AtomNAS: Fine-Grained End-to-End Neural Architecture Search

Anonymous authors
Paper under double-blind review

Abstract

Designing of search space is a critical problem for neural architecture search (NAS) algorithms. We propose a fine-grained search space comprised of atomic blocks, a minimal search unit much smaller than the ones used in recent NAS algorithms. This search space facilitates direct selection of channel numbers and kernel sizes in convolutions. In addition, we propose a resource-aware architecture search algorithm which dynamically selects atomic blocks during training. The algorithm is further accelerated by a dynamic network shrinkage technique. Instead of a search-and-retrain two-stage paradigm, our method can simultaneously search and train the target architecture in an end-to-end manner. Our method achieves state-of-the-art performance under several FLOPs configurations on ImageNet with a negligible searching cost.

1 Introduction

Human-designed neural networks are already surpassed by machine-designed ones. Neural Architecture Search (NAS) has become the mainstream approach to discover efficient and powerful network structures (Zoph & Le (2017); Pham et al. (2018); Tan et al. (2019); Liu et al. (2019a)). Although the tedious searching process is conducted by machines, humans still involve extensively in the design of the NAS algorithms. Designing of search spaces is critical for NAS algorithms and different choices have been explored. Cai et al. (2019) and Wu et al. (2019) utilize supernets with multiple choices in each layer to accommodate a sampled network on the GPU. Chen et al. (2019b) progressively grow the depth of the supernet and remove unnecessary blocks during the search. Tan & Le (2019a) propose to search the scaling factor of image resolution, channel multiplier and layer numbers in scenarios with different computation budgets. Stamoulis et al. (2019) propose to use different kernel sizes in each layer of the supernet and reuse the weights of larger kernels for small kernels. Howard et al. (2019); Tan & Le (2019b) adopts Inverted Residuals with Linear Bottlenecks (MobileNetV2 block) (Sandler et al., 2018), a building block with light-weighted depth-wise convolutions for highly efficient networks in mobile scenarios.

However, the proposed search spaces generally have only a small set of choices for each block. DARTS and related methods (Liu et al., 2019a; Chen et al., 2019b; Liang et al., 2019) use around 10 different operations between two network nodes. Howard et al. (2019); Cai et al. (2019); Wu et al. (2019); Stamoulis et al. (2019) search the expansion ratios in the MobileNetV2 block but still limit them to a few discrete values. We argue that more fine-grained search space is essential to find optimal neural architectures. Specifically, the searched building block in a supernet should be as small as possible to generate the most diversified model structures.

We revisit the architectures of state-of-the-art networks (Howard et al. (2019); Tan & Le (2019b); He et al. (2016)) and find a commonly used building block: convolution - channel-wise operation - convolution. We reinterpret such structure as an ensemble of computationally independent blocks, which we call *atomic blocks*. This new formulation enables a much larger and more fine-grained search space. Starting from a supernet which is built upon atomic blocks, the search for exact channel numbers and various operations can be achieved by selecting a subset of the atomic blocks.

For the efficient exploration of the new search space, we propose a NAS algorithm named AtomNAS to conduct architecture search and network training simultaneously. Specifically, an importance factor is introduced to each atomic block. A penalty term proportional to the computation cost of the atomic block is enforced on the network. By jointly learning the importance factors along with the weights of the network, AtomNAS selects the atomic blocks which contribute to the model capacity with relatively small computation cost.

Training on large supernets is computationally demanding. We observe that the scaling factors of many atomic blocks permanently vanish at the early stage of model training. We propose a dynamic network shrinkage technique which removes the ineffective atomic blocks on the fly and greatly reduce the computation cost of AtomNAS.

In our experiment, our method achieves 75.9% top-1 accuracy on ImageNet dataset around 360M FLOPs, which is 0.9% higher than state-of-the-art model (Stamoulis et al., 2019). By further incorporating additional modules, our method achieves 77.6% top-1 accuracy. It outperforms MixNet by 0.6% using 363M FLOPs, which is a new state-of-the-art under the mobile scenario.

In summary, the major contributions of our work are:

1. We propose a fine-grained search space which includes the exact number of channels and mixed operations (e.g., combination of different convolution kernels).
2. We propose an efficient end-to-end NAS algorithm named AtomNAS which can simultaneously search the network architecture and train the final model. No finetuning is needed after AtomNAS finishes.
3. With the proposed search space and AtomNAS, we achieve state-of-the-art performance on ImageNet dataset under mobile setting.

2 Related Work

2.1 Neural Architecture Search

Recently, there is a growing interest in automated neural architecture design. Reinforce learning based NAS methods (Zoph & Le, 2017; Tan et al., 2019; Tan & Le, 2019b;a) are usually computational intensive, thus hampering its usage with limited computational budget. To accelerate the search procedure, ENAS (Pham et al., 2018) represents the search space using a directed acyclic graph and aims to search the optimal subgraph within the large supergraph. A training strategy of parameter sharing among subgraphs is proposed to significantly increase the searching efficiency. The similar idea of optimizing optimal subgraphs within a supergraph is also adopted by Liu et al. (2019a); Wu et al. (2019); Guo et al. (2019); Cai et al. (2019). A prominent disadvantage of the above methods is their coarse search spaces only include limited categories of properties, e.g. kernel size, expansion ratio, the number of layer, etc. Because of the restriction of search space, it is difficult to learn optimal architectures under computational resource constraints. On the contrary, our method proposes the fine-grained search space to enable searching more flexible network architectures under various resource constraints.

