Prioritized Subnet Sampling for Resource-Adaptive Supernet Training

Bohong Chen, Mingbao Lin, Rongrong Ji, Senior Member, IEEE, and Liujuan Cao

Abstract—A resource-adaptive supernet adjusts its subnets for inference to fit the dynamically available resources. In this paper, we propose prioritized subnet sampling to train a resource-adaptive supernet, termed PSS-Net. We maintain multiple subnet pools, each of which stores the information of substantial subnets with similar resource consumption. Considering a resource constraint, subnets conditioned on this resource constraint are sampled from a pre-defined subnet structure space and high-quality ones will be inserted into the corresponding subnet pool. Then, the sampling will gradually be prone to sampling subnets from the subnet pools. Moreover, the one with a better performance metric is assigned with higher priority to train our PSS-Net, if sampling is from a subnet pool. At the end of training, our PSS-Net retains the best subnet in each pool to enable a fast switch of high-quality subnets for inference when the available resources vary. Experiments on ImageNet using MobileNet-V1/V2 and ResNet-50 show that our PSS-Net can well outperform state-of-the-art resource-adaptive supernets. Our project is publicly available at https://github.com/chenbong/PSS-Net.

Index Terms—Prioritized subnet sampling, resource-adaptive, supernet.

I. INTRODUCTION

Deep neural networks (DNNs) have advanced many tasks of artificial intelligence. Traditional DNNs are designed with fixed network structures, where the hardware resources, such as computation ability and memory footprint, have to be allocated in advance to enable the running of DNNs. Unfortunately, the equipped resources are generally great among heterogeneous devices. Even for the same device, the availability of hardware resources varies over time. These limitations barricade DNNs to be deployed. Though many works obtain a compact network via model pruning [1], [2], network architecture search (NAS) [3], [4], etc., the resulting model structures are still fixed, making it hard to make full use of changeable hardware resources.

By adjusting the runtime subnets to fit the dynamically available resources, designing resource-adaptive supernets has attracted increasing attention from the research community. According to the subnet sampling strategy during supernet training, traditional resource-adaptive supernets can be generally divided into two groups including uniform subnet sampling and random subnet sampling.

For uniform subnet sampling, some fixed multipliers are applied to all layers to scale the network width, such as USNet [8], thus providing different subnets at runtime. Two techniques are introduced to train different subnets in the supernet, including “sandwich rule” where three subnets with the maximum, medium, and minimum sizes are sampled to process a batch of images; as well as “inplace distillation” where the largest subnets are supervised by the ground-truth labels and its outputs guide the learning of medium and smallest subnets. Besides uniform width, MutualNet [7] also considers the resolution of input images to enhance subnet performance. It also improves the “inplace distillation” in a style of Kullback-Leibler divergence. As for random subnet sampling, these methods [3], [9], [10], [11] randomly sample subnets from a supernet for training in each iteration, and then search for optimal subnets of different sizes from the pre-trained supernet using heuristic algorithms. The learning is also known as one-shot NAS. However, the performance is far from satisfactory for real-world applications. We attribute it to three open issues, i.e., homogeneity of subnets, search overhead, and subnet mismatch.

In terms of homogeneity of subnets, uniform subnet sampling [5], [7], [8] simply enforces shared multipliers to all layers without investigating the diversity of layer-wise width. Consequently, only one subnet structure can be made given a particular resource constraint such as FLOPs, latency, etc. As previously studied in network pruning [6], [12], [13] and NAS [3], [14], the optimal subnet at a particular size often results from the non-uniform sampling. To verify this, we perform a validation experiment on ImageNet by replacing the uniform width of MobileNet-V1 [5] with a non-uniform form from [6]. Fig. 1(Left) shows a better trade-off between accuracy and FLOPs from the non-uniform strategy. Thus, the homogeneity of subnets, if not well addressed, results in suboptimal performance.
In terms of search overhead, though random subnet sampling may produce different subnet structures over supernet training w.r.t. a particular resource, which alleviates the homogeneity of subnets, additional overheads are required to search for the optimal subnet of a particular size. It takes around 40 GPU hours to find out each target subnet in the trained supernet [3]. Though this additional cost is acceptable in offline environment, it greatly prevents resource-adaptive supernet from real-time running environment where the available hardware resources frequently vary a lot due to the concurrently running applications.

Subnet mismatch refers to that the sampled subnets during training deviate considerably from the target subnets in inference [3], [7], [9]. Yang et al. [7] observed that only around 30% of sampled subnets fall into the trade-off curve of accuracy versus FLOPs. This indicates that amounts of computation is wasted on training these low-quality subnets that would not be considered in inference. Besides, without sufficient training, these high-quality subnets may result in suboptimal performance in the real-world applications as shown in Fig. 1(Right). Thus, it is necessary to gradually reduce the subnet structure space during the network training, so as to concentrate more on these high-quality subnets [15].

Overall, great challenges still remain in training a resource-adaptive supernet. On one hand, we pursue high-performing DNNs for practical deployment. On the other hand, the deployed model should be switchable instantly if the available hardware resources are changed. Reflecting upon the subnet mismatch problem mentioned above, we can know that uniform subnet sampling and random subnet sampling fail to accurately model subnets towards the available resources. A tedious search process is executed to find out the target subnet when the available resources vary.

To solve these problems, we propose prioritized subnet sampling to train a resource-adaptive supernet, termed PSS-Net, as illustrated in Fig. 2. We first build a subnet structure space by applying varying multipliers to different layers, which eliminates the homogeneity of subnets resulting from uniform subnet sampling. Then, we set up a group of subnet pools, each of which stores the information of these high-quality subnets conditioned on the same resource constraint. In the early training batches of PSS-Net, we sample a medium-sized subnet from a pre-built subnet structure space according to the structure distribution [11]. To measure the quality of a sampled subnet, we consider its batch loss as the performance metric, alternative to validation accuracy which requires a computationally heavy subnet training. We update the performance metric using moving average if the same subnet structure is sampled multiple times. For each high-quality subnet, its information is inserted into the corresponding subnet pool. When the subnet pool is full, subnet sampling will gradually shift from the subnet structure space to each subnet pool to ensure that high-quality subnets are entitled with sufficient training, which solves subnet mismatch. Moreover, when sampling is from a subnet pool, the one with a better performance metric than other high-quality subnets is given with higher priority to being sampled to train PSS-Net. At the end of PSS-Net training, we only need to preserve the best subnet in each pool. Thus, PSS-Net results in a set of subnets that can be alternatively deployed to adapt to the currently available resources, without additional overhead for subnet search. PSS-Net introduces the subnet pools to collect high-quality subnets during training, which eliminates the search consumption. This implies the idea of trading space for time.

