

Table 1. Comparison with other popular networks on TX2.

this paper.

Compared with ResNet-18 and GoogLeNet, our DF1 obtains a higher accuracy 69.8% but the inference latency is 43%, 51% lower than two baselines respectively. our DF2 has a similar latency but the accuracy is 4.9% and 5.2% higher than the baselines respectively. Furthermore, DF2A achieves a surpassing ResNet-50-level accuracy with a 39% lower latency. Note we use the same building blocks with ResNet-18/50. So we attribute the better speed/accuracy trade-off to the better balancing between depth and width in our architectures. Specifically, our DF1/DF2A are slimmer and deeper than ResNet-18/50 for obtaining the same accuracy.

MobileNet [13, 25] and ShuffleNet [32, 20] are state-of-the-art efficient networks that are designed for mobile applications. We also compare our DF networks to them on TX2 in Table 1 and Figure 7. It can be seen our DF1 achieves higher accuracy but lower inference latency. The MobileNet/ShuffleNet have less FLOPs but higher latency. This is because they have higher memory access cost. The total memory cost (*i.e.* intermediate features) for ShuffleNet_V2 and DF1 is $4.86M$ and $2.91M$ respectively. This also indicates the FLOPs may be inconsistent with latency on the target platform [27, 20]. Therefore, taking characteristics of target platform into consideration is necessary for achieving the best speed/accuracy trade-off.

We also compare our DF networks with other models searched by NAS methods [36, 18, 28, 2]. As shown in Table 1, NASNet [36] and PNASNet [18] have not taken latency into consideration, leading to higher latency. Comparing to FBNet [28] and ProxylessNAS [2], which also take target platform-related objectives into neural architecture search, our DF networks show better speed/accuracy trade-off. This can be explained as (a) DF networks are specifically searched for TX2 platform; (b) FBNet and Proxyless-NAS use an inverted bottleneck module, which brings more memory access cost; (c) FBNet and ProxylessNAS aim at

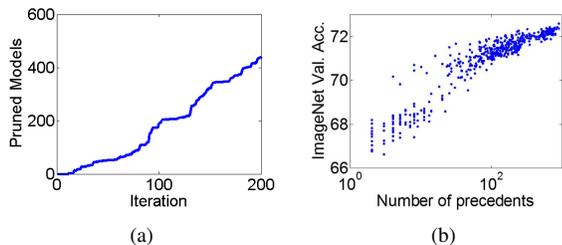| Stage	Layer	Output size	DF1	DF2	DF2A
1	Conv1	112×112	$3 \times 3, 32$	$3 \times 3, 32$	$3 \times 3, 32$
2	Conv2	56×56	$3 \times 3, 64$	$3 \times 3, 64$	$3 \times 3, 64$
3	Res3_x	28×28	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$ $\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 1$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$ $\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 1$
4	Res4_x	14×14	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 10$ $\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 1$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 10$ $\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 1$
5	Res5_x	7×7	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 3$ $\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 1$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 4$ $\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 4$ $\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 1024 \end{bmatrix} \times 2$
6	FC	1×1	Global Average Pooling, 1000-d FC, Softmax.		
	Depth		23	43	60

Table 2. Detailed architecture of DF networks.

Figure 8. (a) Number of pruned architectures. (b) We empirically find that accuracy of an architecture is correlated to the number of precedents.

searching for better building block architectures while we balance the width and depth of the overall architecture.

We then discuss the search efficiency of our proposed algorithm. Figure 8(a) shows the number of pruned architectures in the search process. We prune 438 architectures after training 200 architectures.Therefore, our POP algorithm accelerates the architecture search process for 2.2 times. Each model takes $5 \sim 7$ hours on a server with 8-GPUs. Training 200 architectures takes ~ 400 GPU days in total. The computational cost of our architecture for searching on ImageNet is lower than the building block architecture searching [36, 23] on CIFAR-10 by an order.

