TIPS: Topologically Important Path Sampling for Anytime Neural Networks

Guihong Li [1] **Kartikeya Bhardwaj** [2] **Yuedong Yang** [1] **Radu Marculescu** [1]

Abstract

Anytime neural networks (AnytimeNNs) are a promising solution to adaptively adjust the model complexity at runtime under various hardware resource constraints. However, the manually-designed AnytimeNNs are biased by designers' prior experience and thus provide sub-optimal solutions. To address the limitations of existing hand-crafted approaches, we first model the training process of AnytimeNNs as a discrete-time Markov chain (DTMC) and use it to identify the paths that contribute the most to the training of AnytimeNNs. Based on this new DTMC-based analysis, we further propose *TIPS*, a framework to automatically design AnytimeNNs under various hardware constraints. Our experimental results show that TIPS can improve the convergence rate and test accuracy of AnytimeNNs. Compared to the existing AnytimeNNs approaches, TIPS improves the accuracy by 2%-6.6% on multiple datasets and achieves SOTA accuracy-FLOPs tradeoffs.

1. Introduction

In recent years, deep neural networks (DNNs) have been successful in many areas, such as computer vision or natural language processing (Vaswani et al., 2017; Dosovitskiy et al., 2021). However, the intensive computational requirements of existing large models limit their deployment on resource-constrained devices for Internet-of-Things (IoT) and edge applications. To improve the hardware efficiency of DNNs, multiple techniques have been proposed, such as quantization (Qin et al., 2020; Han et al., 2016), pruning (Luo et al., 2017; Han et al., 2015), knowledge distillation (Hinton et al., 2015), and neural architecture search (NAS) (Zoph & Le, 2017; Liu et al., 2019; Stamoulis et al., 2019; Li et al., 2020; 2023). We note that all these techniques focus on generating *static* neural architectures that can achieve high accuracy under specific hardware constraints.

Recently, anytime neural networks (AnytimeNNs) have been proposed as an orthogonal direction to static neural networks (Huang et al., 2018; Yu & Huang, 2019a; Bengio et al., 2015; Wang et al., 2021; Yang et al., 2021). AnytimeNNs adjust the model size at runtime by selecting subnetworks from a static supernet (Chen et al., 2019; Li et al., 2019; Yu & Huang, 2019a;b; Yu et al., 2019). Compared to the static techniques, AnytimeNNs can automatically adapt (at runtime) the model complexity based on the available hardware resources. However, the existing AnytimeNNs are manually designed by selecting a few candidate subnetworks. Hence, such hand-crafted AnytimeNNs are likely to miss the subnetworks that can offer better performance. These limitations of existing manual design approaches motivate us to analyze the properties of AnytimeNNs and then provide a new algorithmic solution. Specifically, in this work, we address two **key questions:**

1. *How can we quantify the importance of various operations (e.g., convolutions, residual additions, etc.) to the convergence rate and accuracy of AnytimeNNs?*
2. *Are there topological (i.e., related to network structure) properties that can help us design better AnytimeNNs?*

To answer these questions, we analyze the AnytimeNNs from a *graph theory* perspective. This idea is motivated by the observation that the topological features of DNNs can accurately indicate their gradient propagation properties and test performance (Bhardwaj et al., 2021; Li et al., 2021b). Inspired by the network structure analysis, given an AnytimeNN, we propose a Discrete-Time Markov Chain (DTMC)-based framework to explore the relationships among different subnetworks. We then propose two new topological metrics, namely *Topological Accumulated Score* (TAS) and *Topological Path Score* (TPS) to analyze the gradient properties of AnytimeNNs. Based on these two metrics, we finally propose a new training method, *i.e.*, *Topologically Important Path Sampling* (TIPS), to improve the convergence rate and test performance of AnytimeNNs. The experimental results show that our proposed approach outperforms SOTA approaches by a significant margin across

[1]Department of ECE, The University of Texas at Austin, Austin, TX [2]Qualcomm AI Research, San Deigo, CA; work done while Kartikeya Bhardwaj was at Arm, Inc. Correspondence to: Radu Marculescu <radum@utexas.edu>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

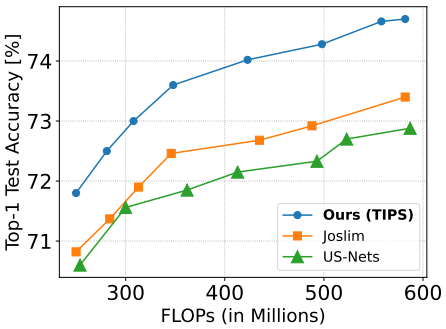


Figure 1. Test Accuracy vs. FLOPs on ImageNet. TIPS achieves higher accuracy (given the same or even fewer FLOPs) than SOTA AnytimeNNs: Joslim (Chin et al., 2021) and US-Nets (Yu & Huang, 2019b).

many models and datasets (see Fig. 1). Overall, we make the following **key contributions:**

- We propose a new importance analysis framework by modeling the AnytimeNNs as DTMCs; this enables us to capture the relationships among different subnetworks of AnytimeNNs.
- Based on the DTMC-based framework, we propose two new topological metrics, *Topological Accumulated Score* (TAS) and *Topological Path Score* (TPS), which can characterize the operations that contribute the most to the training of AnytimeNNs.
- We propose a new theoretically-grounded training strategy for AnytimeNNs, namely, *Topologically Important Path Sampling* (TIPS), based on our importance analysis framework. We show that TIPS achieves a faster convergence rate compared to SOTA training methods.
- We demonstrate that TIPS enables the automatic design of better AnytimeNNs under various hardware constraints. Compared to existing AnytimeNN methods, TIPS improves the accuracy by 2%-6.6% and achieves SOTA accuracy-FLOPs tradeoffs on multiple datasets, under various hardware constraints (see Fig. 1).

The rest of the paper is organized as follows. In Section 2, we discuss related work. In Section 3, we formulate the problem and introduce our proposed solution (TIPS). Section 4 presents our experimental results and outline directions for future work. Finally, Section 5 concludes the paper.

2. Related Work

There are three major directions related to our work:

2.1. Anytime Inference

Anytime neural networks (AnytimeNNs) can adapt the model complexity at runtime to various hardware constraints; this is achieved by selecting the optimal subnetworks of a given (static) architecture (*supernet*), while maintaining the test accuracy. The runtime adaptation of AnytimeNNs is primarily driven by the available hardware resources (Yuan et al., 2020). For instance, early-exit networks can stop the computation at some intermediate layers of the supernet and then use individual output layers to get the final results (Wang et al., 2018; Veit & Belongie, 2018; Bolukbasi et al., 2017). Similarly, skippable networks can bypass several layers at runtime (Wang et al., 2020; Larsson et al., 2017; Wu et al., 2018). Alternatively, approaches for slimmable networks remove several channels of some layers at runtime (Lee & Shin, 2018; Bejnordi et al., 2020; Yang et al., 2018; Hua et al., 2019; Li et al., 2021a; Chin et al., 2021; Gao et al., 2019; Tang et al., 2021). Finally, multi-branch networks select the suitable branches of networks to reduce the computation workload to fit the current hardware constraints (Cai et al., 2021; Ruiz & Verbeek, 2021; Huang et al., 2018; Liu et al., 2020).

2.2. Layerwise Dynamical Isometry (LDI)

LDI is meant to quantify the gradient flow properties of DNNs (Saxe et al., 2014; Xiao et al., 2018; Burkholz & Dubatovka, 2019). For a deep neural network, let x_i be the output of layer i; the Jacobian matrix of layer i is defined as: $J_{i,i-1} = \frac{\partial x_i}{\partial x_{i-1}}$. Authors of (Lee et al., 2020) show that if the singular values of $J_{i,i-1}$ for all i at initialization are close to 1, then the network satisfies the LDI, and the magnitude of the gradient does not vanish or explode, thus benefiting the training process.

2.3. Network Topology

Previous works show that the topological properties can significantly impact the convergence rate and test performance of deep networks. For example, by modeling deep networks as graphs, authors in (Bhardwaj et al., 2021) prove that the average node degrees of deep networks are highly correlated with their convergence speeds. Lately, (Chen et al., 2022) developed a similar understanding of neural networks' connectivity patterns on its trainability. Moreover, several works also show that some specific topological properties of deep networks can indicate their test accuracy (Li et al., 2021b; Javaheripi et al., 2021). We note that these existing approaches primarily focus on networks with a static structure. The relationship between topological properties and the convergence/accuracy of networks with varying architectures (*e.g.*, AnytimeNNs) remains an open question. This motivates us to investigate the topological properties of AnytimeNNs.

