FBNetV2: Differentiable Neural Architecture Search for Spatial and Channel Dimensions

Alvin Wan[1]*, Xiaoliang Dai[2], Peizhao Zhang[2], Zijian He[2], Yuandong Tian[2], Saining Xie[2], Bichen Wu[2], Matthew Yu[2], Tao Xu[2], Kan Chen[2], Peter Vajda[2], Joseph E. Gonzalez[1]
[1]UC Berkeley, [2]Facebook Inc.
{alvinwan,jegonzal}@berkeley.edu
{xiaoliangdai,stzpz,zijian,yuandong,s9xie,bichen,mattcyu,xutao,kanchen18,vadjap}@fb.com

Abstract

Differentiable Neural Architecture Search (DNAS) has demonstrated great success in designing state-of-the-art, efficient neural networks. However, DARTS-based DNAS's search space is small when compared to other search methods', since all candidate network layers must be explicitly instantiated in memory. To address this bottleneck, we propose a memory and computationally efficient DNAS variant: DMaskingNAS. This algorithm expands the search space by up to $10^{14}\times$ over conventional DNAS, supporting searches over spatial and channel dimensions that are otherwise prohibitively expensive: input resolution and number of filters. We propose a masking mechanism for feature map reuse, so that memory and computational costs stay nearly constant as the search space expands. Furthermore, we employ effective shape propagation to maximize per-FLOP or per-parameter accuracy. The searched FBNetV2s yield state-of-the-art performance when compared with all previous architectures. With up to 421$\times$ less search cost, DMaskingNAS finds models with 0.9% higher accuracy, 15% fewer FLOPs than MobileNetV3-Small; and with similar accuracy but 20% fewer FLOPs than Efficient-B0. Furthermore, our FBNetV2 outperforms MobileNetV3 by 2.6% in accuracy, with equivalent model size. FBNetV2 models are open-sourced at https://github.com/facebookresearch/mobile-vision.

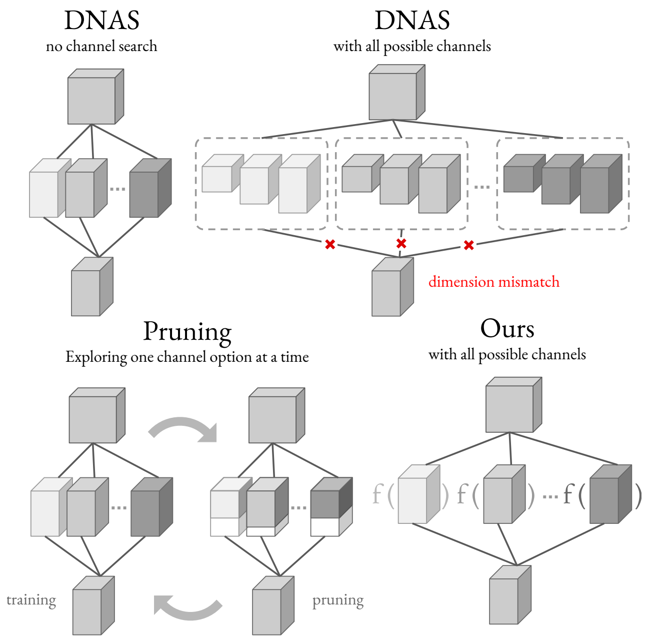
DNAS
no channel search
DNAS
with all possible channels
dimension mismatch
Pruning
Exploring one channel option at a time
training
pruning
Ours
with all possible channels
f( ) f( ) ··· f( )

Figure 1: **DNAS**: Adding all possible numbers of filters to DNAS (top-right) increases computational and memory costs drastically, exacerbating DNAS's memory bottleneck on search space size. **Pruning**: Channel pruning (bottom-left) is limited to training one architecture at a time. **Ours**: With our weight-sharing approximation, DNAS can explore all possible number of filters simultaneously with negligible memory and computation overhead. See Fig. 2 for details.

1. Introduction

Deep neural networks have led to significant progress in many research areas and applications, such as computer vision and autonomous driving. Despite this, designing an efficient network for resource-constrained settings remains a challenging problem. Initial directions involved compressing existing networks [7] or building small networks [23, 26]. However, the design space can easily contain more than 10^{18} candidate architectures [33, 27], making manual design choices sub-optimal and difficult to scale. In lieu of manual tuning, recent work uses neural architecture search (NAS) to design networks automatically.

*Work done while interning at Facebook.

Previous NAS methods utilize reinforcement learning (RL) techniques or evolutionary algorithms (EAs). However, both methods are computationally expensive and consume thousands of GPU hours [40, 29]. As a result, recent NAS literature [33, 20, 24] focuses on differentiable neural architecture search (DNAS); DNAS searches over a supergraph that encompasses all candidate architectures, selecting a single path as the final neural network. Unlike conventional NAS, DNAS can search large combinatorial spaces in the time it takes to train a single model [20, 35, 33, 27]. One class of DNAS methods, based on DARTS [20], suffer from two significant limitations [5]:

- **Memory costs bound the search space.** Short of paging in and out tensors, the supergraph and feature maps must reside in GPU memory for training, which limits the search space.

- **Cost grows linearly with the number of options per layer.** This means that each new search dimension introduces combinatorially more options and combinatorial memory and computational costs.

The other class of DNAS methods, not based on DARTS, suffer from similar issues: For example, ProxylessNAS tackles the memory constraint by training only one path in the supergraph each iteration. However, this means ProxylessNAS would take a prohibitively long time to converge on an order-of-magnitude larger search space. These memory and computation issues, for all DNAS methods, prevent us from expanding the search space to explore larger spaces of configurations. Noting that feature maps typically dominate memory cost [1], we propose a formulation of DNAS (Fig. 1) called DMaskingNAS (Fig. 2) that increases the search space size by orders of magnitude. To accomplish this, we represent multiple channel and input resolution options in the supergraph with masks, which carry negligible memory and computational costs. Furthermore, we reuse feature maps for all options in the supergraph, which enables nearly constant memory cost with increasing search space sizes. These optimizations yield the following three contributions:

- **A memory and computationally efficient DNAS** that optimizes both macro- (resolution, channels) and micro- (building blocks) architectures jointly in a $10^{14}\times$ larger search space using differentiable search. To the best of our knowledge, we are the first to tackle this problem using a differentiable search framework supergraph, with substantially less computational cost and roughly constant memory cost.

