FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search

Bichen Wu[1]*, Xiaoliang Dai[2]*, Peizhao Zhang[3], Yanghan Wang[3], Fei Sun[3], Yiming Wu[3], Yuandong Tian[3], Peter Vajda[3], Yangqing Jia[3], and Kurt Keutzer[1]
[1]UC Berkeley, [2]Princeton University, [3]Facebook Inc.
{bichen, keutzer}@berkeley.edu, xdai@princeton.edu,
{stzpz, yanghan, feisun, wyiming, yuandong, vajdap, jiayq}@fb.com

Abstract

*Designing accurate and efficient ConvNets for mobile devices is challenging because the design space is combinatorially large. Due to this, previous neural architecture search (NAS) methods are computationally expensive. ConvNet architecture optimality depends on factors such as input resolution and target devices. However, existing approaches are too resource demanding for case-by-case redesigns. Also, previous work focuses primarily on reducing FLOPs, but FLOP count does not always reflect actual latency. To address these, we propose a differentiable neural architecture search (DNAS) framework that uses gradient-based methods to optimize ConvNet architectures, avoiding enumerating and training individual architectures separately as in previous methods. FBNets (**F**acebook-**B**erkeley-**Net**s), a family of models discovered by DNAS surpass state-of-the-art models both designed manually and generated automatically. FBNet-B achieves 74.1% top-1 accuracy on ImageNet with 295M FLOPs and 23.1 ms latency on a Samsung S8 phone, 2.4x smaller and 1.5x faster than MobileNetV2-1.3[17] with similar accuracy. Despite higher accuracy and lower latency than MnasNet[20], we estimate FBNet-B's search cost is 420x smaller than MnasNet's, at only 216 GPU-hours. Searched for different resolutions and channel sizes, FBNets achieve 1.5% to 6.4% higher accuracy than MobileNetV2. The smallest FBNet achieves 50.2% accuracy and 2.9 ms latency (345 frames per second) on a Samsung S8. Over a Samsung-optimized FBNet, the iPhone-X-optimized model achieves a 1.4x speedup on an iPhone X. FBNet models are open-sourced at* `https://github.com/facebookresearch/mobile-vision`.

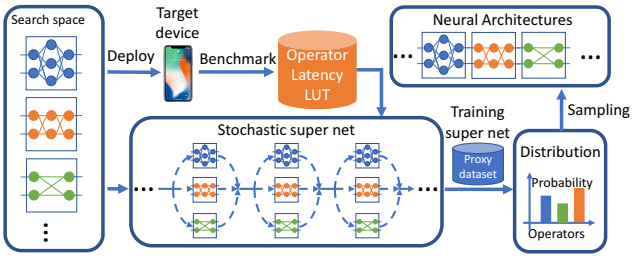


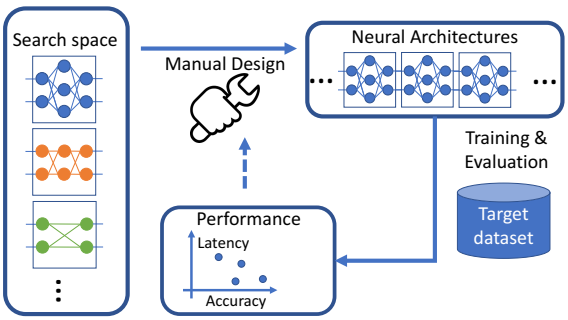
Figure 1. Differentiable neural architecture search (DNAS) for ConvNet design. DNAS explores a layer-wise space that each layer of a ConvNet can choose a different block. The search space is represented by a stochastic super net. The search process trains the stochastic super net using SGD to optimize the architecture distribution. Optimal architectures are sampled from the trained distribution. The latency of each operator is measured on target devices and used to compute the loss for the super net.

1. Introduction

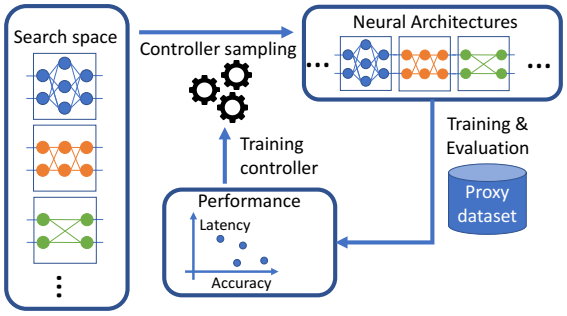
ConvNets are the *de facto* method for computer vision. In many computer vision tasks, a better ConvNet design usually leads to significant accuracy improvement. In previous works, accuracy improvement comes at the cost of higher computational complexity, making it more challenging to deploy ConvNets to mobile devices, on which computing capacity is limited. Instead of solely focusing on accuracy, recent work also aims to optimize for efficiency, especially latency. However, designing efficient and accurate ConvNets is difficult due to the challenges below.

Intractable design space: The design space of a ConvNet is combinatorial. Using VGG16 [18] as a motivating example: VGG16 contains 16 layers. Assume for each layer of the network, we can choose a different kernel size from $\{1, 3, 5\}$ and a different filter number from $\{32, 64, 128, 256, 512\}$. Even with such simplified design choices and shallow layers, the design space contains $(3 \times 5)^{16} \approx 6 \times 10^{18}$ possible architectures. However,

*Work done while interning at Facebook.



(a) A typical flow of manual ConvNet design.



(b) A typical flow of reinforcement learning based neural architecture search.