Based on our architecture search results, we make following observations. 1) Very quick down-sampling is preferred in early stages to obtain higher efficiency. We use 1 convolutional layer in each of stages 1&2, and are still able to achieve good accuracy. 2) Down-sampling with the convolutional layer is preferred to the pooling layer for obtaining higher accuracy. We only use 1 global average pooling at the end of the network. 3) We empirically find that the accuracy of a network is correlated to the number of its precedents, as shown in Figure 8(b). We assume that an ar-

chitecture with more precedents may have a better balance between depth and width.

4.3. Decoder Architecture Search

With our DF1/DF2 backbone networks, we conduct decoder architecture search experiments on two platforms, 1080Ti and TX2. The mIoU$_{class}$ at resolution 1024×2048 is taken as a metric of segmentation accuracy. The profiler of TensorRT is used to evaluate latency of segmentation networks. We evaluate latency at resolution 1024×2048 on 1080Ti, and 640×360 on TX2.

Figure 9 shows our decoder architecture search results. We select three segmentation networks DF1-Seg, DF2-Seg1, DF2-Seg2 from trained networks that provide good speed/accuracy trade-off on both TX2 and 1080Ti. The CC setting in the decoder of these three segmentation networks are $[19, 32, 128]$, $[19, 19, 32]$, $[19, 256, 512]$ respectively (19 is the number of classes). Few previous works have reported inference speed on TX2, thus we provide a comparison between our DF-Seg networks and other methods on 1080Ti, as shown in Table 3. We note [33] explicitly explains how they measure inference speed. Therefore, we add an additional column "FPS(Caffe)" in Table 3 for fair comparison. Inference speed in the "FPS(Caffe)" column is measured by Caffe Time on Titan X (Maxwell) at resolution 1024×2048.

Compared with BiSeNet1, our DF1-Seg achieves comparable inference speed, but the mIoU$_{class}$ on val set is 5.1% higher. Compared with BiSeNet2, DF1-Seg achieves comparable mIoU$_{class}$ on validation set, but the inference speed (FPS) is 1.68 times faster. We attribute the better speed/accuracy trade-off of DF1-Seg to its backbone network DF1. BiSeNet2 employs ResNet-18 as the backbone network. Our DF1 has a comparable accu-

(a) DF1 and TX2 (b) DF1 and GTX 1080Ti (c) DF2 and TX2 (d) DF2 and GTX 1080Ti
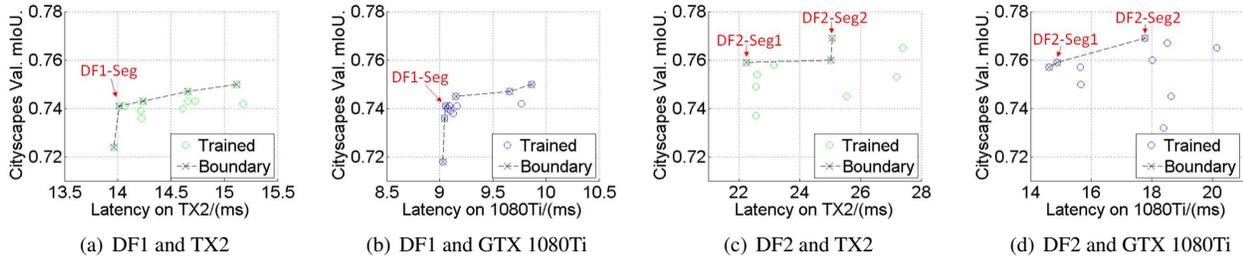
Figure 9. Speed/accuracy trade-off of decoder architecture search results with different backbone networks on different platforms. DF1-Seg/DF2-Seg are two segmentation networks that provide good speed/accuracy trade-off on both TX2 and GTX 1080Ti.