We use the MobileNet-V1 [16]/V2 [17] and ResNet-50 [18] as our supernets, and the experiments are conducted on ImageNet and COCO. Note that the performance increase on MobileNets is challenging for their light-weight designs. Nevertheless, the results show that our PSS-Net significantly outperforms SOTAs under different constraints.

USNet [8] and MutualNet [7] share similarity with our study in supernet training to obtain subnets of different sizes, and switch these subnets to achieve resource-adaptive inference. The difference includes: 1) USNet and MutualNet apply the same width multiplier to all layers and obtain uniform subnet width. Our PSS-Net adopts layer-wise widths and introduces subnet pooling for high-quality subnet collection. 2) Our PSS-Net
can obtain optimal subnets constrained under different metrics such as FLOPs, Parameters, GPU/CPU latency, etc., which is inapplicable for USNet and MutualNet. To train PSS-Net, we borrow “sandwich rule” and “inplace distillation” from previous USNet [8] and MutualNet [7]. Our major contributions include:

- We propose a subnet pooling-based resource-adaptive supernet training that can sample and train subnets for different resource constraints and adjust them adaptively during the inference time.
- We collect subnets by introducing subnet pools during training to obtain high-quality subnet structures with different resource constraints. It eliminates subnet search process after supernet training.
- The experimental performance on classification task and object detection task demonstrates the effectiveness of our proposed PSS-Net.

II. RELATED WORK

Compact Networks. To overcome the hurdle of deploying models on resource-limited devices, many previous works focus on obtaining compact networks, typically through model compression [1], [2], [6], [19], [20], network architecture search (NAS) [3], [11], [21], etc. Network pruning, a form of model compression, removes redundant weights/filters to satisfy resource constraints in general hardware. Han et al. [22] proposed to recursively remove small-magnitude weights and retrain the \( l_2 \)-regularized subnetwork to recover accuracy. Lin et al. [23] discarded filters with low-rank feature map outputs. The lottery ticket hypothesis [24] randomly initializes a dense network and trains it from scratch. The subnets with high-weight values are extracted, and retrained with the initial weight values of the original dense model. In JCW [25], a supernet is used to evaluate subnet performance through joint channel and weight pruning. It searches for subnets considering tradeoff between latency and accuracy in the supernet by an evolutionary algorithm, and trains these subnets individually to obtain the final high-quality subnets. Our PSS-Net is very different from JCW: 1) The subnets of different sizes in JCW do not share weights because they are re-trained independently, while different subnets in our PSS-Net share weights. 2) JCW uses the supernet as a subnet performance evaluator, while in PSS-Net the supernet also offers final weights of all subnets. 3) The search space in JCW includes entire channels and individual weights, while it is channels and resolutions in our PSS-Net. 4) JCW requires a heuristic algorithm in order to conduct subnet search, while by introducing a subnet pool of high-quality subnets, our PSS-Net eliminates the process of subnet search. Besides, [26] and [27] also considered multi-dimension pruning including output channels, model depth, input resolution, etc. Among them, VFS [27] and our PSS-Net are similar in that they sample subnets under the constraints such as FLOPs and latency. The difference is that each subnet of VFS requires additional tuning therefore subnets do not share weights, while subnets in PSS-Net share weights since no fine-tuning is required therefore our PSS-Net saves more storage costs.

Sample-Adaptive Networks. A sample-adaptive network adapts its structure according to the input image samples. Lin et al. [28] modeled the adaption of network width as a Markov decision process, and measured the convolutional kernel importance conditioned on different samples. In [29] the network width is adjusted via a gating module conditioned on the current input. To this end, a “batch-shaping” is introduced to match the marginal aggregated posterior distribution over any intermediate features to a pre-specified prior distribution. Except for adjusting network width, another group realizes sample-adaptive network from the perspective of depth adaption. For instance, [30] proposed to early exit from the inference at shallow layers of the network through training multiple classifiers upon intermediate features. SkipNet [31] skips convolutions via a gating network that maps the last-layer outputs to a binary decision to execute or bypass the current layer.

Traditional compact networks are fixed for deployment. A sample-adaptive network allocates more computational resources to complex images, and fewer to images that are easier for the task. As a core distinction, our PSS-Net produces multiple subnets and switches these subnets for deployment according to available hardware resources.

