HourNAS: Extremely Fast Neural Architecture Search Through an Hourglass Lens

Zhaohui Yang[1], **Yunhe Wang**[2], **Dacheng Tao**[3], **Xinghao Chen**[2], **Jianyuan Guo**[1], **Chunjing Xu**[2], **Chao Xu**[1], **Chang Xu**[3]
[1] Key Lab of Machine Perception (MOE), Dept. of Machine Intelligence, Peking University.
[2] Noah's Ark Lab, Huawei Technologies.
[3] School of Computer Science, Faculty of Engineering, University of Sydney.
{zhaohuiyang,jyguo}@pku.edu.cn; xuchao@cis.pku.edu.cn
{yunhe.wang,xinghao.chen,xuchunjing}@huawei.com
{dacheng.tao,c.xu}@sydney.edu.au

Abstract

Neural Architecture Search (NAS) refers to automatically design the architecture. We propose an hourglass-inspired approach (HourNAS) for this problem that is motivated by the fact that the effects of the architecture often proceed from the vital few blocks. Acting like the narrow neck of an hourglass, vital blocks in the guaranteed path from the input to the output of a deep neural network restrict the information flow and influence the network accuracy. The other blocks occupy the major volume of the network and determine the overall network complexity, corresponding to the bulbs of an hourglass. To achieve an extremely fast NAS while preserving the high accuracy, we propose to identify the vital blocks and make them the priority in the architecture search. The search space of those non-vital blocks is further shrunk to only cover the candidates that are affordable under the computational resource constraints. Experimental results on the ImageNet show that only using 3 hours (0.1 days) with one GPU, our HourNAS can search an architecture that achieves a 77.0% Top-1 accuracy, which outperforms the state-of-the-art methods.

1 Introduction

In the past decade, progress in deep neural networks has resulted in the advancements of various computer vision tasks, such as image classification [22, 37], object detection [12, 27], and segmentation [15]. The big success of deep neural networks is mainly contributed to the well-designed cells and sophisticated architectures. For example, VGGNet [37] suggested the use of smaller convolutional filters and stacked a series of convolution layers for achieving higher performance, ResNet [16] introduced the residual blocks to benefit the training of deeper neural networks, and DenseNet [19] designed the densely connected blocks to stack features from different depths. Besides the efforts on the initial architecture design, extensive experiments are often required to determine the weights and hyperparameters of the deep neural network.

To automatically and efficiently search for neural networks of enjoyable properties (*e.g.*, model size and FLOPs) from a pre-defined search space, a number of Neural Architecture Search (NAS) algorithms have been recently proposed. Wherein, Evolutionary Algorithm (EA) based methods [33] maintain a set of architectures and generate new architectures using genetic operations like mutation and crossover. Reinforcement Learning (RL) based methods [49, 50] sample architectures from the search space and train the controllers accordingly. The differentiable based methods [26, 42, 43, 36]

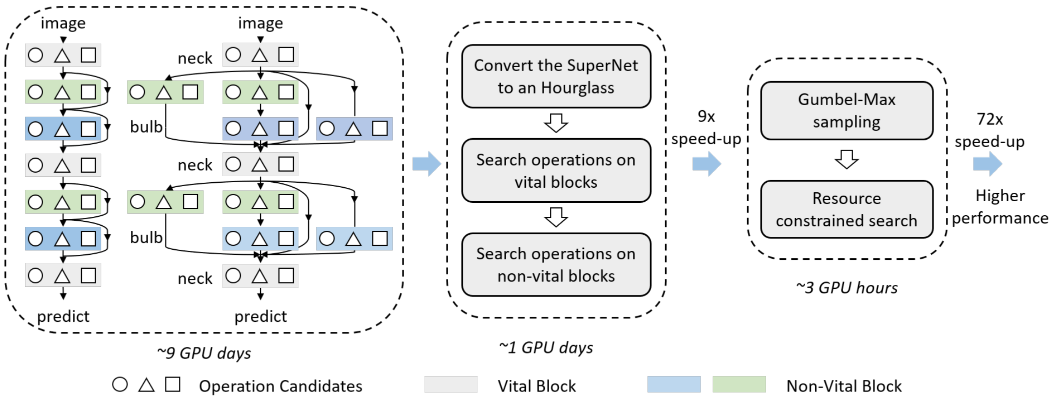


Figure 1: Blocks in the residual network are either "vital" or "non-vital", and they form the neck or bulb parts in the hourglass network. Two-stage search scheme speed up architecture search by 9× and the resource constrained search further accelerates architecture search by 72×.

optimize the shared weights and architecture parameters, which significantly reduces the demand for computation resources and makes the search process efficient.

These methods have made tremendous efforts to greatly accelerate the search process of NAS. Nevertheless, given the huge computation cost on the large-scale dataset [42, 14, 44, 17, 1], *e.g.*, 9 GPU days for NAS on the ImageNet benchmark, most methods execute NAS in a compromised way. The architecture is first searched on a smaller dataset (*e.g.*, CIFAR-10 [21]), and then the network weight of the derived architecture is trained on the large dataset. An obvious disadvantage of this concession is that the performance of the selected architecture on the CIFAR-10 may not be well extended to the ImageNet benchmark [46]. We tend to use the minimal number of parameters without discrimination to construct an architecture that would achieve the maximal accuracy. But just as the popular 80-20 rule goes, only a few parts could be critical to the architecture's success, and we need to give them the most focus while balancing the parameter volume for other parts.

In this paper, we propose HourNAS for an accurate and efficient architecture search on the large-scale ImageNet dataset. Blocks in an architecture are not created equally. Given all the possible paths for the information flow from the input to the output of the network, blocks shared by these paths are *vital*, just as the neck of an hourglass to regulate the flow. We identify these vital blocks and make them the priority in the architecture search. The other non-vital blocks may not be critical to the accuracy of the architecture, but they often occupy the major volume of the architecture (like the bulb of an hourglass) and will carve up the resource budget left by the vital blocks. Instead of directly working in a large search space flooding with architectures that obviously do not comply with the constraints during deployment, we develop a space proposal method to screen out and optimize the most promising architecture candidates under the constraints. By treating the architecture search through an hourglass lens, the limited computation resource can be well allocated across vital and non-vital blocks. We design toy experiments on residual networks to illustrate the varied influence of vital and non-vital blocks on the network accuracy. The resulting HourNAS costs only about 3 hours (0.1 GPU days) on the entire ImageNet dataset to achieve a 77.0% Top-1 accuracy, which outperforms the state-of-the-art methods.

2 Related Works

This section reviews the methods for neural architecture search algorithms. Then, we discuss the layer equality, *i.e.*, the importance of different blocks in an deep neural networks.

Neural Architecture Search. To automate the design of neural models, neural architecture search (NAS) was introduced to discover efficient architectures with competitive performance. Reinforcement learning (RL) and evolution algorithm (EA) were widely adopted in NAS [50, 49, 28, 33, 38, 14, 13]. However, these methods were highly computationally demanding. Many works have been devoted to improving the efficiency of NAS from different perspectives, *e.g.*, by adopting the strategy of weight sharing [31] or progressive search [24]. Among them, differentiable based NAS attracted great interest as it drastically reduces the searching cost to several days or even hours [42, 26, 9, 43, 32, 23].

For example, DARTS [26] adopted the continuous architecture representation to allow efficient architecture search via gradient descent. The most efficient gradient-based NAS which directly searched on target large-scale dataset (*e.g.*, ImageNet) still took dozens of days. Thus, an efficient method for directly searching deep neural architectures on large-scale datasets and search spaces is urgently required.