2.2 Network pruning

Assuming that many parameters in the network are unnecessary, network pruning methods start from a computation-intensive model, identify the unimportant connections and remove them to get a compact and efficient network. Early method (Han et al., 2016) simultaneously learns the important connections and weights. However, non-regularly removing connections in these works makes it hard to achieve theoretical speedup ratio on realistic hardwares due to extra overhead in caching and indexing. To tackle this problem, structured network pruning methods (He et al., 2017b; Liu et al., 2017; Luo et al., 2017; Ye et al., 2018; Gordon et al., 2018) are proposed to prune structured components in networks, e.g. the entire channel and kernel. In this way, empirical acceleration can be achieved on modern computing devices. Liu et al. (2017); Ye et al. (2018); Gordon et al. (2018) encourage channel-level sparsity by imposing the L-1 regularizer on the channel dimension. Recently, Liu et al. (2019b) show that in structured network pruning, the learned weights are unimportant. This suggests structured network pruning is actually a neural architecture search focusing on channel numbers. Our method jointly searches the channel numbers and a mix of operations, which is a much larger search space.

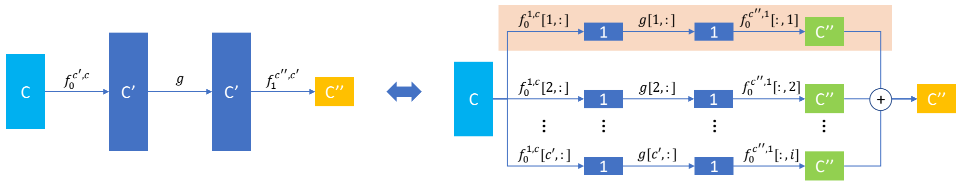


Figure 1: Illustration of the ensemble perspective. Arrow means operators. The structure of two convolutions joined by a channel-wise operation is mathematically equivalent to the ensemble of multiple atomic blocks, according to Eq. (2). Colored rectangles represent tensors, with numbers inside indicating their channel numbers; The shaded path on the right is one example of atomic block.

3 AtomNAS

We formulate our neural architecture search method in a fine-grained search space with the atomic block used as the basic search unit. An atomic block is comprised of two convolutions connected by a channel-wise operation. By stacking atomic blocks, we obtain larger building blocks (*e.g.* residual block and MobileNetV2 block proposed in a variety of state-of-the-art models including ResNet, MobileNet V2/V3 (He et al., 2016; Howard et al., 2019; Sandler et al., 2018). In Section 3.1, We first show larger network building blocks (*e.g.* MobileNetV2 block) can be represented by an ensembles of atomic blocks. Based on this view, we propose a fine-grained search space using atomic blocks. In Section 3.2, we propose a resource-aware atomic block selection method for end-to-end architecture search. Finally, we propose a dynamic network shrinkage technique in Section 3.3, which greatly reduces the search cost.

3.1 Fine-Grained Search Space

Under the typical block-wise NAS paradigm (Tan et al., 2019; Tan & Le, 2019b), the search space of each block in a neural network is represented as the Cartesian product $\mathcal{C} = \prod_{i=1} \mathcal{P}_i$, where each $\mathcal{P}_i$ is the set of all choices of the i-th configuration such as kernel size, number of channels and type of operation. For example, $\mathcal{C} = \{\text{conv}, \text{depth-wise conv}, \text{dilated conv}\} \times \{3, 5\} \times \{24, 32, 64, 128\}$ represents a search space of three types of convolutions by two kernel sizes and four options of channel number. A block in the resulting model can only pick one convolution type from the three and one output channel number from the four values. This paradigm greatly limits the search space due to the few choices of each configuration. Here we present a more fine-grained search space by decomposing the network into smaller and more basic building blocks.

We denote $f^{c',c}(X)$ as a convolution operator, where X is the input tensor and c, c' are the input and output channel numbers respectively. A wide range of manually-designed and NAS architectures share a structure that joins two convolutions by a channel-wise operation:

$$Y = \left(f_1^{c'',c'} \circ g \circ f_0^{c',c}\right)(X) \tag{1}$$

where g is a channel-wise operator. For example, in VGG (Simonyan & Zisserman, 2015) and a Residual Block (He et al., 2016), f_0 and f_1 are convolutions and g is one of Maxpool, ReLU and BN-ReLU; in a MobileNetV2 block (Sandler et al., 2018), f_0 and f_1 are point-wise convolutions and g is depth-wise convolution with BN-ReLU in the MobileNetV2 block. Eq. (1) can be reformulated as follows:

$$Y = \sum_{i=1}^{c'} \left(f_1^{c'',1}[i,:] \circ g[i,:] \circ f_0^{1,c}[:,i]\right)(X), \tag{2}$$

where $f_0^{1,c}[:,i]$ is the i-th convolution kernel of f_0, $g[i,:]$ is the operator of the i-th channel of g, and $\{f_1^{c'',1}[i,:]\}_{i=1}^{c'}$ are obtained by splitting the kernel tensor of f_1 along the the input channel dimension. Each term in the summation can be seen as a computationally independent block, which is called *atomic block*. Fig. (1) demonstrate this reformulation. By determining whether to keep each atomic block in the final model individually, the search of channel number c' is enabled through channel selection, which greatly enlarges the search space.