III. METHODOLOGY

A. Preliminary

The proposed PSS-Net is an \( L \)-layer supernet \( \mathcal{N}_{\phi^{max}, \mathbf{W}} \) with its structure \( \phi^{max} = \{ (\phi^{max}_i)_{i=1}^L, r^{max}_i \} \) and weights \( \mathbf{W} = \{ \mathbf{W}_i \}_{i=1}^L \), where \( \phi^{max}_i \) and \( r^{max}_i \) are the number of output channels in the \( i \)-th layer, the supernet input resolution, and the weights of the \( i \)-th layer, respectively. The supernet \( \mathcal{N}_{\phi^{max}, \mathbf{W}} \) contains subnets of multiple sizes, and each subnet \( \mathcal{N}_{\phi^i, \mathbf{W}^i} \) is allowed to use a different subnet structure \( \phi^i = \{ (\phi^i_L)_{L=1}^1, r^i \} \) and supernet weights \( \mathbf{W}^i = \{ \mathbf{W}^i_1 : \phi^i_1 \} \), similarly, where \( \phi^i_1 \) and \( r^i \) are the number of output channels in the \( i \)-th layer and the supernet input resolution, respectively. \( \mathbf{W}^i_1 : \phi^i_1 \) means the supernet weights of the \( i \)-th layer are the weights of the first \( \phi^i_1 \) channels in \( \mathbf{W}^i \). Therefore, any two subnets share partial weights of the supernet. Then we set separate lower bounds for each layer of the subnet structure \( \{ \phi^i_L \}_{L=1}^1 \) and \( r^i \) as \( \{ \phi^i_{min} \}_{L=1}^1 \) and \( r^i_{min} \), that is, the structure of the smallest subnet can be represented as \( \phi^i_{min} = \{ (\phi^i_{min}^L)_{i=1}^L, r^i_{min} \} \). Finally, we can represent the subnet structure space as \( \mathcal{S} = \{ \phi^i = \{ (\phi^i_L)_{L=1}^1, r^i \} \}, \phi^2 = \{ (\phi^2_L)_{L=1}^2, r^2 \}, \ldots \), where \( \phi^i_{min} \leq \phi^i \leq \phi^{max}, r^i_{min} \leq r^i \leq r^{max} \). In our practical implementation, for any subnet \( \mathcal{N}_{\phi^i, \mathbf{W}^i} \), where the structure \( \phi^i = \{ (\phi^i_L)_{L=1}^1, r^i \} \in \mathcal{S} \), we first sample \( \phi^i \) from \( \{ \phi^i_{min} + 1, \phi^i_{min} + 2, \ldots, \phi^{max} \} \) and then round to its nearest multiple of 8 in order to obtain a hardware-friendly channel number for the deployment on most modern devices. Specifically, in most hardware, setting the channel number to an integer multiple of 8, 16, or 32 aligns the matrix computation with the processor unit, allowing for faster matrix computation.

For example, the GPU often has a wrap size of 32, while many mobile devices have a CPU with a wrap size of 8. Our method is designed to accommodate with more mobile devices, so we choose 8 as the channel divisor. The effect of different divisor sizes on the accuracy of the subnet will be discussed in Section IV-D. Now \( r^i \) can be sampled from the set of \( \{ r^i_{min}, r^i_{min} + 8, r^i_{min} + 16, \ldots, r^{max} \} \).
Given a set of resource constraints $\Theta = \{\theta^t\}_{t=1}^{T}$, the goal in this paper is to train a supernet $N_{\phi^\text{max}}(W)$ such that a total of $T$ subnets $\{N_{\phi^t}(W), \phi^t \in S, W^t \in W\}_{t=1}^{T}$ conditioned on the resource constraints $\Theta$ can be directly extracted from this supernet. Therefore, we can now formally define the problem of our supernet training as:

$$\{(\phi^t)_{\text{best}}\}_{t=1}^{T}, \{(W^t)_{\text{best}}\}_{t=1}^{T} = \arg \max_{\{(\phi^t)_{t=1}^{T}, W\}} \sum_{t=1}^{T} \text{Acc}(N_{\phi^t}^t, W^t),$$

s.t. $\phi^t \in S, \theta^t \in \Theta, C(N_{\phi^t}^t, W^t) \in [\theta^t - \delta, \theta^t + \delta]$, (1)

where $C(N_{\phi^t}^t, W^t)$ calculates the resource consumption of the subnet $N_{\phi^t}^t, W^t$. $\delta$ is a bias tolerance between the actual resource consumption and the given resource constraint $\theta^t$, and Acc$(N_{\phi^t}^t, W^t)$ returns the accuracy of $N_{\phi^t}^t$ on the validation dataset. Note that, weights of each subnet are a subset of the supernet weights, i.e., $W^t \in W$. Thus, to update $W^t$ is to update $W$ in essence. For ease of the following presentation, we remove the superscript “$t$” in $W^t$ and (1) can be simplified as:

$$\{(\phi^t)_{\text{best}}\}_{t=1}^{T}, (W)_{\text{best}} = \arg \max_{\{(\phi^t)_{t=1}^{T}, W\}} \sum_{t=1}^{T} \text{Acc}(N_{\phi^t}^t, W^t),$$

s.t. $\phi^t \in S, \theta^t \in \Theta, C(N_{\phi^t}^t, W) \in [\theta^t - \delta, \theta^t + \delta]$. (2)

To train an $L$-layer supernet $N_{\phi^t}(W)$, a three-step slimmable training paradigm is first proposed in [8], which consists of three steps: “sandwich rule,” “in-place distillation,” and “recalibration of batch normalization layers”. More specifically, the “sandwich rule” is first adopted to select three subnet types of one maximal size (i.e., the supernet, $N_{\phi^\text{max}}(W)$), several medium sizes and one minimal size (i.e., the minimal subnet $N_{\phi^\text{min}}(W)$). Then, the “in-place distillation” is conducted between the supernet and other subnets. Finally, the parameters in the batch normalization layers of the different subnets are recalibrated after supernet training while keeping the convolutional layer parameters being frozen. This paradigm has inspired many subsequent researchers to train resource-adaptive supernets or compact networks [3], [7], [9], [10], [11]. The main difference between these approaches lies in the selection of the medium-sized subnets in the “sandwich rule” step. Traditional approaches [7], [8] obtain several medium-sized subnet using uniform subnet sampling, i.e., applying the same width multipliers to all layers of the supernet to obtain subnets of different sizes in each batch training. However, such a manner causes suboptimal performance due to the homogeneity of subnets. The one-shot NAS studies [9], [10], [11] randomly sample several medium-sized subnets from the pre-defined subnet structure space $S$ to train the one-shot supernet. However, heavy cost is required to search for the target subnet when the available hardware resource constraints change from time to time.

### B. PSS-Net Training

To solve the above problems, we propose prioritized subnet sampling to sample the medium-sized subnet for training our supernet, termed PSS-Net in this paper. Our PSS-Net training generally follows the three-step slimmable training paradigm as shown in Fig. 2. We first outline our PSS-Net training in Algorithm 1, which is then detailed below.

