DetOFA: Efficient Training of Once-for-All Networks for Object Detection using Path Filter

Yuiko Sakuma *   Masato Ishii   Takuya Narihira

Sony Group Corporation, Tokyo, Japan

{Yuiko.Sakuma, Masato.A.Ishii, Takuya.Narihira}@sony.com

Abstract

We address the challenge of training a large supernet for the object detection task, using a relatively small amount of training data. Specifically, we propose an efficient supernet-based neural architecture search (NAS) method that uses search space pruning. The search space defined by the supernet is pruned by removing candidate models that are predicted to perform poorly. To effectively remove the candidates over a wide range of resource constraints, we particularly design a performance predictor for supernet, called path filter, which is conditioned by resource constraints and can accurately predict the relative performance of the models that satisfy similar resource constraints. Hence, supernet training is more focused on the best-performing candidates. Our path filter handles prediction for paths with different resource budgets. Compared to once-for-all, our proposed method reduces the computational cost of the optimal network architecture by 30% and 63%, while yielding better accuracy-floating point operations Pareto front (0.85 and 0.45 points of improvement on average precision for Pascal VOC and COCO, respectively).

1. Introduction

Object detection is one of the fundamental computer vision tasks, which has been widely used in real-life applications, such as on smartphones, for surveillance, and autonomous driving. In the past decade, deep neural network-based methods have become the state-of-the-art method, yielding very high performance [25, 19]. The recently proposed CenterNet [40] is a simple anchor-free, one-stage method that has high computational efficiency.

As the efficiency requirements for detection models get more pronounced for various applications [4, 16], the challenge of efficient deployment arises. In particular, models often are deployed to a diverse set of hardware platforms [2] and, hence, have to meet different resource constraints. For
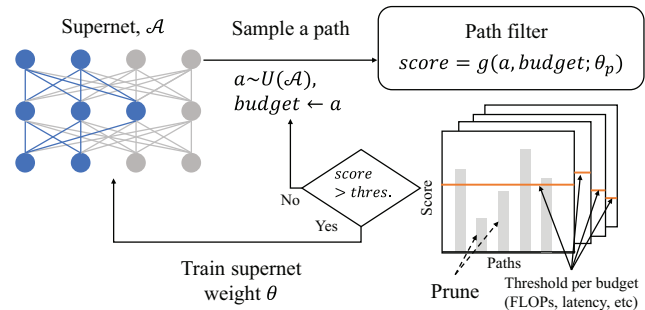
---

*Corresponding author

Figure 1. Our proposed search space pruning method using a path filter. A path is sampled from the supernet, $\mathcal{A}$. The path filter predicts the path score, the relative performance of a path within the budget, from the architecture and budget information. The weak-performing (e.g., low accuracy) paths are evenly pruned from the search space considering the budget. Thus, the supernet training is more focused on good-performing paths.

instance, the latest smartphones and edge devices such as surveillance cameras have different processing speeds and memory capacities. Designing performant model architectures that meet resource constraints across different devices is a laborious task and requires high computation costs. Recently proposed supernet-based neural architecture search (NAS) methods [3, 38, 5, 13, 34] decouple the model training and search and achieve remarkable search efficiency and final model performance. Specifically, supernet-based NAS methods are efficient because the supernet is trained only once and can be scaled to fit any computational budget without any retraining, later. Up to now, supernet-based NAS has mostly been used to optimize network architectures for classification tasks, where large training sets are available, and has only recently been adopted for some object detection tasks [2].

One of the major challenges in supernet-based NAS for object detection is how to efficiently train the supernet with limited training data [11]. For instance, while commonly used ILSVRC2012 (ImageNet) [8] contains 1.2 million images with 1000 classes, Pascal visual object classes (Pascal

VOC) [9] contains 16,000 images with 20 classes, which is only 1% in data size of ImageNet. On the other hand, the search space of supernet-based NAS is large; for example, the search space of Once-for-All (OFA) [3], one of the most referenced supernet-based NAS, contains 10^{19} architectures. Supernet training requires training multiple sub-networks (e.g., paths) in the search space simultaneously. When a small dataset is available, it is prone to under-fit, which leads to a noticeable degradation in the model performance.

We propose a new training method for supernet-based NAS, which is suitable for tasks where only a small amount of training data is available (Figure 1). Although we introduce our method in the example of object detection, it is general and can be used to train supernets for any task. Our efficient NAS method uses search space pruning via a path filter. We propose to prune the supernet after pre-training to effectively reduce the number of candidate paths in the search space. Specifically, we use a path filter to discard weak-performing operations and paths and focus more on good ones. Here, paths are the candidate architectures, which are often called sub-networks contained in the supernet. Operations are the layer configurations of a path. The path filter is trained with a ranking loss, using path configuration/performance pairs, that are collected on the validation data. Further, the path filter is pre-trained by sampling paths from the pre-trained supernet and transferred to the target task as well. Because our method effectively prunes the search space, the supernet can be trained well with relatively small object detection data.

To our knowledge, our work is the first to propose a path filter that is specifically designed for supernet, which is conditioned by constraints and can be used to evenly prune the search space along the resource constraint for training good-performing paths in a one-shot manner. Moreover, the supernet architecture for object detection has not been explored well. Our target includes low-memory edge devices with small FLOPs constraints, thus, we chose CenterNet [40] because of (1) its widespread usage in real applications and (2) its efficiency in FLOPs.

