Complex & Intelligent Systems (2023) 9:6117–6138
https://doi.org/10.1007/s40747-023-01139-8

ORIGINAL ARTICLE

Tiny adversarial multi-objective one-shot neural architecture search

Guoyang Xie[1,2] · Jinbao Wang[1] · Guo Yu[3] · Jiayi Lyu[4] · Feng Zheng[5] · Yaochu Jin[6,7]

Received: 5 January 2023 / Accepted: 13 April 2023 / Published online: 11 July 2023

Abstract

The widely employed tiny neural networks (TNNs) in mobile devices are vulnerable to adversarial attacks. However, more advanced research on the robustness of TNNs is highly in demand. This work focuses on improving the robustness of TNNs without sacrificing the model's accuracy. To find the optimal trade-off networks in terms of the adversarial accuracy, clean accuracy, and model size, we present TAM-NAS, a tiny adversarial multi-objective one-shot network architecture search method. First, we build a novel search space comprised of new tiny blocks and channels to establish a balance between the model size and adversarial performance. Then, we demonstrate how the supernet facilitates the acquisition of the optimal subnet under white-box adversarial attacks, provided that the supernet significantly impacts the subnet's performance. Concretely, we investigate a new adversarial training paradigm by evaluating the adversarial transferability, the width of the supernet, and the distinction between training subnets from scratch and fine-tuning. Finally, we undertake statistical analysis for the layer-wise combination of specific blocks and channels on the first non-dominated front, which can be utilized as a design guideline for the design of TNNs.

Keywords Tiny neural network architecture search · Adversarial attack · One-shot learning · Multi-objective optimization

Introduction

Due to the fragility of tiny neural networks, there is a critical need for systematically investigating how to design robust tiny neural networks (TNNs). It is well known that deep neural networks are susceptible to attacks that introduce subtle perturbations to the input data [1, 2]. To defend attacks, current studies [3, 4] investigate the relationship between the model's capacity and its adversarial robustness via ResNet [5] as the backbone network. It has been shown that adding more neural network parameters may greatly improve the model's resilience. Despite the widespread usage of tiny neural networks for mobile applications, little research concentrates on the improvement of the robustness of TNNs. Typically, they are 10 K to 2 M in size. Hence, the main goal of our research is to re-design the TNN architecture to enhance its robustness.

Guoyang Xie and Jinbao Wang have contributed equally to this work

✉ Yaochu Jin
yaochu.jin@surrey.ac.uk

Guoyang Xie
guoyang.xie@surrey.ac.uk

Jinbao Wang
linkingring@163.com

Guo Yu
gysearch@163.com

Jiayi Lyu
lyujiayi21@mails.ucas.ac.cn

Feng Zheng
zhengf@sustech.edu.cn

1 Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China

2 Department of Computer Science, University of Surrey, Guildford, Surrey GU2 7XH, UK

3 Institute of Intelligent Manufacturing, Nanjing Tech University, Nanjing 211816, China

4 School of Engineering Science, University of Chinese Academy of Sciences, Beijing, China

5 Department of Computer Science, Southern University of Science and Technology, Shenzhen 518055, China

6 Faculty of Technology, Bielefeld University, 33619 Bielefeld, Germany

7 Department of Computer Science, University of Surrey, Guildford GU2 7XH, UK

Designing the neural network architecture is a promising way to enhance robustness against adversarial examples. Previous studies [3, 4, 6, 7] have illustrated the significance of the neural architecture for adversarial robustness. Huang et al. [8] give a comprehensive investigation on the impact of network width and depth on the robustness. Liu et al. [9] employ multi-objective NAS to search for robust neural network architectures. However, most of them do not target on the desired FLOP count and address the tiny architecture design issue. We presume that the tiny network design itself has anti-adversarial capabilities. Thus, the purpose of our work is to investigate the best trade-off architecture and present a design principle for compact, resilient network architectures.