**Subnet Pools (Line 1, Algorithm 1).** Different from existing methods, our PSS-Net training choose to maintain a set of subnet pools, denoted as $\Pi = \{\pi^t = \{\}\}_{t=1}^{T}$. The $t$-th subnet pool $\pi^t$...
is used to collect the information of $M$ high-quality subnets conditioned on $\theta^t$ in (2). The information of each subnet includes its network structure $\phi^t$ and performance metric $m^t$, which will be detailed in the next section.

**Slimmable Training (Lines 2 – 10, Algorithm 1).** In each training batch, we select three subnet types of one maximal size (i.e., the supernet, $\mathcal{N}_{\text{max}}(\mathbf{w})$, one medium size and one minimal size (i.e., the minimal subnet $\mathcal{N}_{\text{min}}(\mathbf{w})$). The “slimmable_training” in Line 8 of Algorithm 1 is the same as Line 3 – Line 16 of Algorithm 1 in USNet [8] and is elaborated in Algorithm 2. To obtain a medium-sized subnet $\phi^t$, USNet simply applies a fixed width multiplier to uniformly reduce the width of the supernet, whereas with the aid of the introduced subnet pools, our PSS-Net gives priority to high-quality subnets, as outlined in Algorithm 3 and detailed in the next section.

**BN Calibration (Lines 11 – 17, Algorithm 1).** After the slimmable training, each subnet pool $\pi^t \in \Pi$ will be filled with these high-quality subnets conditioned on its corresponding resource constraint $\theta^t$. However, the statistics in the batch normalization (BN) layers of these high-quality subnets are not accurately estimated since the forward propagated subnets vary during training our PSS-Net. As a consequence, BN calibration is indispensable in order to evaluate the performance of each subnet. To this end, following [7], [8], we feed each subnet with a small training set of 8,192 images in this paper to update the BN statistics of each subnet in $\pi^t$. Then, the calibrated subnet with the best accuracy is regarded as the optimal subnet $\mathcal{N}(\phi^t)_{\text{best}}(\mathbf{w})_{\text{best}}$ under the resource constraint of $\theta^t$. At the end of training, the subnet pool has collected a sufficient number of high-quality subnet structures for each resource constraint and the corresponding performance metrics detailed in the next section. We only calibrate $k$ subnets within top-$k$ performance metrics, and then pick the one with the best accuracy from these chosen subnets.

**C. Prioritized Subnet Sampling**

We outline our prioritized subnet sampling in Algorithm 3, which serves as a core distinction of our work from the previous studies. More details are discussed in the following.

**Subnet Sampling (Lines 1 – 7, Algorithm 3).** As mentioned above, we intend to sample a medium-sized subnet $\phi^t$ to train our PSS-Net in each training batch. Instead of simply sampling subnets from the subnet structure space $\mathcal{S}$, we also conduct subnet sampling from the subnet pool $\pi^t$ given the resource constraint $\theta^t$. To this end, we introduce a factor $p \in (0, 1)$ to indicate the probability of sampling from the subnet structure space $\mathcal{S}$. At the beginning of PSS-Net training, we sample subnets from $\mathcal{S}$ until $\pi^t$ is full. In this situation, we expect $p = 1$. As the training of PSS-Net, more high-quality structures of subnets will be inserted into $\pi^t$, and we expect to give priority to sampling from $\pi^t$, which indicates a small value of $p$. To this end, we formalize $p$ as:

$$p = p_{\text{end}} \cdot (\pi^t < M) \cdot (e-1) / E,$$

where $p_{\text{end}}$ is a pre-given parameter indicating the probability of sampling from the subnet structure space $\mathcal{S}$ when PSS-Net training ends. In our implementation, we set $p_{\text{end}} = 0.01$. $\mathcal{T}(\cdot)$ is an indicator which returns 1 if the input is true, and 0 otherwise. The $e \in [1, E]$ and $E$ respectively denote the current training epoch and the total training epochs. As can be seen, starting from 1, the probability of sampling from $\mathcal{S}$ decreases to $p_{\text{end}}$ gradually. Consequently, the priority will be entitled to the high-quality subnets.

**Case 1: Sampling from the subnet structure space $\mathcal{S}$ (Line 3 – Line 4, Algorithm 3).** To sample a subnet $\mathcal{N}(\phi^t, \mathbf{w})$ conditioned on $\theta^t$, one naive solution is to go through trial and error until an eligible subnet is found. However, this strategy brings heavy cost. Usually, thousands of sampling are inevitable each time according to our experimental observation. Inspired by [11], we resort to the probability distribution of subnets conditioned on the given resource constraint $\theta^t$, denoted as $\tau(\phi = \{o_i\}_{i=1}^T | \theta^t)$. However, the practical distribution is extremely complex to describe. While the Monte-Carlo approximation can be adopted, it is still intractable since multiple approximations have to be repeatedly constructed for each resource constraint $\theta^t \in \Theta$. To overcome this, we first generate a large-scale set of subnet structures $\mathcal{G}$, and then compute the practical resource consumption of each structure $\phi \in \mathcal{G}$ as $\mathcal{C}(\mathcal{N}(\phi, \mathbf{w}))$. Finally, we split $\mathcal{G}$ into $T$ non-overlapping subnets with $\mathcal{G}_1 \cup \mathcal{G}_2 \cup \cdots \cup \mathcal{G}_T = \mathcal{G}$. For each subnet structure $\phi^t \in \mathcal{G}_t$, its resource consumption satisfies that $\mathcal{C}(\mathcal{N}(\phi^t, \mathbf{w})) \in [\theta^t – \sigma, \theta^t + \sigma]$. To model $\tau(\phi^t = \{o_i\}_{i=1}^T | \theta^t)$, we directly estimate the marginal probability of $\tau(o_k^t | \theta^t)$ as:

$$\tau(o_k^t | \theta^t) \approx \frac{\sum_{\phi^t \in \mathcal{G}_t} \mathcal{I}(o_k^t = o_k^t)}{|\mathcal{G}_t|}.$$

With (4), the sampling times to find the target subnet reduces from $10^6$ to $10^1$ approximately according to our observation, which greatly improves the sampling efficiency.

---

**Algorithm 3: Prioritized Subnet Sampling.**

**Input:** Subnet pool $\pi^t$ conditioned on resource constraint $\theta^t$, subnet structure space $\mathcal{S}$, current epoch $e$, and total epochs $E$.