- **A masking mechanism and effective shape propagation** for feature map reuse. This is applied to both the spatial and channel dimensions in DNAS.

- **State-of-the-art results** on ImageNet classification. With only 27 hours on 8 GPUs, our searched compact models lead to substantial per-parameter, per-FLOP accuracy improvements. The searched models outperform all previous state-of-the-art neural networks, both manually and automatically designed, small and large.

Table 1: The number of DMaskingNAS design choices eclipses that of previous search spaces: number of channels c, kernel size k, number of layers l, bottleneck type b, input resolution r, and expansion rate e.

NAS algorithm	c	k	l	b	r	e
MnasNet [29]	✓		✓	✓		✓
ProxylessNAS [2]		✓	✓	✓		✓
Single-Path NAS [27]		✓	✓			✓
ChamNet [3]	✓		✓		✓	✓
FBNet [33]		✓	✓	✓		✓
DMaskingNAS	✓	✓	✓	✓	✓	✓

2. Related Work

Hand-crafted, efficient neural networks see two predominant approaches: (1) compressing existing architectures and (2) designing compact architectures from scratch.

Network compression includes both architectural and non-architectural modifications. One non-architectural approach is low-bit quantization, where weights and activations alike may be represented with fewer bits. For example, Wang et al. [31] propose hardware-aware automated quantization, which achieves a 1.4-1.95$\times$ latency reduction on MobileNet [12]. These techniques are orthogonal to and can be combined with the methods in this paper. Alternatively, architectural modifications include network pruning [8, 32, 36], where various heuristics govern layer-wise or channel-wise pruning. For example, Han et al. [8] show that magnitude-based pruning can reduce parameter count by orders of magnitude without accuracy loss, and NetAdapt [37] utilizes a filter pruning algorithm that achieves a 1.2$\times$ speedup for MobileNetV2. However, with heuristics-based simplifications, pruning methods train potential architectures separately, one after another – in some cases, pruning methods consider only one architecture [22, 10].

Compact architecture design aims to directly construct efficient networks, rather than trim an expensive one [15, 34]. For example, MobileNet [12] and MobileNetV2 [26] achieve substantial efficiency improvements by exploiting a depth-wise convolution and an inverted residual block, respectively. ShuffleNetV2 [23] shrinks the model size utilizing low-cost group convolutions. Tan et al. propose a compound scaling method, obtaining a family of architectures that achieve state-of-the-art accuracy with an order of magnitude fewer parameters than previous convolutional

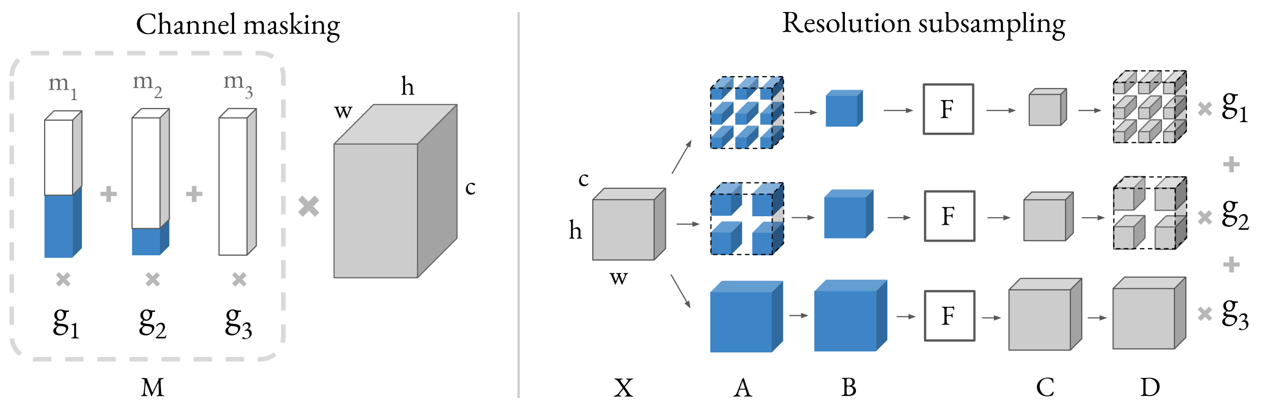

Figure 2: **Channel Masking** for channel search: A column vector mask $M \in \mathbb{R}^c$ is the weighted sum of several masks $m_i \in \mathbb{R}^c$, with Gumbel Softmax weights g_i. Each m_i has ones (white) in the first k entries and zeros (blue) in the next $c - k$ entries, for some $k \in \mathbb{Z}$. Multiplication with this mask speeds up channel search, using a weight-sharing approximation described in Fig. 3. **Resolution Subsampling** for input resolution: X is an intermediate output feature map for the network. A is subsampled from X using nearest neighbors. Values at the blue pixels in column A are assembled to create the smaller feature map in B. Next, run the operation F. Finally, each value in C is placed back into a larger feature map in D. Note we put values back (D) into pixels where we pulled values from (A). This process is motivated in Fig. 4.

networks [30]. However, these models rely on finely-tuned, manual decisions that are bested by automatic design.

Neural architecture search automates the design of state-of-the-art neural networks. Zoph et al. first proposed using RL for automated neural network design in [39]. This and other early NAS approaches are based on RL [39, 29] and EA [25]. However, both approaches consume substantial computational resources.

Later works utilize various techniques to reduce the computational cost of search. One such technique formulates the architecture search problem as a path-finding process in a supergraph [33, 20, 6, 27]. Among them, gradient-based NAS has emerged as a promising tool. Wu et al. show that gradient-based, differentiable NAS yields state-of-the-art compact architectures with $421\times$ less search cost than RL-based approaches. Another direction is to exploit a performance predictor to guide the search process [3, 19]. Such approaches explore the search space by trimming progressively and lead to significant reductions in search cost.