Figure 2. Illustration of manual ConvNet design and reinforcement learning based neural architecture search.

training a ConvNet is very time-consuming, typically taking days or even weeks. As a result, previous ConvNet design rarely explores the design space. A typical flow of manual ConvNet design is illustrated in Figure 2(a). Designers propose initial architectures and train them on the target dataset. Based on the performance, designers evolve the architectures accordingly. Limited by the time cost of training ConvNets, the design flow has to stop after a few iterations, which is far too few to sufficiently explore the design space.

Starting from [30], recent works adopt neural architecture search (NAS) to explore the design space automatically. Many previous works [30, 31, 20] use reinforcement learning (RL) to guide the search and a typical flow is illustrated in Figure 2(b). A controller samples architectures from the search space to be trained. To reduce the training cost, sampled architectures are trained on a smaller proxy dataset such as CIFAR-10 or trained for fewer epochs on ImageNet. The performance of the trained networks is then used to train and improve the controller. Previous works [30, 31, 20] has demonstrated the effectiveness of such methods in finding accurate and efficient ConvNet models. However, training each architecture is still time-consuming, and it usually takes thousands of architectures to train the controller. As a result, the computational cost of such methods is prohibitively high.

Nontransferable optimality: the optimality of ConvNet architectures is conditioned on many factors such as input resolutions and target devices. Once these factors change, the optimal architecture is likely to be different. A common practice to reduce the FLOP count of a network is to shrink the input resolution. A smaller input resolution may require a smaller receptive field of the network and therefore shallower layers. On a different device, the same operator can have different latency, so we need to adjust the ConvNet architecture to achieve the best accuracy-efficiency trade-off. Ideally, we should design different ConvNet architectures case-by-case. In practice, however, limited by the computational cost of previous manual and automated approaches, we can only realistically design one ConvNet and use it for all conditions.

Inconsistent efficiency metrics: Most of the efficiency metrics we care about are dependent on not only the ConvNet architecture but also the hardware and software configurations on the target device. Such metrics include latency, power, energy, and in this paper, we mainly focus on latency. To simplify the problem, most of the previous works adopt hardware-agnostic metrics such as FLOPs (more strictly, number of multiply-add operations) to evaluate a ConvNet's efficiency. However, a ConvNet with lower FLOP count is not necessarily faster. For example, NasNet-A [31] has a similar FLOP count as MobileNetV1 [6], but its complicated and fragmented cell-level structure is not hardware friendly, so the actual latency is slower [17]. The inconsistency between hardware agnostic metrics and actual efficiency makes the ConvNet design more difficult.

To address the above problems, we propose to use differentiable neural architecture search (DNAS) to discover hardware-aware efficient ConvNets. The flow of our algorithm is illustrated in Figure 1. DNAS allows us to explore a layer-wise search space where we can choose a different block for each layer of the network. Following [21], DNAS represents the search space by a super net whose operators execute stochastically. We relax the problem of finding the optimal architecture to find a distribution that yields the optimal architecture. By using the Gumbel Softmax technique [9], we can directly train the architecture distribution using gradient-based optimization such as SGD. The search process is extremely fast compared with previous reinforcement learning (RL) based method. The loss used to train the stochastic super net consists of both the cross-entropy loss that leads to better accuracy and the latency loss that penalizes the network's latency on a target device. To estimate the latency of an architecture, we measure the latency of each operator in the search space and use a lookup table model to compute the overall latency by adding up the latency of each operator. Using this model allows us to quickly estimate the latency of architectures in this enormous search space. More importantly, it makes the latency differentiable with respect to layer-wise block choices.

We name the models discovered by DNAS as FBNets (***F**acebook-**B**erkeley-**Nets***). FBNets surpass the state-of-the-art efficient ConvNets designed manually and automatically. FBNet-B achieves 74.1% top-1 accuracy with 295M FLOPs and 23.1 ms latency on an Samsung S8 phone, 2.4x smaller and 1.5x faster than MobileNetV2-1.3. Being better than MnasNet, FBNet-B's search cost is 216 GPU-hours, 421x lower than the cost for MnasNet estimated based on [20]. Such low search cost enables us to re-design ConvNets case-by-case. For different resolution and channel scaling, FBNets achieve 1.5% to 6.4% absolute gain in top-1 accuracy compared with MobileNetV2 models. The smallest FBNet achieves 50.2% accuracy and 2.9 ms latency (345 frames per second) with a batch size of 1 on Samsung S8. Using DNAS to search for device-specific ConvNet, an iPhone-x-optimized model achieves 1.4x speedup on an iPhone X compared with a Samsung-optimized model.

2. Related work

Efficient ConvNet models: Designing efficient ConvNet has attracted many research attention in recent years. SqueezeNet [8] is one of the early works focusing on reducing the parameter size of ConvNet models. It is originally designed for classification, but later extended to object detection [22] and LiDAR point-cloud segmentation [24, 26]. Following SqueezeNet, SqueezeNext [3] and ShiftNet [23] achieve further parameter size reduction. Recent works change the focus from parameter size to FLOPs. MobileNetV1 and MobileNetV2 [6, 17] use depthwise convolutions to replace the more expensive spatial convolutions. ShuffleNet [29] uses group convolution and shuffle operations to reduce the FLOP count further. More recent works realize that FLOP count does not always reflect the actual hardware efficiency. To improve actual latency, ShuffleNetV2 [13] proposes a series of practical guidelines for efficient ConvNet design. Synetgy [28] combines ideas from ShuffleNetV2 and ShiftNet to co-design hardware friendly ConvNets and FPGA accelerators.

