Improving One-shot NAS by Suppressing the Posterior Fading

Xiang Li*
Brown University
xiang_li_1@brown.edu

Chen Lin*, Chuming Li, Ming Sun, Wei Wu, Junjie Yan
SenseTime Group Limited
{linchen,lichuming,sunming1,wuwei,yanjunjie}@sensetime.com

Wanli Ouyang
The University of Sydney
wanli.ouyang@sydney.edu.au

Abstract

Neural architecture search (NAS) has demonstrated much success in automatically designing effective neural network architectures. To improve the efficiency of NAS, previous approaches adopt weight sharing method to force all models share the same set of weights. However, it has been observed that a model performing better with shared weights does not necessarily perform better when trained alone. In this paper, we analyse existing weight sharing one-shot NAS approaches from a Bayesian point of view and identify the ***Posterior Fading*** *problem, which compromises the effectiveness of shared weights. To alleviate this problem, we present a novel approach to guide the parameter posterior towards its true distribution. Moreover, a hard latency constraint is introduced during the search so that the desired latency can be achieved. The resulted method, namely Posterior Convergent NAS (PC-NAS), achieves state-of-the-art performance under standard GPU latency constraint on ImageNet.*

1. Introduction

Neural network design requires extensive experiments by human experts. In recent years, there has been a growing interest in developing algorithmic NAS solutions to automate the manual process of architecture design [39, 16, 18]. Despite remarkable results [21, 38], early works on NAS [28, 12] are limited to searching only using proxy or subsampled dataset due to the exorbitant computational cost. To overcome this difficulty, [3, 27] attempted to improve search efficiency via sharing weights across models. These approaches utilize an over-parameterized network (supergraph) containing every single model, which can be further divided into two categories.

The first category is continuous relaxation method [23, 6], which keeps a set of so called architecture parameters to represent the model, and updates these parameters alternatively with supergraph weights. The resulting model is obtained using the architecture parameters at convergence. The continuous relaxation method suffers from the rich-get-richer problem [1], which means that a better-performed model at the early stage would be trained more frequently (or have larger learning rates). This introduces bias and instability to the search process.

The other category is referred to as one-shot method [5, 13, 3, 9], which divides the NAS procedure into a training stage and a searching stage. In the training stage, the supergraph is optimized along with either dropping out each operator with certain probability or sampling uniformly among candidate architectures. In the search stage, a search algorithm is applied to find the architecture with the highest validation accuracy with shared weights. The one-shot approach ensures the fairness among all models by sampling architecture or dropping out operator uniformly. However, as identified in [1, 9, 3], the problem of one-shot method is that the validation accuracy of the model with shared weights is not predictive to its true performance.

In this paper, we formulate NAS as a Bayesian model selection problem [8]. This formulation is especially helpful in understanding the one-shot approaches in a theoretical way, which in turn provides us a guidance to fundamentally addressing one of the major issues of one-shot approaches. Specially, we show that shared weights are actually a maximum likelihood estimation of a proxy distribution to the true parameter distribution. Most importantly, we identify the common issue of weight sharing, which we call **Posterior Fading**, i.e., as the number of models in the supergraph increases, the KL-divergence between true parameter posterior and proxy posterior also increases.

To alleviate the Posterior Fading problem, we proposed a practical approach to guide the convergence of the proxy

*Equal Contribution

distribution towards the true parameter posterior. Specifically, we divide the training of supergraph into several intervals and maintain a pool of high potential partial models and progressively update this pool after each interval . At each training step, a partial model is sampled from the pool and complemented to a full model, where full model means an architecture that has full number of layers provided by the search. To update the partial model pool, we first generate candidates by extending each partial model and evaluate their potentials. The top ones among them form the new pool. The search space is effectively shrunk in the upcoming training interval. Consequently, the parameter posterior get closer to the desired true posterior during this procedure. Main contributions of our work is concluded as follows:

- We for the first time analyse one-shot approach from a theoretical point of view and identify the real problem of this method, which we call Posterior Fading. This perspective will provide insights for further study.

- Guided by our Bayesian result, we introduce a novel NAS algorithm fundamentally different from existing ones, which guides the proxy distribution to converge towards the true parameter posterior.

- We benchmark our method's performance on ImageNet [30] against the existing models and a new powerful architecture, PC-NAS, is discovered. In one typical search space [6], our PC-NAS-S attains 76.8% top-1 accuracy, 0.5% higher and 20% faster than EfficientNet-B0 [33], which is the current state-of-the-art model in mobile setting. To further demonstrate the advantage of our method, we test it on a larger space and our PC-NAS-L boosts the accuracy to 78.1%.

2. Related work

Early neural architecture search (NAS) [24, 22, 29, 39, 2, 35] methods normally involves reinforcement learning or neuro-evolution. This type of NAS is typically considered as an agent-based explore and exploit process, where an agent (e.g. an evolution mechanism or a recurrent neural network(RNN)) is introduced to explore a given architecture space with training a network in the inner loop to get an evaluation for guiding exploration. Such methods are computationally expensive and hard to be used on large-scale datasets, e.g. ImageNet.