3. Approach

In this work, for a given deep network (*i.e.*, supernet), our goal is to automatically find its AnytimeNN version under

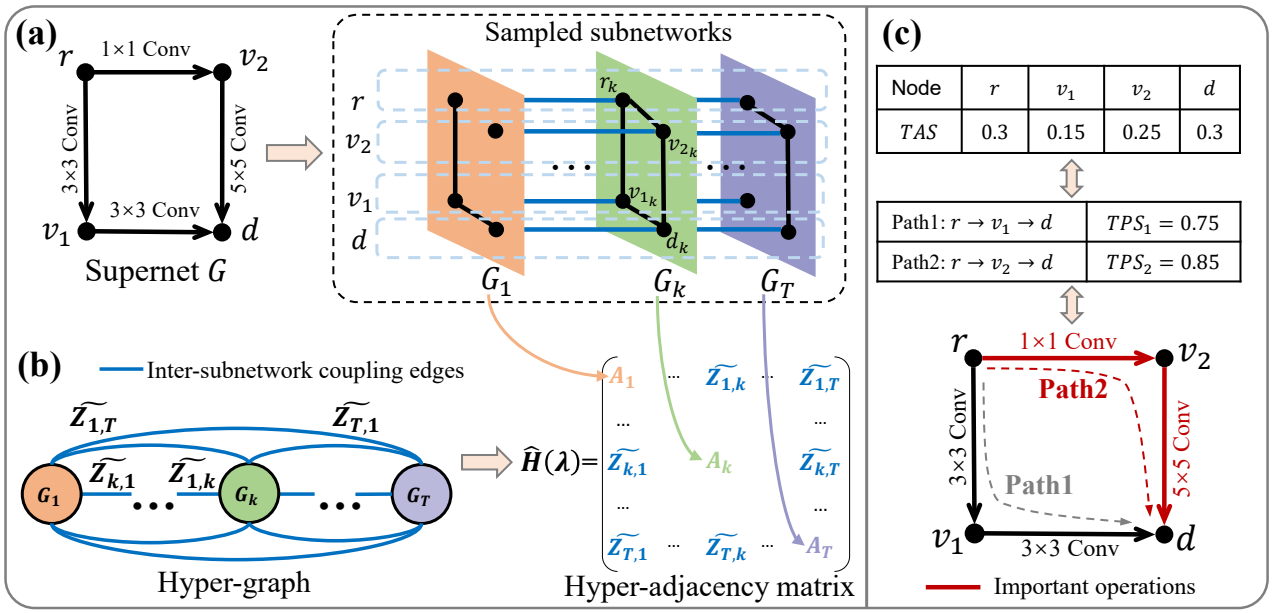


Figure 2. Overview of our proposed DTMC-based analysis (**a**) We model the operations (*e.g.*, 1×1-Conv) as edges; we model the outputs of such operations (*i.e.*, featuremaps) as nodes (*e.g.*, r, v_2); here r and d are the input and output nodes of the supernet G, respectively. By removing some edges from supernet G, we generate multiple subnetworks $\{G_k,\ k = 1, ..., T\}$. (**b**) We then build the adjacency matrices $\{\boldsymbol{A}_k,\ k = 1, ..., T\}$ for each subnetwork. We combine these adjacency matrices and the inter-subnetwork coupling matrices $\{\widetilde{\boldsymbol{Z}_{i,j}},\ i \neq j\}$ to form the hyper-adjacency matrix $\hat{\boldsymbol{H}}(\lambda)$. (**c**) By solving Eq. (6)(7)(8), we get the TAS value of each node. After finding the path with the highest TPS value by Eq. 9 (Path2), we characterize the important operations (*i.e.*, edges) of the AnytimeNN. We provide more examples in Appendix C.

various hardware constraints. To this end, our approach consists of three major steps: (*i*) Characterize the importance of each operation (convolution, residual addition, *etc.*). As such, we model the training process of AnytimeNNs as a DTMC and use it to analyze their topological properties (Fig. 2). (*ii*) Based on this importance analysis, we then propose a new training strategy (TIPS) to improve the accuracy of AnytimeNNs. (*iii*) Finally, we search for the AnytimeNNs under various hardware constraints. Next, we discuss these steps in detail.

3.1. Modeling AnytimeNNs as Markov Chains

3.1.1. Modeling AnytimeNNs as Graphs

As shown in Fig. 2(a), we model various DNN operations (convolutions, residual additions, *etc.*) as *edges*, and model the outputs of such operations (*i.e.*, featuremaps) as *nodes* in a graph. This way, a given architecture (supernet) is represented by a static undirected graph $G = (V, E)$, where V is the set of nodes (with $|V| = N$) and E is the set of edges between nodes (with $|E| = M$). For a given DNN architecture, its corresponding AnytimeNNs select suitable subnetworks under the current hardware constraints. Specifically, these *subnetworks* G_k are obtained by sampling edges from the initial supernet G:

$$G_k = (V, E_k), E_k \subseteq E \tag{1}$$

where, the node set V is the same for all subnetworks, but different subnetworks G_k have different edge sets E_k (see Fig. 2(a)). To ensure that the sampled subnetwork is valid, we always sample the input, output, and down-sample layers (*e.g.*, layers with pooling or stride=2). To ensure the validity of a subnetwork, we first randomly decide whether or not each layer remains in the subnetwork. For the remaining layers, we also ensure that #channels in consecutive layers match. As shown in Fig. 2(a), based on the topology of G_k, we can construct the adjacency matrix $\boldsymbol{A}_k \in R^{N \times N}$ for a subnetwork as follows:

$$\begin{cases} \boldsymbol{A}_k(s,t) = 0, & \text{if } (s,t) \notin E_k \\ \boldsymbol{A}_k(s,t) = 1, & \text{if } (s,t) \in E_k \\ \boldsymbol{A}_k(s,t) = 1, & \text{if } s = t = 1 \text{ or } s = t = N \end{cases} \tag{2}$$

where each edge (s, t) corresponds to an operation in the given network. The intuition behind the values of $\boldsymbol{A}_k(1, 1)$ and $\boldsymbol{A}_k(N, N)$ is that the computation always starts from the input/output layer in the forward/backward path. We note that our objective is to analyze the *layer-wise* gradient properties of AnytimeNNs. Since the singular values of each layer's Jacobian are designed to be around 1 by commonly used initialization schemes (*e.g.*, by maintaining uniform gradient variance at all layers), it is reasonable to assign '1' as the weight of each edge (*i.e.*, operation) in Eq. 2 if it appears in A_k. More details are given in Appendix C.

At each training step, we sample T subnetworks as shown in Fig. 2(a). Let $\mathcal{L}$ denote the loss function (*e.g.*, cross-entropy). Then, the loss for AnytimeNNs at each training step is calculated by passing the same batch of images through these T subnetworks (Huang et al., 2018; Hu et al., 2019):

$$Loss = \sum_{k=1}^{T} \mathcal{L}(y, G_k(x)) \tag{3}$$

where x, y, $G_k(x)$ are the input, ground truth, and output of subnetwork G_k, respectively. Eq. 3 shows that all these subnetworks in Fig. 2(a) share the same input data and use the accumulated loss from all of them to calculate the gradient during the backward propagation. Hence, all these subnetworks are highly coupled with each other. Inspired by the idea in (Taylor et al., 2019), as shown in Fig. 2(b), we integrate multiple subnetworks into a new hyper-graph to capture the coupling impacts among different subnetworks. Specifically, given a sequence of T subnetworks and each subnetwork with N nodes, we construct a *hyper-adjacency matrix* $\hat{\boldsymbol{H}}(\lambda) \in R^{NT \times NT}$:

$$\hat{\boldsymbol{H}}(\lambda) = \begin{bmatrix} \boldsymbol{A}_1 & \widetilde{\boldsymbol{Z}_{1,2}} & ... & \widetilde{\boldsymbol{Z}_{1,T}} \\ \widetilde{\boldsymbol{Z}_{2,1}} & \boldsymbol{A}_2 & ... & \widetilde{\boldsymbol{Z}_{2,T}} \\ ... & ... & ... & ... \\ \widetilde{\boldsymbol{Z}_{T,1}} & ... & ... & \boldsymbol{A}_T \end{bmatrix} \tag{4}$$

where $\widetilde{\boldsymbol{Z}_{i,j}} \in R^{N \times N}$ is the inter-subnetwork coupling matrix between *different* subnetworks G_i and G_j as follows:

$$\widetilde{\boldsymbol{Z}_{i,j}} = \lambda \boldsymbol{I},\ i \neq j\ ,\ 0 < \lambda \leq 1 \tag{5}$$

where $\boldsymbol{I}$ is the identity matrix.

Remark: On the one hand, $\boldsymbol{A}_k$ in $\hat{\boldsymbol{H}}(\lambda)$ capture the connectivity pattern of each individual subnetwork. On the other hand, as shown in Fig. 2(a)(b), $\widetilde{\boldsymbol{Z}_{i,j}}$ in $\hat{\boldsymbol{H}}(\lambda)$ captures the inter-subnetwork coupling effects between every pair of subnetworks by connecting same nodes across different subnetworks[1]. Hence, our methodology does capture both *intra-* and *inter-subnetwork* topological properties. This is crucial since AnytimeNNs have a variable network architecture. (see more discussion in Section 4.5 and Appendix B.4).

3.1.2. Building the DTMC for AnytimeNNs

In this work, we aim to identify the importance of each operation in AnytimeNNs. Inspired by the PageRank algorithm (Berkhin, 2005), we use the *hyper-adjacency matrix* $\hat{\boldsymbol{H}}(\lambda)$ to build the transition matrix $\boldsymbol{P}$ of our DTMC. Specifically, we normalize the adjacency matrix $\hat{\boldsymbol{H}}$ row by row:

$$\boldsymbol{P}_{m,:} = \hat{\boldsymbol{H}}_{m,:}(\lambda) / (\textstyle\sum_n \hat{\boldsymbol{H}}_{m,n}(\lambda)) \tag{6}$$

[1]The parameter λ controls the strength of the interactions between different subnetworks; see details in Sec 4.5.

and obtain an irreducible, aperiodic, and homogeneous DTMC, which has a unique stationary state distribution ($\boldsymbol{\pi}$) (Hajek, 2015)[2]. The stationary distribution $\boldsymbol{\pi}$ of such DTMC has the following property:

$$\boldsymbol{\pi}\boldsymbol{P} = \boldsymbol{\pi} \tag{7}$$

Hence, we can solve Eq. 7 to obtain $\boldsymbol{\pi}$ for our DTMC. Next, we use $\boldsymbol{\pi}$ to analyze the nodes in $\hat{\boldsymbol{H}}(\lambda)$. We denote $\boldsymbol{\pi}(s)$ as the stationary probability of a state s. Note that, as shown in Fig. 2(a), a node r appears in all the T sampled subnetworks, hence it appears T times in $\hat{\boldsymbol{H}}(\lambda)$; each node r from the supernet G corresponds to T nodes $\{r_k,\ k = 1, ..., T\}$ in the DTMC within T subnetworks $\{G_k,\ k = 1, ..., T\}$. For a given node r_k in the DTMC, we denote its stationary probability as $\boldsymbol{\pi}(s_{r_k})$.