Method	mIoU$_{class}$ val	mIoU$_{class}$ test	FPS	FPS (Caffe)
SegNet [1]	-	56.1	-	-
ENet [22]	-	58.3	-	-
ICNet [33]	67.7	69.5	-	30.3
ESPNet [21]	-	60.3	110	-
BiSeNet1† [31]	69.0	68.4	105.8	-
BiSeNet2† [31]	74.8	74.7	65.5	-
DF1-Seg	74.1	73.0	106.4	30.7
DF2-Seg1	75.9	74.8	67.2	20.5
DF2-Seg2	**76.9**	**75.3**	56.3	17.7
DF1-Seg-d8	72.4	71.4	**136.9**	**40.2**

Table 3. Comparison with other real-time segmentation models on 1080Ti. † means FPS is evaluated at 1536×768.

racy with ResNet-18, but is 1.76 times faster (2.5ms vs 4.4ms), as shown in Table 1. Compared with ICNet [33], DF1-Seg achieves comparable inference speed, and the mIoU$_{class}$ is 3.5% higher on test set. Our DF2-Seg1 also achieves faster inference speed and better segmentation accuracy than BiSeNet2. With a wider decoder CC setting ([19, 256, 512]), our DF2-Seg2 achieves the best mIoU$_{class}$ 76.9% on validation set and 75.3% on test set at 56.3 FPS.

We obtain an even faster segmentation network by dropping the final up-sampling layer, and produce a prediction at $1/8$ of input resolution. The images to segment are then up-sampled by 8 times with nearest neighbor interpolation, which can be implemented very efficiently. We then obtain a DF1-Seg-d8 network that achieves 136.9 FPS on 1080Ti. The mIoU$_{class}$ on test set (71.4%) is still 1.9% and 3% better than ICNet (69.5%) and BiSeNet1 (68.4%) respectively.

For fair comparison with previous methods, we compare inference speed on Titan X (Maxwell) at different resolution, as shown in Table 4. Our DF1-Seg and DF1-Seg-d8 achieve 59.9 FPS and 75.9 FPS at resolution 1920×1080, *i.e.* 1080p. Based on the above experimental results, the DF-Seg networks achieve new state-of-the-art in real-time segmentation on high-end GPU, demonstrating better speed/accuracy trade-off is achieved.

Previous works [1, 22] mostly adopt TX1 to analyze their inference speed. In Table 5, we provide a detailed inference speed analysis on TX2. Our DF1-Seg/DF1-Seg-d8 achieve

Method	640×360 ms / FPS	1280×720 ms / FPS	1920×1080 ms / FPS
SegNet [1]	69/14.6	289/3.5	637/1.6
ENet [22]	7/135.4	21/46.8	46/21.6
BiSeNet-1 [31]	5/203.5	12/82.3	24/41.4
BiSeNet-2 [31]	8/129.4	21/47.9	43/23
DF1-Seg	3.65/274.0	8.24/121.4	16.70/59.9
DF2-Seg1	5.88/170.1	13.43/74.5	27.36/36.5
DF2-Seg2	6.57/152.2	15.10/66.2	31.08/32.2
DF1-Seg-d8	**3.25/307.7**	**6.62/151.1**	**13.18/75.9**

Table 4. Speed analysis on Titan X (Maxwell).

Method	480×320 ms / FPS	640×360 ms / FPS	1280×720 ms / FPS
ESPNet [21]	-/-	-/∼20	-/-
DF1-Seg	9.45/105.8	14.01/71.4	45.93/21.8
DF2-Seg1	15.32/65.3	22.25/44.9	73.32/13.6
DF2-Seg2	16.98/58.9	25.07/39.9	82.07/12.2
DF1-Seg-d8	**7.48/133.7**	**10.79/92.7**	**33.41/29.9**

Table 5. Speed analysis on TX2.