Recent works [27, 4, 23, 6] try to alleviate this computation cost via modeling NAS as a single training process of an over-parameterized network that comprises all candidate models, in which weights of the same operators in different models are shared. ENAS [27] reduces the computation cost by orders of magnitude, while requires an RNN agent and focuses on small-scale datasets (e.g. CIFAR10). One-shot NAS [5] trains the over-parameterized network along with dropping out each operator with increasing probability. Then it uses the pre-trained over-parameterized network to evaluate randomly sampled architectures. DARTS [23] additionally introduces a real-valued architecture parameter for each operator and alternately train operator weights and architecture parameters by back-propagation. ProxylessNAS [6] binarizes the real-value parameters in DARTS to save the GPU computation and memory for training the over-parameterized network. SNAS [37] employs Gumbel random variables to directly optimize the NAS objective. [11] develops a differentiable sampler over the search space to achieve impressive speed.

The paradigm of ProxylessNAS [6] and DARTS [23] introduce unavoidable bias since operators of models performing well in the beginning will easily get trained more and normally keep being better than other. But they are not necessarily superior than others when trained from scratch.

Other relevant works are ASAP [26] and XNAS [25], which introduce pruning during the training of over-parameterized networks to improve the efficiency of NAS. Similar to these approaches, we start with an over-parameterized network and then reduce the search space to derive the optimized architecture. Instead of focusing on the speed-up of training, we further improve the rankings of models and evaluate operators directly on validation set.

3. Methods

In this section, we first formulate neural architecture search in a Bayesian manner. Utilizing this setup, we introduce the PC-NAS algorithm and analyse its advantage comparing to previous approaches. Finally, we discuss the search algorithm combined with latency constraint.

3.1. A Probabilistic Setup for Model Uncertainty

The Bayesian setup of model comparison simply involves the use of probabilities to represent uncertainty in the choice of model, along with a consistent application of the sum and product rules of probability. Suppose we want to compare a set of K different models $\mathcal{M} = \{\mathbf{m}_1, ..., \mathbf{m}_K\}$. Here a model refers to a probability distribution over the observed data $\mathcal{D}$ and $p(\mathcal{D}|\theta_k, \mathbf{m}_k)$ describes the probability density of data $\mathcal{D}$ given model $\mathbf{m}_k$ and its associated parameters θ_k. The Bayesian approach proceeds by assigning a prior probability distribution $p(\theta_k|\mathbf{m}_k)$ to the parameters of each model, and a prior probability $p(\mathbf{m}_k)$ to each model.

In order to ensure fairness among all models, we set the model prior $p(\mathbf{m}_k)$ a uniform distribution. Under previous setting, we can drive

$$p(\mathbf{m}_k|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{m}_k)p(\mathbf{m}_k)}{\sum_k p(\mathcal{D}|\mathbf{m}_k)p(\mathbf{m}_k)}, \tag{1}$$

where

$$p(\mathcal{D}|\mathbf{m}_k) = \int p(\mathcal{D}|\theta_k, \mathbf{m}_k) p(\theta_k|\mathbf{m}_k) \mathrm{d}\theta_k. \tag{2}$$

Since $p(\mathbf{m}_k)$ is uniform, the Maximum Likelihood Estimation (MLE) of $\mathbf{m}_k$ is just the maximum of (2), which expresses the preference shown by the data for different models. It can be inferred that, $p(\theta_k|\mathbf{m}_k)$ is crucial to the solution of the model selection. We are interested in attaining the model with highest test accuracy in a trained alone manner, thus the parameter prior is just the posterior $p_{\text{alone}}(\theta_k|\mathbf{m}_k, \mathcal{D})$ which means the distribution of θ_k when $\mathbf{m}_k$ is trained alone on dataset $\mathcal{D}$. Thus we would use the term true parameter posterior to refer $p_{\text{alone}}(\theta_k|\mathbf{m}_k, \mathcal{D})$.

3.2. Network Architecture Selection In a Bayesian Point of View

We constrain our discussion in the setting which is frequently used in NAS literature for simplicity. As a building block of our search space, a mixed operator (mixop), denoted by $\mathbb{O} = \{O_1 \dots, O_N\}$, contains N different choices of candidate operators O_i for $i = 1, \dots N$ in parallel. The search space is defined by L mixed operators (layers) connected sequentially interleaved by downsampling as in Fig. 1(a). The network architecture (model) $\mathbf{m}$ is defined by a vector $[o_1, o_2, ..., o_L]$, $o_l \in \mathbb{O}$ representing the choice of operator for layer l. The parameter for the operator o at the l-th layer is denoted as θ_{lo}. The parameters of the supergraph are denoted by θ which includes $\{\theta_{lo}|l \in \{1, 2, ..., L\}, o \in \mathbb{O}\}$. In this setting, the parameters of each candidate operator are shared among multiple architectures. The parameters related with a specific model $\mathbf{m}_k$ is denoted as $\theta_k = \theta_{1,o_1}, \theta_{2,o_2}, ..., \theta_{L,o_L}$, which is a subset of the parameters of the supergraph θ. Obtaining the $p_{\text{alone}}(\theta_k|\mathbf{m}_k, \mathcal{D})$ or a MLE of it for every model is computationally intractable. Therefore, the one-shot method trains the supergraph by dropping out each operator [5] or sampling different architectures [3, 9] and utilize the shared weights to evaluate single model. In this work, we adopt the latter training paradigm while the former one could be easily generalized. Suppose we sample a model $\mathbf{m}_k$ and optimize the supergraph with a mini-batch of data based on the objective function L_{alone}:

$$\begin{aligned} & -\log p_{\text{alone}}(\theta|\mathbf{m}_k, \mathcal{D}) \\ \propto\ & L_{\text{alone}}(\theta, \mathbf{m}_k, \mathcal{D}) \\ \propto\ & -\log p_{\text{alone}}(\mathcal{D}, \theta|\mathbf{m}_k) - \log p(\mathbf{m}_k) \\ \propto\ & -\log p_{\text{alone}}(\mathcal{D}|\theta, \mathbf{m}_k) - \log p(\theta|\mathbf{m}_k), \end{aligned} \tag{3}$$

where $-\log p(\theta|\mathbf{m}_k)$ could be seen as a regularization term. Thus minimizing this objective equals to making MLE to $p_{\text{alone}}(\theta|\mathbf{m}_k, \mathcal{D})$. When training the supergraph, we sample many models $\mathbf{m}_k$, and then train the parameters for these models, which corresponds to a stochastic approximation of the following objective function:

$$L_{\text{share}}(\theta, D) = \frac{1}{K} \sum_k L_{\text{alone}}(\theta, \mathbf{m}_k, \mathcal{D}). \tag{4}$$

By taking exponential on both sides of the equation 4, it is equivalent to adopting a proxy parameter posterior as follows:

$$p_{\text{share}}(\theta|\mathcal{D}) = \frac{1}{Z} \prod_k p_{\text{alone}}(\theta|\mathbf{m}_k, \mathcal{D}), \tag{5}$$

$$-\log p_{\text{share}}(\theta|\mathcal{D}) = -\sum_k \log p_{\text{alone}}(\theta|\mathbf{m}_k, \mathcal{D}) + \log Z, \tag{6}$$

where K and Z are normalizing factors. One thing worth noting is that $p_{\text{alone}}(\theta|\mathbf{m}_k, \mathcal{D})$ are all independent for different k, since different models have different and independent parameter distributions. Then maximizing $p_{\text{share}}(\theta|D)$ is equivalent to minimizing L_{share}. For each single layer, its parameter $\theta_{l,o}$ is affected by all the remaining layers. By the intrinsic randomness of our uniform model sampling, we could further assume that

$$p_{\text{share}}(\theta_{l,o}|\mathcal{D}) = \prod_k p_{\text{alone}}(\theta_{l,o}|\mathbf{m}_k, \mathcal{D}), \tag{7}$$

which basically means the distribution of parameter $\theta_{l,o}$ is ultimately determined by all its marginal distribution from sampled architectures.

The KL-divergence between $p_{\text{alone}}(\theta_{l,o}|\mathbf{m}_k, \mathcal{D})$ and $p_{\text{share}}(\theta_{l,o}|\mathcal{D})$ follows:

$$\begin{aligned} & D_{\mathcal{KL}}\Big(p_{\text{alone}}(\theta_{l,o}|\mathbf{m}_k, \mathcal{D}) \,\Big\|\, p_{\text{share}}(\theta_{l,o}|\mathcal{D})\Big) \\ &= \int p_{\text{alone}}(\theta_{l,o}|\mathbf{m}_k, \mathcal{D}) \log \frac{p_{\text{alone}}(\theta_{l,o}|\mathbf{m}_k, \mathcal{D})}{p_{\text{share}}(\theta_{l,o}|\mathcal{D})} \mathrm{d}\theta \\ &= \int p_{\text{alone}}(\theta_{l,o}|\mathbf{m}_k, \mathcal{D}) \log \frac{p_{\text{alone}}(\theta_{l,o}|\mathbf{m}_k, \mathcal{D})}{\prod_i p_{\text{alone}}(\theta_{l,o}|\mathbf{m}_i, \mathcal{D})} \mathrm{d}\theta \\ &= \sum_{i \neq k} - \int p_{\text{alone}}(\theta_{l,o}|\mathbf{m}_k, \mathcal{D}) \log p_{\text{alone}}(\theta_{l,o}|\mathbf{m}_i, \mathcal{D}) \mathrm{d}\theta. \end{aligned} \tag{8}$$

The KL-divergence is just the summation of the cross-entropy of $p_{\text{alone}}(\theta_{l,o}|\mathbf{m}_k, \mathcal{D})$ and $p_{\text{alone}}(\theta_{l,o}|\mathbf{m}_i, \mathcal{D})$ where $i \neq k$. The cross-entropy term is always positive. Increasing the number of architectures would push p_{share} away from p_{alone}, namely the **Posterior Fading**. We conclude that non-predictive problem originates naturally from one-shot supergraph training, as KL-divergence grows with the number of architectures in the search space and a typical search space contains huge 10^{21} architectures. Thus if we effectively reduce the number of architectures in (8) during training, the KL divergence would decrease. This is the intuition of our PC-NAS algorithm.