3.2. Topological & Gradient Properties of AnytimeNNs

To analyze the importance of nodes and paths in AnytimeNNs, we propose the following definitions:

Definition 1. Topological Accumulated Score (TAS) A *topological accumulated score* of a node r from the supernet is its accumulated PageRank score across multiple subnetworks. For a given node r in V, its TAS value μ_r is:

$$\mu_r = \sum_{k=1}^{T} \boldsymbol{\pi}(s_{r_k}) \tag{8}$$

TAS quantifies the *accumulated probability* that a node is selected within an AnytimeNN. Next, we use TAS to analyze the importance of various computation paths.

Definition 2. Topological Path Score (TPS) In an AnytimeNN, we define a computation path l from a node r to a node d, as a sequence of edges $\{r \to v_1 \to ... \to v_w \to d\}$. The *topological path score* TPS_l of a computation path l is the sum of the TAS values of *all* nodes traversed in the path:

$$TPS_l = \sum_{s \in \{r, v_1, ..., v_w, d\}} \mu_s \tag{9}$$

The above definitions and the LDI discussion in Section 2 enable us to propose our main result:

Proposition 3.1. *Consider an AnytimeNN initialized by a zero-mean i.i.d. distribution with variance q. Given two computation paths l_S and l_L in this AnytimeNN with same width w_r and number of nodes D, we define w_e^S (w_e^L) as the average degree of l_S (l_L). Assuming $q \leq \epsilon$, $w_e^S \gg w_r$, and $w_e^L \gg w_r$, then the mean singular values $E[\sigma^S]$ and $E[\sigma^L]$ of the Jacobian matrix for l_S and l_L satisfy:*

$$\textit{if } TPS_{l_S} \leq TPS_{l_L}, \textit{ then } E[\sigma^S] \leq E[\sigma^L] \leq 1 \tag{10}$$

[2]More details are given in Appendix B.1

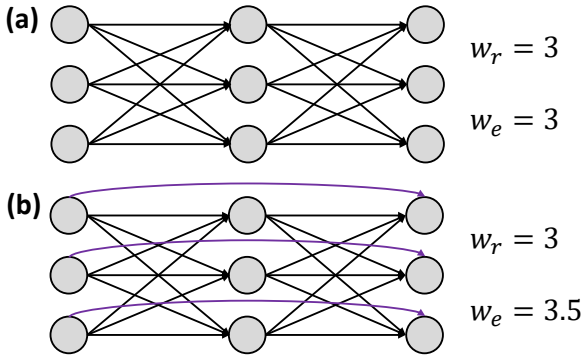


Figure 3. A 2-layer MLP has three neurons per layer; the right version includes skip connections (purple) across layers, while the left does not. Both MLPs have a real width w_r of 3. The average degree w_e is calculated as the total links (weights and skip connections) divided by total neurons (excluding input neurons). The upper MLP in **(a)** has a w_e of $18/6 = 3$, and the lower in **(b)** has a w_e of $21/6 = 3.5$ due to skip connections.

where, $\epsilon = \frac{1}{max(w_e^S, w_e^L) + w_r + 2\sqrt{max(w_e^S, w_e^L) w_r}}$. *That is, the mean singular value of the Jacobian for the computation path with higher TPS values is higher and closer to 1.*

Proof. Authors in (Bhardwaj et al., 2021) prove that for a given neural network initialized by a zero-mean i.i.d. distribution with variance q, the mean singular value $E[\sigma]$ of the Jacobian matrix from the network is bounded by the following inequality:

$$\sqrt{qw_e} - \sqrt{qw_r} \leq E[\sigma] \leq \sqrt{qw_e} + \sqrt{qw_r} \tag{11}$$

where the w_e is the *average node degree* or *effective width* and w_r is the real width of the neural network. In Fig. 3, we give an example of how w_e and w_r are calculated in a neural network.

We now use the above bounds to prove our main result. Let us first prove the right side of the inequality in Proposition 3.1. According to Eq. 11, for two computational paths l_S and l_L, the mean singular values $E[\sigma^S]$ and $E[\sigma^L]$ of their Jacobian matrices are bounded by the following inequalities:

$$\begin{aligned} \sqrt{qw_e^S} - \sqrt{qw_r} \leq E[\sigma^S] \leq \sqrt{qw_e^S} + \sqrt{qw_r} \\ \sqrt{qw_e^L} - \sqrt{qw_r} \leq E[\sigma^L] \leq \sqrt{qw_e^L} + \sqrt{qw_r} \end{aligned} \tag{12}$$

Based on Eq. 12, we note that if initialization variance q satisfies

$$q \leq \frac{1}{max(w_e^L, w_e^S) + w_r + 2\sqrt{max(w_e^L, w_e^S) w_r}} \tag{13}$$

then, the mean singular value is always bounded by 1 for both l_S and l_L; that is:

$$E[\sigma^S] \leq 1 \qquad E[\sigma^L] \leq 1 \tag{14}$$

Inequality 14 proves the right side of inequality in Proposition 3.1. Next, we prove the left side.

Using Eq. 12, if $w_e^S \gg w_r$ and $w_e^L \gg w_r$, then the mean singular values of l_L and l_S are mainly determined by w_e^L and w_e^S; that is:

$$E[\sigma^L] = \sqrt{qw_e^L} \qquad E[\sigma^S] = \sqrt{qw_e^S} \tag{15}$$

From Definition 1, we know that TAS of a given node is the sum of its PageRank across the T subnetworks. As discussed in (Fortunato et al., 2006), under the mean-field approximation, the PageRank of a given node is linearly correlated to its node degree. That is, for the ith node i on the computation path:

$$\mu_i = \frac{k_i}{C} \tag{16}$$

where k_i is the node degree for node i, and C is a constant determined by the topology of supernet[3]. Because both l_S and l_L are sampled from the same supernet, then they share the same value of C. Combining Eq. 16 with Definition 2, the TPS satisfies the following relation:

$$TPS = \sum_{i=1}^{D} \mu_i = \frac{\sum_{i=1}^{D} k_i}{C} \tag{17}$$

Given the definition of average degree w_e, we rewrite Eq. 17 as follows:

$$TPS = \frac{D \times w_e}{C} \implies w_e = \frac{C \times TPS}{D} \tag{18}$$

where D is the number of nodes for a given path. By combining Eq. 15 and Eq. 18, the mean singular value is determined by q, C, D, and TPS:

$$w_e = \frac{C \times TPS}{D} \implies E[\sigma] = \sqrt{\frac{q \times C \times TPS}{D}} \tag{19}$$

Note that q, C, D have the same values for both f_S and f_L. Hence, if $TPS_S \leq TPS_L$, then $E[\sigma^S] \leq E[\sigma^L]$. This proves the left side of inequality in Proposition 3.1.

Therefore, the inequality in Proposition 3.1 holds true for both the left and right sides. That is, the mean singular value of the Jacobian for a computation path with a higher TPS is higher and closer to 1. Moreover, the closeness of $E[\sigma]$ to 1 is determined by the initialization variance q, constant C, the values of TPS_S and TPS_L, and #nodes D. This completes our proof of Proposition 3.1. □

Intuitively, Proposition 3.1 says that the computation paths with high TPS values satisfy the LDI property and the gradient magnitude through such paths would not vanish or

[3]A supernet is the given neural architecture that needs to be converted into its AnytimeNN version.

explode, thus, having a higher impact on AnytimeNNs training. We provide empirical results to verify this in the experiments section.

Algorithm 1 Pareto-optimal subnetwork search

Input: Supernet G, search steps m
Output: Pareto-optimal subnetworks set G_P
Search:
Initialize $G_P = \phi$
for $i = 1$ **to** m **do**
  Sample subnetwork G_i from G
  Evaluate G_i and get its accuracy Θ_{G_i}
  $optimal = TRUE$
  Initialize false-Pareto Set $G_{P_{out}} = \phi$
  for G_j in G_P **do**
    if $FLOPs_{G_j} \leq FLOPs_{G_i}$ **and** $\Theta_{G_j} > \Theta_{G_i}$ **then**
      $optimal = FALSE$
    else if $FLOPs_{G_j} \leq FLOPs_{G_i}$ **and** $\Theta_{G_j} < \Theta_{G_i}$ **then**
      Add G_j to $G_{P_{out}}$
    end if
  end for
  if $optimal$ **then**
    Add G_i to G_P
  end if
  Remove false Pareto-optimal $G_P = G_P \setminus G_{P_{out}}$
end for

3.3. Topologically Important Path Sampling (TIPS)

Among the computation paths with the same number of nodes of an AnytimeNN, we define the operations (*i.e.*, edges) along the path with the highest TPS value as *important operations*; the rest of operations are deemed as *unimportant operations* (see Path2 in Fig. 2(c)). According to Proposition 3.1, the path with higher TPS values has higher singular values of the Jacobian matrix. Hence, these important operations have a significant impact on the training process. Note that, previous works treat all operations uniformly (Chin et al., 2021; Wang et al., 2018). Instead, in our approach, we modify the sampling process during the training process and use a higher sampling probability to sample these important operations. We call this sampling strategy ***Topologically Important Path Sampling*** (***TIPS***). More details are given in the experiments section.

3.4. Pareto-Optimal Subnetwork Search

After the TIPS-based training, we use the Algorithm 1 to search for the Pareto-optimal subnetworks under various hardware constraints. To this end, we consider the number of floating-point operations (FLOPs) as a proxy for hardware resource constraints[4]. At runtime, one can select the proper subnetworks to quickly adapt to various hardware constraints; *e.g.*, if the amount of currently available memory for a device drops below a threshold, we switch to a smaller subnetwork to meet the new memory budget.