21.8 FPS and 29.9 FPS at resolution 1280×720, *i.e.* 720p.

5. Conclusion

We propose a network architecture search algorithm "Partial Order Pruning", which is able to lift the boundary of speed/accuracy trade-off of searched networks on the target platform. By utilizing a partial order assumption, it efficiently prunes the feasible architecture space to speed up the search process. We employ the proposed algorithm to search for both the backbone network and decoder network architectures. The searched DF backbone newtorks provide state-of-the-art speed/accuracy trade-off on target platforms. The searched DF-Seg networks achieve state-of-the-art speed/accuracy trade-off on both embedded devices and high-end GPUs.

Acknowledgement

References

[1] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *TPAMI*, (12):2481–2495, 2017. 1, 2, 8

[2] H. Cai, L. Zhu, and S. Han. ProxylessNAS: Direct neural architecture search on target task and hardware. *ICLR*, 2019. 2, 6

[3] L.-C. Chen, M. D. Collins, Y. Zhu, G. Papandreou, B. Zoph, F. Schroff, H. Adam, and J. Shlens. Searching for efficient multi-scale architectures for dense image prediction. *NIPS*, 2018. 2

[4] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *TPAMI*, 40(4):834–848, 2018. 1

[5] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. *ECCV*, 2018. 2

[6] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. *CVPR*, 2016. 1, 5

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 5

[8] J.-D. Dong, A.-C. Cheng, D.-C. Juan, W. Wei, and M. Sun. Dpp-net: Device-aware progressive search for pareto-optimal neural architectures. *ECCV*, 2018. 2

[9] X. Dong and Y. Yang. Searching for a robust neural architecture in four gpu hours. In *CVPR*, 2019. 2

[10] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch sgd: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017. 6

[11] Y. Guo, A. Yao, and Y. Chen. Dynamic network surgery for efficient dnns. *NIPS*, 2016. 1, 3

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CVPR*, 2016. 1, 3, 4, 6

[13] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 1, 2, 6

[14] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ICML*, 2015. 3

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 1

[16] A. Lavin and G. Scott. Fast algorithms for convolutional neural networks. *CVPR*, 2016. 1

[17] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky. Sparse convolutional neural networks. *CVPR*, 2015. 1, 3

[18] C. Liu, B. Zoph, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. *ECCV*, 2018. 6

[19] K. S. Liu, Hanxiao and Y. Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 1, 2, 3

[20] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. *ECCV*, 2018. 2, 6

[21] S. Mehta, M. Rastegari, A. Caspi, L. Shapiro, and H. Hajishirzi. Espnet: Efficient spatial pyramid of dilated convolutions for semantic segmentation. *ECCV*, 2018. 8

[22] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*, 2016. 1, 2, 8

[23] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. *AAAI*, 2019. 1, 2, 7

[24] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, and A. Kurakin. Large-scale evolution of image classifiers. *ICML*, 2017. 2

[25] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CVPR*, 2018. 1, 2, 6

[26] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CVPR*, 2015. 6

[27] R. J. Wang, X. Li, S. Ao, and C. X. Ling. Pelee: A real-time object detection system on mobile devices. *NIPS*, 2018. 6

[28] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. *CVPR*, 2019. 2, 6

[29] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng. Quantized convolutional neural networks for mobile devices. *CVPR*, 2016. 1, 3

[30] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. *ECCV*, 2018. 3

[31] C. Yu, J. Wang, C. Peng, C. Gao, G. Yu, and N. Sang. Bisenet: Bilateral segmentation network for real-time semantic segmentation. *ECCV*, 2018. 1, 2, 8

[32] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CVPR*, 2018. 2, 6

[33] H. Zhao, X. Qi, X. Shen, J. Shi, and J. Jia. Icnet for real-time semantic segmentation on high-resolution images. *ECCV*, 2018. 1, 2, 5, 7, 8

[34] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. *CVPR*, 2017. 1, 2, 5

[35] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *ICLR*, 2017. 2

[36] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. *CVPR*, 2018. 1, 2, 3, 6, 7