This formulation also naturally includes the selection of operators. To gain a better understanding, we first generalize Eq. (2) as:

$$Y = \sum_{i=1}^{c'} \left(f_{1i}^{c'',1} \circ g_i \circ f_{0i}^{1,c} \right)(X). \tag{3}$$

Note the array indices i are moved to subscripts. In this formulation, we can use different types of operators for f_{0i}, f_{1i} and g_i; in other words, f_0, f_1 and g can each be a combination of different operators and each atomic block can use different operators such as convolution with different kernel sizes.

Formally, the search space is formulated as a supernet which is built based on the structure in Eq. (1); such structure satisfies Eq. (3) and thus can be represented by atomic blocks; each of f_0, f_1 and g is a combination of operators. The new search space includes some state-of-the-art network architectures. For example, by allowing g to be a combination of convolutions with different kernel sizes, the MixConv block in MixNet (Tan & Le, 2019b) becomes a special case in our search space. In addition, our search space facilitates discarding any number of channels in g, resulting in a more fine-grained channel configuration. In comparison, the channels numbers are determined heuristically in Tan & Le (2019b).

3.2 RESOURCE-AWARE ATOMIC BLOCK SEARCH

In this work, we adopt a differentiable neural architecture search paradigm where the model structure is discovered in a full pass of model training. With the supernet defined above, the final model can be produced by discarding part of the atomic blocks during training. Following DARTS (Liu et al. (2019a)), we introduce a scaling factor α to scale the output of each atomic block in the supernet. Eq. (3) then becomes

$$Y = \sum_{i=1}^{c'} \alpha_i \left(f_{1i}^{c'',1} \circ g_i \circ f_{0i}^{1,c} \right)(X). \tag{4}$$

Here, each α_i is tied with an atomic block comprised of three operators $f_{1i}^{c'',1}$,g_i and $f_{0i}^{1,c}$. The scaling factors are learned jointly with the network weights. Once the training finishes, the atomic blocks with factors smaller than a threshold are discarded.

We still need to address two issues related to the factor α. First, where should we put them in the supernet? The scaling parameters in the BN layers can be directly used as such scaling factors(Liu et al. (2017)). In most cases, g contains at least one BN layer and we use the scaling parameters of the last BN layer in g as α. If g has no BN layers, which is rare, we can place α anywhere between f_0 and f_1, as long as we apply regularization terms to the weights of f_0 and f_1 (e.g., weight decays) in order to prevent weights in f_0 and f_1 from getting too large and canceling the effect of α.

The second issue is how to avoid performance deterioration after discarding some of the atomic blocks. For example, DARTS discards operations with small scale factors after iterative training of model parameters and scale factors. Since the scale factors of the discarded operations are not small enough, the performance of the network will be affected which needs re-training to adjust the weights again. In order to maintain the performance of the supernet after dropping some atomics blocks, the scaling factors α of those atomic blocks should be sufficiently small. Inspired by the channel pruning work in Liu et al. (2017), we add L1 norm penalty loss on α, which effectively pushes many scaling factors to near-zero values. At the end of learning, atomic blocks with α close to zero are removed from the supernet. Note that since the BN scales change more dramatically during training due to the regularization term, the running statistics of BNs might be inaccurate and needs to be calculated again using the training set.

With the added regularization term, the training loss is

$$\mathcal{L} = \mathcal{E} + \lambda \sum_{i \in \mathcal{S}} c_i |\alpha_i|, \tag{5}$$

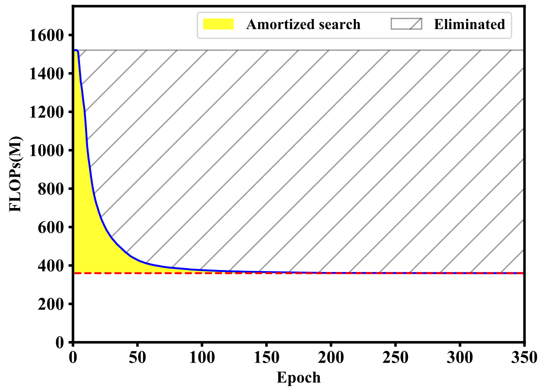


Figure 2: FLOPs change of the supernet during the searching and training for AtomNAS-C. The crossed-out region corresponds to the saved computation compared to training the supernet without the dynamic shrinkage. The region in yellow corresponds to the extra cost compared with training the final model from scratch, the cost of which is the region below the red dashed line.

```
Initialize the supernet and the exponential moving average;
while $epoch \leq max\_epoch$ do
    Update network weights and scaling factors $\alpha$ by minimizing the loss function $\mathcal{L}$ ;
    Update the $\hat{\alpha}$ by Eq. (7);
    if Total FLOPs of dead blocks $\geq \Delta$ then
        Remove dead blocks from the supernet;
    end
    Recalculate BN's statistics by forwarding some training examples;
    Validate the performance of the current supernet;
end
```