Neural Architecture Search: [30, 31] first proposes to use reinforcement learning (RL) to search for neural architectures to achieve competitive accuracy with low FLOPs. Early NAS methods are computationally expensive. Recent works try to reduce the computational cost by weight sharing [16] or using gradient-based optimization [12]. [25, 1] further develop the idea of differentiable neural architecture search combining Gumbel Softmax [9]. Early works of NAS [31, 16, 12] focus on the cell level architecture search, and the same cell structure is repeated in all layers of a network. However, such fragmented and complicated cell-level structures are not hardware friendly, and the actual efficiency is low. Most recently, [20] explores a stage-level hierarchical search space, allowing different blocks for different stages of a network, while blocks inside a stage are still the same. Instead of focusing on FLOPs, [20] aims to optimize the latency on target devices. Besides searching for new architectures, works such as [27, 5] focus on adapting existing models to improve efficiency.

3. Method

In this paper, we use differentiable neural architecture search (DNAS) to solve the problem of ConvNet design. We formulate the neural architecture search problem as

$$\min_{a \in \mathcal{A}} \min_{w_a} \mathcal{L}(a, w_a). \tag{1}$$

Given an architecture space $\mathcal{A}$, we seek to find an optimal architecture $a \in \mathcal{A}$ such that after training its weights w_a, it can achieve the minimal loss $\mathcal{L}(a, w_a)$. In our work, we focus on three factors of the problem: a) the search space $\mathcal{A}$. b) The loss function $\mathcal{L}(a, w_a)$ that considers actual latency. c) An efficient search algorithm.

3.1. The Search Space

Previous works [30, 31, 16, 11, 12] focus on cell level architecture search. Once a cell structure is searched, it is used in all the layers across the network. However, many searched cell structures are very complicated and fragmented and are therefore slow when deployed to mobile CPUs [17, 13]. Besides, at different layers, the same cell structure can have a different impact on the accuracy and latency of the overall network. As shown in [20] and in our experiments, allowing different layers to choose different blocks leads to better accuracy and efficiency.

In this work, we construct a layer-wise search space with a fixed macro-architecture, and each layer can choose a different block. The macro-architecture is described in Table 1. The macro architecture defines the number of layers and the input/output dimensions of each layer. The first and the last three layers of the network have fixed operators. For the rest of the layers, their block type needs to be searched. The filter numbers for each layer are hand-picked empirically. We use relatively small channel sizes for early layers, since the input resolution at early layers is large, and the computational cost (FLOP count) is quadratic to input size.

Each searchable layer in the network can choose a different block from the layer-wise search space. The block structure is inspired by MobileNetV2 [17] and ShiftNet [23], and is illustrated in Figure 3. It contains a point-wise (1x1) convolution, a K-by-K depthwise convolution where K denotes the kernel size, and another 1x1 convolution. "ReLU" activation functions follow the first 1x1 convolution and the depthwise convolution, but there are no activation functions following the last 1x1 convolution. If the output dimension stays the same as the input dimension, we use a skip connection to add the input to the output. Following [17, 23], we use a hyperparameter, the expansion ratio e, to control

Input shape	Block	f	n	s
$224^2 \times 3$	3x3 conv	16	1	2
$112^2 \times 16$	TBS	16	1	1
$112^2 \times 16$	TBS	24	4	2
$56^2 \times 24$	TBS	32	4	2
$28^2 \times 32$	TBS	64	4	2
$14^2 \times 64$	TBS	112	4	1
$14^2 \times 112$	TBS	184	4	2
$7^2 \times 184$	TBS	352	1	1
$7^2 \times 352$	1x1 conv	1504 (1984)	1	1
$7^2 \times 1504$ (1984)	7x7 avgpool	-	1	1
1504	fc	1000	1	-

Table 1. Macro-architecture of the search space. Column-"Block" denotes the block type. "TBS" means layer type needs to be searched. Column-f denotes the filter number of a block. Column-n denotes the number of blocks. Column-s denotes the stride of the first block in a stage. The filter size of the last 1x1 conv is 1504 for FBNet-A and 1984 for FBNet-{B, C}.

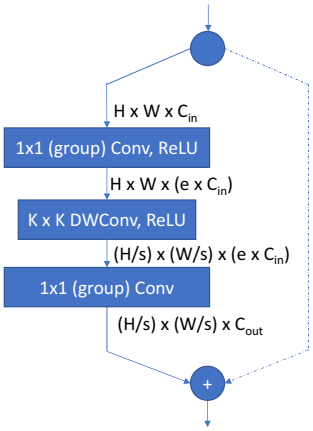


Figure 3. The block structure of the micro-architecture search space. Each candidate block in the search space can choose a different expansion rate, kernel size, and number of groups for group convolution.

the block. It determines how much do we expand the output channel size of the first 1x1 convolution compared with its input channel size. Following [20], we also allow choosing a kernel size of 3 or 5 for the depthwise convolution. In addition, we can choose to use group convolution for the first and the last 1x1 convolution to reduce the computation complexity. When we use group convolution, we follow [29] to add a channel shuffle operation to mix the information between channel groups.

In our experiments, our layer-wise search space contains 9 candidate blocks, with their configurations listed in Table 2. Note we also have a block called "skip", which directly feed the input feature map to the output without actual computations. This candidate block essentially allows us to reduce the depth of the network.

In summary, our overall search space contains 22 layers and each layer can choose from 9 candidate blocks from

Block type	expansion	Kernel	Group
k3_e1	1	3	1
k3_e1_g2	1	3	2
k3_e3	3	3	1
k3_e6	6	3	1
k5_e1	1	5	1
k5_e1_g2	1	5	2
k5_e3	3	5	1
k5_e6	6	5	1
skip	-	-	-

Table 2. Configurations of candidate blocks in the search space.

Table 2, so it contains $9^{22} \approx 10^{21}$ possible architectures. Finding the optimal layer-wise block assignment from such enormous search space is a non-trivial task.