In experiments, we demonstrate that our method yields better-performing supernets for object detection tasks. Our method achieves 30% and 63% reduction in computation cost while 0.85 and 0.45 points improvements in the average precision (AP) over different floating-point operations (FLOPs) constraints for Pascal VOC and Microsoft COCO (COCO) [20], respectively, compared to OFA.

2. Related works

2.1. Object detection

Popular object detection methods are discussed in the supplementary material A. To summarize, among two-stage

detectors [10, 25] and one-stage detectors [24, 19, 1], CenterNet [40] and FCOS [31] are simple anchor-free, one-stage methods. CenterNet models an object as the center point of its bounding box. First, it finds center points, while other object properties like size and orientation are regressed after. The model comprises a convolution neural network (CNN) backbone, up-convolutional layers, and three heads. Each head predicts a keypoint heatmap and a set of local offsets and object sizes. CenterNet is a general method that can be used for other vision tasks such as human pose estimation and 3D object detection, as well.

2.2. NAS

Recently, NAS has been studied to automatically design good-performing neural network architectures under different resource budgets. NAS methods, including evolutionary search [6, 7], reinforcement learning [30], or one-shot method [21], identify the optimal network architecture for a given resource budget from a set of candidate architectures (the search space). This typically involves training and search steps. Recently, supernet-based methods have been proposed [3, 38, 5, 13, 34], which decouple the model training and search. The search space of supernet-based NAS contains multiple paths. OFA [3] is a memory-efficient supernet-based NAS that shares the supernet weights. For training the large supernet with the many shared weights, OFA adapts the progressive shrinking strategy, which gradually trains the large paths to small ones.

NAS for object detection: DetNAS [4], SP-NAS [16], and DetNAS [4] aim to optimize the backbone architecture. However, they only yield one single optimal architecture that meets a single hardware constraint. A problem is, that because the NAS search space contains many weights and because object detection datasets are small, the supernet easily overfits. Transfer learning is a common approach to address the limitation of a small target dataset. Object detection models are for example often pre-trained with classification datasets, i.e., DetNAS uses the backbone initialized with ImageNet. GAIA [2] pre-trains the supernet using a huge data pool with a unified label space from multiple data sources. Then, the downstream fine-tuning is performed on the target dataset. Although GAIA provides a powerful pre-training model, the data unification process requires massive labeling efforts. While GAIA emphasizes the effectiveness of the pre-training dataset, we study the effectiveness of different pre-training methods.

2.3. NAS with search space pruning

2.3.1 Search space pruning for single constraint NAS

Search space pruning techniques are categorized into path [37, 29] and operation levels [14, 27]. GreedyNAS [37] produces a path candidate pool to store good paths and samples

from it under an exploration-exploitation strategy. MCT-NAS [29] proposes a sampling strategy based on the Monte Carlo tree search. ABS [14] proposes an angle-based metric that measures the similarity between initialized and trained weights to predict the generalization ability of paths. BS-NAS [27] proposes a channel-level importance metric.

While operation-level pruning is more efficient because several paths are pruned by pruning one operation, selecting operations to be pruned is not trivial, i.e., evaluating the impact of pruning specific operations on the paths' performance or efficiency is not an easy task. However, it is more straightforward for paths such that validation loss or accuracy can be easily estimated. GreedyNASv2 [15] considers both path and operation level pruning. A path filter, trained to predict weak paths, is used to prune them from the search space. They argue that identifying weak paths is more reliable than identifying good ones. The search space is further pruned by removing operations; the operations with similar path filter embedding are merged into the ones with smaller FLOPs. Similar to GreedyNASv2, our method uses a path filter to identify the paths and operations to be pruned. However, our goal is to train a supernet from which we can extract paths that meet different resource constraints. In comparison, GreedyNASv2 only yields a single network architecture that is optimal for a given resource constraint. In particular, GreedyNASv2 must be run from scratch if the resource constraints change. Moreover, while GreedyNASv2 uses a path filter that predicts the binary label (that is, weak path or not) of each path, our path filter learns a path ranking. Thus, our path filter is more flexible, i.e., once our path filter is trained, different pruning ratios can be applied. In comparison, the path filter proposed for GreedyNASv2 needs to be retrained for each pruning ratio.

2.3.2 Search space pruning for supernet-based NAS

While NAS typically employs uniform sampling for the supernet training [3], iNAS [12] proposes a sampling method with latency grouping, which groups the operations with the same latency across layers, to train a supernet for salient object detection. Although this method limits the operation combinations and, hence prunes the search space, using the latency as a pruning criterion might not be optimal. For instance, the candidate paths do not include the ones with low latency operations in the input side layers and high latency operations in the output side. AttentiveNAS [33] focuses on the best and worst performing paths for supernet training. However, they only focus on path-level pruning. CompOFA [26] prunes the search space by coupling the operations (Table 1). It halves the computation time without performance degradation compared to OFA's progressive shrinking. However, their method is heuristic because the operation combination is determined without considera-

tion of the path performance. Ours is more optimal and general because the operation candidates are determined based on the path score (that is, performance) and FLOPs. Further, no prior works in this category consider joint path and operation-level pruning.