Layer Equality. Most of existing NAS methods treated all layers with the same importance during the search. However, convolution neural networks are always over-parameterized and the effect on the final accuracy of each layer are totally different. Zhang *et al.* [47] re-initialized and re-randomized the pre-trained networks, and found that some layers are robust to the disturbance. For some intermediate layers, the re-initialization and re-randomization steps did not have negative consequences on the accuracy. Veit *et al.* [41] decomposed residual networks and found that skipping some layers does not decrease the accuracy significantly. Ding *et al.* [8] pruned different layers separately and found some layers are more important to the final accuracy. It is obvious that the layers are not created equal, and some layers are more important than others. In this paper, we analyze the causes of the inequality phenomenon in the residual network and exploit this property in neural architecture search to improve its efficiency.

3 Hourglass Neural Architecture Search

In this section, we revisit the neural architecture search from an hourglass way. The vital few blocks should be searched with a higher priority while the non-vital blocks that occupy the major volume are then searched to satisfy the computational resource constraints.

3.1 Vital Blocks: the Neck of the Hourglass

In this paper, we focus on the serial-structure NAS SuperNet [38, 39, 42, 2], as it is hardware-friendly and capable of achieving superior performance. Before we illustrate vital blocks in a general NAS, we first take ResNet [16] as an example for the analysis. ResNet is one of the most popular manually designed architectures. It is established by stacking a series of residual blocks, and the transformation of a residual block is defined as

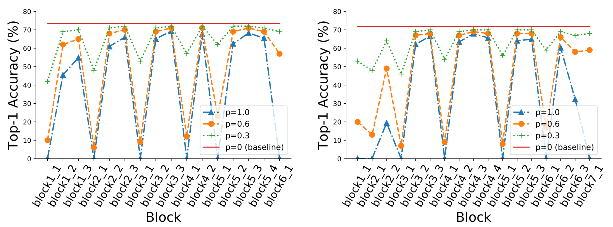
$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \mathbf{w}) + \mathbf{x}, \tag{1}$$

where $\mathbf{x}$ is the input feature map, $\mathcal{F}$ denotes the transformation (*e.g.*, convolution and batch normalization for vision tasks) and $\mathbf{w}$ stands for the trainable parameters.[1] From the information flow perspective, there are two paths to transmit the information from the node $\mathbf{x}$ to the node $\mathbf{y}$, *i.e.*, the shortcut connection and the residual transformation $\mathcal{F}$. If there are m residual blocks in a network, there will be 2^m different paths for the information propagation in total. A general neural network $\mathcal{N}$ based on the residual blocks [16, 38, 42] can therefore be approximated as the ensemble of a number of paths [41] $\{\mathcal{P}_1, \dots, \mathcal{P}_n\}$, *i.e.*, $\mathcal{N}(X) \approx \sum_{i=1}^{n} \mathcal{P}_i(X)$, where each path $\mathcal{P}_i$ is set up by a series of blocks, X is the input data, and n is the number of all the paths.

It is worth noticing that there are a few blocks existing in all the possible paths, *e.g.*, the gray blocks in Figure 1. These self-contained blocks do not participate in forming any residual transformation, but they are *vital*, because of their appearance in every path from the input to the output of the network. On the other hand, the green and blue blocks in Figure 1 are a part of the residual transformations $\mathbf{y} = \mathcal{F}(\mathbf{x}, \mathbf{w}) + \mathbf{x}$, where the information can be transmitted through the plain transformation $\mathcal{F}(\mathbf{x}, \mathbf{w})$ or the shortcut connection $\mathbf{x}$ to the next block.

Given the paths $\{\mathcal{P}_1, \dots, \mathcal{P}_n\}$ in a general residual network $\mathcal{N}$, the vital blocks shared by all the paths can be identified through $\hat{\mathcal{P}} = \mathcal{P}_1 \cap \cdots \cap \mathcal{P}_n$, where $\mathcal{P}_i \cap \mathcal{P}_j$ denotes the intersection set of those blocks in paths $\mathcal{P}_i$ and $\mathcal{P}_j$. In the popular residual networks, such as ResNet [16] and FBNet [42], the vital blocks are exactly the first convolution layer, the last fully connected layer, the downsampling blocks, and the channel expansion blocks. These vital blocks are critical to the accuracy of the whole architecture, as they exist in all paths and act as the neck of the hourglass to control the information flow. In contrast, the other blocks would always find substitutes for themselves to keep the information flow, and they thus play a secondary role in the whole architecture.

[1] As for the downsample blocks (reduce the feature map size) and the channel expansion blocks (increase the channel number), we follow [41] and use $\mathbf{y} = \mathcal{F}(\mathbf{x}, \mathbf{w})$ to express the transformations.



(a) MnasNet (b) MobileNetV2

Figure 2: The diagram of block importance by using the MnasNet and MobileNetV2 pretrained models.

We further take mobile architectures as an example to illustrate the different influence of vital and non-vital blocks on the network accuracy. A random mask function $\mathcal{M}(\mathbf{y}, p)$ is introduced to destroy the output of blocks in the pretrained MnasNet [38] and MobileNetV2 [34], where $\mathbf{y}$ is the output feature map, and $0 \leq p \leq 1$ is the probability. In particular, every channel is reset to 0 with a probability p. Figure 2 shows the accuracy change of MnasNet [38] and MobileNetV2 [34] resulting from the feature distortion on different blocks. The $blocki\_j$ is the j-th block in the i-th stage and the first block in every stage is the vital block. We set $p = \{0.3, 0.6, 1.0\}$ to gradually increase the degree of destroying the output. Each time we only manipulate one block while keeping others unchanged in the network. For the non-vital blocks (*e.g.*, block3_2 and block3_3 in both MnasNet and MobileNetV2), even if all the channels are reset to zero (*i.e.*, p=1.0), the network does not undergo a significant accuracy decrease. However, a small portion (p=0.3) of channels are masked out for those vital blocks (*e.g.*, block1_1 and block2_1) will lead to an obvious accuracy drop.

The goal of NAS is to search for the architecture of a higher accuracy under the constraints of the available computational resources. In other words, NAS can be regarded as a problem of resource allocation. Vital blocks are potentially the most important and need to be put as the priority. As a result, more resources are better to be first allocated to the vital blocks, and the remaining resources are used for the non-vital blocks design. This therefore naturally motivates us to develop a two-stage search algorithm. During the first stage, we construct the minimal SuperNet $\mathcal{S}_{vital}$ and search the vital blocks without worrying about the computational overhead. The weights and architecture parameters are optimized alternatively in a differentiable way [42]. In the second stage, we fix the derived architecture of those vital blocks, and allocate the computational resources to the non-vital blocks.

3.2 Non-Vital Blocks: the Bulb of the Hourglass

Non-vital blocks are often composed of a large number of parameters. They look like the bulb of the hourglass to determine the whole volume size. If the computational resources are unlimited to deploy the neural network, then we can make the network wider and deeper for achieving a higher performance [16, 39]. However the searched architectures are to be deployed on mobile devices which have demanding constraints of the resources, *e.g.*, model size, and FLOPs. Without investigating these constraints, directly sampling the architecture from a large search space would be ineffective. For example, if the sampled architectures cannot fully use the available computation resource, the resulting models might perform poorer than expected, which has been analyzed by a number of multi-objective NAS works [28, 45, 11, 38] (the Pareto front). Otherwise, if the sampled architectures overwhelm the use of computation resources, they would be difficult to be deployed. To tackle the dilemma, we introduce an efficient sampling operation to avoid wasting too much time on search unimportant operations, and a space proposal strategy to put more attention on architectures that meet the target computational resources.