3.2. Latency-Aware Loss Function

The loss function used in (1) has to reflect not only the accuracy of a given architecture but also the latency on the target hardware. To achieve this goal, we define the following loss function:

$$\mathcal{L}(a, w_a) = \text{CE}(a, w_a) \cdot \alpha \log(\text{LAT}(a))^{\beta}. \quad (2)$$

The first term $\text{CE}(a, w_a)$ denotes the cross-entropy loss of architecture a with parameter w_a. The second term $\text{LAT}(a)$ denotes the latency of the architecture on the target hardware measured in micro-second. The coefficient α controls the overall magnitude of the loss function. The exponent coefficient β modulates the magnitude of the latency term.

The cross-entropy term can be easily computed. However, the latency term is more difficult, since we need to measure the actual runtime of an architecture on a target device. To cover the entire search space, we need to measure about 10^{21} architectures, which is an impossible task.

To solve this problem, we use a latency lookup table model to estimate the overall latency of a network based on the runtime of each operator. More formally, we assume

$$\text{LAT}(a) = \sum_l \text{LAT}(b_l^{(a)}), \quad (3)$$

where $b_l^{(a)}$ denotes the block at layer-l from architecture a. This assumes that on the target processor, the runtime of each operator is independent of other operators. The assumption is valid for many mobile CPUs and DSPs, where operators are computed sequentially one by one. This way, by benchmarking the latency of a few hundred operators used in the search space, we can easily estimate the actual runtime of the 10^{21} architectures in the entire search space. More importantly, as will be explained in section 3.3, using the lookup table model makes the latency term in the loss function (2) differentiable with respect to layer-wise block choices, and this allows us to use gradient-based optimization to solve problem (1).

3.3. The Search Algorithm

Solving the problem (1) through brute-force enumeration of the search space is very infeasible. The inner problem of optimizing w_a involves training a neural network. For ImageNet classification, training a ConvNet typically takes several days or even weeks. The outer problem of optimizing $a \in \mathcal{A}$ has a combinatorially large search space.

Most of the early works on NAS [30, 31, 20] follow the paradigm above. To reduce the computational cost, the inner problem is replaced by training candidate architectures on an easier proxy dataset. For example, [30, 31] trains the architecture on the CIFAR10 dataset, and [20] trains on ImageNet but only for 5 epochs. The learned architectures are then transferred to the target dataset. To avoid exhaustively iterating through the search space, [30, 31, 20] use reinforcement learning to guide the exploration. Despite these improvements, solving problem (1) is still prohibitively expensive – training a network on the proxy dataset is still time-consuming, and thousands of architectures need to be trained before reaching the optimal solution.

We adopt a different paradigm of solving problem (1). We first represent the search space by a stochastic super net. The super net has the same macro-architecture as described in Table 1, and each layer contains 9 parallel blocks as described in Table 2. During the inference of the super net, only one candidate block is sampled and executed with the sampling probability of

$$P_{\boldsymbol{\theta}_l}(b_l = b_{l,i}) = \text{softmax}(\theta_{l,i}; \boldsymbol{\theta}_l) = \frac{\exp(\theta_{l,i})}{\sum_i \exp(\theta_{l,i})}. \quad (4)$$

$\boldsymbol{\theta}_l$ contains parameters that determine the sampling probability of each block at layer-l. Equivalently, the output of layer-l can be expressed as

$$x_{l+1} = \sum_i m_{l,i} \cdot b_{l,i}(x_l), \quad (5)$$

where $m_{l,i}$ is a random variable in $\{0, 1\}$ and is evaluated to 1 if block $b_{l,i}$ is sampled. The sampling probability is determined by equation (4). $b_{l,i}(x_l)$ denotes the output of block-i at layer l given the input feature map x_l. We let each layer sample independently, therefore, the probability of sampling an architecture a can be described as

$$P_{\boldsymbol{\theta}}(a) = \prod_l P_{\boldsymbol{\theta}_l}(b_l = b_{l,i}^{(a)}), \quad (6)$$

where $\boldsymbol{\theta}$ denotes the a vector consists of all the $\theta_{l,i}$ for each block-i at layer-l. $b_{l,i}^{(a)}$ denotes that in the sampled architecture a, block-i is chosen at layer-l.

Instead of solving for the optimal architecture $a \in \mathcal{A}$, which has a discrete search space, we relax the problem to optimize the probability P_θ of the stochastic super net to achieve the minimum expected loss. Formally, we re-write the discrete optimization problem (1) as

$$\min_{\boldsymbol{\theta}} \min_{w_a} \mathbf{E}_{a \sim P_{\boldsymbol{\theta}}} \{\mathcal{L}(a, w_a)\}. \quad (7)$$

It is obvious the loss function in (7) is differentiable with respect to the architecture weights w_a and therefore can be optimized by stochastic gradient descent (SGD). However, the loss is not directly differentiable to the sampling parameter $\boldsymbol{\theta}$, since we cannot pass the gradient through the discrete random variable $m_{l,i}$ to $\theta_{l,i}$. To sidestep this, we relax the discrete mask variable $m_{l,i}$ to be a continuous random variable computed by the Gumbel Softmax function [9, 14]

$$\begin{aligned} m_{l,i} &= \text{GumbelSoftmax}(\theta_{l,i} | \boldsymbol{\theta}_l) \\ &= \frac{\exp[(\theta_{l,i} + g_{l,i})/\tau]}{\sum_i \exp[(\theta_{l,i} + g_{l,i})/\tau]}, \end{aligned} \quad (8)$$

where $g_{l,i} \sim \text{Gumbel}(0, 1)$ is a random noise following the Gumbel distribution. The Gumbel Softmax function is controlled by a temperature parameter τ. As τ approaches 0, it approximates the discrete categorical sampling following the distribution in (6). As τ becomes larger, $m_{l,i}$ becomes a continuous random variable. Regardless of the value of τ, the mask $m_{l,i}$ is directly differentiable with respect to the parameter $\theta_{l,i}$. The technique of using Gumbel Softmax for neural architecture search is also proposed in [25, 1].