3. Proposed method

As discussed in Section 1, we propose an efficient supernet-based NAS method that uses search space pruning via a path filter, and apply it to object detection. Let $f(x, a; \theta)$ be a supernet with input x, path a, and weights θ that is initialized by a pre-training task and y be the output. For the CenterNet, the output y is the key-point heatmap, local offset, and object size. We propose to prune the search space while fine-tuning. A path filter $g(a; \theta_p^*)$ that is trained to rank paths in the search space is used to prune the search space. The output of the path filter is the path score. We prune the operations and paths that are predicted to perform badly. Specifically, the supernet is fine-tuned for the target task by computing the optimized shared weights θ^*, according to;

$$
\min_{\theta} \mathbb{E}_{a \sim U(\tilde{A})} \left[\frac{1}{N} \sum_{i}^{N} \mathcal{L}(f(x_i, a; \theta), y_i) \right] \tag{1}
$$
$$
(x_i, y_i) \in \mathcal{D}_{\mathrm{trn}}
$$

where $\tilde{A}$ ($|\tilde{A}| \ll |A|$) and $\mathcal{D}_{\mathrm{trn}}$ are the search space pruned from the original one $\mathcal{A}$ and training dataset, respectively.

After training the supernet, the optimal path a^* is searched that yields the best path score on the target task for a given FLOPs budget as follows:

$$
a^* = \arg\max_{a \in \tilde{A}} g(a; \theta_p^*) \tag{2}
$$
$$
s.t. \, \mathrm{FLOPs}(a) \leq \tau
$$

where $\mathrm{FLOPs}(a)$, and τ are the FLOPs of path a, and FLOPs budget, respectively. Although FLOPs are used in the experiment, other resource constraints such as latency and memory size can also be used. The details of path filter and search space pruning are provided in Section 3.1 and 3.2, respectively. The details of resource constraint search are provided in Section 3.3.

3.1. FLOPs-bounded path filter

Given the architecture configuration of a path, the path filter predicts the relative performance of the path among all path candidates in the search space. It is used to specify weak-performing paths and operations to be pruned from the search space. Details for path filter design and training are provided below.
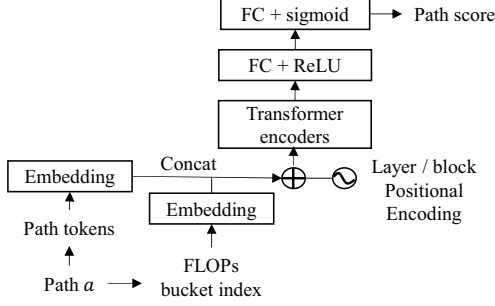
Figure 2. The architecture of the path filter. We use a transformer encoder to predict performance scores from tokenized paths.
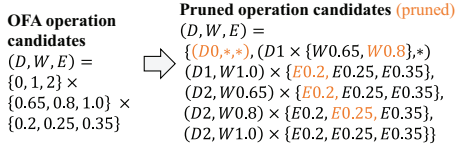

Figure 3. Example of operation pruning. D, W, and E denote the bottleneck depth, channel width expand ratio, and channel expand ratios. The operation candidates are pruned according to the predicted path scores and FLOPs, thus reducing the possible combinations of them.

3.1.1 Path filter design

Figure 2 shows the architecture of our path filter. A transformer encoder architecture [32] is chosen because it yields state-of-the-art performance for many tasks and a high computation efficiency due to the parallelization ability. The architecture configuration and the associated number of FLOPs are represented by path tokens and a FLOPs bucket index, respectively. First, both are converted into feature vectors via a learned embedding and then concatenated. The embeddings are enriched with positional encodings of the architecture configuration and fed to the transformer encoder. The final path score is computed, using two fully connected (FC) layers with sigmoid activation.

Tokenization: The architecture configuration of a path $a \sim U(\mathcal{A})$ is encoded by using layer-wise tokenization with positional encoding. Here, we use OFA ResNet50 [22] as an example to explain how tokenization works. The configuration of a path in OFA ResNet50 has a fixed layer connectivity and is completely defined by the depth (D) of a block, the channel width (W), and the channel expansion ratio (E). Here, W and E define the channel size of a whole block and within a bottleneck in a block, respectively. The OFA network consists of bottleneck blocks with several layers; operation configurations are either block-wise (that is, D and W) or layer-wise (that is, E). We propose to tokenize on a per-layer basis; the block-wise configurations are simply repeated for each block. Each layer can be represented by a

token consisting of W, E, and skip connection (SC) derived from D. Here, an example of tokenization for a bottleneck block with a maximum depth of four (that is, four layers) is presented. When $(\text{D,W,E}) = (0, 1.0, (0.25, 0.35))$, the token would be "W1.0_E0.25 W1.0_E0.35 SC SC" where SC denotes the skip connection, such that the bottleneck with $D = 0$ has only two layers.

As detailed in Section 3.1.2, since the path score is only used to compare paths within the same bucket, FLOPs information is added as the input. The FLOPs bucket is determined from the minimum and maximum FLOPs among all paths in $\tilde{\mathcal{A}}$ by binning values into discrete intervals of equal widths. In the experiment, we set the number of FLOPs buckets as five. The required number of FLOPs to compute the output of a path is represented by the corresponding bucket index. This tokenization scheme can be generalized to other OFA supernets and search spaces. However, the positional encoding is not as simple in the case of more complex layer connectivity.