3.3 Gumbel-Max for Efficient Operation Sampling

The sampling strategy in every iteration directly affects the searching efficiency of gradient-based NAS. The softmax [26] and Gumbel softmax [42] are two popular ways to train the SuperNet. However, some worse performed operations have the same opportunity as well-performed operations to be updated in every iteration, which is an inefficiency solution in terms of both searching efficiency and GPU memory. Hence, we utilize the Gumbel-Max [30, 3, 10] to sample operations according to the learned importance, which avoids wasting searching time on undesired operations.

Suppose P represents the importance matrix of the SuperNet and θ represents the trainable architecture parameters of the NAS SuperNet. The sizes of P and θ are all $L \times O$, where L is the maximum depth, and O is the number of candidate operations. Sampling operations from θ are used either for updating the network weights on the train set $\mathcal{D}_{tr}$ or updating the architecture parameters θ on the validation set $\mathcal{D}_{val}$. The importance $P_{l,o}$ of the o-th operation in layer l is,

$$P_{l,o} = \frac{\exp^{\theta_{l,o}}}{\sum_o \exp^{\theta_{l,o}}}, \quad \sum_o P_{l,o} = 1. \tag{2}$$

To sample an operation in layer l according to the Gumbel-Max,

$$\begin{aligned} z_l &= \theta_l + g, \\ A_{\theta_l} &= \mathcal{I}(\arg\max_i z_{l,i}), \end{aligned} \tag{3}$$

where the $g \sim \text{Gumbel}(0,1)$[2], $\mathbb{P}(\arg\max_i z_{l,i} = o) = P_{l,o}$, $\mathcal{I}$ converts a scalar to the one-hot vector, and A_{θ_l} denotes the one-hot vector for the l-th layer. Therefore, we use $\mathcal{A}(\theta)$ to denote sampling from θ and use the notation A_θ to represent the sampled architecture.

Gumbel-Max samples an operation according to the learned probability distribution (*i.e.*, importance). The sampling frequency of those poor operations tends to be relatively low, so that we can effectively reduce the time spent on them. The Gumbel-Max sampling accelerates every iteration by around O times.

3.4 Space Proposal for Controllable Multi-Objective Search

Assuming the constraints (targets) on computational resources are $T_i, i \in \{1, \dots, n\}$, where n is the number of objectives, an efficient and controllable way is to sample architectures that satisfy the T, neither too much nor too less. Thus, we introduce the concept of space proposal. The space proposal is a subspace from the large search space, and all the sampled architectures from the space proposal satisfy the target resources. The architectures sampled from the space proposal satisfy the targets. As a result, the search phase would not waste resources on optimizing useless architectures. In addition, the space proposal ensures "what you set is what you get". Similar to gradient-based NAS, the space proposal is represented by the architecture parameters.

We take the FLOPs as an example to describe how to optimize a space proposal. The FLOPs table F of the SuperNet $\mathcal{S}$ is of size $L \times O$, where $F_{l,o}$ denotes the FLOPs of the o-th operation in the l-th SuperBlock. The FLOPs for sampled architecture A_θ is calculated as $\mathcal{R}_F(A_\theta) = \text{sum}(A_\theta \odot F)$, where $\odot$ is the element-wise product. Assuming the target FLOPs is T_F, then the goal for the FLOPs objective is as following,

$$\theta^F = \arg\min_\theta |\mathcal{R}_F(\mathcal{A}(\theta)) - T_F| / M_F, \tag{4}$$

where M_F is a constant scalar denotes the maximum FLOPs of the sampled architecture, and this term is used for normalizing the objective to a range of $[0, 1]$. We extend Eqn 4 to n different objectives, the targets for n objectives are $T_i, i \in \{1, \dots, n\}$, and the multi-objectives optimization target is defined as,

$$\theta^T = \arg\min_\theta \sum_i |\mathcal{R}_i(\mathcal{A}(\theta)) - T_i| / M_i, \tag{5}$$

which $\mathcal{R}_i(\mathcal{A}(\theta))$ is the resource demand of the architecture sampled by $\mathcal{A}(\theta)$ on the i-th objective. According to the restrictions T, we can train the randomly initialized structure parameter θ within a few seconds to construct a space proposal. After training, θ^T can be regarded as a space proposal under the constraints T, and the structure A_{θ^T} sampled from θ^T by Eqn 3 satisfies target resources T.

The optimized θ^T could only sample architectures from one subspace, which may not contain a well-performed architecture. Assuming p denotes the probability that one space proposal contains a well-performed architecture, optimizing m different space proposals cover a much larger search space that increases the probability of containing a well-performed architecture to $1 - (1-p)^m$. Thus, we optimize m different randomly initialized space proposals $\Theta^T = \{\theta_1^T, \cdots, \theta_m^T\}$ and we simultaneously optimize a proposal selection parameter $\pi \in \mathbb{R}^m$ initialized by uniform distribution. The selection parameter π is used to sample from m different space proposals. Thus, different space proposals are sampled with the same probability at the beginning, and the importance is gradually learned.

[2] $g = -\log(-\log(u)), u \sim \text{Uniform}(0,1)$.

The overall NAS framework considering the multi-objectives is summarized as,

$$\begin{aligned}\pi^*, \Theta^* &= \arg\min_{\pi,\Theta} \mathcal{H}_{val}(\mathcal{S}(w^*(\pi,\Theta),\pi,\Theta)) + \alpha \times \mathcal{T}(\pi,\Theta),\\ \text{s.t. } \; w^*(\pi,\Theta) &= \arg\min_{w} \mathcal{H}_{train}(w,\pi,\Theta),\end{aligned} \tag{6}$$

where $\mathcal{S}$ is the SuperNet, $\mathcal{H}$ is the cross-entropy loss function, $\mathcal{T}$ is the regularization on space proposal parameters, and α is the slope of the multi-objective loss term.

4 Experiments

In this section, we first describe the experimental settings and then evaluate the proposed HourNAS on several accessible NAS search spaces [42, 38, 39] on ImageNet.

4.1 Experimental Settings

We use the HourNAS to search on the complete ImageNet train set. The subset $\mathcal{D}_{tr}$ takes 80% of the train set to train network parameters and the rest $\mathcal{D}_{val}$ is used to update architecture parameters. We search on three popular search spaces, FBNet [42], MnasNet [38], and EfficientNet [39]. For any of our searched architecture, the training strategy is the same as the corresponding baseline.

The HourNAS first searches the vital blocks for one epoch (about 1 hour). Then, HourNAS optimizes multiple space proposals according to the computational targets and searches the non-vital blocks for one epoch (about 2 hours). The searching process requires one V100 GPU. During searching, we follow FBNet [42] and add the temperature τ on z (Eqn 3) for sharpening the distribution progressively. The temperature τ starts from 5.0 and multiply 0.9999 at every iteration. Slope parameter α is 1.0. Learning rates for optimizing weights and architecture parameters are 0.1 and 0.01, respectively. Adam [20] optimizer is used to update architecture parameters.

4.2 Comparison with State-of-the-arts

FBNet Search Space Starting from the FBNet search space, the HourNAS first searches the vital blocks and the operations with the expansion ratio equal to six have a significantly higher probability than other operations. We choose the highest probability operations to form the vital blocks. This result is in line with our intuition that the complex operations have the greatest feature extraction ability on the large dataset, *i.e.*, ImageNet.

