# IMPROVING ONE-SHOT NAS BY SUPPRESSING THE POSTERIOR FADING

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Neural architecture search (NAS) has demonstrated 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 practical 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. In our small search space, our model PC-NAS-S attains $76.8\%$ top-1 accuracy, $2.1\%$ higher than MobileNetV2 (1.4x) with the same latency. When adopted to the large search space, PC-NAS-L achieves $78.1\%$ top-1 accuracy within 11ms. The discovered architecture also transfers well to other computer vision applications such as object detection and person re-identification.

## 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 (Zoph & Le, 2016; Liu et al., 2018a; Zhong et al., 2018; Li et al., 2019; Lin et al., 2019). Despite remarkable results, early works on NAS (Real et al., 2018; Elsken et al., 2017) are limited to searching only using proxy or subsampled dataset due to the exorbitant computational cost. To overcome this difficulty, (Bender et al., 2018; Pham et al., 2018) attempted to improve search efficiency via sharing weights across models. These approaches utilize an overparameterized network (supergraph) containing every single model, which can be further divided into two categories.

The first category is continuous relaxation method (Liu et al., 2018c; Cai et al., 2018), 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 (Adam & Lorraine, 2019), 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 (Brock et al., 2017b; Guo et al., 2019; Bender et al., 2018; Chu et al., 2019), 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 (Adam & Lorraine, 2019; Chu et al., 2019; Bender et al., 2018), 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 (Chipman et al., 2001). 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 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. To update the partial model pool, we first generate candidates by extending each partial model and evaluate their potentials, keeping the best performancing ones. The search space is effectively shrunk in the upcoming training interval. Consequently, the parameter posterior get close 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 approaches from a Bayesian point of view and identify the associated disadvantage which we call Posterior Fading.
- Guided by the theoretical result, we introduce a novel NAS algorithm fundamentally different from existing one-shot methods, which guides the proxy distribution to converge towards the true parameter posterior.

To examine the effectiveness of our newly proposed method, we benchmark its performance on ImageNet (Russakovsky et al., 2015) against the existing methods. In one typical search space (Cai et al., 2018), our PC-NAS-S attains $76.8\%$ top-1 accuracy, $0.5\%$ higher and $20\%$ faster than EfficientNet-B0 (Tan & Le, 2019a), which is the previous 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) (Miller et al., 1989; Liu et al., 2018b; Real et al., 2017; Zoph & Le, 2016; Baker et al., 2017a; Wang et al., 2019) methods normally involves with 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 (Pham et al., 2018; Brock et al., 2017a; Liu et al., 2018c; Cai et al., 2018) 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 (Pham et al., 2018) 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 (Brock et al., 2017b) trains the over-parameterized network along with droping out each operator with increasing probability. Then it use the pre-trained over-parameterized network to evaluate randomly sampled architectures. DARTS (Liu et al., 2018c) additionally introduces a real-valued architecture parameter for each operator and alternately train operator weights and architecture parameters by back-propagation. ProxylessNAS (Cai et al., 2018) binarize the real-value parameters in DARTS to save the GPU cumputation and memory for training the over-parameterized network.

The paradigm of ProxylessNAS (Cai et al., 2018) and DARTS (Liu et al., 2018c) 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.

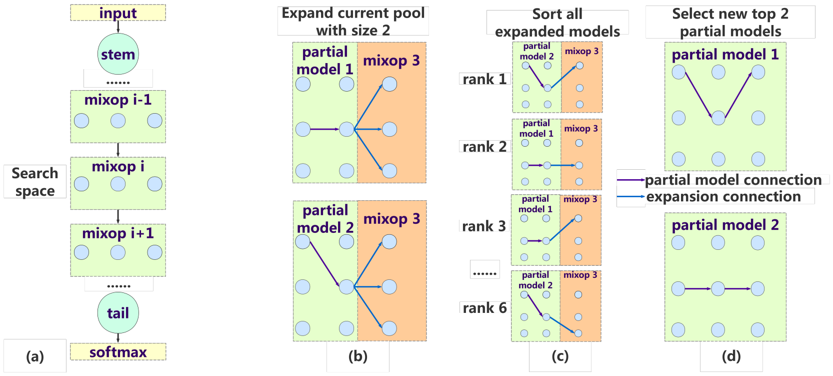


Figure 1: One example of search space(a) and PC-NAS process(b)(c)(d). Each mixed opperator consists of $N$(=3 in this figure) operators. However, only one operator in each mixop is invoked at a time for each batch. In (b), partial models 1 and 2 in the pool consist of choices in mixop 1 and 2. We extend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