Layer and block positional encoding: Since the self-attention mechanism used in transformers is agnostic to the position of the data in a sequence, the layerwise encodings are enriched with positional encodings. More specifically, we use standard cosine/sine-based positional encodings that are defined as

$$P(l, 2i) = \sin\left(l \big/ 10000^{2i/\text{max}\_l}\right)$$
$$P(l, 2i + 1) = \cos\left(l \big/ 10000^{2i/\text{max}\_l}\right) \tag{3}$$

where l and i denote the layer index and embedding dimension, respectively. max_l is the maximum layer index in $\tilde{\mathcal{A}}$. Further, the block index of the bottleneck is encoded. As discussed above, for OFA supernets, some operations are block-wise, while others are layer-wise. Thus, block-level encoding adds extra information to the path filter. The formulation is similar to the layer-wise positional encoding (Equation 3) but replaces l by the block index.

3.1.2 Path filter training

We aim to predict weak-performing (e.g., low accuracy) paths and operations for any pruning ratio with good precision. Our path filter learns a full ranking of the path performance and can be used for different pruning ratios. while GreedyNASv2 can only be used for a fixed pruning ratio. Specifically, the output of our path filter is a path score that indicates the relative performance of the path; higher scores mean lower validation loss. Once the path ranking is learned, it can be used to prune weak-performing paths from the supernet, using any pruning ratio. The learning-to-rank problem is equivalent to the area under the ROC curve (AUC) maximization [39, 36]. The pairwise surrogate loss is often used for AUC maximization. Further, we propose
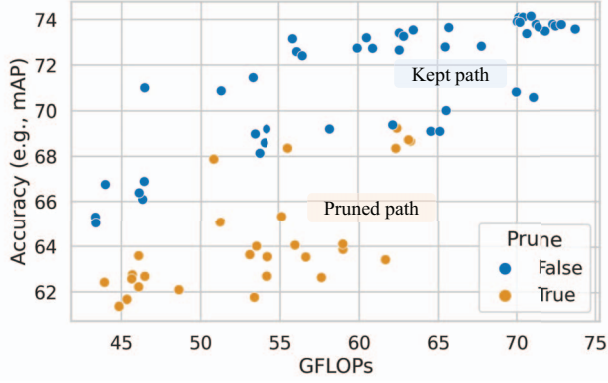
Figure 4. Path pruning via path filter. The weak-performing paths are pruned during supernet training.

the **FLOPs-bounded loss**. Since for supernet-based NAS, the ranking among paths with similar FLOPs matters, the loss is calculated only for path pairs that fall into the same FLOPs bucket, i.e.

$$g_{aa'} := g(a; \theta_p^*) - g(a'; \theta_p^*)$$

$$\theta_p^* = \min_{\theta_p} \mathbb{E}_{a,a' \sim U(\mathcal{A})} \left[\mathcal{L}_p(g_{aa'}, t_{aa'}) \right]$$

$$t_{aa'} = \begin{cases} 1, \text{if } \frac{1}{N_p} \sum_i^N \mathcal{L}(f(x_i, a; \theta^*), y_i) \leq \\ \qquad \frac{1}{N_p} \sum_i^N \mathcal{L}(f(x_i, a'; \theta^*), y_i) \\ \qquad \text{and } \text{FLOPs}(a') \leq \tau_{\text{FLOPs}(a)} \\ 0, \text{otherwise} \end{cases} \quad (4)$$

$$(x_i, y_i) \in \mathcal{D}_{\text{val}}$$

The squared hinge loss

$$\mathcal{L}_p(g_{aa'}, t_{aa'}) = max(0, 1 - t_{aa'} \cdot g_{aa'})^2 \quad (5)$$

is chosen, because it is one of the most commonly used pairwise ranking losses [17, 35]. For training the path filter, m paths are sampled from the supernet for each FLOPs bucket in order to construct a training dataset. Then, the dataset is split into 80% training and 20% validation samples. The path filter with the best validation loss is finally used for search space pruning. The path filter is pre-trained as well. After transferring the path filter to the target task, it is fine-tuned for some warm-up epochs.

3.2. Search space pruning via path filter

First, operations are pruned by using the path score. Then, during supernet fine-tuning, paths are pruned.

Operation pruning (Figure 3): To evaluate the operation candidates by using the path filter that is trained to evaluate each path, n paths are uniformly sampled. Then, the candidate operation is inserted in order to measure its performance for a given combination of D, W, and E. The

Algorithm 1 Supernet training with search space pruning using a path filter

---

Input: Pre-trained supernet weights θ, pre-trained path filter $g(a; \theta_p)$, training dataset $\mathcal{D}_{\text{trn}}$, search space $\mathcal{A}$, operation pruning ratio r_{op}, path pruning ratio r_{path}, Score threshold for path a $\delta_{\text{FLOPs}(a)}$,