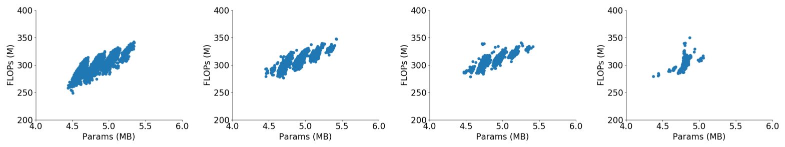
Table 1: Comparison of image classifiers on CIFAR-10 dataset.

Model	Test Error (%)	Params (M)	Search Cost (GPU days)	Type
DenseNet-BC [19]	3.46	25.6	-	manual
PNAS [24]	3.41	3.2	225	SMBO
AmoebaNet-A + cutout [33]	3.12	3.1	3150	evolution
Hierarchical evolution [25]	3.75	15.7	300	evolution
HourNAS (m=1) + cutout	3.86	2.8	0.1	gradient
HourNAS (m=2) + cutout	3.54	2.8	0.1	gradient
HourNAS (m=4) + cutout	3.41	2.7	0.1	gradient
HourNAS (m=8) + cutout	3.39	2.8	0.1	gradient
HourNAS (m=16) + cutout	3.37	2.8	0.1	gradient

After fixing the operations of the vital blocks, we are interested in finding the appropriate proposal number m. We set the computational resources the same as FBNet-B and search architectures using m different space proposals. We first visualize the distribution of sampled architectures from the space proposal by gradually decreasing the temperature τ. The computational targets are set to 4.8M parameters (x-axis) and 300M FLOPs (y-axis), which are consistent with FBNet-B [42]. As shown in Figure 3, with the decrease of temperature of τ, the sampled architectures satisfy the computational targets more precisely.

Table 2: Overall comparison on the ILSVRC2012 dataset.

Model	Type	Search Dataset	Search Cost (GPU days)	Params (M)	FLOPS (M)	Top-1 (%)	Top-5 (%)
ResNet50 [16]	manual	-	-	25.6	4100	75.3	92.2
MobileNetV1 [18]	manual	-	-	4.2	575	70.6	89.5
MobileNetV2 [34]	manual	-	-	3.4	300	72.0	91.0
MobileNetV2 (1.4$\times$)	manual	-	-	6.9	585	74.7	92.5
ShuffleNetV2 [29]	manual	-	-	-	299	72.6	-
ShuffleNetV2 (1.5$\times$)	manual	-	-	3.5	299	72.6	-
AmoebaNet-A [33]	auto	CIFAR-10	3150	5.1	555	74.5	92.0
NASNet-A [50]	auto	CIFAR-10	1800	5.3	564	74.0	91.3
PNAS [24]	auto	CIFAR-10	225	5.1	588	74.2	91.9
DARTS [26]	auto	CIFAR-10	4	4.9	595	73.1	91.0
SNAS (mild) [43]	auto	CIFAR-10	1.5	4.3	522	72.7	90.8
FPNASNet [6]	auto	CIFAR-10	0.8	3.4	300	73.3	-
CARS-I [45]	auto	CIFAR-10	0.4	5.1	591	75.2	92.5
PDARTS [4]	auto	CIFAR-10	0.3	4.9	557	75.6	92.6
MdeNAS [48]	auto	CIFAR-10	0.2	6.1	-	74.5	92.1
GDAS [10]	auto	CIFAR-10	0.2	5.3	581	74.0	91.5
SPOSNAS [14]	auto	ImageNet	13	-	295	74.2	-
ProxylessNAS [2]	auto	ImageNet	8.3	7.1	465	75.1	92.5
RCNet [44]	auto	ImageNet	8	3.4	294	72.2	91.0
FBNet-B [42]	auto	ImageNet	9	4.8	295	74.1	-
FBNet-C [42]	auto	ImageNet	9	5.5	375	74.9	-
HourNAS-A	auto	ImageNet	**0.1**	4.8	298	**74.1**	**91.8**
HourNAS-B	auto	ImageNet	**0.1**	5.5	406	**75.0**	**92.2**
HourNAS-C	auto	ImageNet	**0.1**	4.8	296	**74.1**	**91.6**
HourNAS-D	auto	ImageNet	**0.1**	5.5	394	**75.3**	**92.3**
MnasNet-A1 [38]	auto	ImageNet	3800	3.9	312	75.2	92.5
HourNAS-E	auto	ImageNet	**0.1**	3.8	313	**75.7**	**92.8**
EfficientNet-B0 [39]	auto	ImageNet	-	5.3	390	76.8	-
HourNAS-F	auto	ImageNet	**0.1**	5.3	383	**77.0**	**93.5**



(a) $\tau = 5.0$ (b) $\tau = 1.0$ (c) $\tau = 0.5$ (d) $\tau = 0.1$

Figure 3: The distribution of 10,000 architectures sampled from optimized space proposal under different temperatures τ.

The optimization step for constructing several space proposals takes only a few seconds, which is an efficient solution for controllable multi-objective neural architecture search. We enumerate $m = \{1, 2, 4, 8, 16\}$ and after searching, the operations with the highest probability according to π and Θ are fixed. The architectures are evaluated on the CIFAR-10 dataset to determine the appropriate space proposal number m for finding superior architectures. As shown in Table 1, $m = 8$ could result in a well-performed architecture and we use $m = 8$ in the following experiments to achieve a better trade-off between searching costs and performance.

We train two searched models from scratch on the ImageNet dataset. The HourNAS-A has the same model size and FLOPs as FBNet-B, and the HourNAS-B has the same computational requirements as FBNet-C. The training strategy is the same as FBNet [42] without using bells and whistles. As shown in Table 2, the HourNAS-A and HourNAS-B achieve on par accuracies with FBNet-B and FBNet-C, and the search time accelerates by two magnitudes.

Enlarged FBNet Search Space MixConv [40] indicates that a larger kernel size leads to better performance. To understand the impact of search space and to further verify the effectiveness of HourNAS, we slightly change the search space of FBNet. We add the blocks with kernel size $k = 7$ and remove the blocks with group $g = 2$. The multi-objectives are the same as HourNAS-A and HourNAS-B. We list the searched architectures in Table 2. The HourNAS-C achieves the same Top-1 accuracy with HourNAS-A and HourNAS-D surpasses HourNAS-B by 0.3% Top-1 accuracy.

MnasNet Search Space We further apply our proposed HourNAS to the search space of MnasNet [38]. We select MnasNet-A1 as the baseline and use the number of the parameters and FLOPs as two objectives to optimize 8 space proposals. The searched HourNAS-E achieves a Top-1 accuracy of 75.7% on the ILSVRC2012 dataset, which surpasses MnasNet-A1 by 0.5% Top-1 accuracy.

EfficientNet Search Space To compare with the state-of-the-art architecture EfficientNet-B0 [39], HourNAS directly searches on the same backbone and search space as EfficientNet. Targeting at EfficientNet-B0, we use the model size and FLOPs as two objectives to regularize space proposals and denote the result as HourNAS-F. The AutoAugment [5] is not used during training and the result in Table 2 shows HourNAS-F surpasses EfficientNet-B0 by 0.2% Top-1 accuracy.

4.3 Ablation Study

If we do not restrict the computational resources of the sampled architectures while searching, the most complex block achieves the highest probability after searching for enough time. As shown in Table 3, we train the most complex architectures in both FBNet and EfficientNet search spaces, namely FBNet-Max and EfficientNet-Max. Two structures obtain 75.7%, and 78.3% Top-1 accuracies, respectively. However, the computational resource requirements of these structures are relatively high. Therefore, the neural architecture search could be regarded as the problem of computational resource allocation given the resource constraints.

Table 3: The results of FBNet-Max and EfficientNet-Max on ILSVRC2012 dataset.