Stamoulis et al. [28] introduce weight-sharing to further reduce the computational cost of search. However, kernel weight-sharing doesn't address the primary drawback of DARTS, namely a memory bottleneck yielding small search space size: Say a "mixed kernel" contains weights shared between a 3×3 and 5×5. Since it is impossible to extract a 3×3 convolution's outputs from a 5×5's (and vice versa), this mixed kernel still convolves $2\times$ and still stores 2 feature maps for backpropagation. Thus, 2 kernel-weight-sharing convolutions induce memory and computational costs of 2 vanilla convolutions.

Searching along spatial and channel dimensions has been studied both with and without NAS. Liu et al [18] develop a NAS variant that searches over varying strides for semantic segmentation. However, this method suffers from increasing memory cost as the number of possible input resolutions grows. As described above, network pruning suffers from inefficient and sequential exploration of architectures, one-by-one. Yu et al [38] amend this partially by creating a batchnorm invariant to the number input channels; after training the "supergraph" they see competitive accuracy without further training, for each possible subset of channels. Yu et al [21] expand on these slimmable networks by introducing a test-time greedy channel selection procedure. However, these methods are orthogonal to and can be combined with DMaskingNAS, as we train the sampled architecture from scratch. To address these concerns, our algorithm jointly optimizes over multiple input resolutions and channel options simultaneously, increasing memory cost only negligibly as the number of options grows. This allows DMaskingNAS to support orders of magnitude more possible architectures, under existing memory constraints.

3. Method

We propose DMaskingNAS to search over spatial and channel dimensions, summarized in Fig. 2. The search space would be computationally prohibitive and ill-formed without the optimizations described below; our approach makes it possible to search this expanded search space (Table 1) over channels and input resolutions.

3.1. Channel Search

To support searches over varying numbers of channels, previous DNAS methods simply instantiate a block for ev-

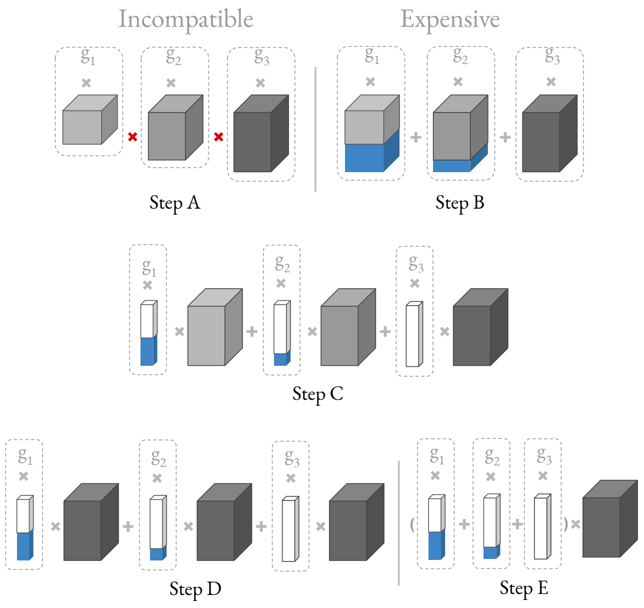

Figure 3: **Channel Search Challenges**: **Step A**: Consider 3 convolutions with varying numbers of filters. Each output (gray) will have varying numbers of channels. Thus, the outputs cannot be naively summed. **Step B**: Zero-padding (blue) outputs allows them to be summed. However, both FLOP and memory cost increases sub-linearly with the number of channel options. **Step C**: This is equivalent to running three convolutions with equal numbers of filters, multiplied by masks of zeros (blue) and ones (white). **Step D**: We approximate using weight sharing – all three convolutions are represented by one convolution. **Step E**: This is equivalent to summing the masks first, before multiplying by the output. Now, FLOP and memory cost are effectively constant w.r.t. the number of channel options.

ery channel option in the supergraph. For a convolution with k filters, this could mean up to $k(k+1)/2 \sim O(k^2)$ convolutions. Previous channel pruning methods [21] suffer from a similar drawback: each option must be trained separately, finding the "optimal" channel count in one shot or iteratively. Furthermore, even without saturating the maximum number of possibilities, there are two problems, the first of which makes this search impossible:

1. **Incompatible dimensions**: DNAS is divided into several "cells". In each cell, we consider a number of different block options; the outputs of all options are combined in a weighted sum. This means that all block outputs must align dimensions. If each block adopts convolutions with different number of filters, each output will have a different number of channels. As a result, DNAS could not perform a weighted sum.

2. **Slower training, increased memory cost**: Even with a workaround, with this nave instantiation, each convolution with a different channel option must be run separately, resulting in a $O(k)$ increase in FLOP cost. Furthermore, each output feature map must be stored separately in memory.

To address the aforementioned issues, we handle the incompatibility (Fig. 3, Step A): consider a block b with varying numbers of filters, where b_i denotes this block with i filters. The maximum number of filters is k. The outputs of all blocks are then zero-padded to have k channels (Fig. 3, Step B). Given input x, the Gumbel Softmax output is thus the following, with Gumbel weights g_i:

$$y = \sum_{i=1}^{k} g_i \textsc{Pad}(b_i(x), k) \tag{1}$$

Note that this is equivalent to increasing the number of filters for all convolutions to k, and masking out the extra channels (Fig. 3, Step C). $\mathbb{1}_i \in \mathbb{R}^k$ is a column vector with i leading 1s and $k - i$ trailing zeros. Note that the search method is invariant to the ordering of 1s and 0s. Since all blocks b_i have the same number of filters, we can approximate by sharing weights, so that $b_i = b$ (Fig. 3, Step D).

$$y = \sum_{i=1}^{k} g_i (b(x) \circ \mathbb{1}_i) \tag{2}$$

Finally, with this approximation, we can handle the computational complexity of the nave channel search approach: this is equivalent to computing the aggregate mask and running the block b only once (Fig. 3, Step E).

$$y = b(x) \circ \underbrace{\sum_{i=1}^{k} g_i \mathbb{1}_i}_{M} \tag{3}$$

This approximation only requires one forward pass and one feature map, inducing no additional FLOP or memory costs other than the negligible M term in Eq. 3 (Fig. 2, Channel Masking). Furthermore, the approximation falls short of equivalence only because weights are shared, which is shown to reduce train time and boost accuracy in DNAS [28]. This allows us to search the number of output channels for any block, including related architectural decisions such as the expansion rate in an inverted residual block.