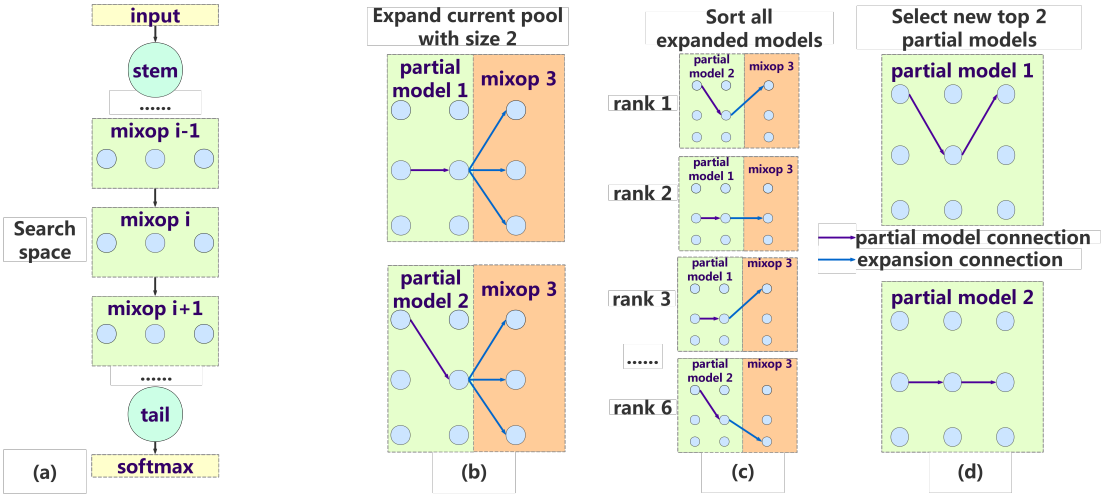

Figure 1. One example of search space(a) and PC-NAS process(b)(c)(d). Each mixed opperator consists of N(=3 in this figure) operators. However, only one operator in each mixop is invoked at a time for each batch. In (b), partial models 1 and 2 in the pool consist of choices in mixop 1 and 2. We extend these 2 partial models to the mixop 3. 6 extended candidate models are evaluated and ranked in(c). In (d), the new pool consists of the top-2 candidate models that are ranked in (c).

Algorithm 1 Potential: Evaluating the Potential of Partial Candidates

Inputs: G(supergraph), L(num of mixops in G), $\mathbf{m}'$(partial candidate), Lat(latency constraint), S(evaluation number), D_{val} (validataion set)
Scores = $\emptyset$
for $i = 1 : S$ **do**
  $\mathbf{m}^* = \text{expand}(\mathbf{m}')$ *randomly expand* $\mathbf{m}'$ *to full depth* L
  if Latency($\mathbf{m}^*$) $>$ Lat **then**
    continue *dump samples that don't satisfy the latency constraint*
  end if
  acc = Acc($\mathbf{m}^*$, D_{val}) *inference* $\mathbf{m}^*$ *for one batch and return its accuracy*
  Scores.append(acc) *save accuracy*
end for
Outputs: Average(Scores)

3.3. Posterior Convergent NAS

The naive approach to mitigate the posterior fading problem is to limit the number of candidate models inside the supergraph. However, large number of candidates is demanded for NAS to discover promising models. Due to this conflict, we present PC-NAS which adopts progressive search space shrinking. The resulting algorithm divides the training of shared weights into L intervals, where L is the number of mixed operators in the search space. The number of training epochs of a single interval is denoted as T_i. We will explain the key components of our method separately.

Partial model pool is a collection of partial models. At the l-th interval, a single partial model should contain $l-1$ selected operators $[o_1, o_2, ..., o_{l-1}]$. The size of partial model pool is denoted as P. After the l-th interval, each partial model in the pool will be extended by the N operators in l-th mixop. Thus there are $P \times N$ candidate extended partial models with length l. These candidate partial models are evaluated and the top-P among which are used as the partial model pool for the interval $l+1$. An illustrative exmaple of partial model pool update is in Fig. 1(b)(c)(d).

Candidate evaluation with latency constraint We define the potential of a partial model to be the expected validation accuracy of the models which contain the partial model.

$$\text{Potential}(o_1, o_2, ..., o_l) = E_{\mathbf{m} \in \{\mathbf{m} | m_i = o_i, \forall i \leq l\}}(Acc(\mathbf{m})). \tag{9}$$

where the validation accuracy of model $\mathbf{m}$ is denoted by $Acc(\mathbf{m})$. We estimate this value by uniformly sampling valid models and computing the average of their validation accuracy using one mini-batch. We use S to denote the evaluation number, which is the total number of sampled models. We observe that when S is large enough, the potential of a partial model is fairly stable and discriminative among candidates. See Algorithm 1 for pseudocode. The latency constraint is imposed by discarding invalid full models when calculating potentials of partial models in the pool. Unlike previous soft constraint training methods [6, 36], our PC-NAS will guarantee the latency constraint to be satisfied.

Training based on partial model pool The training iteration of the supergraph along with the partial model pool

has two steps. First, for a partial model from the pool, we randomly sample the missing operator $\{o_{l+1}, o_{l+2}, ..., o_L\}$ to complement the partial model to a full model. Then we optimize θ using the sampled full model and mini-batch data. Initially, the partial model pool is empty. Thus the supergraph is trained by uniformly sampled models, which is identical to previous one-shot training stage. After the initial training, all operators in the first mixop are evaluated. The top P operators form the partial model pool in the second training stage. Then, the supergraph resumes training and the training procedure is identical to the one discussed in last paragraph. Inspired by warm-up, the first stage is set much more epochs than following stages denoted as T_w. The whole PC-NAS process is elaborated in algorithm 2. The number of models in shrunk search space at the interval l is strictly less than interval $l-1$. At the final interval, the number of cross-entropy terms in (8) are P-1 for each architectures in final pool. Thus the parameter posterior of PC-NAS would move towards the true posterior during these intervals.