Model	Params (M)	FLOPS (M)	Top-1 (%)	Top-5 (%)
FBNet-Max	5.7	583	75.7	92.8
EfficientNet-Max	5.8	738	78.3	94.0

The Impact of Vital Block Priori In order to investigate the impact of the vital block priori, we directly search architectures without using the vital block information. All the blocks in the SuperNet S are treated equally, rather than the two-stage scheme. We use the Gumbel-Max sampling and space proposal strategy to search architectures under the same predefined computational resources.

We use the previously described original and enlarged FBNet search space. We optimize 8 space proposals and train the SuperNet for 6 hours, which is twice of the counterparts. We denote these models as HourNAS-G/H and the Top-1 accuracies drop by 0.9% and 0.6% on the ImageNet validation set, respectively. The results show the necessity of the vital block priori. Searching the vital blocks with higher priority is helpful in finding high-quality architectures.

Table 4: The results comparison on ILSVRC2012 dataset. The search spaces are original (upper) and enlarged (lower) FBNet search space, respectively.

Model	Type	Search Dataset	Search Cost (GPU days)	Params (M)	FLOPS (M)	Top-1 (%)	Top-5 (%)
HourNAS-A	auto	ImageNet	**0.1**	4.8	298	**74.1**	**91.8**
HourNAS-G (w/o priori)	auto	ImageNet	0.2	4.7	297	73.2	91.4
HourNAS-D	auto	ImageNet	**0.1**	4.8	296	**74.1**	**91.6**
HourNAS-H (w/o priori)	auto	ImageNet	0.2	4.8	299	73.5	91.3

The Impact of Gumbel-Max Sampling In the search phase, we use the Gumbel softmax [42] rather than the Gumbel-Max sampling to optimize architecture parameters and network parameters. The search space and target resource constraints are the same as HourNAS-A. The search process takes around 1 GPU day and the finalized architecture is denoted as HourNAS-I. The Gumbel-Max is an efficient strategy for optimizing the SuperNet.

Table 5: The results comparison on ILSVRC2012 dataset.

Model	Type	Search Dataset	Search Cost (GPU days)	Params (M)	FLOPS (M)	Top-1 (%)	Top-5 (%)
HourNAS-A	auto	ImageNet	0.1	4.8	298	74.1	91.8
HourNAS-I	auto	ImageNet	1.0	4.8	318	74.2	91.8

5 Conclusions

This paper investigates an efficient algorithm to search deep neural architectures on the large-scale dataset (*i.e.*, ImageNet) directly. To reduce the complexity of the huge search space, we present an Hourglass-based search framework, namely HourNAS. The entire search space is divided into "vital" and "non-vital" parts accordingly. By gradually search the components in each part, the search cost can be reduced significantly. Since the "vital" parts are more important for the performance of the obtained neural network, the optimization on this part can ensure the accuracy. By exploiting the proposed approach, we can directly search architectures on the ImageNet dataset that achieves a 77.0% Top-1 accuracy using only 3 hours (*i.e.*, about 0.1 GPU days), which outperforms the state-of-the-art methods in both terms of search speed and accuracy.

References

[1] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once for all: Train one network and specialize it for efficient deployment. *ICLR*, 2020.

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

[3] Jianlong Chang, xinbang zhang, Yiwen Guo, Gaofeng Meng, Shiming Xiang, and Chunhong Pan. Data: Differentiable architecture approximation. *NeurIPS*, 2019.

[4] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. *ICCV*, 2019.

[5] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. *CVPR*, 2019.

[6] Jiequan Cui, Pengguang Chen, Ruiyu Li, Shu Liu, Xiaoyong Shen, and Jiaya Jia. Fast and practical neural architecture search. *ICCV*, 2019.

[7] Soham De and Samuel L. Smith. Batch normalization biases deep residual networks towards shallow paths. *arxiv*, 2019.

[8] Xiaohan Ding, guiguang ding, Xiangxin Zhou, Yuchen Guo, Jungong Han, and Ji Liu. Global sparse momentum sgd for pruning very deep neural networks. *NeurIPS*, 2019.

[9] Xuanyi Dong and Yi Yang. One-shot neural architecture search via self-evaluated template network. *ICCV*, 2019.

[10] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. *CVPR*, 2019.

[11] Thomas Elsken, Jan Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. *ICLR*, 2019.

[12] Ross Girshick. Fast r-cnn. *ICCV*, 2015.

[13] Xinyu Gong, Shiyu Chang, Yifan Jiang, and Zhangyang Wang. Autogan: Neural architecture search for generative adversarial networks. *ICCV*, 2019.

[14] 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*, 2019.

[15] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. *ICCV*, 2017.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CVPR*, 2016.

[17] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. *ECCV*, 2018.

[18] 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*, 2017.

[19] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. *CVPR*, 2017.

[20] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

[21] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, 2009.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *NeurIPS*, 2012.

[23] 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. *CVPR*, 2019.

[24] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. *ECCV*, 2018.

[25] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. *ICLR*, 2018.

[26] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *ICLR*, 2019.

[27] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. *ECCV*, 2016.

[28] Zhichao Lu, Ian Whalen, Vishnu Boddeti, Yashesh D. Dhebar, Kalyanmoy Deb, Erik D. Goodman, and Wolfgang Banzhaf. Nsga-net: A multi-objective genetic algorithm for neural architecture search. *GECC*, 2018.

[29] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. *ECCV*, 2018.

[30] Chris J Maddison, Daniel Tarlow, and Tom Minka. A* sampling. *NeurIPS*, 2014.

[31] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *ICML*, 2018.

[32] Ruijie Quan, Xuanyi Dong, Yu Wu, Linchao Zhu, and Yi Yang. Auto-reid: Searching for a part-aware convnet for person re-identification. *ICCV*, 2019.

[33] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *AAAI*, 2019.

[34] Mark B. Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *CVPR*, 2018.

[35] Albert Shaw, Daniel Hunter, Forrest N. Iandola, and Sammy Sidhu. Squeezenas: Fast neural architecture search for faster semantic segmentation. *ICCVW*, 2019.

[36] Albert Shaw, Wei Wei, Weiyang Liu, Le Song, and Bo Dai. Meta architecture search. *NeurIPS*, 2019.

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

[38] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. *CVPR*, 2018.

[39] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *ICML*, 2019.

[40] Mingxing Tan and Quoc V. Le. Mixconv: Mixed depthwise convolutional kernels. *BMVC*, 2019.

[41] Andreas Veit, Michael Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. *NeurIPS*, 2016.

[42] 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. *CVPR*, 2019.

[43] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. *ICLR*, 2019.

[44] Yunyang Xiong, Ronak Mehta, and Vikas Singh. Resource constrained neural network architecture search. *ICCV*, 2019.

[45] Zhaohui Yang, Yunhe Wang, Xinghao Chen, Boxin Shi, Chao Xu, Chunjing Xu, Qi Tian, and Chang Xu. Cars: Continuous evolution for efficient neural architecture search. *CVPR*, 2020.

[46] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. *ICLR*, 2020.

[47] Chiyuan Zhang, Samy Bengio, and Yoram Singer. Are all layers created equal. *ICMLW*, 2019.

[48] Xiawu Zheng, Rongrong Ji, Lang Tang, Baochang Zhang, Jianzhuang Liu, and Qi Tian. Multinomial distribution learning for effective neural architecture search. *ICCV*, 2019.

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

[50] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. *CVPR*, 2018.