[4]In practice, this can be easily replaced by some other hardware resource, such as memory or power consumption.

3.5. Summary of Our Approach

In brief, our method consists of the following steps:

- **Step 1: TPS analysis** (Fig. 2) We sample subnetworks and exploit TPS (our DTMC-based metric for AnytimeNNs) to identify important operations.
- **Step 2: AnytimeNN training** We use TIPS by assigning a higher sampling probability to the important operations (as given by TPS) to *train* AnytimeNNs.
- **Step 3: Pareto-optimal search** Before model deployment, we do an *offline* search under *various* hardware constraints. We store the full supernet and #channel configurations of the obtained subnetworks.

Remarks: Our framework involves two steps where we perform sampling. In Step 1, we conduct the TAS and TPS analysis without knowing the importance of each operation. Hence, to build the DTMC with the sampled subnetworks (Section 3.1), we uniformly sample these operations and ensure each operation is selected at least once during this stage. Once we compute the TAS and TPS values, we can identify the important and unimportant operations in a given supernet (Sections 3.2 and 3.3). During Step 2 (i.e., AnytimeNN training), operations are not sampled uniformly. Instead, important operations are sampled with a higher probability compared to unimportant operations. We provide more details in Section 4.3.

After Steps 1-3, *at runtime*, we use the best subnetwork configurations under various budgets. We provide the storage overhead and time efficiency analysis in Appendix B.8.

In terms of **time cost**: Step 1 takes only 39 seconds on a Xeon CPU for MobileNet-v2. Step 2 takes 97 hours on an 8-GPU server for 150 epochs, and Step 3 takes 8 minutes on an RTX3090 GPU. We note that Step 1 only has negligible time costs compared to AnytimeNN training. Moreover, Steps 1-3 are conducted offline and, hence, they result in zero overhead for the online inference.

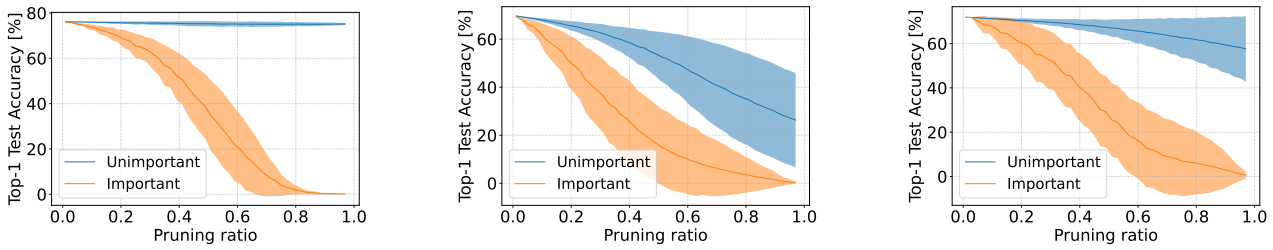
4. Experimental Results

4.1. Experimental Setup

In this section, we present the following experiments: (*i*) Verification of Proposition 3.1, (*ii*) Validation of TIPS, and (*iii*) Model complexity vs. accuracy results.

For the experiments (*i*) on Proposition 3.1, we build several MLP-based supernets on MNIST dataset by stacking several linear layers with 80 neurons (and adding residual connections between each two consecutive layers), then verify our Proposition 3.1 on these supernets.

For the experiments (*ii*) on TIPS, we use TPS and TAS to identify the importance of various operations in several networks (MobileNet-v2, ResNet, WideResNet, ResNext, and



(a) EfficientNet-B0 (b) ResNet18 (c) MobileNet-v2

Figure 4. Pruning ratio of important and unimportant operations (as identified by TAS/TPS) vs. mean test accuracy on ImageNet (std. dev. shown with shade) on EfficientNet-B0, ResNet18 and MobileNet-v2. More results are given in Fig. 8 in Appendix A.

EfficientNet) for the ImageNet dataset. We also present the comparisons between the training convergence for our proposed TIPS strategy and the previous SOTA methods (Chin et al., 2021; Yu & Huang, 2019b) with the exact same setup (*i.e.*, same data augmentation, optimizer, and learning rate schedule). More training details are given in Appendix B.2.

Finally, for experiments (*iii*), we take the MobileNet-v2 and ResNet34 trained with TIPS as supernets, and then search for Pareto-optimal subnetworks. We compare the accuracy-FLOPs tradeoffs of the obtained subnetworks with various training strategies.

4.2. Verification of Proposition 3.1

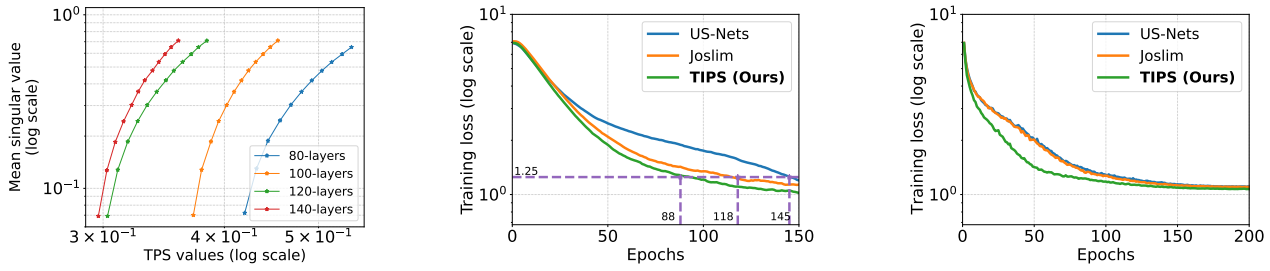
To empirically validate Proposition 3.1, we consider several supernets with 80, 100, 120, and 140 layers (along with residual connections). We then randomly sample 8 subnetworks from these supernets and use Eq. 2, Eq. 4, and Eq. 6 to build the DTMC. After solving Eq. 7, we calculate the TAS for each node (*i.e.*, output of various operations). Next, we set the path length to 50 as an example, then randomly sample multiple computation paths with 50 nodes from these supernets and calculate the corresponding {TPS, mean singular value} pairs. As shown in Fig. 5(a), for a supernet with specific depth (*e.g.*, 80 layers), higher TPS values always lead to higher mean singular values (closer to 1). These results empirically validate our Proposition 3.1.

4.3. Validation of TIPS

In order to verify the effectiveness of our topological analysis, we first explore the relationship between the *important operations* and the test accuracy for various networks. To this end, before training, we first use our DTMC based framework to obtain the TAS for each node (Eq. 8). Next, among all the computation paths from input to output in the supernet, we find the path that has the highest TPS value (Eq. 9); we mark all operations along this path as important operations. Then, we prune the output channels of each operation *individually* (with pruning ratios ranging from 1% to 99%), without pruning the channels of any other operation in the network. Meanwhile, we measure the test accuracy of the network after each pruning step. This way, we can analyze the impact of each operation on the test accuracy of a given network. Note that, we prune the last channels first. For example, to prune a convolution layer with 64 (0-63) channels with a pruning ratio of 75%, we directly set the output of the last 75% (16-63) channels to zero. As shown in Fig. 4, for various pruning ratios, pruning the important operations has a much higher accuracy drop than the unimportant ones. These experimental results show that the important operations found by our framework have a significant impact on the test accuracy of AnytimeNNs. Therefore, the proposed TAS and TPS metrics clearly identify the important operations and computation paths in AnytimeNNs.

Next, we evaluate the impact of TIPS on *training convergence* of the AnytimeNNs. For this experiment, we train MobileNet-v2 with width-multiplier 1.4 on ImageNet dataset with (*i*) SOTA training strategies: Joslim, US-Nets, and DS-Net (Chin et al., 2021; Yu & Huang, 2019b; Li et al., 2021a), and (*ii*) our proposed TIPS. As explained in Section 3.2, a higher sampling probability for important operations helps more with the training process. However, if the sampling probability for important operations is too high, it hurts the diversity of sampled subnetworks. In the extreme case, we always end up sampling and training only the important operations, while the unimportant ones never get sampled and trained; this can hurt the test accuracy of AnytimeNNs. Hence, for our proposed TIPS strategy, we use a 50% higher sampling probability for important operations compared to unimportant operations. For example, if 40% of the output channels of unimportant operations are sampled, then 60% of the output channels of important operations are sampled (since $40\% \times (1+0.5) = 60\%$).

We note that previous methods (*e.g.*, Joslim and US-Nets) use a uniform sampling for every subnetwork, *i.e.*, the same sampling probability for all operations during the training process. In contrast, TIPS focuses more on important operations thus improving the LDI properties. As shown in



(a) Mean singular values vs. TPS (b) MBN-v2 on ImageNet (c) MBN-v2 on Tiny-ImageNet

Figure 5. **(a)** Mean singular values ($E[\sigma]$) vs. TPS for various supernets (MLPs with various #layers) on MNIST. Clearly, paths with higher TPS values have higher $E[\sigma]$ for a specific supernet (*e.g.*, 80-layers). **(b, c)** Training loss of MobileNet-v2 (MBN-v2) based supernet vs. #Epochs on ImageNet and Tiny-ImageNet. TIPS requires much fewer epochs to achieve a target training loss. For example, on ImageNet, to make the training loss less than 1.25 on ImageNet, US-Nets takes 145 epochs while TIPS only needs 88 epochs.

Table 1. Comparison of Top-1 test accuracy vs. FLOPS (Million [M]) with SOTA training methods on MobileNet-v2. The best results are shown with bold fonts. The results are averaged over three runs. The std. dev. values are given in Table 4 in Appendix B.3.