To identify the best tiny neural network architecture with clean accuracy, adversarial accuracy, and model size, we employ a multi-objective architecture search algorithm to find the best trade-off architecture design. Most present works adopt adversarial training [2, 10, 11] to increase the robustness of the model. However, most are solely concerned with enhancing resilience, ignoring the degradation of clean accuracy. Hence, we employ a multi-objective approach-based NAS [12–14] to find the best architecture for the trade-off solutions between the adversarial accuracy, the clean accuracy, and the mode size. In our work, we mainly address the trade-off problem based on the ShuffleNetV2 architecture [15], Xception block [16], SE layer [17], Non-Local block [18], and their variants.

The contributions of this work can be summarized as follows:

- To find the best trade-off neural networks between the adversarial accuracy, clean accuracy, and model size, we propose three novel tiny robust blocks. Due to the inertial self-attention mechanism, the layer-wise combination of these three blocks can increase the robustness without substantially degrading the clean performance.
- We explore a new adversarial training paradigm for the supernet. Because the subnets heavily rely on the supernet in one-shot NAS, the adversarial performance of the subnets can be further improved using our proposed training paradigm. To this end, we examine how the width of the supernet, the perturbation range, and the number of attack steps for the supernet adversarial training affect the performance of the subnets.
- We seamlessly integrate a multi-objective search algorithm with a one-shot NAS algorithm. After the search process, we can get the non-dominated front immediately, which makes it easier to find the best trade-off subnets. In addition, we discover that training from scratch outperforms fine-tuning for the non-dominated subnets.
- We provide guidelines for how to design tiny robust neural networks. First, pure robust blocks and small robust blocks should be placed in the shallow levels, while pure tiny blocks should be placed in the deep layers. Second, larger intermediate channels should be placed in the shallow levels, whereas intermediate channels should decrease gradually in the remaining layers. Finally, we rebuild a tiny neural network using the guidelines and find that it can reduce the model size and increase the adversarial accuracy and the clean accuracy.

Related work

Adversarial training is the most popular defensive mechanism against adversarial attacks [2], which uses both clean and adversarial images for training. Derived from game theory [19], the work in Ref. [10] reformulates the min–max optimization problem of adversarial learning as Nash equilibrium [20]. The game-theory-based optimization method [21] can effectively reduce the high computational cost without sacrificing adversarial accuracy. In addition, the work [11] has empirically proved the trade-off information between the adversarial accuracy and clean accuracy. Kannan et al. [22] and Zhang et al. [23] develop a surrogate loss function to reduce the disparity between clean images and adversarial counterparts. Madry et al. [3] conclude that increasing the capacity size of a neural network might enhance their performance against adversarial attacks. It is well known that most neural network models deployed in our electronic devices are quite small due to energy consumption and storage limitations. Hence, we aim to figure out which kind of tiny neural network architectures can be effective for the resilience of adversarial perturbations.

White-box attacks assume that the adversary knows detailed information about the targeted models, including model architecture, hyperparameters, gradients, and training data. In the following, we use X^* and X to denote the adversarial and clean examples, respectively. Then, ∇_X measures the gradients of the loss function l for X.

Fast Gradient Sign Method (FGSM). It is a one-step and non-target attack, which generates adversarial examples by adding perturbations along the direction of the gradient sign at each pixel [2]. The generated adversarial examples can be calculated by

$$X^* = X + \epsilon \cdot \text{sign}\left(\nabla_X \, l(X, y_{true})\right), \tag{1}$$

where ϵ is a hyperparameter that controls the magnitude of the disturbance, and y_{true} is the ground truth label.

Projected Gradient Descent (PGD). The PGD attack [3], which combines randomized initialization with multi-step attacks, is one of the strongest adversarial attacks against

adversarial training. The adversarial examples generated by the PGD attack can be expressed as

$$X_0^* = X + \mathcal{U}(-\epsilon, \epsilon), \tag{2}$$

$$X_{n+1}^* = \prod_{X,\epsilon} \left\{ X_n^* + \alpha \cdot \text{sign}(\nabla_{X_n^*} l(X_n^*, y_{true}) \right\}, \tag{3}$$