1: Initialize $\tilde{\mathcal{A}}$ with $\mathcal{A}$
2: Supernet warm-up training
3: Uniformly sample m paths for each FLOPs bucket and validate
4: Fine-tune path filter to $g(a; \theta_p^*)$
5: Prune $r_{\text{op}}\%$ operations from $\tilde{\mathcal{A}}$ via path filter
6: Calculate path score threshold $\delta_{\text{FLOPs}(a)}$ with pruning ratio $r_{\text{path}}\%$ for each FLOPs bucket
7: **for** $epoch = 1, ..., \text{max\_epoch}$ **do**
8:    **for** $i = 1, ..., \text{max\_iter}$ **do**
9:       Sample $(x_i, y_i) \in \mathcal{D}_{\text{trn}}$
10:      $a \sim U(\tilde{\mathcal{A}})$ (uniform sampling of a path)
11:      **while** $g(a; \theta_p^*) < \delta_{\text{FLOPs}(a)}$ **do**
12:        $a \sim U(\tilde{\mathcal{A}})$
13:      **end while**
14:      Update θ via gradient descent (Equation 1)
15:    **end for**
16: **end for**

---

average of n path scores is used as the operation score. This gives the score and FLOPs information of candidate operations for all layers as depicted in Figure 3. Three pruning strategies are proposed as follows:

- FLOPs: For each FLOPs bucket, $r_{\text{op}}\%$ operations are pruned with uniform distribution.

- FLOPs & score (per bucket): For each FLOPs bucket, worst $r_{\text{op}}\%$ operations are pruned.

- FLOPS & score (all): For all FLOPs buckets, worst $r_{\text{op1}}\%$ operations are pruned. Then, for each FLOPs bucket, worst r_{op2} operations are pruned.

Path pruning: During supernet fine-tuning, r_{path} weak performing paths are pruned for each FLOPs bucket. Specifically, sampling of pruned paths is skipped and the supernet is only trained for the kept paths. Figure 4 presents the pruned and kept paths, evaluated by path filter when 15 paths are uniformly sampled from the fine-tuned supernet. Algorithm 1 summarizes our proposed supernet training method with the path filter. For path pruning, when a path a is sampled (Line 10), it is evaluated by the path filter. If it is larger than the threshold, the supernet is trained for a, and if not, a new path is sampled (Line 12).

Method	Search space	Size
OFA	$(D,W,E) = \{\{0, 1, 2\} \times \{0.65, 0.8, 1.0\} \times \{0.2, 0.25, 0.35\}\}$	$((3^2 + 3^2) \times (3^3 + 3^3) \times (3^4 + 3^4))^4$ $\approx 10^{20}$
CompOFA*	$(D,W,E) = \{(0, 0.65), (1, 0.8), (2, 1.0)\} \times \{0.2, 0.25, 0.35\}$	$(3^2 + 3^3 + 3^4) \approx 10^8$

Table 1. The search spaces used in our experiments. D, W, and E denote the bottleneck depth, channel width expand ratio, and channel expand ratios.
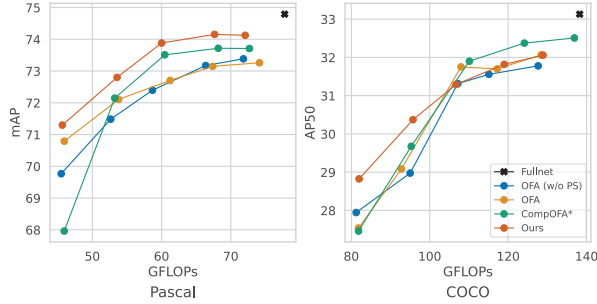

Figure 5. The performance of the optimal architecture a^*, for object detection on the Pascal VOC (left) and the COCO (right) dataset. Our method outperforms OFA (w/o progressive shrinking, PS) across all given FLOP bounds. CompOFA* performs well only for larger FLOPs. Fullnet denotes the largest path after fullnet training (Line1 in Algorithm 1).

3.3. Resource-constraint search

Similar to OFA [3], we adopt the evolutionary search [23] algorithm to search architectures for the given resource constraint. Because evaluating the actual performance of each path during the search phase is expensive, the path score is used as the proxy as done by OFA [3]. OFA for example uses an accuracy predictor that regresses the exact validation accuracy. The path filter can also be used for standard supernet-based NAS, and not only for our proposed method. We used the path filter as the accuracy predictor in the search phase. Note, that before the search the path filter is re-trained. Similar to the supernet warm-up (Line 2 in Algorithm 1), m paths are uniformly sampled for each FLOPs bucket to obtain training data.

4. Experiments

4.1. Experimental setup

The experiments are implemented, using the NNablaNAS framework [28]. Experiments on Pascal VOC and COCO were performed on NVIDIA V100 and A100 GPUs, respectively. We use the same search space as [22], which yields the OFA-ResNet50 supernet summarized in Table 1.

Path filter: The path filter uses an embedding layer with 128 dimensions, three transformer encoders (with four-head
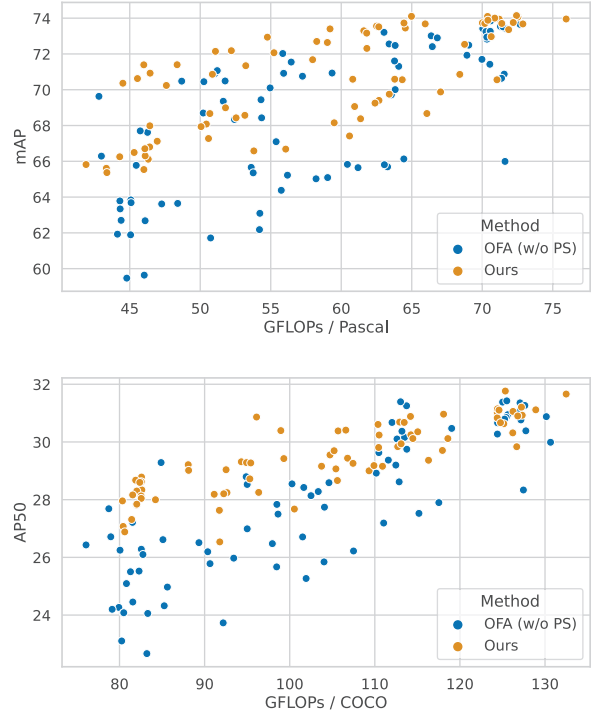