**Output:** Subnet structure $\phi^t$, and subnet pool $\pi^t$

1: $p = p_{\text{end}} \cdot (\pi^t < M) \cdot (e-1) / E$,
2: if $\text{rand}(0, 1) < p$ then
3: Sample a subnet structure $\phi^t \in \mathcal{S}$, s.t. $\mathcal{C}(\mathcal{N}(\phi^t, \mathbf{w})) \in [\theta^t – \delta, \theta^t + \delta]$ using (4);
4: else
5: $\eta = \eta_{\text{end}} \cdot (\pi^t < M) \cdot (e-1) / E$;
6: Sample a subnet structure $\phi^t \in \pi^t$ using (6);
7: end if
8: if $\phi^t$ not in $\pi^t$ then
9: $m^t = -$batch_loss$;$
10: Insert ($\phi^t, m^t$) into $\pi^t$;
11: if $|\pi^t| > M$ then
12: Remove ($\phi^t_M, (m^t)_M$);
13: end if
14: else
15: $m^t = \lambda \cdot m^t + (1 - \lambda) \cdot (-$batch_loss$)$;
16: Update $m^t$ of $\phi^t$ in $\pi^t$;
17: end if
18: return $\phi^t, \pi^t$. 

---

111111 IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 45, NO. 9, SEPTEMBER 2023
Case 2: Sampling from the subnet pool $\pi^t$ (Line 5 – Line 7, Algorithm 3). For $\pi^t = \{(\phi^t_j, (m^t_j))\}_{j=1}^M$, to obtain the best subnet conditioned on $\theta$ in (2), we resort to preserving the best subnet structure in $\pi^t$ for deployment, i.e.,

$$ (\phi^t)_{\text{best}} = \arg\max_{(\phi^t_j)} \text{Acc}(N(\phi^t_j), \mathbf{w}), $$

after our PSS-Net training. Thus, it is natural to prioritize the subnet structure with the best performance $m^t$ in $\pi^t$ to train our PSS-Net. To this end, we perform weighted sampling upon $\pi^t$ so that better subnets are assigned with higher probabilities. Thus, we can obtain a distribution set $\{(q^t_j)_{j=1}^M$ with $\sum_{j=1}^M (q^t_j) = 1$ where $(q^t_j)$ denotes the probability of sampling $(\phi^t_j)$ to train PSS-Net, defined as:

$$ (q^t_j) = \frac{\exp((m^t_j) / \eta)}{\sum_{i=1}^M \exp((m^t_i) / \eta)}, $$

where $\eta$ is a temperature to control the smoothness of the distribution. In the early training stage, the subnets are not well trained, and thus $m^t$ is not a reliable measure to reflect the actual performance of each subnet. At this time, we expect each subnet to be sampled equally. As training goes, $m^t$ will become stable and we expect the distribution to be a one-hot form. Similar to (3), we achieve this by setting $\eta = \eta_{\text{end}} \sum_{i=1}^M ((\pi^t_i) < M) (e-1)/E$, where $\eta_{\text{end}} = 0.01$.

Subnet Pool Manipulation (Lines 8 – 16, Algorithm 3). We continuously collect the information tuple $(\phi^t, m^t)$ until the pool $\pi^t$ is full. When receiving a $(\phi^t, m^t)$, we insert it into $\pi^t$ such that the information tuples in $\pi^t$ are arranged in a decreasing order according to the performance $m^t$. In the case of full pool, we can obtain $\pi^t = \{(\phi^t_j, (m^t_j))\}_{j=1}^M$ with $m^t_{j+1} \leq (m^t)_j$ and $\phi^t_j \subseteq \phi^t_{j+1}$. If one more tuple received, we will remove the worst performing tuple, i.e., $(\phi^t_M, (m^t_M))$, before inserting this tuple.

To model $m^t$ of $\phi^t$, the best way is to define it as the accuracy of the subnet $N(\phi^t_j, \mathbf{w})$. However, due to the large subnet structure space, it is almost impossible to execute a complete training for every subnet. Inspired by [11], [32], we define $m^t_j = \text{batch\_loss}$ if $\phi^t$ is inserted into $\pi^t$ for the first time. If $\phi^t$ is sampled from $\pi^t$ for updating PSS-Net, we update $m^t$ using moving average as $m^t = \lambda \cdot m^t + (1 - \lambda) \cdot (\text{batch\_loss})$ after the current training batch. Here, we define $\text{batch\_loss}$ as the training loss in the current batch.

IV. EXPERIMENTS

A. Experimental Settings

We use the light-weight MobileNet-V1 [16] and MobileNet-V2 [17] as well as heavy ResNet-50 [18] as the backbones of PSS-Net and compared methods, and then conduct classification experiments on ImageNet [33]. We set $q^t_{\text{max}}$ to be the same as the channel number of the $i$-th layer in the backbone network, and $r_{\text{max}}^{\text{w}}$ is 224. Then we define $q^t_{\text{min}} = 0.75 \cdot r_{\text{max}}^{\text{w}}, r_{\text{min}}^{\text{w}} = 128$. After that we can obtain the supernet $N(\phi^t_{\text{min}}, \mathbf{w})$ and the subnet space $S$. We perform three resource constraints in this paper, including FLOPs, GPU and CPU latencies. The resource constraint set $\Theta$ ranges from $C(N(\phi^t_{\text{min}}, \mathbf{w}))$ to $C(N(\phi^t_{\text{max}}, \mathbf{w}))$ with steps of 10 M, 10 ms and 1 ms for FLOPs, GPU latency and CPU latency, respectively. The subnet pool size $M$ is set to 50.

We construct lookup tables of CPU and GPU latencies for a quick calculation of resource consumption $C(\cdot)$ during PSS-Net training. The details of the lookup tables and more ablations are provided in Section IV-B. We use PyTorch as our training framework. The initial learning rate is set to 0.3 which is then adjusted by cosine annealing. SGD optimizer is adopted with a weight decay of 1e-4. We train our PSS-Net for a total of 250 epochs with a batch size of 1024.