where $\mathcal{U}$ is a uniform distribution, X_n^* is the adversarial examples after n steps, and $\prod_{X,\epsilon}(B)$ is the projections of $B(X, \epsilon)$. *Neural network architecture search* aims to replace hand-crafted architecture design approaches with automated design using machine learning techniques. Representative search algorithms include evolutionary algorithms [24, 25], reinforcement learning [26, 27], and gradient-based methods [28–30]. In one-shot NAS [31], the authors [31] construct a supernet that is capable of generating any potential architecture in the search space. The work in Ref. [31] trains a supernet for once, and then during the search, they can retrieve multiple fitness values for different subnets through weight sharing from the supernet. However, most of them employ a single-objective optimization approach to search, which is not well suited for solving the trade-off problem [11]. To solve the multi-objective optimization problem, we adopt the elitist non-dominated sorting genetic algorithm (NSGA-II) [32] as our search algorithm. Recently, several papers have been published that explore the impact of the network width and depth on robustness [8, 9]. In addition, Huang et al. [33] propose a robust residual block and a compound scaling rule to investigate the influence of network width and depth. By contrast, TAM-NAS pays more attention to the resilience attack ability of tiny neural networks.

TAM-NAS

Our TAM-NAS approach consists of four steps as follows. First, design a supernet search space and uniformly sample different candidates from the supernet to increase our supernet representation ability for many subnet architectures when using a single supernet. Second, train the candidates sampled from the supernet using adversarial examples and make them more robust in the presence of adversarial attacks. Third, perform multi-objective search of new subnets using NSGA-II [32] and evaluate the clean accuracy, the adversarial accuracy, and the number of parameters of each subnet by cloning its weight from the pre-trained supernet. Finally, fine-tune each subnet on the first non-dominated front and evaluate their performance on the test dataset. Figure 1 shows the overall framework.

Problem definition

Without loss of generality, our supernet search space $\mathcal{A}$ can be represented by a directed acyclic graph (DAG), denoted as $\mathcal{N}(\mathcal{A}, W)$, where W is the weight of supernet. A subnet architecture is a subgraph $a \in \mathcal{A}$, denoted as $\mathcal{N}(a, w)$, where w is the weight of the subnet. $\Gamma(\mathcal{A})$ is a prior distribution of $a \in \mathcal{A}$. $\mathcal{L}_{adv-train}(\cdot)$ is the adversarial training loss function on the adversarial training examples. The most important factor for TAM-NAS is that the performance of the subnets using inherited weights from the supernet (without extra fine-tuning or training from scratch) should be highly predictive. In other words, the supernet weights $W_{\mathcal{A}}$ should be optimized, so that all subnet architectures in the search space A are optimized simultaneously. It can be expressed as

$$W_{\mathcal{A}} = \underset{W}{\text{argmin}} \mathbb{E}_{a \sim \Gamma(\mathcal{A})} \left[\mathcal{L}_{adv-train}(\mathcal{N}(a, W(a))) \right]. \tag{4}$$

After finishing the training of supernet, the next step is to find a set of Pareto optimal subnets $a^* \in \mathcal{A}$ in terms of our objectives: the adversarial error, the clean error, and the model size. It can be expressed as

$$\begin{aligned} &\min \left\{ f_1(a^*), f_2(a^*), f_3(a^*) \right\}, \\ &\text{s.t. } a^* \in \mathcal{A}, \end{aligned} \tag{5}$$

where f_1, f_2, f_3 are the three objectives, the adversarial error, the clean error, and the model size, respectively. Figure 2 shows the pipeline of multi-objective one-shot NAS. Actually, it cannot get the minimum value for three objectives simultaneously, since each objective has a strong trade-off relationship with the other two objectives. For instance, if the model size is larger, the adversarial error and the clean error will become smaller. Our aim is to obtain a tiny model with compatible performance in the adversarial dataset and clean dataset.