Other relevant works are ASAP (Noy et al., 2019) and XNAS (Nayman et al., 2019), 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 and only pruning by evaluating the architecture parameters, 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

Suppose we have $K$ different models $\mathcal{M} = \{\mathbf{m}_1, ..., \mathbf{m}_K\}$ under consideration for 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 $\theta_k$. 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). 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 $p(\theta_k|\mathbf{m}_k)$ is just the posterior $p_{\text{alone}}(\theta_k|\mathbf{m}_k, \mathcal{D})$ which means the distribution of $\theta_k$ 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 $\mathbb{O} = \{O_1 \ldots, O_N\}$, contains $N$ different choices of candidate operators $O_i$ for $i = 1, \ldots N$ 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 $\theta_{lo}$. The parameters of the supergraph are denoted by $\theta$ 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 $\theta$. 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 (Brock et al., 2017b) or sampling different architectures (Bender et al., 2018; Chu et al., 2019) 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 $L_{\text{alone}}$:

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

where $-\log p(\theta|\mathbf{m}_k)$ is 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:

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

This is equivalent to adopting a proxy parameter posterior as follows:

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

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

Maximizing $p_{\text{share}}(\theta|D)$ is equivalent to minimizing $L_{\text{share}}$.

We take one step further to assume that the parameters at each layer are independent, i.e.

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

Due to the independence, we have

$$p_{\text{share}}(\theta|\mathcal{D}) = \prod_k \prod_{l,o} p_{\text{alone}}(\theta_{l,o}|\mathbf{m}_k, \mathcal{D}) = \prod_{l,o} p_{\text{share}}(\theta_{l,o}|\mathcal{D}), \tag{8}$$

where

$$p_{\text{share}}(\theta_{l,o}|\mathcal{D}) = \prod_k p_{\text{alone}}(\theta_{l,o}|\mathbf{m}_k, \mathcal{D}). \tag{9}$$

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

$$\begin{aligned} D_{\mathcal{KL}}\Big(p_{\text{alone}}(\theta_{l,o}|\mathbf{m}_k, \mathcal{D}) \Big|\Big| \, p_{\text{share}}(\theta_{l,o}|\mathcal{D})\Big) &= \int p_{\text{alone}}(\theta_{l,o}|\mathbf{m}_k, \mathcal{D}) \log \frac{p_{\text{alone}}(\theta_{l,o}|\mathbf{m}_k, \mathcal{D})}{p_{\text{share}}(\theta_{l,o}|\mathcal{D})} \mathrm{d}\theta \\ &= \int p_{\text{alone}}(\theta_{l,o}|\mathbf{m}_k, \mathcal{D}) \log \frac{p_{\text{alone}}(\theta_{l,o}|\mathbf{m}_k, \mathcal{D})}{\prod_i p_{\text{alone}}(\theta_{l,o}|\mathbf{m}_i, \mathcal{D})} \mathrm{d}\theta \\ &= \sum_{i \neq k} - \int p_{\text{alone}}(\theta_{l,o}|\mathbf{m}_k, \mathcal{D}) \log p_{\text{alone}}(\theta_{l,o}|\mathbf{m}_i, \mathcal{D}) \mathrm{d}\theta. \end{aligned} \tag{10}$$

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 $p_{\text{share}}$ away from $p_{\text{alone}}$, namely the **Posterior Fading**. We conclude that non-predictive problem originates naturally from one-shot supergraph training, as a typical search space contains a huge $10^{21}$ architectures. Thus if we effectively reduce the number of architectures in (10) during training, the KL divergence would decrease. This is the intuition of our PC-NAS algorithm.

**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
    $\mathbf{m}^* =$ 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 adopt progressive search space shrinking. The resulted 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 $T_i$.

**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.

$$\text{Potential}(o_1, o_2, ..., o_l) = E_{\mathbf{m} \in \{\mathbf{m} | m_i = o_i, \forall i \leq l\}}(Acc(\mathbf{m})). \tag{11}$$

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.

**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 $\theta$ 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 (10) 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, a large scale benchmark dataset, which contains 1,000,000 training samples with 1000 classes. 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 ProxylessNAS (Cai et al., 2018) and FBNet (Wu et al., 2018) 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 4.4. 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 and MixNet, We add squeeze-and-excitation (SE) layers (Hu et al., 2018) to our model at the end of each operator. Results with and without SE layers are listed separately in the Table 1. Since the authors didn't list no SE versions of EfficientNet or MixNet in their original papers, we train them as well as PC-NAS (w/o SE) with exactly the same script and the results are in Table 4.