Figure 6. The validation performance of 15 uniformly sampled paths across 5 FLOPs buckets on the Pascal VOC (top) and the COCO (bottom) dataset. Our proposed method yields paths with a better average performance.

self-attention and an FC layer with 128 dimensions), and two final FC layers with 128 dimensions. The path filter was pre-trained with path configuration/performance pairs obtained by sampling 10000 paths in total from the supernet trained on ImageNet. Path filter fine-tuning was conducted by sampling 500 paths after fine-tuning the supernet on the detection dataset.

Transfer learning details: The CenterNet backbone is transferred from the supernet pre-trained by the classification task. A max-pooling layer and classification head are replaced by task-specific layers, which are up-convolution layers and detection heads.

Supernet training and search details: The supernet was trained for 5 and 10 warm-up epochs for Pascal VOC and COCO, respectively. Then, it was fine-tuned for 65 and

60 epochs, respectively. Training details are described in the supplementary materials B. The number of generations in the evolutionary search was 500.

4.2. Comparison with prior NAS approaches

The proposed method is compared to the prior supernet-based NAS methods, OFA and CompOFA. Note that the ResNet50-based supernet architecture was originally not proposed for CompOFA. However, to be comparable, we define a similar supernet proposed in the CompOFA paper[26] as given in Table 1 and denote it as CompOFA*. In this experiment, OFA with and without progressive shrinking (denoted as w/ and w/o PS, respectively) were compared. OFA (w/ PS) were trained by ×3 epochs than the proposed method. OFA (w/o PS) and CompOFA were trained for the same epochs (that is, warm-up + fine-tuning epochs) as the proposed method for a fair comparison. The detailed training schedules are presented in supplementary materials B. Here, the operation pruning method is "FLOPS & score (all)", removing 10% of worst performing operations overall and 30% of them per each FLOPs range. The path pruning ratio was 25%.

Figure 5 presents the evolution search results on the average of three runs. Transfer learning improves the accuracy by 6.55 and 1.85 points on average for Pascal VOC and COCO, respectively. Our method achieves better performance over OFA (w/ and w/o PS) over the different FLOPs. For instance, compared to OFA (w/o PS), the proposed method outperforms by 0.85 and 0.45 points on average for Pascal VOC and COCO, respectively. Specifically, our method performs better than OFA (w/ and w/o PS). CompOFA* performs well for large FLOPs, but weaker for smaller paths. This result presents the limitation of handcrafted search space pruning. Paths with larger depths and smaller channels on the output side performed better for small paths. This corresponds to the intuition that detection models have to keep rich feature information in the backbone to be processed in up-convolutional and head layers. CompOFA* removes these paths by coupling small depths and channels together. This result suggests that handcrafted search space in CompOFA is well-tuned only for a specific task and a search space, while our method is more general and potentially works well for arbitrary search spaces and tasks.

Table 3 presents the computation costs of our method compared to prior methods. The extra computation cost of our proposed method to OFA is path sampling and path filter fine-tuning after supernet warm-up (lines 4 and 5 in Algorithm 1) and path pruning (lines 12 and 13 in Algorithm 1). The computation cost for path filter fine-tuning was negligible (less than 2 min on one RTX A4000 GPU) compared to other processes. Also, the path pruning cost did not affect the training cost for COCO. It was about a +10% increase
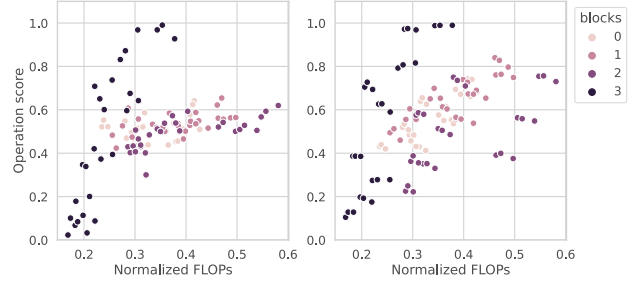

Figure 7. Operation scores for the supernet trained by (left) ImageNet and (right) Pascal VOC. For pascal VOC, the score is more affected by operation configurations such as block id and depth.

for Pascal VOC. The proposed method outperforms OFA (w/ PS) although the computation cost is reduced by approximately 30% and 60% for Pascal VOC and COCO, respectively because our method efficiently trains the supernet without progressive shrinking by pruning the search space. This suggests that our method effectively prunes the search space to be trained with small detection datasets. Furthermore, we confirmed that when OFA and CompOFA* supernets are trained for the same GPU costs as ours, our method still outperforms (detailed in the supplementary materials, Section D). Compared to the fullnet performance, the best-performing paths found by the evolutionary search were approximately 0.50 and 1.00 points worse for Pascal VOC and COCO, respectively; the method trains multiple paths well without fine-tuning.