Algorithm 1: Dynamic network shrinkage

$$c_i = \hat{c}_i / \sum_{k \in \mathcal{S}} \hat{c}_k \tag{6}$$

where λ is the coefficient of L1 penalty term, $\mathcal{S}$ is the index set of all atomic blocks, and $\mathcal{E}$ is the conventional training loss (e.g. cross-entropy loss combined with the weight decay term). $|\alpha_i|$ is weighted by coefficient c_i which is proportional to the computation cost of i-th atomic block, i.e. $\hat{c}_i$. By using computation costs aware regularization, we encourage the model to learn network structures that strike a good balance between accuracy and efficiency. In this paper, we use FLOPs as the criteria of computation cost. Other metrics such as latency and energy consumption can be used similarly. As a result, the whole loss function $\mathcal{L}$ trades off between accuracy and FLOPs.

3.3 Dynamic Network Shrinkage

Usually, the supernet is much larger than the final search result. We observe that many atomic blocks become "dead" starting from the early stage of the search, i.e., their scaling factors α are close to zero till the end of the search. To utilize computational resources more efficiently and speed up the search process, we propose a dynamic network shrinkage algorithm which cuts down the network architecture by removing atomic blocks once they are deemed "dead".

We adopt a conservative strategy to decide whether an atomic block is "dead": for scaling factors α, we maintain its momentum $\hat{\alpha}$ which is updated as

$$\hat{\alpha} \leftarrow \beta\hat{\alpha} + (1-\beta)\alpha^t, \tag{7}$$

where α^t is the scaling factors at t-th iteration and β is the decay term. An atomic block is considered "dead" if both $\hat{\alpha}$ and α^t are smaller than a threshold, which is set to 1e-3 throughout experiments.

Once the total FLOPs of "dead" blocks reach a predefined threshold, we remove those blocks from the supernet. As discussed above, we recalculate BN's running statistics before deploying the network. The whole training process is presented in Algorithm 1.

We show the FLOPs of a sample network during the search process in Fig. 2. We start from a supernet with 1521M FLOPs and dynamically discard "dead" atomic blocks to reduce search cost. The overall search and train cost only increases by 17.2% compared to that of training the searched model from scratch.

4 Experiment

We first describe the implementation details in Section 4.1 and then compare AtomNAS with previous state-of-the-art methods under various FLOPs constraints in Section 4.2. Finally, we provide more analysis about AtomNAS in Section 4.3.

4.1 Implementation Details

The picture on the left of Fig. 3 illustrates a search block in the supernet. Within this search block, f_0 is a 1×1 pointwise convolutions that expands the input channel number from C to $3 \times 6C$; g is a mix of three depth-wise convolutions with kernel sizes of 3×3, 5×5 and 7×7, and f_1 is another 1×1 pointwise convolutions that projects the channel number to the output channel number. Similar to Sandler et al. (2018), if the output dimension stays the same as the input dimension, we use a skip connection to add the input to the output. In total, there are $3 \times 6C$ atomic blocks in the search block. The overall architecture of the supernet is shown in the table on the right of Fig. 3. The supernet has 21 search blocks.

We use the same training configuration (e.g., RMSProp optimizer, EMA on weights and exponential learning rate decay) as Tan et al. (2019); Stamoulis et al. (2019) and do not use extra data augmentation such as MixUp (Zhang et al., 2018) and AutoAugment (Cubuk et al., 2018). We find that using this configuration is sufficient for our method to achieve good performance. Our results are shown in Table 1 and Table 3. When training the supernet, we use a total batch size of 2048 on 32 Tesla V100 GPUs and train for 350 epochs. For our dynamic network shrinkage algorithm, we set the momentum factor β in Eq. (7) to 0.9999. At the beginning of the training, all of the weights are randomly initialized. To avoid removing atomic blocks with high penalties (i.e., FLOPs) prematurely, the penalty term in Eq. (5) is increased from 0 to the target λ by a linear scheduler during the first 25 epochs. By setting the weight of the L1 penalty term λ to be 1.8×10^{-4}, 1.2×10^{-4} and 1.0×10^{-4} respectively, we obtain networks with three different sizes: AtomNAS-A, AtomNAS-B, and AtomNAS-C. They have the similar FLOPs as previous state-of-the-art networks under 400M: MixNet-S (Tan & Le, 2019b), MixNet-M (Tan & Le, 2019b) and SinglePath (Stamoulis et al., 2019).

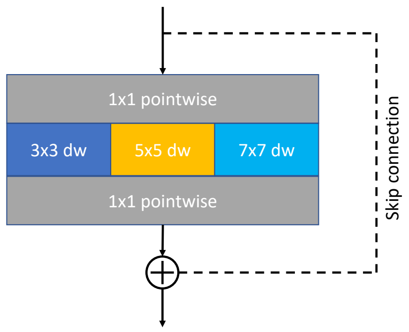
4.2 Experiments on ImageNet

We apply AtomNAS to search high performance light-weight model on ImageNet 2012 classification task (Deng et al., 2009). Table 1 compares our methods with previous state-of-the-art models, either manually designed or searched.

With models directly produced by AtomNAS, our method achieves the new state-of-the-art under all FLOPs constraints. Especially, AtomNAS-C achieves 75.9% top-1 accuracy with only 360M FLOPs, and surpasses all other models, including models like PDARTS and DenseNAS which have much higher FLOPs.