A Examine The Vital Blocks in NAS SuperNet

The NAS search space is an extended formulation of residual network [16]. For each SuperBlock [35] in the NAS SuperNet $\mathcal{S}$, it contains several operations parallel to the identity mapping (*e.g.*, $O = 9$ for FBNet [42] and one of them is zero-mapping). NAS searches for the most appropriate operation for each block and constructs the searched architecture by stacking them. The identity mapping is served as the shortcut for some SuperBlocks. Thus, we could also divide all the SuperBlocks into "vital" and "non-vital" categories. The transformations in the vital SuperBlocks are to learn the $\mathcal{F}$, while the transformations in non-vital SuperBlocks are to learn the residual.

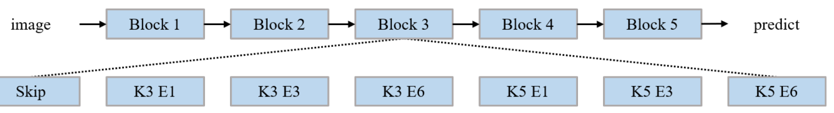
We further examine that the vital blocks play a more important role in the NAS search space by experiments. We could not train all the models (*e.g.*, 9^{22} for FBNet) in the search space. Thus we train a subset (504 models) on the ImageNet dataset to verify the hypothesis of vital blocks. The backbone is a shallower FBNet [42]. We aim to find which blocks would affect most on the final performance, in other words, which blocks are more sensitive to the final performance. If block l is to be evaluated, we fix all the other blocks with a pre-defined operation and enumerate block l with seven different operations (*e.g.*, block3 in Figure 4(a)). Then we report the mean/std. We examine six different backbones, where all the blocks are separately $ir\_k\{3,5\}\_e\{1,3,6\}$ [42][3]. All the 12 blocks in the middle are separately evaluated, and there are a total of $12 \times 7 \times 6 = 504$ models.

The statistics are shown in Figure 4. While other blocks are fixed, changing the transformations of vital blocks has the greatest impact on the final accuracy, which is reflected in greatly changing the mean or std. This phenomenon is especially obvious for experiments ir_k3_e6 and ir_k5_e6, by changing the operations of vital blocks, the averaged accuracies decrease by a large margin, and the standard deviations also increase significantly. Both analyses and experiments verify that vital blocks are more important to the final accuracy. Thus the vital blocks should be searched with a higher priority. Denoting $\mathcal{S}$ as the NAS SuperNet, We use $\mathcal{S}_{vital}$ to represent the minimal SuperNet which contains all the vital SuperBlocks.

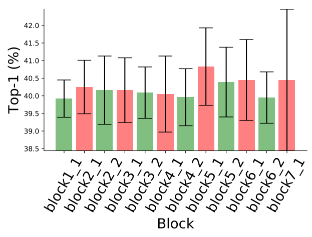
Table 6: Network definition for the sensitivity property experiments. The 'Super' denotes that this layer will be used to evaluate the mean value and standard deviation while other layers are fixed. In FBNet, each stage is constructed by 4 SuperBlocks, the first SuperBlock in each stage is vital, and the rest three of them are non-vital. In our experiments, we use 2 SuperBlocks in each stage, the first SuperBlock is vital and the next SuperBlock is non-vital.

Layer	Type	Out	Stride	vital
conv1	Conv	16	2	-
layer1_1	Super	16	1	non-vital
layer2_1	Super	24	2	vital
layer2_2	Super	24	1	non-vital
layer3_1	Super	32	2	vital
layer3_2	Super	32	1	non-vital
layer4_1	Super	64	2	vital
layer4_2	Super	64	1	non-vital
layer5_1	Super	112	1	vital
layer5_2	Super	112	1	non-vital
layer6_1	Super	184	2	vital
layer6_2	Super	184	1	non-vital
layer7_1	Super	352	1	vital
conv1	Conv	1984	1	-
avg	Avg Pool	-	-	-
fc	Fc	100	-	-

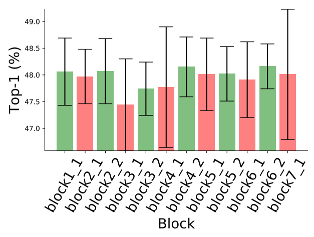
[3]The inverted residual (ir) block [34] with kernel size k and expansion e.



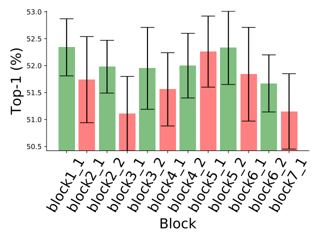
(a) Sensitivity property experiments



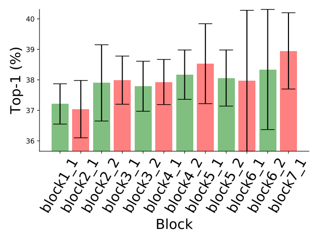
(b) ir k5 e1



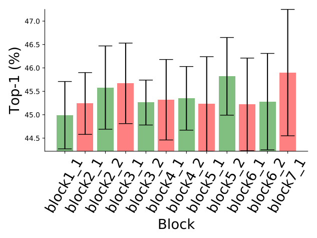
(c) ir k5 e3



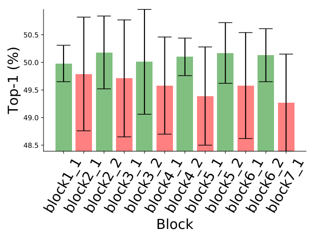
(d) ir k5 e6



(e) ir k3 e1



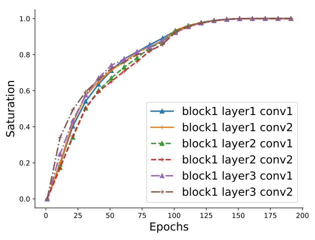
(f) ir k3 e3



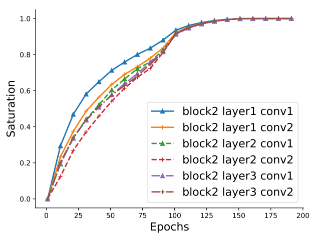
(g) ir k3 e6

Figure 4: The diagram of the block sensitivity experiments. Figure 4(b) to Figure 4(g) show the mean value and standard deviation of every block. The red histograms and the green histograms are the mean/std for the vital and non-vital blocks, respectively. A block with larger std means this block is more sensitive to the final accuracy by enumerating operations. In general, the vital blocks (red) are more sensitive to the final accuracy.

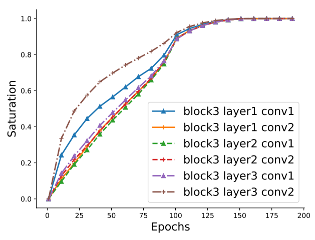
B Other Property of Vital Blocks



(a) block 1



(b) block 2



(c) block 3

Figure 5: The saturation curves for weights of all the convolution layers in the ResNet20 trained on the CIFAR-10 dataset. From left to right are different layers in block1, block2, and block3, respectively. We train the network for 200 epochs. Weight decay is 1e-4. The learning rate starts from 0.1 and decays by a factor of 10 in epoch 100, 150, and 180. The 'padding' mode shortcut for downsampling blocks is used.

During the network optimization period, different paths have some unique and interesting properties, *e.g.*, BN bias to the shallow path [7]. In this section, we study the convergence speed of different layers in network optimization.