3.2. Input Resolution Search

For spatial dimensions, we search over input resolutions. As with channels, previous DNAS methods would simply instantiate each block with every input resolution. This nave method's downfalls are twofold: increased memory cost and incompatible dimensions. As before, we address both issues directly by zero-padding the result. However, there are two caveats:

1. **Pixel misalignment**: means padding cannot occur navely as before. It would not make sense to zero-pad the periphery of the image, since the sum in Eq. 1 would result in misaligned pixels (Fig. 4, B). To handle pixel misalignment, we zero-pad such that zeros are interspersed spatially (Fig. 4, C). This zero-padding pattern is uniform; except for the zeros, this is a nearest neighbors upsampling. For example, a 2× increase in size would involve zero-padding every other row and column. Zero-padding instead of upsampling minimizes "pixel contamination" across input resolutions (Fig. 5).

2. **Receptive field misalignment**: Since subsets of the feature map correspond to different resolutions, navely convolving over the full feature map would result in a reduced receptive field (Fig. 4, D). To handle receptive field misalignment, we convolve over subsampled input instead. (Fig. 4, E). Using Gumbel Softmax, we arrive at "resolution subsampling" in Fig. 2.

NASNet [40] introduces a similar notion of combining hidden states. These combinations are also used to efficiently explore a combinatorially large search space but are used to determine – instead of input resolution or channels – the number of times to repeat a searched cell. With the above insights, the input resolution search thus incurs constant memory cost, regardless of the number of input resolutions. On the other hand, computational cost increases sub-linearly as the number of resolutions grows.

3.3. Effective Shape Propagation

Note this calculation for effective shape is only used during training. In our formulation of the weighted sum Eq. 1, the output y retains the maximum number of channels. However, there exists a non-integral number of *effective* channels: say a 16-channel output has Gumbel weight $g_i = 0.8$ and a 12-channel output has weight $g_i = 0.2$. This means the effective number of channels is $0.8 * 16 + 0.2 * 12 = 15.2$. These effective channels are necessary for both FLOP and parameter computation, as assigning higher weight to more channels should incur a larger cost penalty. This effective shape is how we realize effective resource costs introduced in previous works [33, 35]: First, define the gumbel softmax weights as

$$g_i^l = \frac{\exp[(\alpha_i^l + \epsilon_i^l)/\tau]}{\Sigma_i \exp[(\alpha_i^l + \epsilon_i^l)/\tau]} \tag{4}$$

with sampling parameter α, Gumbel noise ϵ, temperature τ. For a convolution with Gumbel Softmax in the l^{th} layer, we define its effective output shape $\bar{S}_{out}^l$ in Eq. 7 using effective output channel ($\bar{C}_{out}^l$, Eq. 5), and effective height, width ($\bar{h}_{out}^l, \bar{w}_{out}^l$, Eq. 6).

$$\bar{C}_{out}^l = \Sigma_i g_i^l \cdot C_{i,out}^l \tag{5}$$

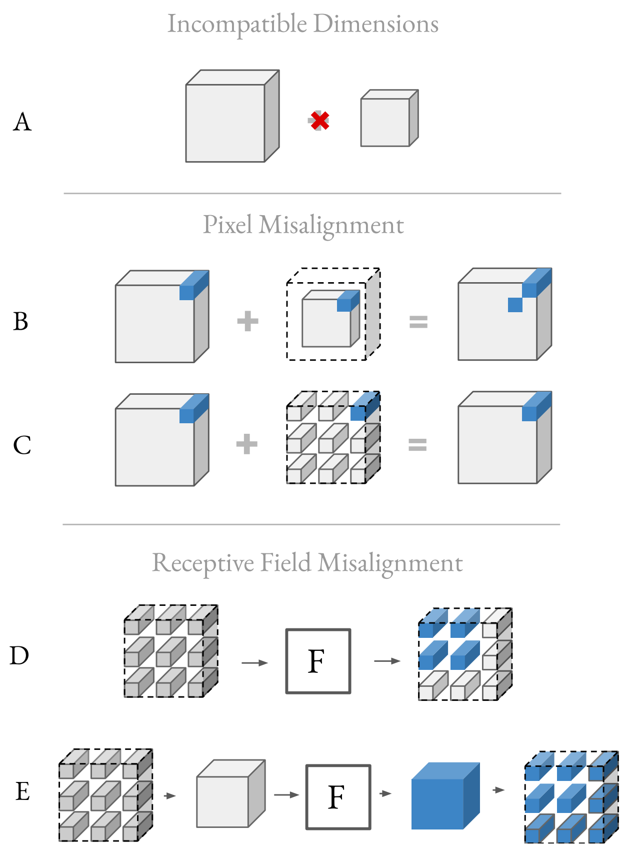

Figure 4: **Spatial Search Challenges**: **A**: Tensors with different spatial dimensions cannot be summed due to incompatible dimensions. **B**: Zero-padding along the periphery of the smaller feature map makes summing possible. However, the top-right pixels (blue) are not aligned correctly. **C**: Interspersing zero-padding spatially results in a sum that aligns pixels correctly. Note the top-right pixels of both feature maps are correctly overlapping in the sum. **D**: Say F is a convolution with 3×3 kernels. Convolving navely with the feature map, containing a subset (gray), results in reduced receptive field (2×2, blue) for the subset. **E**: To preserve receptive field for all searched input resolutions, the input must be subsampled before convolving. Note the receptive field (blue) is still 3×3. Furthermore, note we can achieve the same effect, without the need to construct a smaller tensor, with appropriately-strided dilated convolutions; we subsample to avoid modifying the operation F.

$$\bar{h}_{out}^l = \Sigma_i g_i^l \cdot \bar{h}_{in}^l, \bar{w}_{out}^l = \Sigma_i g_i^l \cdot \bar{w}_{in}^l \tag{6}$$

$$\bar{S}_{out}^l = (n, \bar{C}_{out}^l, \bar{h}_{out}^l, \bar{w}_{out}^l) \tag{7}$$

with batch size n, effective input width $\bar{w}_{in}$ and height $\bar{h}_{in}$.