B. Latency Lookup Table

We respectively measure the CPU latency and GPU latency on a single core of Intel Xeon Platinum 8268 CPU and on a single NVIDIA Tesla V100 GPU with a batch size of 1 and 128. To realize a quick calculation of resource consumption $C(\cdot)$, we construct lookup tables of CPU and GPU latencies for our PSS-Net training. To this end, the most naive approach is to pre-compute the practical consumption on the target platform for all subnet structures $\phi^t \in S$. However, the large volume of $S$ makes it intractable. Instead, we choose to compute the latency of each subnet block in $S$ and build latency tables for these blocks. During PSS-Net training, the latency of a given subnet structure is predicted by summing up the latencies of the blocks making up this subnet. In Fig. 3, we randomly sample 1,000 subnet structures using MobileNet-V1 as the supernet, and show the actual GPU latencies of these subnets versus their block summation from the lookup tables. In Fig. 3, the predicted latency is very close to the real latency, which indicates that our lookup tables are very accurate and reliable.

C. Experimental Results

In this subsection, we compare the top-1 accuracy of subnets obtained by different methods, including the resource-adaptive
TABLE I

Comparison With Resource-Adaptive Methods. We first compare our PSS-Net with the resource-adaptive studies [7], [8]. We train four types of PSS-Net constrained by different resources including: (1) PSS-NetF is trained under only FLOPs constraint; (2) PSS-NetC is trained under only CPU latency constraint; (3) PSS-NetG is trained under only GPU latency constraint; (4) PSS-NetM is trained under multiple constraints of FLOPs, CPU latency and GPU latency.

In Table I, we compare the accuracies of the subnets resulting from different methods under similar FLOPs, GPU latency, and CPU latency. First, our PSS-Net provides subnets with better performance when compared to MutualNet [7] and USNet [8] under similar or less resource consumption on FLOPs, GPU latency, and CPU latency. Second, USNet using MobileNet-V1 as its supernet backbone fails to produce subnets with low GPU/CPU latency. Third, though PSS-NetM needs to accommodate more resource constraints, its performance is still comparable or even better than PSS-NetF, PSS-NetG, and PSS-NetC which are dedicated to a single type of resource constraint. Fourth, PSS-Net well improves the performance of backbone networks, for instance, PSS-NetM well increases the accuracy of MobileNet-V1/V2 from 70.9%/71.8% to 74.5%/73.4% with FLOPs reduction.

To better demonstrate the advantage of our PSS-Net, we further plot the resource-accuracy curves of different methods in Fig. 4. We can observe that our PSS-Net consistently outperforms USNet and MutualNet by a large margin in all subfigures. Under similar resource consumption, our PSS-Net brings subnets with higher accuracy. In addition, similar to Table I, USNet using MobileNet-V2 as the supernet backbone only produces subnets with high GPU/CPU latency. Specifically, the GPU latency is over 225 μs and the CPU latency is over 30 ms. Such a limitation greatly barricades the deployment of USNet, particularly when the available resources are in short supply. The reasons are attributed to two points. First, the input image resolution of USNet is fixed to 224, which is a large size and thus results in heavy GPU/CPU latency. Second, USNet adopts uniform subnet sampling, which ignores the diversity of layer-wise width. The resulting subnets are usually hardware-unfriendly. Our PSS-Net searches for the optimal image resolution for each subnet and the sampling strategy is non-uniform, which brings about subnets with low latency in comparison with existing methods and the compact network methods under similar resource constraints.

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
methods. In addition to that, the proposed prioritized subnet sampling makes sure that high-performing subnets are trained sufficiently, which in turn entitles high performance of these subnets.

**Comparison With Compact Models.** We compare PSS-Net with some compact models including model pruning and network architecture search (NAS) methods. Model pruning methods include VFS [27], PFS [2], MetaPruning [6], MDP [34], AMC [21], NetAdapt [35], EagleEye [20], AutoSlim [12], DPFPSS [1], CC [36], DMCP [37], CPLI [19], ThilNet [38], ABCPruner [13], HRank [23], FPGM [39], CURL [40] and DMC [41]. NAS methods include DARTS [42], PNAS [43], NASNet [44] and ChamNet [4]. The subnet search space of PSS-Net is limited to the size of the backbone networks, while the subnet search space of most NAS methods is much larger than that of the backbone networks. Therefore, for a fair comparison, we mainly compare PSS-Net with the model pruning methods. We compare the accuracy under similar FLOPs consumption since the FLOPs metric is more often adopted in the compact model research. To that effect, we train our supernet PSS-Net with MobileNet-V1/V2 and ResNet-50 as backbones respectively, and then extract the best subnets from PSS-NetF and PSS-NetF, whose FLOPs consumption is similar to the compact models for a fair comparison. Table II displays the performance comparison. As can be seen, our PSS-Net results in high-capacity subnets which not only surpass the compact model methods significantly, but also are better than the original MobileNet-V1/V2 and ResNet-50 sometimes. Taking MobileNet-V1 as an example, our PSS-Net provides a subnet that reduces the FLOPs of MobileNet-V1 from 569 M to 278 M, and meanwhile increases the accuracy from 70.6% to 71.8%. Moreover, these compact methods produce static models for deployment while our PSS-Net offers switchable subnets to adapt the change of available resources.

**Latency on Mobile Devices.** Following JCW [25], we also report the single core ARM Cortex-A72 CPU latencies with TensorFlow Lite. Table III provides the comparison results with JCW [25], USNet [8], AutoSlim [12], AMC [21], FSC [45], DMCP [37] and APS [46]. The advantage of our PSS-Net is two-fold: First, PSS-Net enjoys a weight-shared mechanism, leading to less storage requirement when deploying multiple subnets. Second, under similar or even less latency, PSS-Net results in better performance in most cases. Therefore, the applicability of PSS-Net on mobile devices is also well demonstrated.