Algorithm 2 PC-NAS: Posterior Convergent Architecture Search

Inputs: P(size of partial model pool), G(supergraph), O_i (the ith operator in mixed operator), L(num of mixed operators in G), T_w(warm-up epochs), T_i(interval between updation of partial model pool), D_{train}(train set), D_{val} (validataion set), Lat(latency constraint)
PartialModels = $\emptyset$
Warm-up(G, D_{train}, T_w) *uniformly sample models from G and train*
for I = 0:($L \cdot T_i - 1$) **do**
  if I mod T_i == 0 **then**
    ExtendedPartialModels = $\emptyset$
    if PartialModels == $\emptyset$ **then**
      ExtendedPartialModels.append($[O_i]$) *add all operator in the first mixop*
    end if
    for $\mathbf{m}$ in PartialModels **do**
      ExtendedPartialModels.append(Extend($\mathbf{m}$,O_1), ..., Extend($\mathbf{m}$,O_N))
    end for
    for $\mathbf{m}'$ in ExtendedPartialModels **do**
      $\mathbf{m}'$.potential = Potential($\mathbf{m}'$, D_{val}, Lat, S) *evaluate the extended partial model*
    end for
    PartialModels = Top(ExtendedPartialModels, P) *keep P best partial models*
  end if
  Train(PartialModels, D_{train}) *train one epoch using partial models*
end for
Outputs: PartialModels

4. Experiments Results

We demonstrate the effectiveness of our methods on ImageNet [30], a large scale benchmark dataset, whose training set contains 1,000,000 training samples. For this task, we focus on models that have high accuracy under certain GPU latency constraint. We search models using PC-NAS, which progressively updates a partial model pool and trains shared weights. Then, we select the model with the highest potential in the pool and report its performance on the test set after training from scratch. Finally, we investigate the transferability of the model learned on ImageNet by evaluating it on two tasks, object detection and person re-identification.

4.1. Training Details

Dataset and latency measurement: As a common practice, we randomly sample 50,000 images from the train set to form a validation set during the model search. We conduct our PC-NAS on the remaining images in train set. The original validation set is used as test set to report the performance of the model generated by our method. The latency is evaluated on Nvidia GTX 1080Ti and the batch size is set 16 to fully utilize GPU resources.

Search spaces: We use two search spaces. We benchmark our **small space** similar to recent state-of-the-art NAS systems ProxylessNAS [6], FBNet [36] and EfficientNet [33] for fair comparison. To test our PC-NAS method in a more complicated search space, we add 3 more kinds of operators to the small space's mixoperators to construct our **large space**. Details of the two spaces are in A.1.

PC-NAS hyperparameters: We use PC-NAS to search in both small and large space. To balance training time and performance, we set evaluation number $S = 900$ and partial model pool size $P = 5$ in both experiments. Ablation study of the two values is in 5. When updating weights of the supergraph, we adopt mini-batch nesterov SGD optimizer with momentum 0.9, cosine learning rate decay from 0.1 to 5e-4, batch size 512, and L2 regularization with weight 1e-4. The warm-up epochs T_w and shrinking interval T_i are set 100 and 5, thus the total training of supergraph lasts $100 + 20 \times 5 = 200$ epochs. After searching, we select the best one from the top 5 final partial models and train it from scratch. Similar to EfficientNet [33] and MixNet [34], We add squeeze-and-excitation (SE) layers [15] to our model at the end of each operator. However, given that squeeze-and-excitation is relatively new and many existing models don't have this extra optimization, results with and without SE layers are listed separately in the Table 1. Our PC-NAS models perform consistently the best in either case.

4.2. ImageNet Results

Table 1 shows the performance of our model on ImageNet. We set our target latency at $10ms$ according to our

Table 1. PC-NAS' Imagenet results compared with state-of-the-art methods in the *mobile* setting.

model	space	params	latency	top-1	top-1(+SE)
MobileNetV2 1.4x [31]	-	6.9 M	10 ms	74.7%	-
AmoebaNet-A [28]	-	5.1 M	23 ms	74.5%	-
PNASNet [21]	$5.6\text{x}10^{14}$	5.1 M	25 ms	74.2%	-
FBNet-C [36]	10^{21}	5.5 M	-	74.9%	-
MnasNet [32]	-	4.4 M	11 ms	74.8%	76.1%
ProxylessNAS-gpu [6]	7^{21}	7.1 M	8 ms	75.1%	-
MixNet-S [34]	-	4.1 M	13 ms	-	75.8%
EfficientNet-B0 [33]	-	5.3 M	13 ms	-	76.3%
RandomSearch	10^{21}	3.6M	10 ms	75.5%	-
PC-NAS-S	10^{21}	5.1 M	10 ms	76.1%	76.8%
PC-NAS-L	20^{21}	15.3 M	11 ms	77.5%	78.1%

measurement of mobile setting models on GPU. Our search result in the small space, namely PC-NAS-S, achieves 76.8% top-1 accuracy under our latency constraint, which is 0.5% higher than EffcientNet-B0 (in terms of absolute accuracy improvement), 1% higher than MixNet-S. If we slightly relax the time constraint, our search result from the large space, namly PC-NAS-L, achieves 78.1% top-1 accuracy, which improves top-1 accuracy by 1.8% compared to EfficientNet-B0, 2.3% compared to MixNet-S. Both PC-NAS-S and PC-NAS-L are faster than previous state-of-the-art models EffcientNet-B0 and MixNet-S.