For a convolution layer without a Gumbel Softmax, effective output shape simplifies to Eq. 8, where effective channel count is equal to actual channel count. For a depthwise convolution, effective output shape simplifies to Eq. 9,

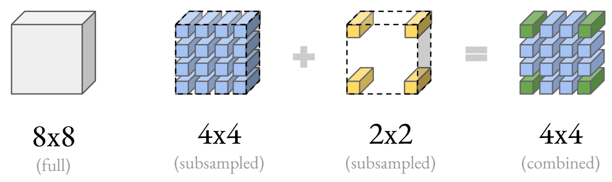

Figure 5: **Minimizing Pixel "Contamination"**: On the far left, we have the original 8×8 feature map. The blue 4×4 is a feature map subsampled with nearest neighbors and zero-padded uniformly. The yellow 2×2 is also subsampled and zero-padded. Summing the 2×2 with the 4×4 yields the combined feature map to the far right. Only the green pixels in the corners hold values from both feature map sizes; these green values are "contaminated" by the lower resolution feature maps.

where effective channel count is simply propagated.

$$\bar{C}^l_{out} = C^l_{out} \tag{8}$$

$$\bar{C}^l_{out} = \bar{C}^l_{in} \tag{9}$$

with actual output channel count C_{out}, effective input channel count $\bar{C}_{in}$. Then, we define the cost function for the l^{th} layer as follow:

$$\text{cost}^l = \begin{cases} k^2 \cdot \bar{h}^l_{out} \cdot \bar{w}^l_{out} \cdot \bar{C}^l_{in} \cdot \bar{C}^l_{out} \,/\, \gamma & \text{if } \texttt{FLOP} \\ k^2 \cdot \bar{C}^l_{in} \cdot \bar{C}^l_{out} \,/\, \gamma & \text{if } \texttt{param} \end{cases} \tag{10}$$

with γ convolution groups. The effective input channels for the $(l+1)^{th}$ layer are $\bar{C}^{l+1}_{in} = \bar{C}^l_{out}$. The total training loss consists of (1) cross-entropy loss and (2) total cost, which is the sum of cost from all layers: $\text{cost}_{total} = \Sigma_l \text{cost}^l$.

In the forward pass, for all convolutions, we calculate and return both the output tensor and effective output shape. Additionally, τ in the Gumbel Softmax Eq. 4 decreases throughout training, [16], forcing g^l to approach a one-hot distribution. $\text{argmax}_i g^l_i$ would thus select a path of blocks in the supergraph; a single channel and expansion rate option for each block; and a single input resolution for the entire network. This final architecture is then trained. Note this final model does not employ masking or require effective shapes.

4. Experiments

We use DMaskingNAS to search for convolutional network architectures under different objectives. We compare our search space, performance of searched models, and search cost to previously state-of-the-art networks. Detailed numerical results are listed in Table 4.

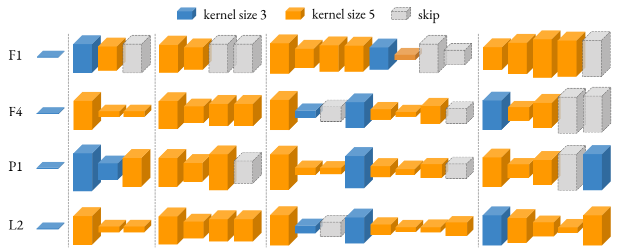

Figure 6: Searched FBNetV2 architectures, with colors denoting different kernel sizes and heights denoting different expansion rates. The heights are drawn to scale.

4.1. Experimental Setup

We implement DMaskingNAS using PyTorch on 8 Tesla V100 GPUs with 16GB memory. We use DMaskingNAS to search for convolutional neural networks on the ImageNet (ILSVRC 2012) classification dataset [4], a widely-used NAS evaluation benchmark. We use the same training settings as reported in [33]: we randomly select 10% of classes from the original 1000 classes and train the supergraph for 90 epochs. In each epoch, we train the network weights with 80% of training samples using SGD. We then train the Gumbel Softmax sampling parameter α with the remaining 20% using Adam [17]. We set initial temperature τ to 5.0 and exponentially anneal by $e^{-0.045} \approx 0.956$ every epoch.

4.2. Search Space

Previous cell-level searches produced fragmented, complicated, and latency-unfriendly blocks. Thus, we adopt a layer-wise search space for known, latency-friendly blocks.

Table 3 describes the micro-architecture search space: the block structure is inspired by [26, 11] and sequentially consists of a 1×1 point-wise convolution, a 3×3 or 5×5 depth-wise convolution, and another 1×1 point-wise convolution. Table 2 describes the macro-architecture. The search space contains more than 10^{35} candidate architectures, which is $10^{14}\times$ larger than DNAS's [33].

4.3. Memory Cost

Our memory optimizations yield a $\sim$1MB increase in memory cost for every 2 orders of magnitude the channel search space grows by; for context, this 1 MB increase is just 0.1% of the total memory cost during training. This is due to our feature map reuse as described in Sec. 3.1. We compare memory costs for DNAS and DMaskingNAS as the number of channel options increases (Fig. 7, left). With only 8 channel options for each convolution, DNAS fails to fit in memory during training, exceeding the 16GB memory supported by a Tesla V100 GPU. On the other hand, DMaskingNAS supports 32-option channel search, for a $32^{22} \sim 10^{33}$ in search space size (given our 22-layer search space), at nearly constant memory cost. Here, k-

Table 2: **Macro-architecture** for our largest search space, describing block type b, block expansion rate e, number of filters f, number of blocks n, stride of first block s. "TBS" means layer type needs to be searched. Tuples of three values represent the lowest value, highest, and steps between options (low, high, steps). The maximum input resolution for FBNetV2-P models is 288, for FBNetV2-F is 224, and for FBNetV2-L is 256. See supplementary material for all search spaces.