Techniques like Swish activation function (Ramachandran et al., 2018) and Squeeze-and-Excitation (SE) module (Hu et al., 2018) consistently improve the accuracy with marginal FLOPs cost. For a fair comparison with methods that use these techniques, we directly modify the searched network by replacing all ReLU activation with Swish and add SE module with ratio 0.5 to every block and then retrain the network from scratch. Note that unlike other methods, we do not search the configuration of Swish and SE, and therefore the performance might not be optimal. Extra data augmentations such as MixUp and AutoAugment are still not used. We train the models from scratch with a total batch size of 4096 on 32 Tesla V100 GPUs for 250 epochs.

Simply adding these techniques improves the results further. AtomNAS-A+ achieves 76.3% top-1 accuracy with 260M FLOPs, which outperforms many heavier models including MnasNet-A2. It performs as well as Efficient-B0 (Tan & Le, 2019a) by using 130M less FLOPs and without extra data augmentations. It also outperforms the previous state-of-the-art MixNet-S by 0.5%. In



Input Shape	Block	f	n	stride
$224^2 \times 3$	3x3 conv	32(16)	1	2
$112^2 \times 32(16)$	3x3 MB	16	1	1
$112^2 \times 16$	searchable	24	4	2
$56^2 \times 24$	searchable	40	4	2
$28^2 \times 40$	searchable	80	4	2
$14^2 \times 80$	searchable	96	4	1
$14^2 \times 96$	searchable	192	4	2
$7^2 \times 192$	searchable	320	1	1
$7^2 \times 320$	avgpool	-	1	1
1280	fc	1000	1	-

Figure 3: (a) The searchable block of the supernet. f_0 and f_1 are fixed to 1×1 pointwise convolutions; g here is a mix of three convolutions with kernel sizes of 3×3, 5×5 and 7×7. f_0 expands the input channel number from C to $18C$ and f_1 projects the channel number to the output channel number. If the output dimension stays the same as the input dimension, we use a skip connection to add the input to the output. (b) Architecture of the supernet. Column-Block denotes the block type; MB denotes MobileNetV2 block; "searchable" means a searchable block shown on the left. Column-f denotes the output channel number of a block. Column-n denotes the number of blocks. Column-s denotes the stride of the first block in a stage. The output channel numbers of the first convolution are 16 for AtomNAS-A, 32 for AtomNAS-B and AtomNAS-C.

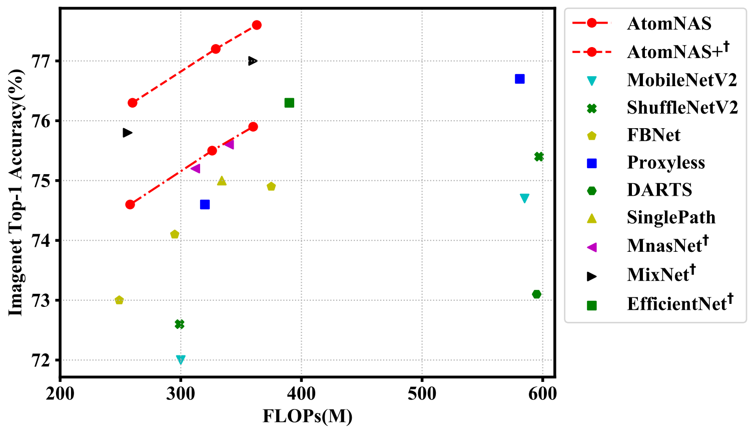


Figure 4: FLOPs versus accuracy on ImageNet. † means methods use extra techniques like Swish activation and Squeeze-and-Excitation module.

addition, AtomNAS-C+ improves the top-1 accuracy on ImageNet to 77.6%, surpassing previous state-of-the-art MixNet-M by 0.6% and becomes the overall best performing model under 400M FLOPs.

Fig. 4 visualizes the top-1 accuracy on ImageNet for different models. It's clear that our fine-grained search space and the end-to-end resource-aware search method boost the performance significantly.

4.3 Analysis

4.3.1 Visualization

We plot the structure of the searched architecture AtomNAS-C in Fig. 5, from which we see more flexibility of channel number selection, not only among different operators within each block, but also across the network. In Fig. 6a, we visualize the ratio between atomic blocks with different kernel sizes in all 21 search blocks. First, we notice that all search blocks have convolutions of all three kernel sizes, showing that AtomNAS learns the importance of using multiple kernel sizes in network architecture. Another observation is that AtomNAS tends to keep more atomic blocks at

Table 1: Comparision with state-of-the-arts on ImageNet under the mobile setting. † denotes methods using extra network modules such as Swish activation and Squeeze-and-Excitation module. ‡ denotes using extra data augmentation such as MixUp and AutoAugment. * denotes models searched and trained simultaneously.