**Storage-Constrained Devices.** In addition to dynamically adapting to different FLOPs, CPU latency, and GPU latency constraints, as in the compact network methods, our PSS-Net can also be applied in storage resource-constrained scenarios by simply setting the type of resource constraints to the number of parameters at training time. Table IV shows the performance...
of different subnets with different number of parameters in the same PSS-Net supernet, which can be deployed independently on devices with different requirements of storage constraints.

**Complexity Comparison.** We further compare the storage and training costs of PSS-Net, resource-adaptive methods [7], [8], and the compact methods of model pruning. Denote the training cost of the supernet as $O_T$ and the storage cost as $O_S$. Given $N$ subnets, we summarize the cost complexity in Table V. For resource-adaptive networks, $k$ subnets of different sizes ($k = 4$ for USNet and MutualNet, $k = 3$ for PSS-Net) are trained in each batch, and thus the total training cost is not more than $kO_T$. The resource-adaptive network methods require to store the parameters of means and variances of the BN layers for each of the $N$ subnets (the number of BN parameters for each MobileNet-V1/V2 subnet is about 1% of the supernet’s parameters $O_S$), and thus the total storage overhead is about $(1 + 0.01N)O_S$. Most of the model pruning methods require a complete training for each compact subnet in isolation, so the total training cost is about $NO_T$ and the total storage cost is $NO_S$. Taking $N = 100$ as an example, our PSS-Net achieves better performance than the model pruning methods (Table II) with about 3% of the training cost and about 2% of the storage.
cost compared to the model pruning methods (Table V). Note that the MDP [34] prunes model in three dimensions: channel, resolution and depth. In contrast, the subnet search space of PSS-Net only has two dimensions: resolution and channels. Nevertheless, the performance of the PSS-Net still exceeds the compact model trained by MDP, further demonstrating the effectiveness of PSS-Net.

D. Ablation Studies

We further conduct ablation studies to analyze the effectiveness of our PSS-Net. All the experiments are conducted using MobileNet-V1 as the supernet which is trained under the FLOPs constraint.

**Compare With a Random Search.** We compare PSS-Net with a random search under the same subnet space with resource constraints of 300 M, 400 M and 500 M. As shown in Table VI, PSS-Net improves over a random search for different sizes of resource constraints, especially for smaller FLOPs constraints. This is due to the fact that the selection space of subnets with small FLOPs tends to be larger than that of large FLOPs, so that random searches are less likely to find better subnet structures. It also illustrates the effectiveness of PSS-Net in using subnet pools to collect high-quality subnets during training.

**Low Bound of Subnet Space.** Recall that we set $o^\text{max}_i$ to be the same as the channel number of the $i$-th layer in the backbone network, and the low bound of subnets $o^\text{min}_i = 0.75 \cdot o^\text{max}_i$. In this subsection, we consider different low bounds adopted by MutualNet [7] and show the experimental results in Table VII. Our PSS-Net shows better performance when the low bounds are set to different scales.

**BN Calibration.** Our PSS-Net follows the slimmable training paradigm which requires BN calibration for each subnet. However, the large volume of subnet space disables one-by-one calibration. Luckily, our introduced subnet pool encourages to collect more high-quality subnets. Consequently, we only need to calibrate $k$ subnets within top-$k$ performance metrics, and then pick the one with the best accuracy. To verify this, in Table VIII, we choose different values of $k$ and show the best subnets after BN calibration, which are constrained by the FLOPs of 300 M, 400 M and 500 M. When $k \geq 2$, better subnets can be observed than simply considering the top-1. However, the performance gains are limited when $k > 5$, which indicates the best subnets fall into these subnets within top-5 performance metrics. Thus, we pick up 5 subnets in each pool for BN calibration, which is also our setting in all the experiments.

**Other Hyper-Parameters.** We also perform an ablation study for the hyper-parameters $p_{\text{end}}, \eta_{\text{end}}, M$, and divisor in the algorithms of PSS-Net Training and Prioritized Subnet Sampling. As shown in Table IX, $p_{\text{end}}$ is insensitive to the hyper-parameter value and $\eta_{\text{end}}$ is slightly better given 0.01. For simplicity, we use $p_{\text{end}} = \eta_{\text{end}} = 0.001$ in our experiments. It can also be seen from Table IX that the performance of the subnet increases with the subnet pool size $M$, but using too large a subnet pool size leads to too much storage overhead, so we take $M = 50$ in this paper. As for the channel divisor, a smaller divisor results in better performance. This is due to the fact that a smaller divisor leads to a larger subnet sampling space, which improves the subnet performance. As mentioned in Section III-A, we choose $\text{divisor} = 8$ to align channel number with the warp size of most mobile CPUs for more efficient computation.

**Subnet Fine-Tuning.** As mentioned in the introduction section, existing researches suffer the problem of subnet mismatch, which causes insufficient training of high-quality subnets. Our PSS-Net solves this problem by giving priority to sampling high-quality subnets in the subnet pool. In this subsection, we further investigate whether the best subnets have been well trained in PSS-Net. To this end, we select the best subnets from the 300M-, 400M-, and 500M-subnet pools, and further fine-tune these subnets for 25 epochs. Table X shows the performance comparison before/after fine-tuning. As can be seen,
fine-tuning incurs slight performance drops in these three cases. This experiment demonstrates that our high-quality subnets are already well trained and our PSS-Net well solves the subnet mismatch problems.

E. Detection Performance

We further verify the detection performance on COCO [47] using Faster-RCNN [48]. The network backbone is ResNet-50. Table XI manifests the experiment comparison. We observe similar conclusion as in classification where the proposed PSS-Net results in subnets that have similar or less computation cost, but significant increase in mAP performance when comparing to existing USNet and MutualNet. The superiority is outstanding in particular to a smaller backbone such as 0.25×. Therefore, the efficacy of our PSS-Net is well demonstrated even upon different tasks.

V. LIMITATIONS

We construct lookup tables based on the block latency of a supernet for a quick calculation of resource consumption in PSS-Net training. Though effective enough, the predicted latency sometimes still deviates slightly from the real latency, which would misguide the sample of subnets. Besides, theoretical analyses on why the proposed subnet sampling works well and the relationship between the subnet pools and subnet design space remain to be further explored.