CIFAR100						
FLOPS	20M	30M	35M	40M	45M	50M
US-Nets	61.5	62.9	64.8	65.5	65.6	66.5
Joslim	62.0	62.7	63.1	63.7	64.1	65.0
DS-Net	61.8	63.8	64.8	65.3	65.5	66.7
TIPS (Ours)	**66.4**	**66.9**	**67.0**	**67.6**	**67.7**	**68.2**
Tiny-ImageNet						
FLOPS	80M	120M	140M	160M	180M	200M
US-Nets	47.0	47.3	48.3	49.0	50.2	51.4
Joslim	47.4	47.9	48.7	49.5	50.3	50.7
DS-Net	46.9	47.4	48.1	48.7	50.3	50.8
TIPS (Ours)	**53.5**	**53.8**	**54.0**	**54.4**	**54.9**	**55.1**
ImageNet						
FLOPS	260M	320M	400M	450M	500M	600M
US-Nets	70.6	71.6	71.8	72.1	72.3	72.9
Joslim	70.8	71.9	72.5	72.7	72.9	73.4
DS-Net	70.6	72.1	72.5	72.6	73.0	73.3
TIPS (Ours)	**71.8**	**73.2**	**73.6**	**74.0**	**74.3**	**74.7**

Table 2. Comparison of Top-1 test accuracy vs. FLOPS (Million/Giga [M/G]) with SOTA training methods on ResNet34. The best results are shown with bold fonts. The results are averaged over three runs. The std. dev. values are shown in Table 5 in Appendix B.3.

CIFAR100						
FLOPS	120M	180M	200M	220M	240M	260M
US-Nets	63.1	63.9	64.4	64.8	65.0	65.4
Joslim	65.8	66.2	66.7	67.0	67.3	67.4
DS-Net	64.4	65.9	66.2	66.4	66.5	66.6
TIPS (Ours)	**67.3**	**67.4**	**67.8**	**67.9**	**68.1**	**68.2**
Tiny-ImageNet						
FLOPS	130M	190M	220M	250M	270M	300M
US-Nets	42.9	43.2	44.3	44.7	44.9	45.2
Joslim	44.9	**45.0**	45.3	45.4	45.5	45.8
DS-Net	41.8	43.0	43.8	43.9	44.1	44.2
TIPS (Ours)	**44.1**	44.6	**45.4**	**45.8**	**45.9**	**46.0**
ImageNet						
FLOPS	1.5G	2.2G	2.8G	3.0G	3.2G	3.6G
US-Nets	67.8	69.2	69.7	70.1	70.2	70.5
Joslim	68.0	**69.4**	69.6	70.0	70.2	70.4
DS-Net	66.0	67.0	68.8	69.4	69.9	70.0
TIPS (Ours)	**68.4**	69.3	**70.8**	**71.1**	**71.4**	**71.9**

Fig. 5(b,c), by changing the sampling strategy, our TIPS based-training achieves a much faster training convergence for the supernet compared to Joslim and US-Nets. Hence, this validates that TIPS results in better trainability of the supernet.

4.4. Pareto-Optimal AnytimeNN Search

We use the Algorithm 1 to search for Pareto-optimal subnetworks under various hardware constraints for MobileNet-v2 and ResNet34. After the search, we evaluate the obtained Pareto-optimal subnetworks and get their real test accuracy.

Table 1 demonstrates that our proposed TIPS achieves significantly higher accuracy than SOTA given similar FLOPs for ImageNet on MobileNet-v2. For example, assuming the hardware constraint is 500M FLOPs, TIPS has a 1.4%-2% higher accuracy on ImageNet than the SOTA; on Tiny-ImageNet with an 80M FLOPs budget, TIPS has 6.1%-6.6% higher accuracy than SOTA.

Moreover, as shown in Table 2, given the ResNet34 supernet with a 3.6G FLOPs budget on ImageNet, TIPS achieves 1.4%, 1.5%, and 1.9% higher test accuracy than Joslim, US-Nets and DS-Net, respectively (Chin et al., 2021; Yu & Huang, 2019b; Li et al., 2021a). On CIFAR100 dataset with a 120M FLOPs budget, TIPS has 1.5%, 2.9%, and 4.2% higher accuracy Joslim, US-Nets and DS-Net, respectively.

Table 3. Top-1 test accuracy vs. latency of MobileNet-v2 on ImageNet for RaspberryPi-3B+. The results are averaged over three runs.

Method	Metric	Results				
Joslim	Latency (ms):	176	232	305	341	406
	Accuracy (%)	70.8	71.9	72.5	72.9	73.4
TIPS (Ours)	Latency (ms):	190	245	298	362	413
	Accuracy (%)	71.8	73.2	73.6	74.3	74.7

Latency vs. Accuracy Trade-off Besides FLOPs vs. accuracy, we also compare the latency vs. accuracy tradeoff of subnetworks obtained by TIPS and Joslim. As shown in Table 3, TIPS achieves higher accuracy than Joslim, given a similar latency. For example, assuming the latency constraint is around 300ms, TIPS has a 1.1% higher accuracy on ImageNet than the Joslim.

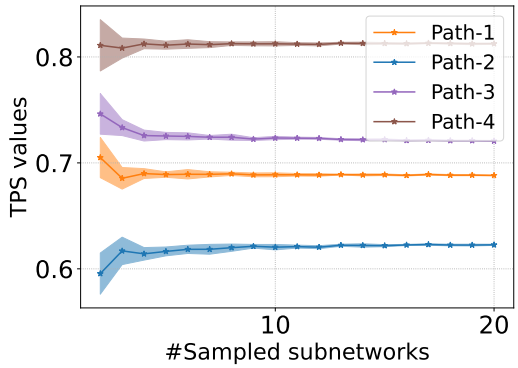


Figure 6. Stability of TPS values vs. #subnetworks over 10 runs (std. dev. shown with shade) for MobileNet-v2. The variation is negligible when #subnetworks is larger than 4.

4.5. Ablation Studies

Stability of TPS analysis We vary the #sampled subnetworks from 2 to 20 for MobileNet-v2. As shown in Fig. 6, four subnetworks are typically enough to make the TPS value converge. In practice, we sample 8 subnetworks and the standard deviation of TPS values is less than 2.5% of the mean values.

Effect of λ Finally, we fix the #sampled subnetworks to 8 and vary the λ value in the hyper-adjacency matrix (Eq. 4) from 0.1 to 1 for MobileNet-v2. As shown in Fig. 7, the ranking among different paths remains the same under various λ values. Therefore, our approach is robust to λ values variation. In our approach, we set the value of λ to '1'. We discuss this in Appendix B.4.

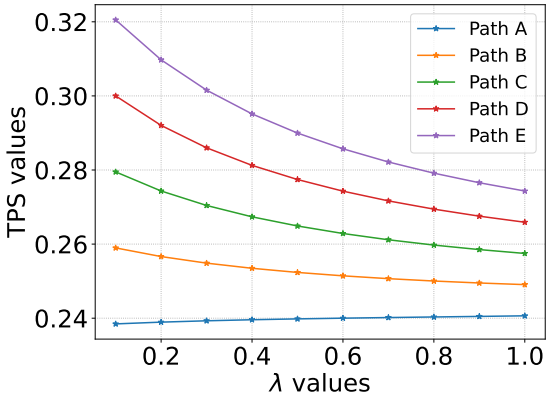


Figure 7. TPS values vs. λ values for MobileNet-v2. The ranking among different paths remains the same for various λ values.

4.6. Limitations and Future Work

Our current framework (TIPS) has been primarily verified on CNNs with variable width and depth. We plan to explore it with other AnytimeNNs (*e.g.*, multi-branch and early-exit networks) and other types of networks (*e.g.*, transformers and graph neural networks). Also, in the current version, the hardware constraints are considered *after* the supernet training; we intend to consider incorporating hardware awareness into the training process as well.

5. Conclusion

In this work, we have proposed a new methodology to *automatically design the AnytimeNNs* under various hardware budgets. To this end, by modeling the training process of AnytimeNNs as a DTMC, we have proposed two metrics – TAS and TPS – to characterize the important operations in AnytimeNNs. We have shown that these important operations and computation paths significantly impact the accuracy of AnytimeNNs. Based on this, we have proposed a new training method called *TIPS*. Experimental results show that TIPS has a faster training convergence speed than SOTA training methods for anytime inference. Our experimental results demonstrate that our framework can achieve SOTA accuracy-FLOPs trade-offs, while achieving 2%-6.6% accuracy improvements on CIFAR100, Tiny-ImageNet and ImageNet datasets compared to existing approaches for anytime inference.

Acknowledgement

This work was supported in part by the US National Science Foundation (NSF) grants CNS-2007284 and CCF-2107085.

References

Bejnordi, B. E., Blankevoort, T., and Welling, M. Batch-shaping for learning conditional channel gated networks. In *International Conference on Learning Representations (ICLR)*, 2020.

Bengio, E., Bacon, P., Pineau, J., and Precup, D. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*, 2015.

Berkhin, P. A survey on pagerank computing. *Internet mathematics*, 2(1):73–120, 2005.

Bhardwaj, K., Li, G., and Marculescu, R. How does topology influence gradient propagation and model performance of deep networks with densenet-type skip connections? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

Bolukbasi, T., Wang, J., Dekel, O., and Saligrama, V. Adaptive neural networks for efficient inference. In *Proceedings of International Conference on Machine Learning (ICML)*, 2017.

Burkholz, R. and Dubatovka, A. Initialization of relus for dynamical isometry. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Cai, H., Gan, C., Wang, T., Zhang, Z., and Han, S. Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations (ICLR)*, 2020.

Cai, S., Shu, Y., and Wang, W. Dynamic routing networks. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2021.