Model	Parameters	FLOPs	Top-1(%)	Top-5(%)
MobileNetV1 (Howard et al., 2017)	4.2M	575M	70.6	89.5
MobileNetV2 (Sandler et al., 2018)	3.4M	300M	72.0	91.0
MobileNetV2 (our impl.)	3.4M	301M	73.6	91.5
MobileNetV2 (1.4)	6.9M	585M	74.7	92.5
ShuffleNetV2 (Ma et al., 2018)	3.5M	299M	72.6	-
ShuffleNetV2 2×	7.4M	591M	74.9	-
FBNet-A (Wu et al., 2019)	4.3M	249M	73.0	-
FBNet-C	5.5M	375M	74.9	-
Proxyless (mobile) (Cai et al., 2019)	4.1M	320M	74.6	92.2
SinglePath (Stamoulis et al., 2019)	4.4M	334M	75.0	92.2
NASNet-A (Zoph & Le, 2017)	5.3M	564M	74.0	91.6
DARTS (second order) (Liu et al., 2019a)	4.9M	595M	73.1	-
PDARTS (cifar 10) (Chen et al., 2019b)	4.9M	557M	75.6	92.6
DenseNAS-A (Fang et al., 2019)	7.9M	501M	75.9	92.6
FairNAS-A (Chu et al., 2019b)	4.6M	388M	75.3	92.4
AtomNAS-A*	3.9M	258M	74.6	92.1
AtomNAS-B*	4.4M	326M	75.5	92.6
AtomNAS-C*	4.7M	360M	75.9	92.7
SCARLET-A† (Chu et al., 2019a)	6.7M	365M	76.9	93.4
MnasNet-A1† (Tan et al., 2019)	3.9M	312M	75.2	92.5
MnasNet-A2†	4.8M	340M	75.6	92.7
MixNet-S† (Tan & Le, 2019b)	4.1M	256M	75.8	92.8
MixNet-M†	5.0M	360M	77.0	93.3
EfficientNet-B0†‡ (Tan & Le, 2019a)	5.3M	390M	76.3	93.2
SE-DARTS+†‡ (Liang et al., 2019)	6.1M	594M	77.5	93.6
AtomNAS-A+†	4.7M	260M	76.3	93.0
AtomNAS-B+†	5.5M	329M	77.2	93.5
AtomNAS-C+†	5.9M	363M	77.6	93.6

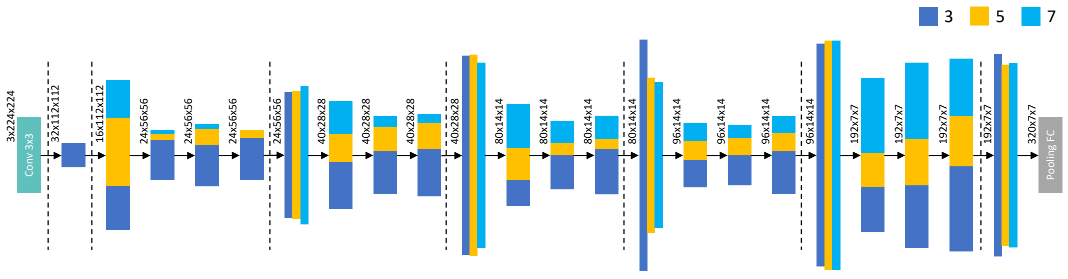
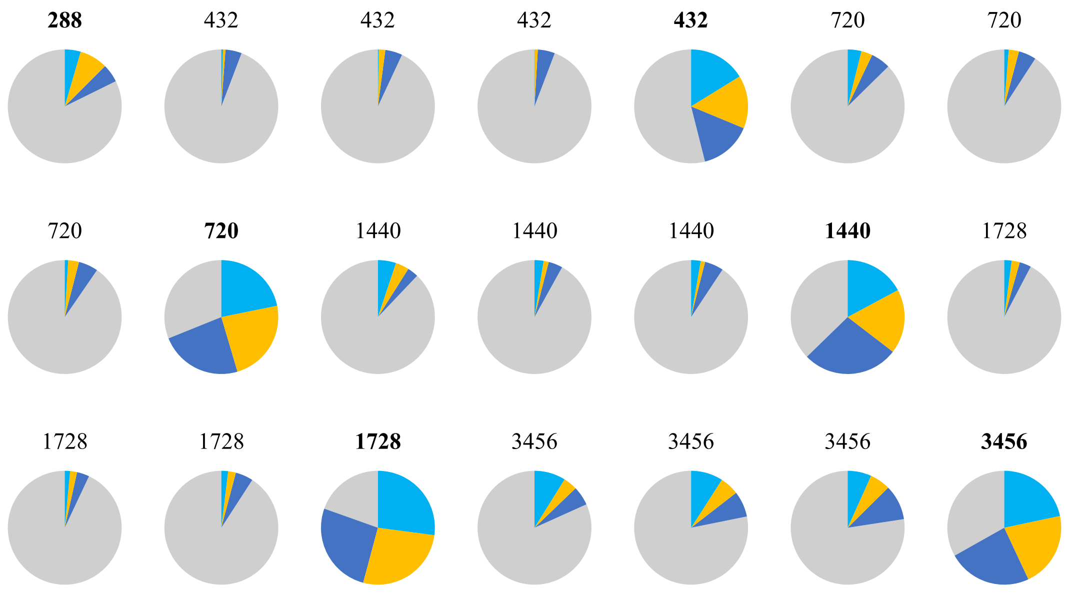
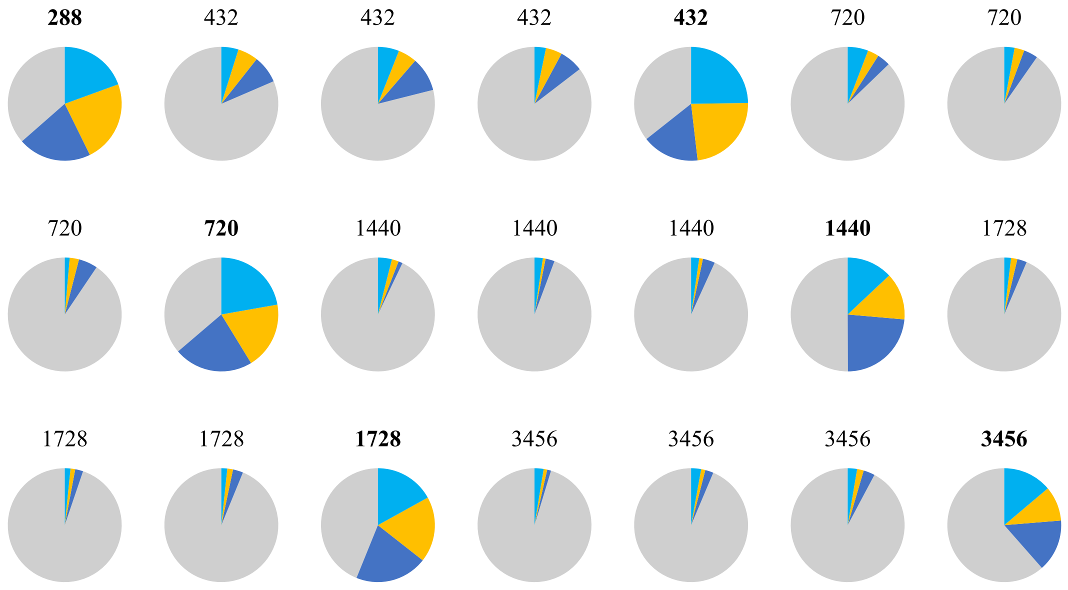