While training the residual network, all the paths are jointly optimized. However, the depths are different for these paths. Therefore, optimization difficulties are different. Since the minimal path is the shortest path that directly connects the input images to the ground truths. For any other path that contains more random initialized layers, we argue these paths are harder to be optimized compared to the minimal path. Thus, the minimal path converges faster. We propose a new metric to measure the convergence degree of the parameters. Networks are randomly initialized and are gradually trained until converge using the back-propagation algorithm. Each layer (transformation) learns a series of patterns and extracts the features. Denote the weights of one transformation $\mathcal{L}$ as $\mathbf{w}$, $\mathbf{w}_0$ is the randomly initialized weights, and $\mathbf{w}_t$ denotes the weights after training for t epochs. T is the maximum epoch number for training the parameter $\mathbf{w}$ until converge. We define a metric $\mathcal{C}$ to judge the convergence degree of parameter $\mathbf{w}$, and the metric is used for measuring the saturation speed of the weights. The metric $\mathcal{C}$ is defined as follows,

$$\mathcal{C}_t = 1 - \frac{\mathcal{D}(\mathbf{w}_t, \mathbf{w}_T)}{\mathcal{D}(\mathbf{w}_0, \mathbf{w}_T)}, \quad \mathcal{C}_t = \begin{cases} 0, & \text{if} \ \ t = 0, \\ 1, & \text{if} \ \ t = T. \end{cases} \tag{7}$$

where $\mathcal{D}$ is the Frobenius norm measuring the distance between two tensors. Different layers have different convergence speed, so $\mathcal{C}$ is a function related to layers.

By following the proposed saturation measurement in Eqn 7. The saturation curves are shown in Figure 5. All the weights in block1 (non-vital) converge at a similar speed. However, the vital layers show a faster saturation speed. This experiment demonstrates that different layers converge with different speeds, and the vital layers converge faster because the non-vital layers would be affected by the noisy initialization at the beginning.

C The Diagram of Space Proposal

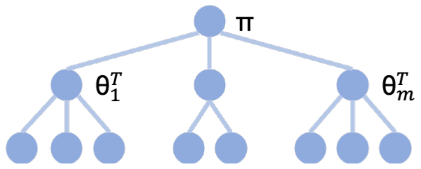


Figure 6: The diagram of space proposal selection. In this diagram, π represents the sampler for sampling the space proposals from $\Theta^T = \{\theta_1^T, \cdots, \theta_m^T\}$. The $\theta_i^T,\ i \in \{1, \ldots, m\}$ represents the distributions for sampling the architectures $A_{\theta_i^T}$.

The figure 6 illustrates the strategy of space proposal selection. After defining the computational targets T, we optimize m different space proposals, and each space proposal is utilized for sampling architectures that satisfy the targets. The π is used for sampling space proposals. The two-level sampling strategy is used to sample architecture in every iteration. At each iteration, π is used to sample a space proposal θ_i^T and the θ_i^T is used to sample an individual architecture $A_{\theta_i^T}$.

D Overall Search Algorithm

Algorithm 1 The searching algorithm of HourNAS.

```
Input: The NAS supernet $\mathcal{S}$, the computational targets $T_i, i \in \{1,\dots,n\}$, the train set $D_{tr}$ and
    validation set $D_{val}$, the searching epochs for vital blocks $E_{vital}$ and non-vital blocks $E_{n-vital}$,
    the number of space proposals $m$, iterations $I_{sp}$ for training space proposals.
 1: // Search Vital Blocks
 2: Constructing the minimal SuperNet $\mathcal{S}_{vital}$ and the architecture parameter $\theta_{vital}$.
 3: for e = 1 to $E_{vital}$ do
 4:     for Data and target pair $(X_{tr}, Y_{tr})$ in $D_{tr}$ do
 5:         Sample network $A$ from $\theta_{vital}$, calculating loss and update network parameters.
 6:     end for
 7:     for Data and target pair $(X_{val}, Y_{val})$ in $D_{val}$ do
 8:         Sample network $A$ from $\theta_{vital}$, calculating loss and update $\theta_{vital}$.
 9:     end for
10: end for
11: The operations which have the highest importance are selected to form the vital layers.
12: // Search Non-Vital Blocks
13: According to the computational targets $T$, HourNAS trains $m$ proposals $\Theta^T = \{\theta_1^T, \cdots, \theta_m^T\}$
    for $I_{sp}$ iterations, and construct the proposal sampler $\pi$ .
14: for e = 1 to $E_{n-vital}$ do
15:     for Data and target pair $(X_{tr}, Y_{tr})$ in $D_{tr}$ do
16:         Sampling network $A$ from $\pi$ and $\Theta$, calculating loss and update network parameters.
17:     end for
18:     for Data and target pair $(X_{val}, Y_{val})$ in $D_{val}$ do
19:         Sampling network $A$ from $\pi$ and $\Theta$, calculating loss and update $\pi$ and $\Theta$.
20:     end for
21: end for
22: Fixing operations by selecting the space proposal and operations with the highest probability.
Output: The architecture $A$ which satisfies the computational targets $T_i, i \in \{1,\dots,n\}$.
```

E The Architectures on FBNet Search Space

HourNAS-A/B/G/I use the same search space and backbone as FBNet [42]. HourNAS searches for the kernel size, expansion ratio, and operations. The search space is defined as follows, and we also list the searched architectures in detail. It is worth noticing that we do not use bells and whistles like swish, SE modules in this experiment.

Table 7: The inverted residual block with the following settings.

Block Type	Expansion	Kernel	Group
k3g2	1	3	2
k3e1	1	3	1
k3e3	3	3	1
k3e6	6	3	1
k5g2	1	5	2
k5e1	1	5	1
k5e3	3	5	1
k5e6	6	5	1
skip	-	-	-

Table 8: Detailed architectures for HourNAS-A/B/G/I. The backbone is same as FBNet [42].

Layer	Out	Stride	SuperNet	HourNAS-A	HourNAS-B	HourNAS-G w/o crit priori	HourNAS-I	FBNet-Max
Conv	16	2	conv3 × 3	conv3 × 3	conv3 × 3	conv3 × 3	conv3 × 3	conv3 × 3
layer1_1	16	1	Super	k3e1	k3e3	k3e3	k3e1	k5e6
layer2_1	24	2	Super	k5e6	k5e6	k5g2	k5e6	k5e6
layer2_2	24	1	Super	k5g2	k3e3	k3g2	k5e1	k5e6
layer2_3	24	1	Super	skip	k5g2	k3e3	31	k5e6
layer2_4	24	1	Super	k3g2	skip	k3e3	31	k5e6
layer3_1	32	2	Super	k5e6	k5e6	k5e1	56	k5e6
layer3_2	32	1	Super	k3g2	k3e3	k3e1	53	k5e6
layer3_3	32	1	Super	k5g2	k3e3	k3e3	53	k5e6
layer3_4	32	1	Super	k5e1	k3e3	k3e3	33	k5e6
layer4_1	64	2	Super	k5e6	k5e6	k5e3	56	k5e6
layer4_2	64	1	Super	k5e3	k5e6	k3e6	53	k5e6
layer4_3	64	1	Super	k3e1	k5e6	k3e3	53	k5e6
layer4_4	64	1	Super	k5e3	k3e6	k5e6	56	k5e6
layer5_1	112	1	Super	k5e6	k5e6	k3e3	56	k5e6
layer5_2	112	1	Super	k3e3	k3e6	k3e3	53	k5e6
layer5_3	112	1	Super	k3e3	k3e6	k5e3	31	k5e6
layer5_4	112	1	Super	k3e3	k5e3	k5e3	53	k5e6
layer6_1	184	2	Super	k5e6	k5e6	k3e6	56	k5e6
layer6_2	184	1	Super	k5e3	k3e6	k5e6	33	k5e6
layer6_3	184	1	Super	k5e3	k3e6	k3e6	53	k5e6
layer6_4	184	1	Super	k3e6	k5e6	k5e3	56	k5e6
layer7_1	352	1	Super	k5e6	k5e6	k3e3	56	k5e6
Params	-	-	-	4.8	5.5	4.7	4.8	5.7
FLOPs	-	-	-	298	406	297	318	583
Top-1 (%)	-	-	-	74.1	75.0	73.2	74.2	75.7
Top-5 (%)	-	-	-	91.8	92.2	91.4	91.8	92.8

F The Enlarged Search Space on FBNet Backbone

For our HourNAS-C/D/H experiment, the search space and the SuperNet are defined as below. The backbone is the same as FBNet [42] and the search space is slightly enlarged. The architectures are detailed in Table 10.