As a result, it is clear that the cross-entropy term from the loss function (2) is differentiable with respect to the mask $m_{l,i}$ and therefore $\theta_{l,i}$. For the latency term, since we use the lookup table based model for efficiency estimation, equation (3) can be written as

$$\text{LAT}(a) = \sum_l \sum_i m_{l,i} \cdot \text{LAT}(b_{l,i}). \quad (9)$$

The latency of each operator $\text{LAT}(b_{l,i})$ is a constant coefficient, so the overall latency of architecture-a is differentiable with respect to the mask $m_{l,i}$, therefore $\theta_{l,i}$.

As a result, the loss function (2) is fully differentiable with respect to both weights w_a and the architecture distribution parameter $\boldsymbol{\theta}$. This allows us to use SGD to efficiently solve problem (1).

Our search process is now equivalent to training the stochastic super net. During the training, we compute $\partial\mathcal{L}/\partial w_a$ to train each operator's weight in the super net. This is no different from training an ordinary ConvNet. After operators get trained, different operators can have a different contribution to the accuracy and the efficiency of the overall network. Therefore, we compute $\partial\mathcal{L}/\partial\boldsymbol{\theta}$ to update the sampling probability $P_{\boldsymbol{\theta}}$ for each operator. This step selects operators with better accuracy and lower latency and suppresses the opposite ones. After the super net training finishes, we can then obtain the optimal architectures by sampling from the architecture distribution $P_{\boldsymbol{\theta}}$.

As will be shown in the experiment section, the proposed DNAS algorithm is orders of magnitude faster than previous RL based NAS while generating better architectures.

4. Experiments

4.1. ImageNet Classification

To demonstrate the efficacy of our proposed method, we use DNAS to search for ConvNet models on ImageNet 2012 classification dataset [2], and we name the discovered models FBNets. We aim to discover models with high accuracy and low latency on target devices. In our first experiment, we target Samsung Galaxy S8 with a Qualcomm Snapdragon 835 platform. The model is deployed with Caffe2 with int8 inference engine for mobile devices.

Before the search starts, we first build a latency lookup table described in section 3.2 on the target device. Next, we train a stochastic super net with search space described in section 3.3. We set the input resolution of the network to 224-by-224. To reduce the training time, we randomly choose 100 classes from the original 1000 classes to train the stochastic super net. We train the stochastic super net for 90 epochs. In each epoch, we first train the operator weights w_a and then the architecture probability parameter $\boldsymbol{\theta}$. w_a is trained on 80% of ImageNet training set using SGD with momentum. The architecture distribution parameter $\boldsymbol{\theta}$ is trained on the rest 20% of ImageNet training set with Adam optimizer [10]. To control the temperature of the Gumbel Softmax from equation (8), we use an exponentially decaying temperature. After the search finishes, we sample several architectures from the trained distribution $P_{\boldsymbol{\theta}}$, and train them from scratch. Our architecture search framework is implemented in pytorch [15] and searched models are trained in Caffe2. More training details will be provided in the supplementary materials.

Our experiment results are summarized in Table 3. We compare our searched models with state-of-the-art efficient models both designed automatically and manually. The primary metrics we care about are top-1 accuracy on the ImageNet validation set and the latency. If the latency is not available, we use FLOP as the secondary efficiency metric. For baseline models, we directly cite the parameter size, FLOP count, and top-1 accuracy from the original paper. Since our network is deployed with caffe2 with highly efficient in8 implementation, we have an unfair latency advantage against other baselines. Therefore, we implement the baseline models ourselves and measure their latency under the same environment for a fair comparison. For automatically designed models, we also compare the search method, search space, and search cost.

Table 3 divides the models into three categories according to their accuracy level. In the first group, FBNet-A achieves 73.0% accuracy, better than 1.0-MobileNetV2 (+1.0%), 1.5-ShuffleNet V2 (+0.4%), and CondenseNet (+2%), and are on par with DARTS and MnasNet-65. Regarding latency, FBNet-A is 1.9 ms (relative 9.6%), 2.2 ms (relative 11%), and 8.6 ms (relative 43%) better than the MobileNetV2, ShuffleNetV2, and CondenseNet counterparts. Although we did not optimize for FLOP count directly, FBNet-A's FLOP count is only 249M, 50M smaller (relative 20%) than MobileNetV2 and ShuffleNetV2, 20M (relative 8%) smaller than MnasNet, and 2.4X smaller than DARTS. In the second group, FBNet-B achieves comparable accuracy with 1.3-MobileNetV2, but the latency is 1.46x lower, and the FLOP count is 1.73x smaller, even smaller than 1.0-MobileNetV2 and 1.5-ShuffleNet V2. Compared with MnasNet, FBNet-B's accuracy is 0.1% higher, latency is 0.6ms lower, and FLOP count is 22M (relative 7%) smaller. We do not have the latency of NASNet-A and PNASNet, but the accuracy is comparable, and the FLOP count is 1.9x and 2.0x smaller. In the third group, FBNet-C achieves 74.9% accuracy, same as 2.0-ShuffleNetV2 and better than all others. The latency is 28.1 ms, 1.33x and 1.19x faster than MobileNet and ShuffleNet. The FLOP count is 1.56x, 1.58x, and 1.03x smaller than MobileNet, ShuffleNet, and MnasNet-92.