Chen, W., Huang, W., Gong, X., Hanin, B., and Wang, Z. Deep architecture connectivity matters for its convergence: A fine-grained analysis. *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Chen, Z., Li, Y., Bengio, S., and Si, S. You look twice: Gaternet for dynamic filter selection in cnns. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

Chin, T.-W., Morcos, A. S., and Marculescu, D. Joslim: Joint widths and weights optimization for slimmable neural networks. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD*, 2021.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.

Fortunato, S., Boguñá, M., Flammini, A., and Menczer, F. Approximating pagerank from in-degree. In *International workshop on algorithms and models for the web-graph*, pp. 59–71. Springer, 2006.

Gao, X., Zhao, Y., Dudziak, L., Mullins, R. D., and Xu, C. Dynamic channel pruning: Feature boosting and suppression. In *International Conference on Learning Representations (ICLR)*, 2019.

Hajek, B. *Random processes for engineers*. Cambridge university press, 2015.

Han, S., Pool, J., Tran, J., and Dally, W. J. Learning both weights and connections for efficient neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations (ICLR)*, 2016.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Hu, H., Dey, D., Hebert, M., and Bagnell, J. A. Learning anytime predictions in neural networks via adaptive loss balancing. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

Hua, W., Zhou, Y., Sa, C. D., Zhang, Z., and Suh, G. E. Channel gating neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Huang, G., Chen, D., Li, T., Wu, F., van der Maaten, L., and Weinberger, K. Q. Multi-scale dense networks for resource efficient image classification. In *International Conference on Learning Representations (ICLR)*, 2018.

Javaheripi, M., Rouhani, B. D., and Koushanfar, F. Swann: Small-world architecture for fast convergence of neural networks. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 11(4):575–585, 2021.

Larsson, G., Maire, M., and Shakhnarovich, G. Fractalnet: Ultra-deep neural networks without residuals. In *International Conference on Learning Representations (ICLR)*, 2017.

Lee, H. and Shin, J. Anytime neural prediction via slicing networks vertically. *arXiv preprint arXiv:1807.02609*, 2018.

Lee, N., Ajanthan, T., Gould, S., and Torr, P. H. S. A signal propagation perspective for pruning neural networks at initialization. In *International Conference on Learning Representations (ICLR)*, 2020.

Li, C., Wang, G., Wang, B., Liang, X., Li, Z., and Chang, X. Dynamic slimmable network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021a.

Li, G., Mandal, S. K., Ogras, U. Y., and Marculescu, R. Flash: Fast neural architecture search with hardware optimization. *ACM Transactions on Embedded Computing Systems (TECS)*, 20(5s):1–26, 2021b.

Li, G., Yang, Y., Bhardwaj, K., and Marculescu, R. Zico: Zero-shot NAS via inverse coefficient of variation on gradients. In *International Conference on Learning Representations (ICLR)*, 2023.

Li, H., Zhang, H., Qi, X., Yang, R., and Huang, G. Improved techniques for training adaptive deep networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

Li, Y., Hao, C., Zhang, X., Liu, X., Chen, Y., Xiong, J., Hwu, W.-m., and Chen, D. Edd: Efficient differentiable dnn architecture and implementation co-search for embedded ai solutions. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6. IEEE, 2020.

Liu, H., Simonyan, K., and Yang, Y. DARTS: differentiable architecture search. In *International Conference on Learning Representations (ICLR)*, 2019.

Liu, L., Deng, L., Chen, Z., Wang, Y., Li, S., Zhang, J., Yang, Y., Gu, Z., Ding, Y., and Xie, Y. Boosting deep neural network efficiency with dual-module inference. In *Proceedings of International Conference on Machine Learning (ICML)*, 2020.

Luo, J.-H., Wu, J., and Lin, W. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2017.

Qin, H., Gong, R., Liu, X., Bai, X., Song, J., and Sebe, N. Binary neural networks: A survey. *Pattern Recognition*, 105:107281, 2020.

Ruiz, A. and Verbeek, J. Anytime inference with distilled hierarchical neural ensembles. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Saxe, A. M., McClelland, J. L., and Ganguli, S. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *International Conference on Learning Representations (ICLR)*, 2014.

Stamoulis, D., Ding, R., Wang, D., Lymberopoulos, D., Priyantha, B., Liu, J., and Marculescu, D. Single-path NAS: designing hardware-efficient convnets in less than 4 hours. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD*, 2019.

Tang, Y., Wang, Y., Xu, Y., Deng, Y., Xu, C., Tao, D., and Xu, C. Manifold regularized dynamic network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

Taylor, D., Porter, M. A., and Mucha, P. J. Supracentrality analysis of temporal networks with directed interlayer coupling. In *Temporal Network Theory*, pp. 325–344. Springer, 2019.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Veit, A. and Belongie, S. Convolutional networks with adaptive inference graphs. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

Veit, A., Wilber, M. J., and Belongie, S. Residual networks behave like ensembles of relatively shallow networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

Wang, L., Dong, X., Wang, Y., Ying, X., Lin, Z., An, W., and Guo, Y. Exploring sparsity in image super-resolution for efficient inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

Wang, X., Yu, F., Dou, Z.-Y., Darrell, T., and Gonzalez, J. E. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

Wang, Y., Shen, J., Hu, T.-K., Xu, P., Nguyen, T., Baraniuk, R., Wang, Z., and Lin, Y. Dual dynamic inference: Enabling more efficient, adaptive, and controllable deep inference. *IEEE Journal of Selected Topics in Signal Processing*, 14(4):623–633, 2020.

Wu, Z., Nagarajan, T., Kumar, A., Rennie, S., Davis, L. S., Grauman, K., and Feris, R. S. Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Xiao, L., Bahri, Y., Sohl-Dickstein, J., Schoenholz, S., and Pennington, J. Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks. In *Proceedings of International Conference on Machine Learning (ICML)*, 2018.

Yang, T.-J., Howard, A., Chen, B., Zhang, X., Go, A., Sandler, M., Sze, V., and Adam, H. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

Yang, Y., Xue, Z., and Marculescu, R. Anytime depth estimation with limited sensing and computation capabilities on mobile devices. In *Conference on Robot Learning*, pp. 609–618. PMLR, 2021.

Yu, J. and Huang, T. Autoslim: Towards one-shot architecture search for channel numbers. *arXiv preprint arXiv:1903.11728*, 2019a.

Yu, J. and Huang, T. S. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019b.

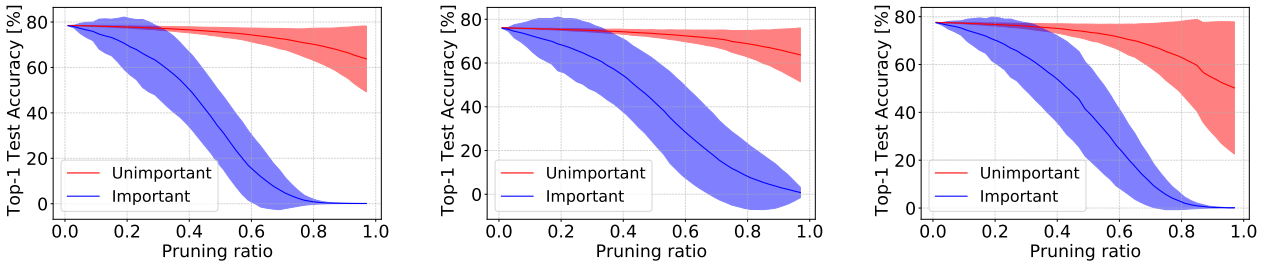
Yu, J., Yang, L., Xu, N., Yang, J., and Huang, T. Slimmable neural networks. In *International Conference on Learning Representations (ICLR)*, 2019.

Yuan, Z., Wu, B., Sun, G., Liang, Z., Zhao, S., and Bi, W. S2DNAS: transforming static CNN model for dynamic inference via neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2017.

A. Supplementary Results for Importance Analysis

The plots below (Fig. 8) supplement the results in Fig. 4 in the main paper. As shown in Fig. 8, for various pruning ratios, pruning the important operations has a much higher accuracy drop than the unimportant ones. These experimental results show that the important operations found by our framework have a significant impact on the test accuracy of AnytimeNNs.



(a) WideResNet50-2 (b) ResNet50 (c) ResNext50-32x4d

Figure 8. Pruning ratio of important and unimportant operations vs. mean test accuracy on ImageNet (standard deviations are drawn with shade). We prune the output of each operation with various pruning ratios and obtain the test accuracy. We calculate the mean accuracy under various pruning ratios for important operations and unimportant operations. As shown, for all networks, given the same pruning ratio, important operations have a much higher impact on accuracy than unimportant ones.

B. Details of the Training Methods

B.1. Construction of DTMC for AnytimeNNs

An irreducible, aperiodic, and homogeneous DTMC has a unique state stationary distribution ($\boldsymbol{\pi}$) (Hajek, 2015). We analyze the above three requirements (irreducibility, aperiodicity, and homogeneity) for our problem as follows:

Irreducibility To ensure the constructed DTMC is irreducible, and following a similar idea from PageRank (Berkhin, 2005), we add a small transition probability κ between each pair of states to the original $\hat{\boldsymbol{H}}(\lambda)$ (Eq. 4 in the main paper) as follows:

$$\tilde{\boldsymbol{H}}(\lambda) = (1-\kappa)\hat{\boldsymbol{H}}(\lambda) + \kappa \boldsymbol{U} \tag{20}$$

where $\boldsymbol{U}$ is a all-one matrix with all elements equal to '1'. Next, we use the slightly modified $\tilde{\boldsymbol{H}}(\lambda)$ to construct the DTMC. Hence, the introduced transition probability κ guarantees that every two states in the DTMC are accessible to each other with a probability at least κ. As such, we ensure that the DTMC constructed by $\tilde{\boldsymbol{H}}(\lambda)$ is always irreducible. In practice, we set the value of κ very small (*e.g.*, $\kappa = 10^{-5}$) to minimize the impact of the introduced transition probability.