Max. Input	b	e	f	n	s
$256^2 \times 3$	3x3	1	16	1	2
$128^2 \times 16$	TBS	1	(12, 16, 4)	1	1
$128^2 \times 16$	TBS	(0.75, 3.25, 0.5)	(16, 28, 4)	1	2
$64^2 \times 28$	TBS	(0.75, 3.25, 0.5)	(16, 28, 4)	2	1
$64^2 \times 28$	TBS	(0.75, 3.25, 0.5)	(16, 40, 8)	1	2
$32^2 \times 40$	TBS	(0.75, 3.25, 0.5)	(16, 40, 8)	2	1
$32^2 \times 40$	TBS	(0.75, 3.75, 0.5)	(48, 96, 8)	1	2
$16^2 \times 96$	TBS	(0.75, 3.75, 0.5)	(48, 96, 8)	2	1
$16^2 \times 96$	TBS	(0.75, 4.5, 0.75)	(72, 128, 8)	4	1
$16^2 \times 128$	TBS	(0.75, 4.5, 0.75)	(112, 216, 8)	1	2
$8^2 \times 216$	TBS	(0.75, 4.5, 0.75)	(112, 216, 8)	3	1
$8^2 \times 216$	1x1	-	1984	1	1
$8^2 \times 1984$	avgpl	-	-	1	1
1984	fc	-	1000	1	-

Table 3: **Micro-architecture** search space for block design: non-linearities, kernel sizes, and Squeeze-and-Excite [13].

block type	kernel	squeeze-and-excite	non-linearity
ir_k3	3	N	relu
ir_k5	5	N	relu
ir_k3_hs	3	N	hswish
ir_k5_hs	5	N	hswish
ir_k3_se	3	Y	relu
ir_k5_se	5	Y	relu
ir_k3_se_hs	3	Y	hswish
ir_k5_se_hs	5	Y	hswish
skip	-	-	-

option channel search means that for each convolution with c channels, we search over $\{c/k, 2c/k, ..., c\}$ channels. To compare larger numbers of channel options, we reduce the number of blocks options in the search space (Fig. 7, right). To compute memory cost, we average the maximum memory allocated during each training step, across 10 epochs.

4.4. Search for ImageNet Models

FLOP-efficient models: We first use DMaskingNAS to find compact models (Fig. 6) for low computational budgets, with models ranging from 50 MFLOPs to 300 MFLOPs in Fig. 8. The searched FBNetV2s outperform all existing networks.

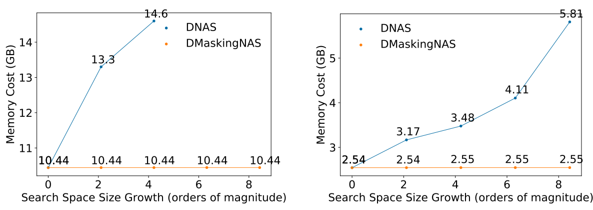

Figure 7: **Memory Cost of DNAS vs. DMaskingNAS** (Left) Conventional DNAS does not fit into memory with just 8 options per block in channel search. On the other hand, DMaskingNAS's memory cost remains roughly constant, even with 32 channel options per block. (Right) We reduce the number of block options in the search space to fit conventional DNAS into memory. The memory cost growth, as the search space increases, is significantly steeper than that of DMaskingNAS; in fact, DMaskingNAS's memory cost is nearly constant.

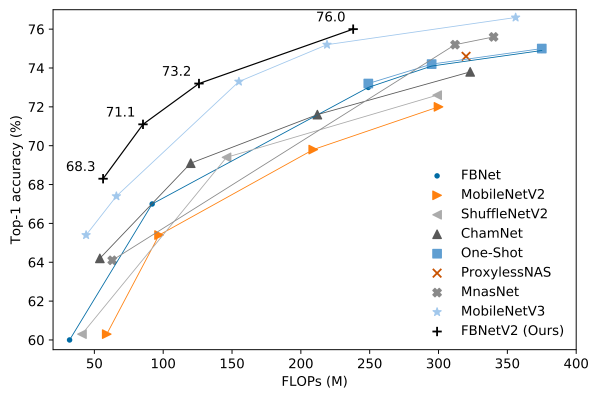

Figure 8: **ImageNet Accuracy vs. Model FLOPs.** We refer to these FLOP-efficient FBNetV2s as FBNetV2-F{1, 2, 3, 4} from left to right.

Storage-efficient models: Many real world scenarios face limited on-device storage space. Thus, we next perform searches for models minimizing parameter count, in Fig. 9. With similar or smaller model size (4M parameters), FBNetV2 achieves 2.6% and 2.9% absolute accuracy gains over MobileNetV3 [11] and FBNet [33], respectively.

Large models: We finally use DMaskingNAS to explore larger models for high-end devices. We compare FBNetV2-Large with networks of 300+ MFLOPs in Fig. 10.

5. Conclusions

We propose a memory-efficient algorithm, drastically expanding the search space for DNAS by supporting searches over spatial and channel dimensions. These contributions target the main bottleneck for DNAS – high memory cost that induces constraints on the search space size – and yield state-of-the-art performance.

Model	Search			FLOPs	Top-1
	Method	Space	Cost (GPU hours)		Acc (%)
MobileNetV2-0.35× [26]	manual	-	-	59M	60.3
ShuffleNetV2-0.5× [23]	manual	-	-	41M	60.3
MnasNet-0.35× [29]	RL	stage-wise	91K*	63M	64.1
ChamNet-E [3]	EA	stage-wise	28K†	54M	64.2
FBNet-0.35× [33]	gradient	layer-wise	0.2K	72M	65.3
MobileNetV3-Small [11]	RL/NetAdapt	stage-wise	>91K‡	66M	67.4
FBNetV2-F1 (ours)	**gradient**	**layer-wise**	**0.2K**	**56M**	**68.3**
MobileNetV2-1.0× [26]	manual	-	-	300M	72.0
ShuffleNetV2-1.5× [23]	manual	-	-	299M	72.6
DARTS [20]	gradient	cell	0.3K	595M	73.1
FBNetV2-F3 (ours)	**gradient**	**layer-wise**	**0.2K**	**126M**	**73.2**
ChamNet-B [3]	EA	stage-wise	28K†	323M	73.8
FBNet-B [33]	gradient	layer-wise	0.2K	295M	74.1
One-Shot NAS [6]	EA	layer-wise	0.3K	295M	74.2
ProxylessNAS [2]	gradient/RL	layer-wise	0.2K	320M	74.6
MobileNetV3-Large [11]	RL/NetAdapt	stage-wise	>91K‡	219M	75.2
MnasNet-A1 [29]	RL	stage-wise	91K*	312M	75.2
FBNetV2-F4 (ours)	**gradient**	**layer-wise**	**0.2K**	**238M**	**76.0**
ResNet-50 [9]	manual	-	-	4.1B	76.0
DenseNet-169 [14]	manual	-	-	3.5B	76.2
EfficientNet-B0 [30]	RL/scaling	stage-wise	>91K‡	390M	77.3
FBNetV2-L1 (ours)	**gradient**	**layer-wise**	**0.6K**	**325M**	**77.2**