Among all the automatically searched models, FBNet's performance is much stronger than DARTS, PNAS, and NAS, and better than MnasNet. However, the search cost is orders of magnitude lower. MnasNet [20] does not disclose the exact search cost (in terms of GPU-hours). However, it mentions that the controller samples 8,000 models during the search and each model is trained for five epochs. According to our experiments, training of MNasNet for one epoch takes 17 minutes using 8 GPUs. So the estimated cost for training 8,000 models for 5 epochs is about $17/60 \times 5 \times 8 \times 8,000 \approx 91 \times 10^3$ GPU hours. In comparison, the FBNet search takes 8 GPUs for only 27 hours, so the computational cost is only 216 GPU hours, or 421x faster than MnasNet, 222x faster than NAS, 27.8x faster than PNAS, and 1.33x faster than DARTS.

We visualize some of our searched FBNets, MobileNetV2, and MnasNet in Figure 4.

4.2. Different Resolution and Channel Size Scaling

A common technique to reduce the computational cost of a ConvNet is to reduce the input resolution or channel size without changing the ConvNet structure. This approach is likely to be sub-optimal. We hypothesize that with a different input resolution and channel size scaling, the optimal ConvNet structure will be different. To test this, we use DNAS to search for several different combinations of input resolution and channel size scaling. Thanks to the superior efficiency of DNAS, we can finish the search very quickly. The result is summarized in Table 4. Compared with MobileNetV2 under the same input size and channel size scal-

Model	Search method	Search space	Search cost (GPU hours / relative)	#Params	#FLOPs	CPU Latency	Top-1 acc (%)
1.0-MobileNetV2 [17]	manual	-	-	3.4M	300M	21.7 ms	72.0
1.5-ShuffleNetV2 [13]	manual	-	-	3.5M	299M	22.0 ms	72.6
CondenseNet (G=C=8) [7]	manual	-	-	2.9M	274M	28.4 ‡ ms	71.0
MnasNet-65 [13]	RL	stage-wise	91K* / 421x	3.6M	270M	-	73.0
DARTS [12]	gradient	cell	288 / 1.33x	4.9M	595M	-	**73.1**
FBNet-A (ours)	gradient	layer-wise	216 / 1.0x	4.3M	**249M**	**19.8 ms**	73.0
1.3-MobileNetV2 [17]	manual	-	-	5.3M	509M	33.8 ms	**74.4**
CondenseNet (G=C=4) [7]	manual	-	-	4.8M	529M	28.7‡ ms	73.8
MnasNet [20]	RL	stage-wise	91K* / 421x	4.2M	317M	23.7 ms	74.0
NASNet-A [31]	RL	cell	48K / 222x	5.3M	564M	-	74.0
PNASNet [11]	SMBO	cell	6K† / 27.8x	5.1M	588M	-	74.2
FBNet-B (ours)	gradient	layer-wise	216 / 1.0x	4.5M	**295M**	**23.1 ms**	74.1
1.4-MobileNetV2 [17]	manual	-	-	6.9M	585M	37.4 ms	74.7
2.0-ShuffleNetV2 [13]	manual	-	-	7.4M	591M	33.3 ms	**74.9**
MnasNet-92 [20]	RL	stage-wise	91K* / 421x	4.4M	388M	-	74.8
FBNet-C (ours)	gradient	layer-wise	216 / 1.0x	5.5M	**375M**	**28.1 ms**	**74.9**

Table 3. ImageNet classification performance compared with baselines. For baseline models, we directly cite the parameter size, FLOP count and top-1 accuracy on the ImageNet validation set from their original papers. For CPU latency, we deploy and benchmark the models on the same Samsung Galaxy S8 phone with Caffe2 int8 implementation. The details of MnasNet-{64, 92} are not disclosed from [20] so we cannot measure the latency. *The search cost for MnasNet is estimated according to the description in [20]. † The search cost is estimated based on the claim from [11] that PNAS [11] is 8x lower than NAS[31]. ‡ The inference engine is faster than other models.

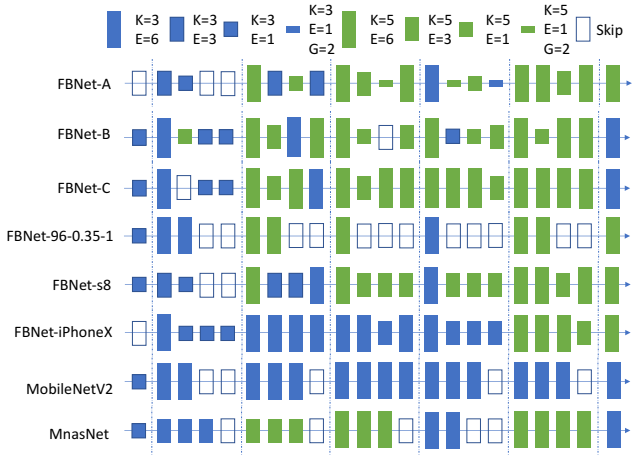


Figure 4. Visualization of some of the searched architectures. We use rectangle boxes to denote blocks for each layer. We use different colors to denote the kernel size of the depthwise convolution, blue for kernel size of 3, green for kernel size of 5, and empty for skipping. We use height to denote the expansion rate of the block: 6, 3, 1, and 1 with group-2 convolution.

ing, our searched models achieve 1.5% to 6.4% better accuracy with similar latency. Especially the FBNet-96-0.35-1 model achieves 50.2% (+4.7%) accuracy and 2.9 ms latency (345 frames per second) on a Samsung Galaxy S8.

We visualize the architecture of FBNet-96-0.35-1 in Figure 4, we can see that many layers are skipped, and the network is much shallower than FBNet-{A, B, C}, whose input size is 224. We conjecture that this is because with smaller input size, the receptive field needed to parse the image also becomes smaller, so having more layers will not effectively increase the accuracy.