Search space design

Since we aim to search tiny robust neural networks, our supernet adopts one of state-of-the-art hand-crafted tiny network architecture—ShuffleNetV2 [15] as the backbone model. Since our experiments are mainly conducted on the CIFAR10 [34] and SVHN [35] datasets, the depth and width of the supernet are different from the original ShuffleNetV2, which is adapted from the Imagenet dataset [36]. Table 1 shows the parameter settings of the overall architecture of the supernet. BN represents the batch norm layer. 3×3 Conv represents a convolutional layer, and its kernel size is 3. CB refers to the choice block chosen from our pre-defined block search space. SE refers to the SE layer [17]. Moreover, we design a search space for the channel number search of each choice

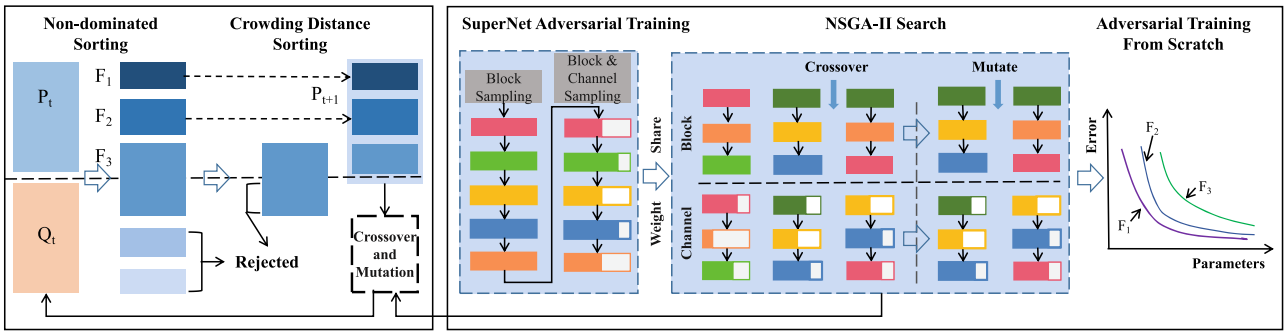


Fig. 1 The overall framework of TAM-NAS. The first step is to design a supernet search space and uniformly sample the new subnet candidates. Our sampling strategy is divided into two phases. One is the block sampling phase, and the other is the block and channel jointly sampling phase. The second step is to train the subnets sampled from the supernet adversarially. The third step is to perform a multi-objective search for the best trade-off subnets. The fitness value of the subnets is achieved by cloning the weight from the supernet. The final step is to train from scratch or fine-tune the non-dominated subnets

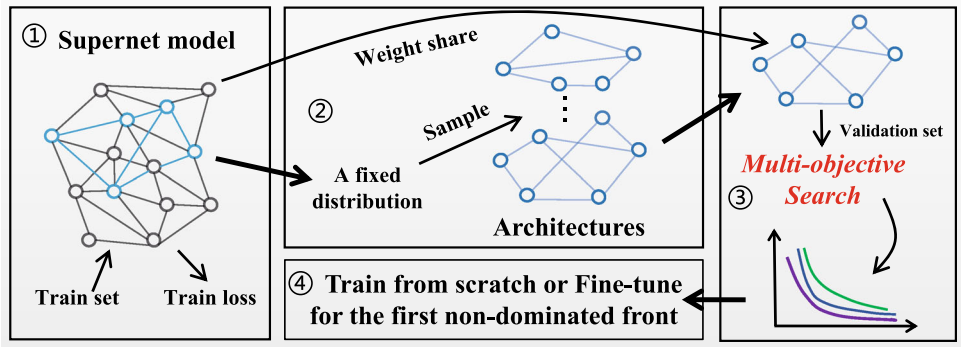


Fig. 2 The proposed multi-objective one-shot NAS framework. (1) Design the supernet search space and the loss function. (2) Sample different subnets architectures from the supernet under a fixed distribution and train the subnets for a small number of epochs. (3) Perform multi-objective search for the best trade-off subnets and obtain the fitness value of the subnets by cloning the weight from the supernet. (4) Train the non-dominated subnets