Figure 5: The architecture of AtomNAS-C. Blue, orange, cyan blocks denote atomic blocks with kernel size 3, 5 and 7 respectively; the heights of these blocks are proportional to their expand ratios.

the later stage of the network. This is because in earlier stage, convolutions of the same kernel size costs more FLOPs; AtomNAS is aware of this (thanks to its resource-aware regularization) and try to keep as less as possible computationally costly atomic blocks.



(a) AtomNAS-C



(b) Baseline

Figure 6: Ratio of different types of atomic blocks in all 21 searchable blocks. The text above each pie tells the total number of atomic blocks of the corresponding block in the original supernet. Grey denotes dead atomic blocks; blue, orange, and cyan represent atomic blocks using depth-wise convolutions with kernel size $3, 5, 7$ respectively. Blocks without skip connection are highlighted by bold text. (a) Visualization for AtomNAS-C. (b) Visualization for baseline (i.e., without FLOPs related coefficients c_i).

4.3.2 Resource-Aware Regularization

To demonstrate the effectiveness of the resource-aware regularization in Section 3.2, we compare it with a baseline without FLOPs-related coefficients c_i, which is widely used in network pruning (Liu et al., 2017; He et al., 2017b). Table 2 shows the results. First, by using the same L1 penalty coefficient $\lambda = 1.0 \times 10^{-4}$, the baseline achieves a network with similar performance but using much more FLOPs; then by increasing λ to 1.5×10^{-4}, the baseline obtain a network which has similar FLOPs but inferior performance (i.e., about 1.0% lower). In Fig. 6b we visualized the ratio of different types of atomic blocks of the baseline network obtained by $\lambda = 1.5 \times 10^{-4}$. The baseline network keeps more atomic blocks in the earlier blocks, which have higher computation cost due to higher input resolution. On the contrary, AtomNAS is aware of the resource constraint, thus keeping more atomic blocks in the later blocks and achieving much better performance.

Table 2: Influence of awareness of resource metric. The upper block uses equal penalties for all atomic blocks. The lower part uses our resource-aware atomic block selection.

λ	FLOPs	Top-1(%)
1.0×10^{-4}	445M	76.1
1.5×10^{-4}	370M	74.9
1.0×10^{-4}	360M	75.9

4.3.3 BN Recalibration

As the BN's running statistics might be inaccurate as explained in Section 3.2 and Section 3.3, we re-calculate the running statistics of BN before inference, by forwarding 131k randomly sampled training images through the network. Table 3 shows the impact of the BN recalibration. The top-1 accuracies of AtomNAS-A, AtomNAS-B, and AtomNAS-C on ImageNet improve by 1.4%, 1.7%, and 1.2% respectively, which clearly shows the benefit of BN recalibration.

Table 3: Influence of BN recalibration.

Model	w/o Recalibration	w/ Recalibration
AtomNAS-A	73.2	74.6 (+1.4)
AtomNAS-B	73.8	75.5 (+1.7)
AtomNAS-C	74.7	75.9 (+1.2)

4.3.4 Cost of Dynamic Network Shrinkage

Our dynamic network shrinkage algorithm speedups the search and train process significantly. For AtomNAS-C, the total time for search-and-training is 25.5 hours. For reference, training the final architecture from scratch takes 22 hours. Note that as the supernet shrinks, both the GPU memory consumption and forward-backward time are significantly reduced. Thus it's possible to dynamically change the batch size once having sufficient GPU memory, which would further speed up the whole procedure.