### 4.2 ImageNet Results

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 (Sandler et al., 2018) | - | 6.9M | 10ms | 74.7% | - |
| AmoebaNet-A (Real et al., 2018) | - | 5.1M | 23ms | 74.5% | - |
| PNASNet (Liu et al., 2018a) | 5.6x$10^{14}$ | 5.1M | 25ms | 74.2% | - |
| FBNet-C (Wu et al., 2018) | $10^{21}$ | 5.5M | - | 74.9% | - |
| MnasNet (Tan et al., 2018) | - | 4.4M | 11ms | 74.8% | 76.1% |
| ProxylessNAS-gpu (Cai et al., 2018) | $7^{21}$ | 7.1M | 8ms | 75.1% | - |
| MixNet-S (Tan & Le, 2019b) | - | 4.1M | 13 ms | - | 75.8% |
| EfficientNet-B0 (Tan & Le, 2019a) | - | 5.3M | 13 ms | - | 76.3% |
| PC-NAS-S | $10^{21}$ | 5.1M | 10 ms | 76.1% | 76.8% |
| PC-NAS-L | $20^{21}$ | 15.3M | 11 ms | 77.5% | 78.1% |

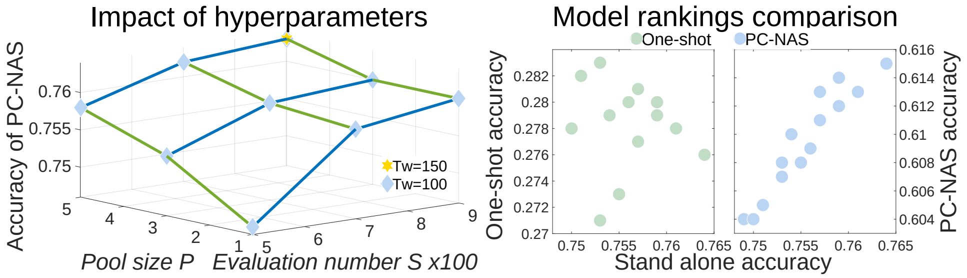
Table 1 shows the performance of our model on ImageNet. We set our target latency at $10ms$ according to our measurement of mobile setting models on GPU. Our search result in the small space, namely PC-NAS-S, achieves 76.8% top-1 accuracy under our latency constraint, which is $0.5\%$ higher than EffcientNet-B0 (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 EffcientNet-b0 and MixNet-S.

Table 2: Performance Comparison on COCO and Market-1501

| **backbone** | **params** | **latency** | **COCO mAP** | **Market-1501 mAP** |
|---|---|---|---|---|
| MobileNetV2 | 3.5M | 7ms | 31.7 | 76.8 |
| ResNet50 | 25.5M | 15ms | 36.8 | 80.9 |
| ResNet101 | 44.4M | 26ms | 39.4 | 82.1 |
| PC-NAS-L | 15.3M | 11ms | 38.5 | 81.0 |

### 4.3 Transferability of PC-NAS

We validate our PC-NAS's transferability on two tasks, object detection and person re-identification. We use COCO (Lin et al., 2014) dataset as benchmark for object detection and Market-1501 (Zheng et al., 2015) for person re-identification. For the two dataset, PC-NAS-L pretrained on ImageNet is utilized as feature extractor, and is compared with other models under the same training script. For object detection, the experiment is conducted with the two-stage framework FPN (Lin et al., 2017). Table 2 shows the performance of our PC-NAS model on COCO and Market-1501. For COCO, our approach significantly surpasses the mAP of MobileNetV2 as well as ResNet50. Compare to the standard ResNet101 backbone, our model achieves comparable mAP quality with almost $1/3$ parameters and $2.3\times$ faster speed. Similar phenomena are found on Market-1501.



(a)

(b)

Figure 2: (a): Influence of warm-up epochs $T_w$, partial model pool size $P$, and evaluation number $S$ to the resulted model. (b): Comparison of model rankings for One-Shot (left) and PC-NAS (right).

### 4.4 Ablation Study

**Impact of hyperparameters:** In this section, we further study 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 resulted models of these two settings show no significant difference in top-1 accuracy (less than $0.1\%$), shown as in Fig. 2a. 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. 2a. 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 improvement when further increasing these two hyperparameters.