Table 4: **ImageNet classification performance**: For baselines, we cite statistics on ImageNet from the original papers. Our results are bolded. *: The search cost is estimated based on the experimental setup in [29]. †: [3] discovers 5 models with the cost of training 240 networks. ‡: The cost estimation is a lower bound. [11] and [30] combines the approach proposed in [29] with [37] and compound scaling.

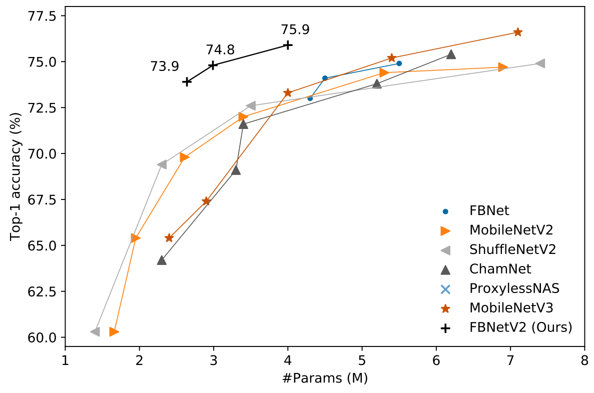

Figure 9: **ImageNet Accuracy vs. Model Size.** We refer to these as parameter-efficient FBNetV2s as FBNetV2-P{1, 2, 3} from left to right.

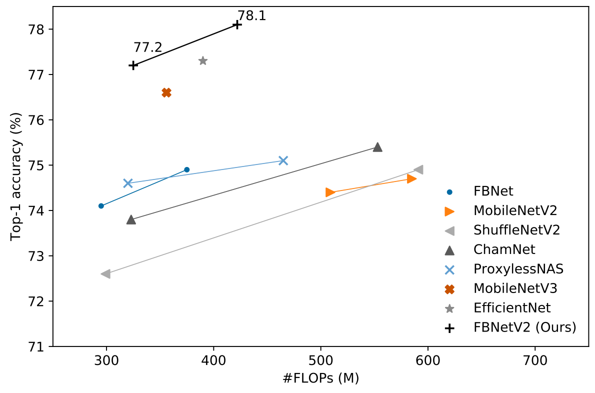

Figure 10: **ImageNet Accuracy vs. Model FLOPs for Large Models.** We refer to these large FBNetV2s as FBNetV2-L{1, 2} from left to right.

Acknowledgements In addition to NSF CISE Expeditions Award CCF-1730628, UC Berkeley research is supported by gifts from Alibaba, Amazon Web Services, Ant Financial, CapitalOne, Ericsson, Facebook, Futurewei, Google, Intel, Microsoft, Nvidia, Scotiabank, Splunk and VMware. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE 1752814.

References

[1] Deep learning performance guide. https://docs.nvidia.com/deeplearning/sdk/dl-performance-guide/index.html.

[2] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

[3] Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, et al. Chamnet: Towards efficient network design through platform-aware model adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11398–11407, 2019.

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[5] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

[6] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. *arXiv preprint arXiv:1904.00420*, 2019.

[7] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[8] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Proc. Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[10] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866*, 2018.

[11] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. *arXiv preprint arXiv:1905.02244*, 2019.

[12] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[13] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[14] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[15] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and $<$0.5 MB model size. *arXiv preprint arXiv:1602.07360*, 2016.

[16] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[17] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L. Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[19] Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. *arXiv preprint arXiv:1712.00559*, 2017.

[20] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[21] Ning Liu, Xiaolong Ma, Zhiyuan Xu, Yetang Wang, Jian Tang, and Jieping Ye. Autoslim: An automatic dnn structured pruning framework for ultra-high compression rates, 07 2019.

[22] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2736–2744, 2017.

[23] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. ShuffleNet V2: Practical guidelines for efficient CNN architecture design. *arXiv preprint arXiv:1807.11164*, 2018.

[24] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[25] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2902–2911. JMLR. org, 2017.

[26] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *arXiv preprint arXiv:1801.04381*, 2018.

[27] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-path nas: Designing hardware-efficient convnets in less than 4 hours. *arXiv preprint arXiv:1904.02877*, 2019.

[28] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-path nas: Device-aware efficient convnet design, 05 2019.

[29] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V Le. MnasNet: Platform-aware neural architec-

ture search for mobile. *arXiv preprint arXiv:1807.11626*, 2018.

[30] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

[31] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8612–8620, 2019.

[32] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Proc. Advances in Neural Information Processing Systems*, pages 2074–2082, 2016.

[33] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019.

[34] Bichen Wu, Forrest Iandola, Peter H Jin, and Kurt Keutzer. Squeezedet: ified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 129–137, 2017.

[35] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: Stochastic neural architecture search. *arXiv preprint arXiv:1812.09926*, 2018.

[36] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. *arXiv preprint arXiv:1611.05128*, 2016.

[37] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. NetAdapt: Platform-aware neural network adaptation for mobile applications. In *Proc. European Conf. Computer Vision*, volume 41, page 46, 2018.

[38] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. In *International Conference on Learning Representations*, 2019.

[39] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[40] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc Le. Learning transferable architectures for scalable image recognition. pages 8697–8710, 06 2018.