5 Conclusion

In this paper, we revisit the common structure, i.e., two convolutions joined by a channel-wise operation, and reformulate it as an ensemble of atomic blocks. This perspective enables a much larger and more fine-grained search space. For efficiently exploring the huge fine-grained search space, we propose an end-to-end algorithm named AtomNAS, which conducts architecture search and network training jointly. The searched networks achieve significantly better accuracy than previous state-of-the-art methods while using small extra cost.

REFERENCES

Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*, 2019.

Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019a.

Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. *CoRR*, abs/1904.12760, 2019b.

Xiangxiang Chu, Bo Zhang, Jixiang Li, Qingyuan Li, and Ruijun Xu. Scarletnas: Bridging the gap between scalability and fairness in neural architecture search. *CoRR*, abs/1908.06022, 2019a.

Xiangxiang Chu, Bo Zhang, Ruijun Xu, and Jixiang Li. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. *CoRR*, abs/1907.01845, 2019b.

Ekin Dogus Cubuk, Barret Zoph, Dandelion Mané, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation policies from data. *CoRR*, abs/1805.09501, 2018.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. In *CVPR*, pp. 248–255, 2009.

Jiemin Fang, Yuzhu Sun, Qian Zhang, Yuan Li, Wenyu Liu, and Xinggang Wang. Densely connected search space for more flexible neural architecture search. *CoRR*, abs/1906.09607, 2019.

Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *CVPR*, pp. 1586–1595, 2018.

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. *CoRR*, abs/1904.00420, 2019.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *ICLR*, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pp. 770–778, 2016.

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pp. 2980–2988, 2017a.

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, pp. 1398–1406, 2017b.

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. *Corr*, abs/1905.02244, 2019.

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. *CoRR*, abs/1704.04861, 2017.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *CVPR*, pp. 7132–7141, 2018.

Hanwen Liang, Shifeng Zhang, Jiacheng Sun, Xingqiu He, Weiran Huang, Kechen Zhuang, and Zhenguo Li. Darts+: Improved differentiable architecture search with early stopping, 2019.

Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. In *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V*, pp. 740–755, 2014.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. In *ICLR*, 2019a.

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, pp. 2755–2763, 2017.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019b.

Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *ICCV*, pp. 5068–5076, 2017.

Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet V2: practical guidelines for efficient CNN architecture design. In *ECCV*, pp. 122–138, 2018.

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *ICML*, pp. 4092–4101, 2018.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*, 2018.

Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, pp. 4510–4520, 2018.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

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. *CoRR*, abs/1904.02877, 2019.

Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, pp. 6105–6114, 2019a.

Mingxing Tan and Quoc V. Le. Mixconv: Mixed depthwise convolutional kernels. *CoRR*, abs/1907.09595, 2019b.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, pp. 2820–2828, 2019.

Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*, pp. 10734–10742, 2019.

Jianbo Ye, Xin Lu, Zhe Lin, and James Z. Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In *ICLR*, 2018.

Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*, 2018.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

Table 4: Comparision with baseline backbones on COCO object detection and instance segmentation. Cls denotes the ImageNet top-1 accuracy; detect-mAP and seg-mAP denotes mean average precision for detection and instance segmentation on COCO dataset. The detection results of baseline models are from (Stamoulis et al., 2019).

Model	FLOPs	Cls (%)	detect-mAP (%)	seg-mAP (%)
MobileNetV2 (Sandler et al., 2018)	301M	73.6	30.5	-
Proxyless (mobile) (Cai et al., 2019)	320M	74.9	32.9	-
SinglePath (Stamoulis et al., 2019)	334M	75.0	33.0	-
AtomNAS-A	258M	74.6	32.7	30.1
AtomNAS-B	326M	75.5	33.6	30.8
AtomNAS-C	360M	75.9	34.1	31.4

A Appendix

A.1 Experiments on COCO Detection and Instance Segmentation

In this section, we assess the performance of AtomNAS models as feature extractors for object detection and instance segmentation on COCO dataset (Lin et al., 2014). We first pretrain AtomNAS models (without Swish activation function (Ramachandran et al., 2018) and Squeeze-and-Excitation (SE) module (Hu et al., 2018)) on ImageNet, use them as drop-in replacements for the backbone in the Mask-RCNN model (He et al., 2017a) by building the detection head on top of the last feature map, and finetune the model on COCO dataset.

We use the open-source code MMDetection (Chen et al., 2019a). All the models are trained on COCO train2017 with batch size 16 and evaluated on COCO val2017. Following the schedule used in the open-source implementation of TPU-trained Mask-RCNN [1], the learning rate starts at 0.02 and decreases by a scale of 10 at 15-th and 20th epoch respectively. The models are trained for 23 epochs in total.

Table 4 compares the results with other baseline backbone models. The detection results of baseline models are from (Stamoulis et al., 2019). We can see that all three AtomNAS models outperform the baselines on both object detection task. The results demonstrate that our models have better transferability than the baselines.

[1] https://github.com/tensorflow/tpu/tree/master/models/official/mask_rcnn