4.3. Different Target Devices

In previous ConvNet design practices, the same ConvNet model is deployed to many different devices. However, this is sub-optimal since different computing platforms and software implementation can have different characteristics. To validate this, we conduct search targeting two mobile devices: Samsung Galaxy S8 with Qualcomm Snapdragon 835 platforms, and iPhone X with A11 Bionic processors. We use the same architecture search space, but different latency lookup tables collected from two target devices. All the architecture search and training protocols are the same. After we searched and trained two models, we deploy them to both Samsung Galaxy S8 and iPhone X to benchmark the overall latency. The result is summarized in Table. 5.

As we can see, the two models reach similar accuracy (73.20% vs. 73.27%). FBNet-iphoneX model's latency is 19.84 ms on its target device, but when deployed to a Samsung S8, its latency increases to 23.33 ms. On the other hand, FBNet-S8 reaches a latency of 22.12 ms on a Samsung S8, but when deployed to an iPhone X, the latency hikes to 27.53 ms, 7.69 ms (relatively 39%) higher than FBNet-iPhone X. This demonstrates the necessity of re-designing ConvNets for different target devices.

Two models are visualized in Figure 4. Note that FBNet-

Input size & Channel Scaling	Model	#Parameters	#FLOPs	CPU Latency	Top-1 acc (%)
(224, 0.35)	MobileNetV2-224-0.35	1.7M	59M	9.3 ms	60.3
	MNasNet-scale-224-0.35	1.9M	76M	10.7 ms	62.4 (+2.1)
	FBNet-224-0.35	2.0M	72M	10.7 ms	65.3 (+5.0)
(192, 0.50)	MobileNetV2	2.0M	71M	8.4 ms	63.9
	MnasNet-search-192-0.5	-	-	-	65.6 (+1.7)
	FBNet-192-0.5 (ours)	2.6M	73M	9.9 ms	65.9 (+2.0)
(128, 1.0)	MobileNetV2	3.5M	99M	8.4 ms	65.3
	MnasNet-scale-128-1.0	4.2M	103M	9.2 ms	67.3 (+2.0)
	FBNet-128-1.0 (ours)	4.2M	92M	9.0 ms	67.0 (+1.7)
(128, 0.50)	MobileNetV2	2.0M	32M	4.8 ms	57.7
	FBNet-128-0.5 (ours)	2.4M	32M	5.1 ms	60.0 (+2.3)
(96, 0.35)	MobileNetV2	1.7M	11M	3.8 ms	45.5
	FBNet-96-0.35-1 (ours)	1.8M	12.9M	2.9 ms	50.2 (+4.7)
	FBNet-96-0.35-2 (ours)	1.9M	13.7M	3.6 ms	51.9 (+6.4)

Table 4. FBNets searched for different input resolution and channel scaling. MnasNet-scale is the MnasNet model with input and channel size scaling. MnasNet-search-192-0.5 is a model searched with an input size of 192 and channel scaling of 0.5. Details of it are not disclosed in [20], so we only cite the accuracy.

Model	#Parameters	#FLOPs	Latency on iPhone X	Latency on Samsung S8	Top-1 acc (%)
FBNet-iPhoneX	4.47M	322M	19.84 ms (target)	23.33 ms	73.20
FBNet-S8	4.43M	293M	27.53 ms	22.12 ms (target)	73.27

Table 5. FBNets searched for different devices.

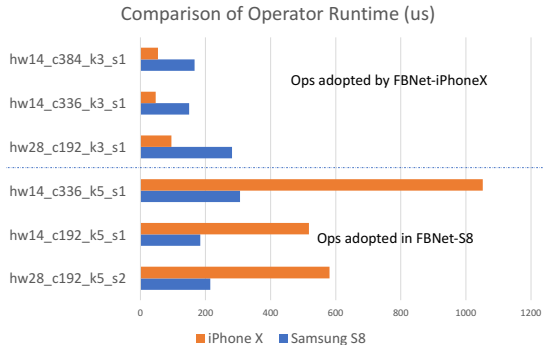


Figure 5. Comparison of operator runtime on two devices. Runtime is in micro-second (us). Orange bar denotes the runtime on iPhone X and blue bar denotes the runtime on Samsung S8. The upper three operators are faster on iPhone X, therefore they are automatically adopted in FBNet-iPhoneX. The lower three operators are faster on Samsung S8, and they are also automatically adopted in FBNet-S8.

S8 uses many blocks with 5x5 depthwise convolution while FBNet-iPhoneX only uses them in the last two stages. We examine the depthwise convolution operators used in the two models and compare their runtime on both devices. As shown in Figure 5, the upper three operators are faster on iPhone X, therefore they are automatically adopted in FBNet-iPhoneX. The lower three operators are significantly faster on Samsung S8, and they are also automatically adopted in FBNet-S8. Notice the drastic runtime differences of the lower three operators on two target devices. It explains why the Samsung-S8-optimized model performs poorly on an iPhone X. This shows DNAS can automatically optimize the operator adoptions and generate different ConvNets optimized for different devices.