4.3. Transferability of PC-NAS

We validate our PC-NAS's transferability on object detection. We use COCO [20] dataset as benchmark. For the dataset, PC-NAS-L pretrained on ImageNet is utilized as feature extractor, and is compared with other models under the same training script. The experiment is conducted with the two-stage framework FPN [19]. Table 2 shows the performance of our PC-NAS model on COCO. Our approach significantly surpasses the mAP of MobileNetV2 [31] as well as ResNet50 [14]. Compare to the standard ResNet101 [14] backbone, our model achieves comparable mAP quality with almost $1/3$ parameters and $2.3\times$ faster speed.

5. Ablation Study

In this section, we study the impact of hyperparameters, and discuss the effectiveness of our space shrinking technique and search method.

Impact of hyperparameters: We investigate the impact of hyperparameters on our method within our small space on ImageNet. The hyperparameters include warm-up, training epochs Tw, partial model pool size P, and evaluation number S. We tried setting Tw as 100 and 150 with fixed $P = 5$ and $S = 900$. The resulting models of these two settings show no significant difference in top-1 accuracy (less than 0.1%), shown as in Fig. 2. Thus we choose warm-up training epochs as 100 in our experiment to save computation resources. For the influence of P and S, we show the results in Fig. 2. It can be seen that the top-1 accuracy of the models found by PC-NAS increases with both P and S. Thus we choose $P = 5$, $S = 900$ in the experiments for better performance. we did not observe significant improvements when further increasing these two hyperparameters in our small space.

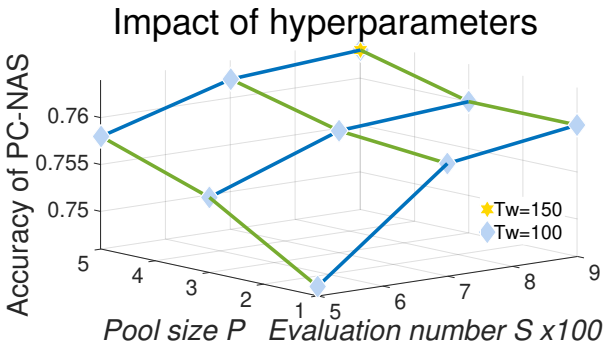

Figure 2. Influence of warm-up epochs T_w, partial model pool size P, and evaluation number S to the resulted model.

Effectiveness of shrinking search space: One advantage of one-shot method is that the shared weights of the supergraph could be used to conveniently predict the performance of various architectures. However, previous models [5, 21] turn out to be unsatisfactory when ranking models. To assess the extent space shrinking could mitigate this issue, we do the first comparison as follows. Initially, we select a bunch of models from the candidates of our final pool under small space, train them from scratch and evaluate their stand alone top-1 accuracy. Then we use One-Shot to train the supergraph also under small space without shrinking. Finally, we show the model rankings of PC-NAS and One-Shot using the accuracy obtained from inferring the models in the supergraphs trained with two methods. The difference is shown in Fig. 3: the pearson correlation

Table 2. Performance Comparison on COCO

backbone	params	latency	COCO
MobileNetV2 [31]	3.5 M	7 ms	31.7 mAP
ResNet50 [14]	25.5 M	15 ms	36.8 mAP
ResNet101 [14]	44.4 M	26 ms	39.4 mAP
ProxylessNAS [6]	6.3 M	11 ms	38.7 mAP
PC-NAS-L	15.3 M	11 ms	38.5 mAP

coefficients between stand-alone accuracy and accuracy in supergraph of One-Shot and PC-NAS are 0.11 and 0.92, thus models under PC-NAS's space shrinking can be ranked by their accuracy evaluated on sharing weights much more precisely than One-Shot. Repeated experiments have shown similar phenomenon.

Next, we explore how space shrinking influences our final searched results. We train the supergraph of our large space using One-Shot [5] method without any shrinking of the search space. Then we conduct model search on this supergraph by progressively updating a partial model pool in our method. The resulting model using this setting attains 77.1% top-1 accuracy on ImageNet, which is 1% lower than our PC-NAS-L. Since shrinking takes 100 additional epochs and training is more expensive than evaluation, a natural question to ask is, could additional models (larger P, larger S) instead of space shrinking compensate the unshrunk space. To address this question, we double the value of P or S, and the results are listed in Table 3. PC-NAS without space shrinking attains 77.4% and 77.2% respectively with larger P or S. From these results, we can conclude that our search method alone already lead to impressive performance. Larger P and S increase the final result around 0.2%, space shrinking strategy however improves model ranking and brings extra 0.7%-1% accuracy boost.

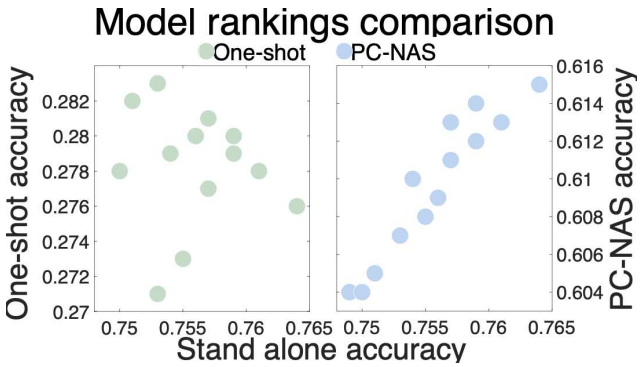

Figure 3. Comparison of model rankings for One-Shot (left) and PC-NAS (right).