Aperiodicity According to (Hajek, 2015), for a given DTMC, if it is irreducible and there exist some self-loop transition among its states, then the DTMC is a aperiodic DTMC. (*i*) The modified $\tilde{\boldsymbol{H}}(\lambda)$ in the above discussion already ensures the DTMC is irreducible. (*ii*) Recall that in Eq. 2 (in the main paper), when we build the adjacency matrix $\boldsymbol{A}_k$, we set values of $\boldsymbol{A}_k(1,1)$ and $\boldsymbol{A}_k(N,N)$ as '1'. Hence, there are self-loops for the first and last states (*i.e.*, nodes) of each sub-networks. These two conditions ensure the constructed DTMC is also aperiodic.

Homogeneity For a given DTMC, if the probabilities of state transitions are independent of time, then the DTMC is a homogeneous DTMC. In our case, the probability of state transition is determined by the sampling strategy. Intuitively, if the sampling strategy remains the same over time, then the probabilities of state transitions are the same for different time moments. Hence, in this work, it is reasonable to assume that the constructed DTMC is homogeneous as well.

In summary, by ensuring the irreducibility, aperiodicity and the assumption of homogeneity of our constructed DTMC, we can always find the stationary state distribution $\boldsymbol{\pi}$ and use it to conduct the TAS and TPS analysis.

B.2. Training Hyperparameters

We use the SGD with a momentum of 0.9 as the optimizer and set the initial learning rate as 0.04. We set the batch-size as 512 and train the MobileNet-v2 for 150 epochs with a cosine annealing learning rate schedule on ImageNet dataset. We

train the ResNet-34 for 90 epochs with the same optimizer, batch-size, and learning rate schedule on ImageNet dataset. When we train the MobileNet-v2 and ResNet-34 on CIFAR100 and Tiny-ImageNet datasets, we use the same optimizer; we reduce the batch-size to 256 and train these networks for 200 epochs with the initial learning rate as 0.08 and a cosine annealing learning rate schedule.

Loss function For each training step, we randomly sample three sub-networks $G_k,\ k = 1, 2, 3$. In practice, to further increase the diversity of sub-networks, we conduct the sampling process at a finer level of granularity, *i.e.*, at *channel-level*. For example, in MobileNet-v2, we found that the layers within the block with stride=2 are important operations. Consequently, for each sub-network, we sample each channel with a 50% higher probability for these important operations compared to the channels that correspond to the unimportant operations.

Overall, we use the cross entropy loss together with the knowledge distillation function to train the AnytimeNNs for all these baseline methods and TIPS. For the same batch of input images, we combine these three subnetworks $G_k,\ k = 1, 2, 3$ as well as the entire supernet G, as follows:

$$Loss = \sum_{Net \in \{G, G_1, G_2, G_3\}} \mathcal{L}_{\mathcal{CE}}(y, Net(x)) + \sum_{Net \in \{G_1, G_2, G_3\}} \mathcal{L}_{\mathcal{KD}}(Net(x), G(k)) \tag{21}$$

where x, y, $\mathcal{L}_{\mathcal{CE}}$ and $\mathcal{L}_{\mathcal{KD}}$ are the input batch of images, labels, cross-entropy loss function and distillation function, respectively. In our work, the distillation function $\mathcal{L}_{\mathcal{KD}}$ is the same as the one used in (Chin et al., 2021).

Table 4. Comparison of Top-1 test accuracy vs. FLOPS (in millions [M]) with SOTA training methods on MobileNet-v2. Best results are shown with bold fonts. Results are averaged over three runs.

	FLOPS	20M	30M	35M	40M	45M	50M
CIFAR100	US-Nets (Yu & Huang, 2019b)	61.5±0.4	62.9±0.6	64.8±0.3	65.5±0.3	65.6±0.1	66.5±0.1
	Joslim (Chin et al., 2021)	62.0±0.4	62.7±0.4	63.1±0.3	63.7±0.2	64.1±0.3	65.0±0.2
	DS-Net (Li et al., 2021a)	61.8±0.6	63.8±0.3	64.8±0.2	65.3±0.2	65.5±0.3	66.7±0.2
	TIPS	**66.4±0.5**	**66.9±0.1**	**67.0±0.1**	**67.6±0.3**	**67.7±0.1**	**68.2±0.3**
	FLOPS	80M	120M	140M	160M	180M	200M
Tiny-ImageNet	US-Nets (Yu & Huang, 2019b)	47.0±0.5	47.3±0.1	48.3±0.3	49.0±0.1	50.2±0.3	51.4±0.2
	Joslim (Chin et al., 2021)	47.4±0.4	47.9±0.4	48.7±0.1	49.5±0.2	50.3±0.3	50.7±0.4
	DS-Net (Li et al., 2021a)	46.9±0.3	47.4±0.3	48.1±0.2	48.7±0.1	50.3±0.2	50.8±0.2
	TIPS	**53.5±0.3**	**53.8±0.2**	**54.0±0.1**	**54.4±0.3**	**54.9±0.2**	**55.1±0.2**
	FLOPS	260M	320M	400M	450M	500M	600M
ImageNet	US-Nets (Yu & Huang, 2019b)	70.6±0.3	71.6±0.2	71.8±0.1	72.1±0.2	72.3±0.4	72.9±0.2
	Joslim (Chin et al., 2021)	70.8±0.1	71.9±0.3	72.5±0.2	72.7±0.2	72.9±0.2	73.4±0.3
	DS-Net (Li et al., 2021a)	70.6±0.2	72.1±0.1	72.5±0.3	72.6±0.1	73.0±0.2	73.3±0.2
	TIPS	**71.8±0.4**	**73.2±0.3**	**73.6±0.3**	**74±0.2**	**74.3±0.3**	**74.7±0.1**

B.3. Additional Results for Table 1 and Table 2

We show the std. dev. values in Table 4 and Table 5 for MobileNet-v2 and ResNet34, respectively.

B.4. Details of TIPS

Given the supernet, we randomly sample 8 subnetworks and obtain the adjacency matrices $\boldsymbol{A}_k$ as described in Eq. 2. As discussed in Section 4.5, the ranking among multiple paths remains the same with varying value of λ. For simplicity, we set λ to '1' for the inter-subnetwork coupling matrix $\widetilde{\boldsymbol{Z}_{i,j}}$ in Eq. 5 to build the hyper-adjacency matrix $\hat{\boldsymbol{H}}(\lambda)$. Then we construct the DTMC as described in Eq. 6 and solve Eq. 7 to obtain the stationary state distribution. Next, we exploit the TAS and TPS analysis to characterize the important operations as discussed in Section 3.2 and Section 3.3.

B.5. Observations of DTMC-based Analysis

Based on our experiments on MLPs, MobileNet-v2, and ResNet34 (see Sec. 4), we can draw the following conclusions:

Table 5. Comparison of Top-1 test accuracy vs. FLOPS (in millions/Giga [M/G]) with SOTA training methods on ResNet-34. Best results are shown with bold fonts. Results are averaged over three runs.

CIFAR100	FLOPS	120M	180M	200M	220M	240M	260M
	US-Nets (Yu & Huang, 2019b)	63.1±0.2	63.9±0.1	64.4±0.3	64.8±0.2	65.0±0.1	65.4±0.1
	Joslim (Chin et al., 2021)	65.8±0.1	66.2±0.3	66.7±0.4	67.0±0.2	67.3±0.4	67.4±0.4
	DS-Net (Li et al., 2021a)	64.4±0.3	65.9±0.1	66.2±0.3	66.4±0.1	66.5±0.3	66.6±0.1
	TIPS	**67.3±0.1**	**67.4±0.2**	**67.8±0.2**	**67.9±0.3**	**68.1±0.2**	**68.2±0.3**
Tiny-ImageNet	FLOPS	130M	190M	220M	250M	270M	300M
	US-Nets (Yu & Huang, 2019b)	42.9±0.1	43.2±0.2	44.3±0.3	44.7±0.3	44.9±0.2	45.2±0.3
	Joslim (Chin et al., 2021)	**44.9±0.2**	**45.0±0.4**	45.3±0.4	45.4±0.3	45.5±0.3	45.8±0.2
	DS-Net (Li et al., 2021a)	41.8±0.3	43.0±0.1	43.8±0.2	43.9±0.3	44.1±0.4	44.2±0.1
	TIPS	44.1±0.4	44.6±0.1	**45.4±0.2**	**45.8±0.1**	**45.9±0.1**	**46.0±0.2**
ImageNet	FLOPS	1.5G	2.2G	2.8G	3.0G	3.2G	3.6G
	US-Nets (Yu & Huang, 2019b)	67.8±0.4	69.2±0.1	69.7±0.2	70.1±0.1	70.2±0.3	70.5±0.2
	Joslim (Chin et al., 2021)	68.0±0.2	**69.4±0.4**	69.6±0.4	70.0±0.1	70.2±0.1	70.4±0.1
	DS-Net (Li et al., 2021a)	66.0±0.6	67.0±0.3	68.8±0.1	69.4±0.2	69.9±0.1	70.0±0.2
	TIPS	**68.4±0.5**	69.3±0.2	**70.8±0.2**	**71.1±0.1**	**71.4±0.2**	**71.9±0.2**

- The TPS values and the important operations identified by our framework depend on the specific structure of a given supernet. Hence, we need to conduct the DTMC-based analysis individually for different supernets in order to have a meaningful understanding of operations importance.

- Empirically, we found that for inverted bottleneck-based MobileNet-v2 supernet and BasicBlock-based ResNet supernet, the first convolution layer was more important and more channels were sampled at those layers.