**Effectiveness of shrinking search space:** To assess the role of space shrinking, we train the supergraph of our large space using One-Shot (Brock et al., 2017b) 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 resulted model using this setting attains $77.1\%$ top-1 accuracy on ImageNet, which is $1\%$ lower than our PC-NAS-L. We further double the value of $P$ or $S$, and the results are listed in Table 3. From these results, we could conclude that our search method alone already lead to impressive performance. And our space shrinking training strategy improves model ranking and brings extra $0.7\%$-$1\%$ accuracy boost.

We add another comparison as follows. First, we select a batch of models from the candidates of our final pool under small space 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 shows the model rankings of PC-NAS and One-Shot using the accuracy obtained from inferring the models in the supergraphs trained with the two methods. The difference is shown in Fig. 2b, the pearson correlation 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.

**Effectiveness of our search method:** To investigate the importance of our search method, we utilize Evolution Algorithm (EA) to search for models with the above supergraph of our large space trained with One-Shot. The top-1 accuracy of discovered model drops furthur to $75.9\%$ accuracy, which is $2.2\%$ lower than PC-NAS-L . We implement EA with population size 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 ours. The results are also shown in Table 3.

## 5 Conclusion

In this paper, a new architecture search approach called PC-NAS is proposed. We 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, a practical 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 | 75.9% |

Table 4: Accuracy comparison trained with exactly the same script

| models | top-1 acc |
|---|---|
| EfficientNet-B0 (w/o SE) | 74.1% |
| MixNet-S (w/o SE) | 74.6% |
| PC-NAS-S (w/o SE) | 76.1% |
| PC-NAS-L (w/o SE) | 77.5% |

is devised to mitigate the issue. Experimental results demonstrate the effectiveness of our method, which achieves state-of-the-art performance on ImageNet, and transfers well to COCO detection and person re-identification too.

## References

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

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

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

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

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

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

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*, pp. 65–116, 2001.

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

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

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.

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

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.

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

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*, pp. 740–755. Springer, 2014.

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*, pp. 2117–2125, 2017.

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)*, pp. 19–34, 2018a.

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

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

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

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.

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.

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.

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

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

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*, pp. 115(3):211–252, 2015.

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*, pp. 4510–4520, 2018.

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

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

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.

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.

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.

Liang Zheng, Liyue Shen, Lu Tian, Shengjin Wang, Jingdong Wang, and Qi Tian. Scalable person re-identification: A benchmark. In *Proceedings of the IEEE international conference on computer vision*, pp. 1116–1124, 2015.

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*, pp. 2423–2432, 2018.

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

# 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 (Sandler et al., 2018)) 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 5, 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 lised in the right column of Table 5, where notation OP_X_Y represents the specific operator OP with expand ratio X and K=Y.

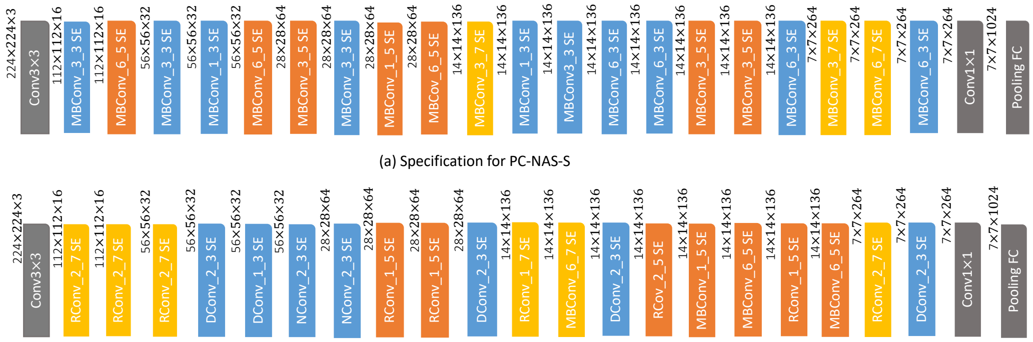
There are altogether 21 mixoperators in both small and large search spaces. Thus our small search space contains $10^{21}$ models, while the large one contains $20^{21}$.

Table 5: operator table

| Operators in both large and small space | | Operators 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 resulted models:

The specifications of PC-NAS-S and PC-NAS-L are shown in Fig. 3. 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.



(a) Specification for PC-NAS-S

(b) Specification for PC-NAS-L

Figure 3: The architectures of PC-NAS-S and PC-NAS-L.