Effectiveness of our search method: Generally speaking, our method can be classified as a sampling based NAS method.There are two popular counterparts in previous NAS works: Evolutionary Algorithm [7, 10] and random search [17]. It is widely admitted that EA methods outperform random search consistently. Thus to evaluate the performance of our search method, we utilize Evolutionary Algorithm (EA) as the baseline to search for models in the same supergraph trained with One-Shot. We implement EA with population size $p = 5$, aligned to the value of pool size P in our method, and set the mutation operation as randomly replace the operator in one mixop operator to another. We constrain the total number of validation images in EA the same as PC-NAS. The top-1 accuracy of discovered model drops to 75.9% accuracy, which is 2.2% lower than PC-NAS-L. When we further enlarge the population size of EA method to 20, the accuracy increases to 76.5%, but still not comparable to our search strategy. We conclude the performance gap comes from the better efficiency of our sampling method. In PC-NAS approach, every architecture is evaluated batch-wise, whose size is 512. Then average accuracy is computed to represent the the potential of each mixop. However, in classical EA methods, the whole validation set which contains 50000 images has to be traversed to evaluate a single model. This is not efficient especially when the shared weights are not predictive, as shown in Fig. 3. Thus the results summarized in Table 3 suggest the effectiveness of our novel search method.

Table 3. Comparison of different one-shot search methods

training method	search method	top-1 acc
PC-NAS	PC-NAS	**78.1%**
One-Shot	PC-NAS	77.1%
One-Shot	PC-NAS ($P = 10$)	77.4%
One-Shot	PC-NAS ($S = 1800$)	77.2%
One-Shot	EA ($p = 5$)	75.9%
One-Shot	EA ($p = 20$)	76.5%

6. Conclusion

In this paper, a novel architecture search approach called PC-NAS is proposed. We for the first time study the conventional weight sharing approach from Bayesian point of view and identify a key issue that compromises the effectiveness of shared weights. With the theoretical motivation, an original method is devised to mitigate this issue. Experi-

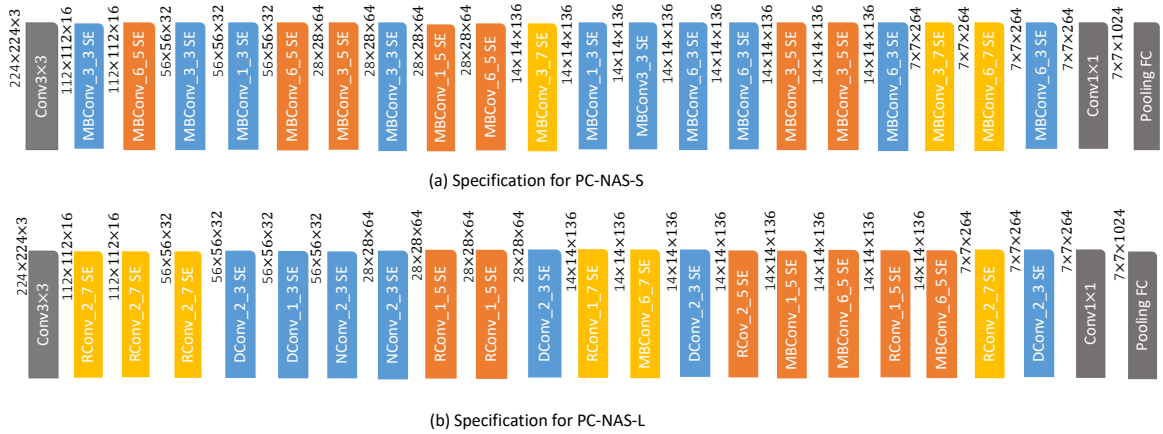

Figure 4. The architectures of PC-NAS-S and PC-NAS-L.

mental results demonstrate our approach can automatically find significantly better mobile setting models than existing approaches, and achieve new state-of-the-art results.

A. Appendix

A.1. Construction of the Search Space

The operators in our spaces have structures described by either Conv1x1-ConvNxM-Conv1x1 or Conv1x1-ConvNxM-ConvMxN-Conv1x1. We define **expand ratio** as the ratio between the channel numbers of the ConvNxM in the middle and the input of the first Conv1x1.

Small search space Our small search space contains a set of MBConv operators (mobile inverted bottleneck convolution [31]) with different kernel sizes and expand ratios, plus Identity, adding up to 10 operators to form a mixoperator. The 10 operators in our small search space are listed in the left column of Table 4, where notation OP_X_Y represents the specific operator OP with expand ratio X and kernel size Y.

Large search space We add 3 more kinds of operators to the mixoperators of our large search space, namely NConv, DConv, and RConv. We use these 3 operators with different kernel sizes and expand ratios to form 10 operators exclusively for large space, thus the large space contains 20 operators. For large search space, the structure of NConv, DConv are Conv1x1-ConvKxK-Conv1x1 and Conv1x1-ConvKxK-ConvKxK-Conv1x1, and that of RConv is Conv1x1-Conv1xK-ConvKx1-Conv1x1. The kernel sizes and expand ratios of operators exclusively for large space are listed in the right column of Table 4, where notation OP_X_Y represents the specific operator OP with expand ratio X and K=Y. The difference between these three kinds of operators lie in the choice of middle convolution operators. In principle, these three kinds of operators are increasingly general and capable, but at the same time, more and more time consuming. There is a trade off between expressiveness and speed for our PC-NAS to balance.