B.6. Societal Impact of TIPS

Our method does accelerate the convergence speed of the training process and thus reduces the total training costs. Indeed, as shown in Fig. 5(b,c) in the main paper, to achieve the same training loss, our method requires far fewer training epochs compared to previous SOTA methods (Joslim (Chin et al., 2021) and US-Nets (Yu & Huang, 2019b)). Hence, our method is clearly more environment-friendly than SOTA and implicitly addresses an important societal concern.

B.7. Comparison with One-shot NAS

We remark that our method focuses on the training methods for anytime inference in order to improve the test accuracy of anytime inference for neural networks instead of improving the accuracy of single networks; this is the key difference between anytime inference and neural architecture search (NAS). To demonstrate the benefits of our proposed training method, we compare our proposed TIPS with the training method of the one-shot NAS method Once-For-All (OFA) (Cai et al., 2020).

Table 6. Comparison of Top-1 test accuracy vs. FLOPS (in millions/Giga [M/G]) with representative one-shot NAS method OFA on MobileNet-v2 under the same training setup. The best results are shown with bold fonts. Results are averaged over three runs.

#FLOPs	**260M**	**320M**	**400M**	**450M**	**500M**	**600M**
OFA	70.4	71.4	72.3	72.8	73.4	74
TIPS	**71.8**	**73.2**	**73.6**	**74**	**74.3**	**74.7**

To make an apples-to-apples comparison with OFA, we took the official training code for OFA and then trained our MobileNet-v2-based supernet on ImageNet under the same setup as ours (150 epochs, batchsize=512). As shown in Table 6, our proposed TIPS achieves far better than OFA #FLOPs-accuracy tradeoffs consistently; *e.g.*, when the FLOPs budget is 320M, TIPS has a 1.8% higher accuracy than OFA, which is a significant improvement on ImageNet.

B.8. Overhead of Network Switch at Runtime

In our method, we store only the supernet and the configuration of each subnetwork (*i.e.*, only the #channels values for the layers in the supernet). This way, we do *not* need to store and load the pretrained weights of different subnetworks separately. We provide the pseudo-code in Algorithm 2 to better illustrate how we conduct the inference of AnytimeNNs.

Algorithm 2 Pseudo code: Inference of AnytimeNNs

1: **Input:** Supernet checkpoint G, Pareto-optimal subnetworks' width configurations Θ
2: **Run:**
3: Load the supernet checkpoint G
4: Load subnetworks' width configurations Θ
5: **while** Running inference **do**
6: Index the suitable subnetwork configurations θ from Θ
7: **for** each layer i in G **do**
8: Load C_{IN_i} and C_{OUT_i} from θ
9: Set #input channels to C_{IN_i}
10: Set #output channels to C_{OUT_i}
11: **end for**
12: **while** hardware resources budget doesn't change **do**
13: Run inference
14: **end while**
15: **end while**

To quantitatively demonstrate the hardware efficiency of our method, we use MobileNet-v2 as the supernet then select twelve Pareto-optimal subnetworks under different FLOP budgets. We calculate the storage costs of these subnetworks. Specifically, storing these twelve subnetworks separately requires 117.8MB in total. In contrast, in our method, the storage cost of all these subnetwork configuration is quite negligible, *i.e.*, requiring only 1.9 KB in total (6176× smaller) since it only requires storing layerwise width information for each subnetwork. Hence, our method is very hardware-efficient as it has much less overhead than storing all these subnetworks individually.

We also verify the hardware efficiency of our method as follows: As shown in Algorithm 2, we only need to load the checkpoint for the supernet G once and the Pareto-optimal subnetwork configurations. To switch the subnetwork, we just select the suitable subnetwork configuration and reconfigure the width value of each layer (see line 6-11 in Algorithm 2). For the same twelve Pareto-optimal subnetworks from MobileNet-v2, on a NVIDIA RTX3090 GPU with PyTorch Framework, we repeat the switching process 1000 times. We measure that reloading a new subnetwork checkpoint consumes 287ms, on average. In contrast, in our method, it takes only a negligible 0.037ms, on average, to switch the subnetwork (7756× faster than reloading the subnetworks checkpoint).

C. Illustration of Our Modeling Method and Sampling Process

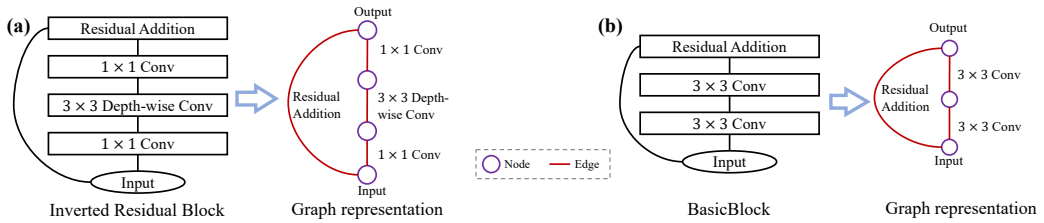


Figure 9. Illustration of how we model neural networks as graphs. **(a)** Inverted Residual block from MobileNet-v2 (Sandler et al., 2018). **(b)** BasicBlock from ResNet-18/34 (Veit et al., 2016). As we mention in Section 3.1, we model each operation (linear layers, convolutional layers, residual additions, pooling layers, etc.) as *edges* in a graph; we model the input featuremaps and output featuremaps of these operations as *nodes* in a graph.

C.1. Modeling Neural Networks as Graphs

As shown in Fig, 9, we illustrate how we model two commonly used blocks as graphs: Inverted Residual block from MobileNet-v2 (Sandler et al., 2018) and BasicBlock from ResNet-18/34 (He et al., 2016).

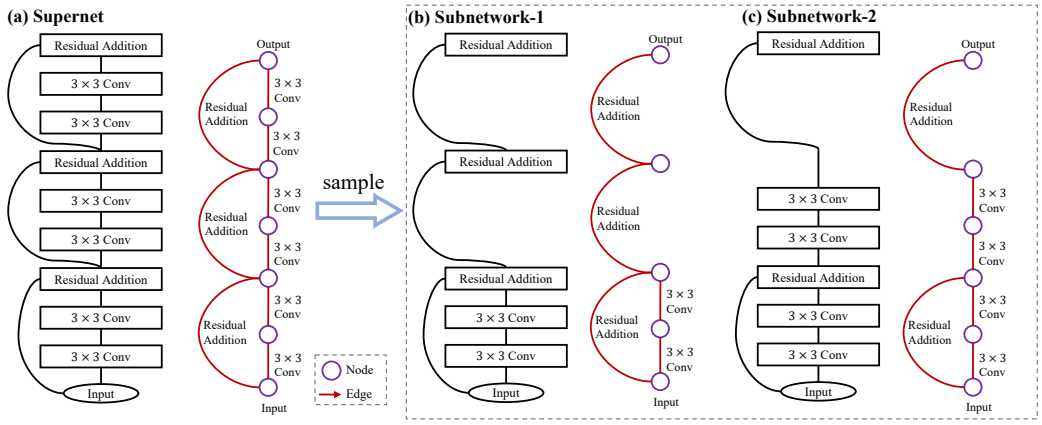


Figure 10. An illustration of sampling subnetworks and then converting subnetworks to graphs. We use a network with three BasicBlocks from ResNet18/34 (Veit et al., 2016), for simplicity.

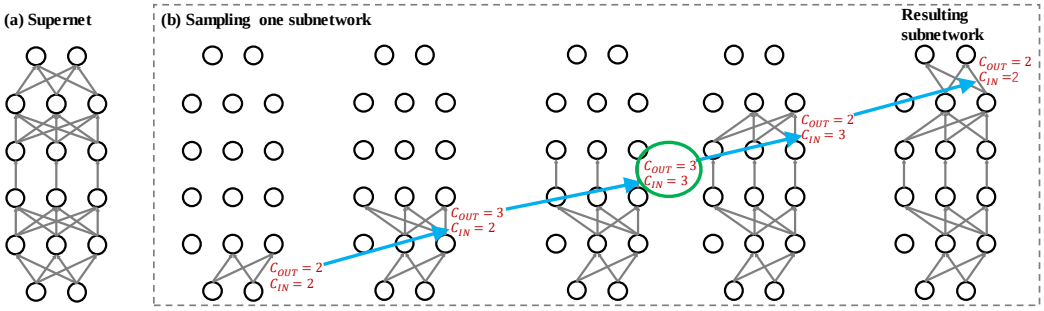


Figure 11. Sampling a subnetwork from the supernet. We show a supernet with 4 convolution layers and one depthwise convolution layer, for simplicity. We sample these layers based on input to output order in the supernet. The number of input channels of a given layer (C_{IN}) is always set to the same value as the number of output channels (C_{OUT}) of the previous layer; see the blue arrows in the figure. In particular, for a depthwise convolution layer, C_{OUT} is always set to the values its C_{IN}; see the green circle in the figure.

C.2. Sampling Subnetworks from the Supernet

As shown in Fig. 10, to further demonstrate how we model subnetworks as graphs, we use a network with three BasicBlocks as the supernet. Clearly, the same operation from the supernet can be skipped or kept in different subnetworks (this is temporally dependent). Our method captures these temporal relationships among multiple subnetworks; this is why we combine the adjacency matrices of multiple subnetworks into a hyper-adjacency matrix, as shown in Eq. 4.

C.3. Illustration of Validity of Subnetworks

As shown in Fig. 11, given the supernet, to ensure the validity of the sampled networks, we sample #channels of each layer from input to output as follows:

1. The number of input channels of a given layer is always set to the same value as the number of output channels of the previous layer (see the blue arrows in Fig. 11).
2. For a depthwise convolution layer, we always set the number of output channels to the same value as its input channels (see the green circle in Fig. 11).