FBNetV2 Supplementary Materials

Model	Input	Flops	Top-1 (%)
FBNetV2-F1	128	56M	68.3
FBNetV2-F2	160	85M	71.1
FBNetV2-F3	192	126M	73.2
FBNetV2-F4	224	238M	76.0
FBNetV2-L1	224	325M	77.2
FBNetV2-L2	256	422M	78.1

Table 1: **ImageNet FLOP-efficient classification**: These are the FBNetV2 models yielded by DMaskingNAS optimizing for FLOP count and accuracy.

Model	Input	Params	Top-1 (%)
FBNetV2-P1	288	2.64M	73.9
FBNetV2-P2	288	2.99M	74.8
FBNetV2-P3	288	4.00M	75.9

Table 2: **ImageNet parameter-efficient classification**: These are the FBNetV2 models yielded by DMaskingNAS optimizing for parameter count and accuracy.

1. FBNetV2 on ImageNet

We include numeric results for all three categories of FBNetV2s, optimized for various resource constraints: FLOP-efficient FBNetV2-F and large FBNetV2-L in Table 1, parameter-efficient FBNetV2-P in Table 2. See the main manuscript for comparison with previously state-of-the-art results.

2. Macro-architecture Search Spaces

We list the DMaskingNAS macro-architecture search spaces for all three categories of FBNetV2s, optimized for various resource constraints: FLOP-efficient FBNetV2-F in Table 4, parameter-efficient FBNetV2-P in Table 5, and large FBNetV2-L in Table 3. Note that in all classes of models, the micro-architecture search space over blocks remains the same.

Table 3: Macro-architecture for our largest search space for **FBNetV2-L**, describing block type b, block expansion rate e, number of filters f, number of blocks n. "TBS" means layer type needs to be searched. Tuples of three values additionally represent steps between options (low, high, steps). The maximum input resolution for FBNetV2-L is 256.

Max. Input	b	e	f	n	s
$256^2 \times 3$	3x3	1	16	1	2
$128^2 \times 16$	TBS	1	(12, 16, 4)	1	1
$128^2 \times 16$	TBS	(0.75, 3.25, 0.5)	(16, 28, 4)	1	2
$64^2 \times 28$	TBS	(0.75, 3.25, 0.5)	(16, 28, 4)	2	1
$64^2 \times 28$	TBS	(0.75, 3.25, 0.5)	(16, 40, 8)	1	2
$32^2 \times 40$	TBS	(0.75, 3.25, 0.5)	(16, 40, 8)	2	1
$32^2 \times 40$	TBS	(0.75, 3.75, 0.5)	(48, 96, 8)	1	2
$16^2 \times 96$	TBS	(0.75, 3.75, 0.5)	(48, 96, 8)	2	1
$16^2 \times 96$	TBS	(0.75, 4.5, 0.75)	(72, 128, 8)	4	1
$16^2 \times 128$	TBS	(0.75, 4.5, 0.75)	(112, 216, 8)	1	2
$8^2 \times 216$	TBS	(0.75, 4.5, 0.75)	(112, 216, 8)	3	1
$8^2 \times 216$	1x1	-	1984	1	1
$8^2 \times 1984$	avgpl	-	-	1	1
1984	fc	-	1000	1	-

Table 4: Macro-architecture for our FLOP-efficient search space for **FBNetV2-F**. The maximum input resolution for FBNetV2-F is 224. See Table 3 for column names.

Max. Input	b	e	f	n	s
$224^2 \times 3$	3x3	1	16	1	2
$112^2 \times 16$	TBS	1	(12, 16, 4)	1	1
$112^2 \times 16$	TBS	(0.75, 4.5, 0.75)	(16, 24, 4)	1	2
$56^2 \times 24$	TBS	(0.75, 4.5, 0.75)	(16, 24, 4)	2	1
$56^2 \times 24$	TBS	(0.75, 4.5, 0.75)	(16, 40, 8)	1	2
$28^2 \times 40$	TBS	(0.75, 4.5, 0.75)	(16, 40, 8)	2	1
$28^2 \times 40$	TBS	(0.75, 4.5, 0.75)	(48, 80, 8)	1	2
$14^2 \times 80$	TBS	(0.75, 4.5, 0.75)	(48, 80, 8)	2	1
$14^2 \times 80$	TBS	(0.75, 4.5, 0.75)	(72, 112, 8)	3	1
$14^2 \times 112$	TBS	(0.75, 4.5, 0.75)	(112, 184, 8)	1	2
$7^2 \times 184$	TBS	(0.75, 4.5, 0.75)	(112, 184, 8)	3	1
$7^2 \times 184$	1x1	-	1984	1	1
$7^2 \times 1984$	avgpl	-	-	1	1
1984	fc	-	1000	1	-

Table 5: Macro-architecture for our parameter-efficient search space for **FBNetV2-P**. The maximum input resolution for FBNetV2-P is 288. See Table 3 for column names.

Max. Input	b	e	f	n	s
$288^2 \times 3$	3x3	1	32	1	2
$144^2 \times 16$	TBS	1	(16, 28, 4)	1	1
$144^2 \times 28$	TBS	(0.75, 4.5, 0.75)	(16, 40, 4)	1	2
$72^2 \times 40$	TBS	(0.75, 4.5, 0.75)	(16, 40, 4)	2	1
$72^2 \times 40$	TBS	(0.75, 4.5, 0.75)	(16, 48, 8)	1	2
$36^2 \times 48$	TBS	(0.75, 4.5, 0.75)	(16, 48, 8)	2	1
$36^2 \times 48$	TBS	(0.75, 4.5, 0.75)	(48, 96, 8)	1	2
$18^2 \times 96$	TBS	(0.75, 4.5, 0.75)	(48, 96, 8)	2	1
$18^2 \times 96$	TBS	(0.75, 4.5, 0.75)	(72, 128, 8)	4	1
$18^2 \times 128$	TBS	(0.75, 4.5, 0.75)	(112, 216, 8)	1	2
$9^2 \times 216$	TBS	(0.75, 4.5, 0.75)	(112, 216, 8)	3	1
$9^2 \times 216$	TBS	(0.75, 4.5, 0.75)	(112, 216, 8)	1	1
$9^2 \times 216$	1x1	-	1280	1	1
$9^2 \times 1280$	avgpl	-	-	1	1
1280	fc	-	1000	1	-