There are altogether 21 mixoperators in both small and large search spaces. Thus our small search space contains 10^{21} models, while the large one contains 20^{21}.

Table 4. operator table

Ops in both large and small space		**Ops exclusively in large space**	
MBConv_1_3	MBConv_3_3	NConv_1_3	NConv_2_3
MBConv_6_3	MBConv_1_5	DConv_1_3	DConv_2_3
MBConv_3_5	MBConv_6_5	RConv_1_5	RConv_2_5
MBConv_1_7	MBConv_3_7	RConv_4_5	RConv_1_7
MBConv_6_7	Identity	RConv_2_7	RConv_4_7

A.2. Specifications of Discovered Models

The specifications of PC-NAS-S and PC-NAS-L are shown in Fig. 4. We observe that PC-NAS-S adopts either high expansion rate or large kernel size at the tail end, which enables a full use of high level features. However, it tends to select small kernels and low expansion rates to ensure the model remains lightweight. PC-NAS-L chooses lots of powerful bottlenecks exclusively contained in the large space to achieve the accuracy boost. The high expansion rate is not quite frequently seen which is to compensate the computation utilized by large kernel size. Both PC-NAS-S and PC-NAS-L tend to use heavy operator when the resolution reduces, circumventing too much information loss in these positions.

References

[1] Georege Adam and Jonathan Lorraine. Understanding neural architecture search techniques. *arXiv preprint arXiv:1904.00438*, 2019.

[2] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *International Conference on Learning Representations*, 2017a.

[3] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc V. Le. Understanding and simplifying one-shot architecture search. *ICML*, 2018.

[4] Andrew Brock, Theodore Lim, James M. Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks. *NIPS Workshop on Meta-Learning*, 2017.

[5] Andrew Brock, J.M. Ritchie, Theodore Lim, and Nick Weston. Smash: One-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.

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

[7] Yukang Chen, Gaofeng Meng, Qian Zhang, Shiming Xiang, Chang Huang, Lisen Mu, and Xinggang Wang. Renas: Reinforced evolutionary neural architecture search. *arXiv:1808.00193*, 2019.

[8] Hugh Chipman, Edward I. George, and Robert E. McCulloch. The practical implementation of bayesian model selectio. In *Institute of Mathematical Statistics Lecture Notes - Monograph Series, 38*, pages 65–116, 2001.

[9] XiangXiang Chu, Bo Zhang, Ruijun Xu, and Jixiang Li. Fairnas:rethinking evaluation of weight sharing neural architecture search. *arXiv preprint arXiv:1907.01845v2*, 2019.

[10] Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, Peter Vajda, Matt Uyttendaele, and Niraj K. Jha. Chamnet: Towards efficient network design through platform-aware model adaptation. *arXiv:1812.08934*, 2018.

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

[12] Thomas Elsken, Jan-Hendrik Metzen, and Frank Hutter. Simple and efficient architecture search for convolutional neural networks. *ICLR workshop*, 2017.

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

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[15] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. Squeeze-and-excitation networks. *CVPR*, 2018.

[16] Chuming Li, Xin Yuan, Chen Lin, Minghao Guo, Wei Wu, Wanli Ouyang, and Junjie Yan. Am-lfs: Automl for loss function search. *arXiv preprint arXiv:1905.07375*, 2019.

[17] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.0763*, 2019.

[18] Chen Lin, Minghao Guo, Chuming Li, Xin Yuan, Wei Wu, Dahua Lin, Wanli Ouyang, and Junjie Yan. Online hyperparameter learning for auto-augmentation strategy. *arXiv preprint arXiv:1905.07373*, 2019.

[19] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[20] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[21] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Fei-Fei Li, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

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

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

[24] Geoffrey F. Miller, Peter M. Todd, and Shailesh U. Hegde. Designing neural networks using genetic algorithms. *ICGA*, pages volume 89, pages 379–384, 1989.

[25] Niv Nayman, Asaf Noy, Tal Ridnik, Itamar Friedman, Rong Jin, and Lihi Zelnik-Manor. Xnas: Neural architecture search with expert advice. *arXiv preprint arXiv:1906.08031*, 2019.

[26] Asaf Noy, Niv Nayman, Tal Ridnik, Nadav Zamir, Sivan Doveh, Itamar Friedman, Raja Giryes, and Lihi Zelnik-Manor. Asap: Architecture search, anneal and prune. *arXiv preprint arXiv:1904.04123*, 2019.

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

[28] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2018.

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

[30] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, pages 115(3):211–252, 2015.

[31] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted

residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[32] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. *arXiv preprint arXiv:1807.11626*, 2018.

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

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

[35] Linnan Wang, Yiyang Zhao, Yuu Jinnai, Yuandong Tian, and Rodrigo Fonseca. Alphax: exploring neural architectures with deep neural networks and monte carlo tree search. *arXiv preprint arXiv:1903.11059*, 2019.

[36] 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. *arXiv preprint arXiv:1812.03443*, 2018.

[37] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926*, 2018.

[38] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2423–2432, 2018.

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