Further, the results of a uniform sampling of 15 paths in each FLOPs bucket are presented in Figure 6. Paths are sampled from $\mathcal{A}$ and $\tilde{\mathcal{A}}$ for OFA (w/o PS) and ours, respectively. For both datasets, the proposed method presents a larger population of paths with better performance for overall FLOPs than OFA (w/o PS). This confirms our method's ability to obtain good-performing paths for multiple resource constraints by search space pruning via the path filter.

4.3. Path filter performance

Table 2 summarizes the path filter performance under different settings explained in Section 3.1.1 for the weakest 25% path prediction. The proposed FLOPs-bounded loss, budget conditioning, and block positional encoding improve the precision by 24% 2%, and 2%, respectively.

4.4. Analysis on path and operation pruning

Figure 7 presents the operation scores calculated by the method explained in Section 3.2 for the supernet trained by ImageNet (after pre-training) and Pascal VOC (after supernet warm-up). Compared to the pre-trained supernet, the fine-tuned supernet presents a stronger correlation between

FLOPs-bounded loss	Budget condition	Block positional encoding	Accuracy	Precision	Recall
−	✓	✓	0.790	0.590	0.174
✓	−	✓	0.900	0.830	0.209
✓	✓	−	0.940	0.850	0.190
✓	✓	✓	0.930	0.870	0.224

Table 2. Evaluation of the path filter for detecting the weakest 25% of the paths. Results with the best precision are presented.

Stage	Pascal VOC	COCO
OFA (w/ PS)	1.150	14.41
OFA (w/o PS)	0.383	4.804
CompOFA*	0.383	4.804
Ours	0.800 (0.414)	5.260 (0.456)

Table 3. Computation cost for sampling paths and supernet training in GPU days. The values in brackets denote computation time for path sampling for path filter fine-tuning.
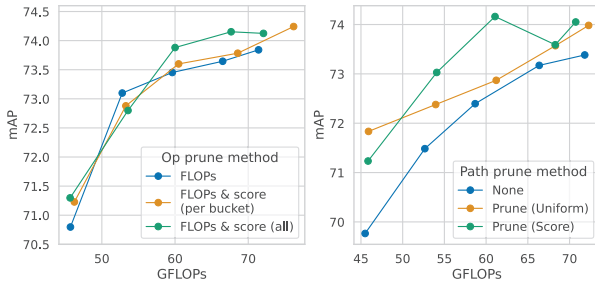

Figure 8. Results for different operation and path pruning methods. For operation pruning (left), FLOPs & score (all), which removes weak performing operations over all layers, performs the best. For path pruning (right), score-based pruning outperforms uniform sampling.
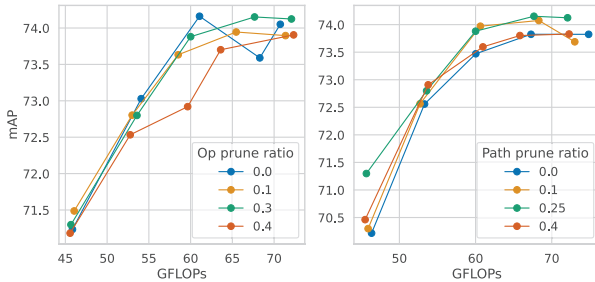

Figure 9. Results for different pruning ratios for (left) operations and (right) paths on Pascal VOC dataset. Without operation nor path pruning ($r_{\text{path}}, r_{\text{op}} = 0$), the performance degrades.

operation scores and configuration such that within each block, the larger the operation size, the larger the operation score. This implies that detection models' performance dif-

fers more depending on its architecture than classification; it motivates our operation pruning strategy of removing weak ones because pruning operations without considering performance might remove good ones.

The effectiveness of score-based search space pruning is analyzed in Figure 8. For both, only operation or path pruning was performed. The score-based pruning outperforms other methods. For operation (left in 8), "FLOPs & score (all)" performs the best. This highlights the effectiveness of removing weak-performing operations. For path (right in 8), the score-based method performs well except for the smallest FLOPs budget. This may be due to the lack of operation pruning; removing weak-performing operations is especially important to train smaller paths. The performances across different pruning ratios are evaluated (Figure 9). Without path pruning (that is, $r_{\text{path}} = 0$) performs worse for all FLOPs-bounds. Without operation pruning performs the worst for the smallest FLOPs. This suggests that operation pruning is more effective for small paths where performance degradation is more obvious without training tricks. Too much pruning (for example, $r = 0.4$) also performs weaker than other ratios; this may be due to removing some good-performing candidate operations or paths.

5. Conclusion

In this paper, we propose a search space pruning technique for training a supernet. The search space is pruned by using a path filter that learns the relative performance of paths. More specifically, based on the given resource constraints and on the predicted path rankings, weak operations and paths are removed from the search space. The experimental results show that our proposed method yields better-performing supernets for object detection. Future works include extending our path filter to more complex search spaces and further optimizing architectures of upconvolutional and head structures in addition to the backbone. Our proposed method could also be extended to other vision tasks such as segmentation tasks and human pose estimation.