VI. CONCLUSION

We have proposed Prioritized Subnet Sampling (PSS-Net) to train a resource-adaptive supernet. Compared to existing resource-adaptive networks, our PSS-Net offers subnets with a better accuracy-resource trade-off by maintaining subnet pools, each of which is conditioned on a particular resource constraint, and assigning priority to high-quality subnets for training. Our PSS-Net produces a set of high-quality subnets with different resource consumptions, thus allowing a fast switch of high-quality subnets for inference when the available resources vary. Besides, compared to the compact network methods such as pruning and NAS, our PSS-Net significantly reduces the training and storage cost.

| Table X | PERFORMANCE COMPARISON BEFORE/AFTER FINE-TUNING THE BEST SUBNETS |
|---------|---------------------------------------------------------------|
| Subnet  | FLOPs | Acc_{25%} (%) | Acc_{top-1} (%) |
| A       | 300M  | 72.3         | 71.9 (-0.4)    |
| B       | 400M  | 72.9         | 72.7 (-0.2)    |
| C       | 500M  | 73.8         | 73.5 (-0.3)    |

| Table XI | PERFORMANCE OF FASTER-RCNN ON COCO WITH RESNET-50 BACKBONE |
|----------|-------------------------------------------------------------|
| Method   | FLOPs (G) | Box mAP (%) | ∆Box mAP (%) |
| Uniform 1.0× | 236   | 36.4 | +0.0   |
| USNet    | 236   | 36.8 | +0.4   |
| MutualNet | 186   | 36.8 | +0.4   |
| PSS-Net  | 174   | 36.8 | +0.4   |
| Uniform 0.75× | 170  | 34.7 | +0.0   |
| USNet    | 170   | 36.3 | +1.6   |
| MutualNet | 167   | 36.6 | +1.9   |
| PSS-Net  | 165   | 36.7 | +2.0   |
| Uniform 0.5× | 144   | 32.7 | +0.0   |
| USNet    | 144   | 34.5 | +1.8   |
| MutualNet | 143   | 34.8 | +2.4   |
| PSS-Net  | 142   | 36.0 | +3.3   |
| Uniform 0.25× | 129  | 27.5 | +0.0   |
| USNet    | 129   | 30.4 | +2.9   |
| MutualNet | 126   | 31.8 | +4.3   |
| PSS-Net  | 126   | 35.6 | +8.1   |

REFERENCES

[1] X. Ruan, Y. Liu, B. Li, C. Yuan, and W. Hu, “DPFFS: Dynamic and progressive filter pruning for compressing convolutional neural networks from scratch,” in Proc. AAAI Conf. Artif. Intell., 2021, pp. 2495–2503.
[2] Y. Wang et al., “Pruning from scratch,” in Proc. AAAI Conf. Artif. Intell., 2020, pp. 12 273–12 280.
[3] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, “Once-for-all: Train one network and specialize it for efficient deployment,” in Proc. Int. Conf. Learn. Representations, 2020.
[4] X. Dai et al., “ChamNet: Towards efficient network design through platform-aware model adaptation,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2019, pp. 11 398–11 407.
[5] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, “Slimmable neural networks,” in Proc. Int. Conf. Learn. Representations, 2019.
[6] Z. Liu et al., “MetaPruning: Meta learning for automatic neural network channel pruning,” in Proc. IEEE/CVF Conf. Comput. Vis., 2019, pp. 3296–3305.
[7] T. Yang, S. Zhu, C. Chen, S. Yan, M. Zhang, and A. Willis, “MutualNet: Adaptive ConvNet via mutual learning from network width and resolution,” in Proc. Eur. Conf. Comput. Vis., 2020, pp. 299–315.
[8] J. Yu and T. S. Huang, “Universally slimmable networks and improved training techniques,” in Proc. IEEE/CVF Int. Conf. Comput. Vis., 2019, pp. 1803–1811.
[9] W. Lou, L. Xun, A. Sabet, J. Bi, J. Hare, and G. V. Merrett, “Dynamic- OFA: Runtime DNN architecture switching for performance scaling on heterogeneous embedded platforms,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2021, pp. 3110–3118.
[10] J. Yu et al., “BigNAS: Scaling up neural architecture search with big single-stage models,” in Proc. Eur. Conf. Comput. Vis., 2020, pp. 702–717.
[11] D. Wang, M. Li, C. Gong, and V. Chandra, “AttentionNAS: Improving neural architecture search via attentive sampling,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2021, pp. 6418–6427.
[12] J. Yu and T. Huang, “AutoSlim: Towards one-shot architecture search for channel numbers,” 2019, arXiv:1903.11726.
[13] M. Lin, B. H. Y. Zhang, B. Zhang, Y. Wu, and Y. Tian, “Channel pruning via automatic structure search,” in Proc. Int. Joint Conf. Artif. Intell., 2020, pp. 673–679.
[14] Z. Guo et al., “Single path one-shot neural architecture search with uniform sampling,” in Proc. Eur. Conf. Comput. Vis., 2020, pp. 544–560.
[15] M. Sahni, S. Varshini, A. Khare, and A. Tumanov, “CompOFA: Compound once-for-all networks for faster multi-platform deployment,” in Proc. Int. Conf. Learn. Representations, 2021.
[16] A. G. Howard et al., “MobileNets: Efficient convolutional neural networks for mobile vision applications,” 2017, arXiv:1704.04861.
[17] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobilenetV2: Inverted residuals and linear bottlenecks,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2018, pp. 4510–4520.
[18] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2016, pp. 770–778.
[19] J. Guo, W. Ouyang, and D. Xu, “Channel pruning guided by classification loss and feature importance,” in Proc. AAAI Conf. Artif. Intell., 2020, pp. 10 885–10 892.
[20] B. Li, B. Wu, J. Su, and G. Wang, “EagleEye: Fast sub-net evaluation for efficient neural network pruning,” in Proc. Eur. Conf. Comput. Vis., 2020, pp. 639–654.
[21] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, “AMC: AutoML for model compression and acceleration on mobile devices,” in Proc. Eur. Conf. Comput. Vis., 2018, pp. 784–800.
Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.