5. Conclusion

We present DNAS, a differentiable neural architecture search framework. It optimizes over a layer-wise search space and represents the search space by a stochastic super net. The actual target device latency of blocks is used to compute the loss for super net training. FBNets, a family of models discovered by DNAS surpass state-of-the-art models, both manually and automatically designed: FBNet-B achieves 74.1% top-1 accuracy with 295M FLOPs and 23.1 ms latency, 2.4x smaller and 1.5x faster than MobileNetV2-1.3 with the same accuracy. It also achieves better accuracy and lower latency than MnasNet, the state-of-the-art efficient model designed automatically; we estimate the search cost of DNAS is 420x smaller. Such efficiency allows us to conduct searches for different input resolutions and channel scaling. Discovered models achieve 1.5% to 6.4% accuracy gains. The smallest FBNet achieves 50.2% accuracy with a latency of 2.9 ms (345 frames/sec) with batch size 1. Over the Samsung-optimized FBNet, the improved FBNet achieves 1.4x speed up on an iPhone X, showing DNAS is able to adapt to different target devices automatically.

References

[1] Anonymous. Snas: stochastic neural architecture search. In *Submitted to International Conference on Learning Representations*, 2019. under review.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009.

[3] A. Gholami, K. Kwon, B. Wu, Z. Tai, X. Yue, P. Jin, S. Zhao, and K. Keutzer. Squeezenext: Hardware-aware neural network design. *arXiv preprint arXiv:1803.10615*, 2018.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[5] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018.

[6] 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.

[7] G. Huang, S. Liu, L. van der Maaten, and K. Q. Weinberger. Condensenet: An efficient densenet using learned group convolutions. *group*, 3(12):11, 2017.

[8] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[9] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[10] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[11] 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. *arXiv preprint arXiv:1712.00559*, 2017.

[12] H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[13] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. *arXiv preprint arXiv:1807.11164*, 2018.

[14] C. J. Maddison, A. Mnih, and Y. W. Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[15] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[16] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[17] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[18] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[19] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[20] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le. Mnasnet: Platform-aware neural architecture search for mobile. *arXiv preprint arXiv:1807.11626*, 2018.

[21] T. Veniat and L. Denoyer. Learning time/memory-efficient deep architectures with budgeted super networks. *arXiv preprint arXiv:1706.00046*, 2017.

[22] B. Wu, F. N. Iandola, P. H. Jin, and K. Keutzer. Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In *CVPR Workshops*, pages 446–454, 2017.

[23] B. Wu, A. Wan, X. Yue, P. Jin, S. Zhao, N. Golmant, A. Gholaminejad, J. Gonzalez, and K. Keutzer. Shift: A zero flop, zero parameter alternative to spatial convolutions. *arXiv preprint arXiv:1711.08141*, 2017.

[24] B. Wu, A. Wan, X. Yue, and K. Keutzer. Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1887–1893. IEEE, 2018.

[25] B. Wu, Y. Wang, P. Zhang, Y. Tian, P. Vajda, and K. Keutzer. Mixed precision quantization of convnets via differentiable neural architecture search. *arXiv preprint arXiv:1812.00090*, 2018.

[26] B. Wu, X. Zhou, S. Zhao, X. Yue, and K. Keutzer. Squeezesegv2: Improved model structure and unsupervised domain adaptation for road-object segmentation from a lidar point cloud. *arXiv preprint arXiv:1809.08495*, 2018.

[27] 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. *Energy*, 41:46, 2018.

[28] Y. Yang, Q. Huang, B. Wu, T. Zhang, L. Ma, G. Gambardella, M. Blott, L. Lavagno, K. Vissers, J. Wawrzynek, et al. Synetgy: Algorithm-hardware co-design for convnet accelerators on embedded fpgas. *arXiv preprint arXiv:1811.08634*, 2018.

[29] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. arxiv 2017. *arXiv preprint arXiv:1707.01083*.

[30] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[31] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2(6), 2017.

A. Experiment details

We describe more experiment details in this appendix to facilitate other researchers to reproduce our work. Our architecture search is divided into two stages. In the first stage, we train the stochastic super net to find an optimal architecture distribution. In the second stage, we sample architectures from the distribution and train them from scratch.

To train the stochastic super net, we randomly sample 100 classes from the original 1,000 classes of ImageNet. Training the super net on this smaller proxy dataset is much faster. We train the stochastic super net for 90 epochs with a batch size of 192. In each epoch, we first train the operator parameters w_a on 80% of the training set using stochastic gradient descent with momentum. The initial learning rate is 0.1, and decay following a cosine decaying schedule. The momentum is 0.9, and weight decay is 10^{-4}. Next, we train the architecture distribution parameter $\boldsymbol{\theta}$ on the rest 20% of the training set with Adam optimizer [10] with a learning rate of 10^{-2} and weight decay of 5×10^{-4}. The split of weight and architecture parameter training ensure the architecture generalize to the validation dataset. To control the Gumbel Softmax in (8), we use an initial temperature of 5.0 and exponentially anneal it by $exp(-0.045) \approx 0.956$ every epoch. For the loss function in (2), we set α to 0.2 and β to 0.6. We use the standard ResNet data augmentation [4] to process the input images. We found that at the beginning of the training, operators are usually not sufficiently trained, so their contributions to the accuracy are not clear. However, their computational costs are always significantly different from each other. As a consequence, the super net may always pick low computational cost operators at the beginning of the training. To prevent this, we postpone the training of the architecture parameter $\boldsymbol{\theta}$ by 10 epochs to allow operator weights to be sufficiently trained first. At the end of the super net training, we sample 6 architectures from the final distribution to be trained from scratch.

To train the sampled architectures, the training protocols are different for different models. Here we describe the training protocol for FBNet-{A, B, C}. These models have an input resolution of 224, channel size scaling of 1.0. We train the models with a batch size of 256 on 8 GPUs for 360 epochs. We set the initial learning rate to be 0.1, and decay 10x at 90, 180, and 270 epochs. The momentum is 0.9, weight decay is 4×10^{-5}. We use dropout at the last convolution layer of the network, and the dropout ratio is 0.2. We use the standard GoogleNet data augmentation [19] to randomly resize the image during training.