Table 9: The inverted residual block with the following settings.

Block Type	Expansion	Kernel	Group
k3e1	1	3	1
k3e3	3	3	1
k3e6	6	3	1
k5e1	1	5	1
k5e3	3	5	1
k5e6	6	5	1
k7e1	1	7	1
k7e3	3	7	1
k7e6	6	7	1
skip	-	-	-

Table 10: Detailed architectures for HourNAS-C/D/H. The backbone is same as FBNet [42].

Layer	Out	Stride	SuperNet	HourNAS-C	HourNAS-D	HourNAS-H w/o crit priori
Conv	16	2	conv3 $\times$ 3	conv3 $\times$ 3	conv3 $\times$ 3	conv3 $\times$ 3
layer1_1	16	1	Super	skip	k3e3	k3e3
layer2_1	24	2	Super	k3e6	k3e6	k3e1
layer2_2	24	1	Super	k3e3	k5e1	k3e3
layer2_3	24	1	Super	skip	k5e1	k3e3
layer2_4	24	1	Super	k3e3	k5e3	k5e1
layer3_1	32	2	Super	k5e6	k5e6	k5e3
layer3_2	32	1	Super	k3e3	k5e3	k3e3
layer3_3	32	1	Super	k3e3	k3e6	k3e3
layer3_4	32	1	Super	k3e3	k5e1	k7e1
layer4_1	64	2	Super	k5e6	k5e6	k7e3
layer4_2	64	1	Super	k3e3	k3e6	k7e3
layer4_3	64	1	Super	k5e3	k3e6	k7e3
layer4_4	64	1	Super	k7e1	k3e6	k7e3
layer5_1	112	1	Super	k7e6	k7e6	k7e3
layer5_2	112	1	Super	k5e3	k7e3	k5e3
layer5_3	112	1	Super	k5e1	k5e3	k5e3
layer5_4	112	1	Super	k5e1	k5e3	k7e3
layer6_1	184	2	Super	k7e6	k7e6	k7e3
layer6_2	184	1	Super	k5e3	k7e6	k3e6
layer6_3	184	1	Super	k3e3	k5e6	k7e3
layer6_4	184	1	Super	k3e6	k7e6	k7e3
layer7_1	352	1	Super	k7e6	k7e6	k7e6
Params	-	-	-	4.8	5.5	4.8
FLOPs	-	-	-	296	394	299
Top-1 (%)	-	-	-	74.1	75.3	73.5
Top-5 (%)	-	-	-	91.6	92.3	91.3

G The Architectures on MnasNet Search Space

For our HourNAS-E experiment, the search space and the SuperNet are defined as below, which are the same as MnasNet [38]. The architectures are detailed in Table 12.

Table 11: The inverted residual block with the following settings.

Block Type	Expansion	Kernel	Group	SE
k3e1	1	3	1	False
k3e3	3	3	1	False
k3e6	6	3	1	False
k5e1	1	5	1	False
k5e3	3	5	1	False
k5e6	6	5	1	False
k3e1se	1	3	1	True
k3e3se	3	3	1	True
k3e6se	6	3	1	True
k5e1se	1	5	1	True
k5e3se	3	5	1	True
k5e6se	6	5	1	True
skip	-	-	-	-

Table 12: Detailed architectures for HourNAS-E. The backbone is same as MnasNet [38].

Layer	Out	Stride	SuperNet	HourNAS-E
Conv	32	2	conv3 $\times$ 3	conv3 $\times$ 3
layer1_1	16	1	SepConv3 $\times$ 3	SepConv3 $\times$ 3
layer2_1	24	2	Super	k5e6
layer2_2	24	1	Super	k3e3se
layer2_3	24	1	Super	k3e1se
layer2_4	24	1	Super	k3e3
layer3_1	40	2	Super	k5e6se
layer3_2	40	1	Super	k5e1
layer3_3	40	1	Super	k3e1se
layer3_4	40	1	Super	k5e1
layer4_1	80	2	Super	k5e6se
layer4_2	80	1	Super	k3e3se
layer4_3	80	1	Super	k3e3
layer4_4	80	1	Super	k3e3s3
layer5_1	112	1	Super	k5e6se
layer5_2	112	1	Super	k3e3se
layer5_3	112	1	Super	k3e3
layer5_4	112	1	Super	k3e3
layer6_1	160	2	Super	k5e6se
layer6_2	160	1	Super	k3e6
layer6_3	160	1	Super	k5e3se
layer6_4	160	1	Super	k5e3se
layer7_1	320	1	Super	k5e6se
Params	-	-	-	3.8
FLOPs	-	-	-	313
Top-1 (%)	-	-	-	75.7
Top-5 (%)	-	-	-	92.8

H The Architectures on EfficientNet Search Space

For our HourNAS-F experiment, we use the backbone the same as EfficientNet [39]. The search space and the architectures are detailed below.

Table 13: The inverted residual block with the following settings.

Block Type	Expansion	Kernel	Group	SE
k3e1se	1	3	1	True
k3e3se	3	3	1	True
k3e6se	6	3	1	True
k5e1se	1	5	1	True
k5e3se	3	5	1	True
k5e6se	6	5	1	True
skip	-	-	-	-

Table 14: Detailed architectures for HourNAS-F. The backbone is same as EfficientNet [39].

Layer	Out	Stride	SuperNet	HourNAS-F	EfficientNet-Max
Conv	32	2	conv3 $\times$ 3	conv3 $\times$ 3	conv3 $\times$ 3
layer1_1	16	1	k3e1se	k3e1se	k5e6se
layer2_1	24	2	Super	k5e6se	k5e6se
layer2_2	24	1	Super	k5e1se	k5e6se
layer2_3	24	1	Super	k5e1se	k5e6se
layer2_4	24	1	Super	k3e1se	k5e6se
layer3_1	40	2	Super	k5e6se	k5e6se
layer3_2	40	1	Super	k5e1se	k5e6se
layer3_3	40	1	Super	k3e1se	k5e6se
layer3_4	40	1	Super	k5e1se	k5e6se
layer4_1	80	2	Super	k5e6se	k5e6se
layer4_2	80	1	Super	k3e6se	k5e6se
layer4_3	80	1	Super	k3e6se	k5e6se
layer4_4	80	1	Super	k3e6se	k5e6se
layer5_1	112	1	Super	k5e6se	k5e6se
layer5_2	112	1	Super	k5e3se	k5e6se
layer5_3	112	1	Super	k5e3se	k5e6se
layer5_4	112	1	Super	k5e3se	k5e6se
layer6_1	192	2	Super	k5e6se	k5e6se
layer6_2	192	1	Super	k5e6se	k5e6se
layer6_3	192	1	Super	k3e6se	k5e6se
layer6_4	192	1	Super	k5e6se	k5e6se
layer7_1	320	1	Super	k5e6se	k5e6se
Params	-	-	-	5.3	5.8
FLOPs	-	-	-	383	738
Top-1 (%)	-	-	-	77.0	78.3
Top-5 (%)	-	-	-	93.5	94.0