Acknowledgement

References

[1] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020. 2

[2] Xingyuan Bu, Junran Peng, Junjie Yan, Tieniu Tan, and Zhaoxiang Zhang. Gaia: A transfer learning system of object detection that fits your needs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 274–283, 2021. 1, 2

[3] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019. 1, 2, 3, 6

[4] Yukang Chen, Tong Yang, Xiangyu Zhang, Gaofeng Meng, Xinyu Xiao, and Jian Sun. Detnas: Backbone search for object detection. *Advances in Neural Information Processing Systems*, 32, 2019. 1, 2

[5] Xiangxiang Chu, Bo Zhang, and Ruijun Xu. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. In *Proceedings of the IEEE/CVF International Conference on computer vision*, pages 12239–12248, 2021. 1, 2

[6] Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Bichen Wu, Zijian He, Zhen Wei, Kan Chen, Yuandong Tian, Matthew Yu, Peter Vajda, et al. Fbnetv3: Joint architecture-recipe search using neural acquisition function. *arXiv preprint arXiv:2006.02049*, 1(2):3, 2020. 2

[7] Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, et al. Chamnet: Towards efficient network design through platform-aware model adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11398–11407, 2019. 2

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 1

[9] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88:303–308, 2009. 2

[10] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015. 2, 11

[11] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Region-based convolutional networks for accurate object detection and segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 38(1):142–158, 2015. 1

[12] Yu-Chao Gu, Shang-Hua Gao, Xu-Sheng Cao, Peng Du, Shao-Ping Lu, and Ming-Ming Cheng. Inas: integral nas for device-aware salient object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4934–4944, 2021. 3

[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. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVI 16*, pages 544–560. Springer, 2020. 1, 2

[14] Yiming Hu, Yuding Liang, Zichao Guo, Ruosi Wan, Xiangyu Zhang, Yichen Wei, Qingyi Gu, and Jian Sun. Angle-based search space shrinking for neural architecture search. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIX 16*, pages 119–134. Springer, 2020. 2, 3

[15] Tao Huang, Shan You, Fei Wang, Chen Qian, Changshui Zhang, Xiaogang Wang, and Chang Xu. Greedynasv2: Greedier search with a greedy path filter. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11902–11911, 2022. 3, 11

[16] Chenhan Jiang, Hang Xu, Wei Zhang, Xiaodan Liang, and Zhenguo Li. Sp-nas: Serial-to-parallel backbone search for object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11863–11872, 2020. 1, 2

[17] Majdi Khalid, Indrakshi Ray, and Hamidreza Chitsaz. Scalable nonlinear auc maximization methods. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2018, Dublin, Ireland, September 10–14, 2018, Proceedings, Part II 18*, pages 292–307. Springer, 2019. 5

[18] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 11

[19] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017. 1, 2, 11

[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 *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014. 2

[21] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 2

[22] Mit-Han-Lab. Mit-han-lab/once-for-all. 4, 6

[23] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019. 6

[24] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. 2, 11

[25] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015. 1, 2, 11

[26] Manas Sahni, Shreya Varshini, Alind Khare, and Alexey Tumanov. CompOFA: Compound once-for-all networks for faster multi-platform deployment. In *Proc. of the 9th Inter-*

national Conference on Learning Representations, 2021. 3, 7

[27] Zan Shen, Jiang Qian, Bojin Zhuang, Shaojun Wang, and Jing Xiao. Bs-nas: Broadening-and-shrinking one-shot nas with searchable numbers of channels. *arXiv preprint arXiv:2003.09821*, 2020. 2, 3

[28] Sony. Sony/nnabla-nas: Neural architecture search for neural network libraries. 6

[29] Xiu Su, Tao Huang, Yanxi Li, Shan You, Fei Wang, Chen Qian, Changshui Zhang, and Chang Xu. Prioritized architecture sampling with monto-carlo tree search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10968–10977, 2021. 2, 3

[30] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2820–2828, 2019. 2

[31] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: Fully convolutional one-stage object detection. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9627–9636, 2019. 2

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 4

[33] Dilin Wang, Meng Li, Chengyue Gong, and Vikas Chandra. Attentivenas: Improving neural architecture search via attentive sampling. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6418–6427, 2021. 3

[34] Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. Hat: Hardware-aware transformers for efficient natural language processing. *arXiv preprint arXiv:2005.14187*, 2020. 1, 2

[35] Lian Yan, Robert H Dodier, Michael Mozer, and Richard H Wolniewicz. Optimizing classifier performance via an approximation to the wilcoxon-mann-whitney statistic. In *Proceedings of the 20th international conference on machine learning (icml-03)*, pages 848–855, 2003. 5

[36] Tianbao Yang and Yiming Ying. Auc maximization in the era of big data and ai: A survey. *ACM Computing Surveys*, 55(8):1–37, 2022. 4

[37] Shan You, Tao Huang, Mingmin Yang, Fei Wang, Chen Qian, and Changshui Zhang. Greedynas: Towards fast one-shot nas with greedy supernet. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1999–2008, 2020. 2

[38] Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. Bignas: Scaling up neural architecture search with big single-stage models. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VII 16*, pages 702–717. Springer, 2020. 1, 2

[39] Zhuoning Yuan, Yan Yan, Milan Sonka, and Tianbao Yang. Large-scale robust deep auc maximization: A new surrogate loss and empirical studies on medical image classification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3040–3049, 2021. 4

[40] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *arXiv preprint arXiv:1904.07850*, 2019. 1, 2