Table 1 Supernet architecture

Input shape	Block	Channels	Repeat	Stride
$32^2 \times 3$	3×3 Conv	24	1	1
$32^2 \times 24$	BN	24	1	
$32^2 \times 24$	CB	48	1	2
$16^2 \times 48$	CB	48	3	1
$16^2 \times 48$	CB	96	1	2
$8^2 \times 96$	CB	96	7	1
$8^2 \times 96$	CB	192	1	2
$4^2 \times 192$	CB	192	3	1
$4^2 \times 192$	1×1 Conv	176	1	1
$4^2 \times 176$	BN	176	1	
$1^2 \times 176$	Pooling	176	1	
$1^2 \times 176$	SE	920	1	
$1^2 \times 920$	1×1 Conv	1024	1	1
$1^2 \times 1024$	FC	10	1	

block. In total, we provide 22 block choices and 10 channel number choices for the search space. We will describe our search space in detail.

Block search spaces

In block search spaces, we design three types of blocks, namely pure tiny blocks, pure robust blocks, and tiny robust blocks.

(1) Pure Tiny Blocks: Pure tiny blocks mainly come from ShuffleNetV2 [15]. We add the self-attention layer—SE layer [17] into the main branch for balancing between the accuracy and inference speed. Figure 3a will become the pure tiny blocks if the non-local layer from the main branch is removed.
(2) Pure Robust Blocks: First, we design a non-local block for image denoising, inspired by Refs. [18, 37]. We also add another self-attention layer—SE layer [17] as the last

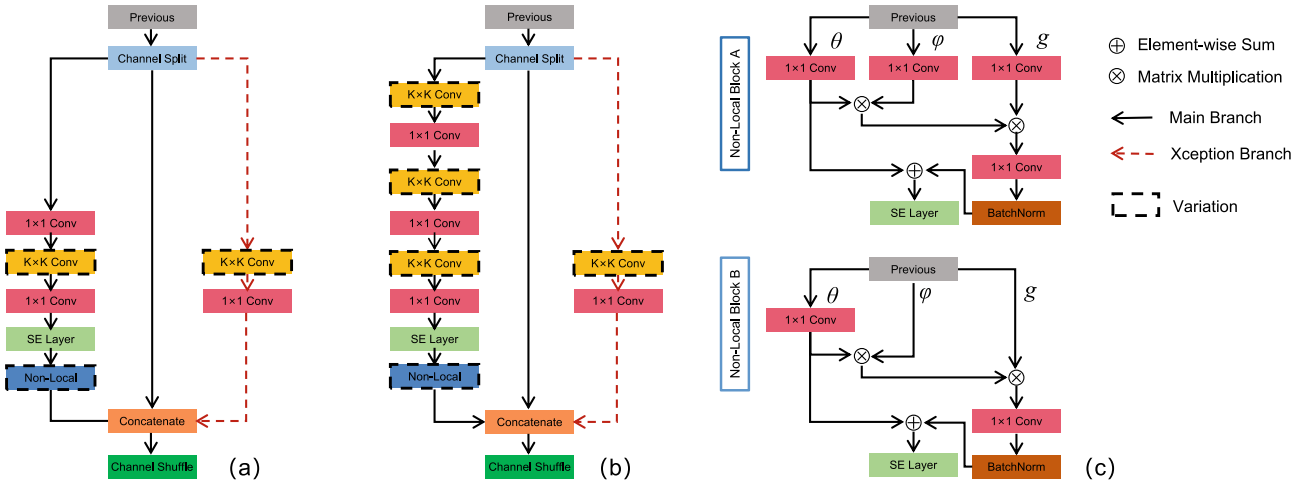


Fig. 3 **a** and **b** Internal architectures of the tiny robust blocks. K refers to the kernel size, and it ranges from 3, 5, and 7. The dashed line indicates the internal search space for the tiny robust blocks. Therefore, if we removed the non-local layer from the main branch of (**a**) and (**b**), **a** and **b** will denote the pure tiny blocks. **c** Internal architectures of the pure robust blocks. The upper part of (**c**) represents the combination of SE layer and Embedded-Gaussian non-local layer. The bottom part of (**c**) represents the combination of SE layer and Gaussian non-local layer

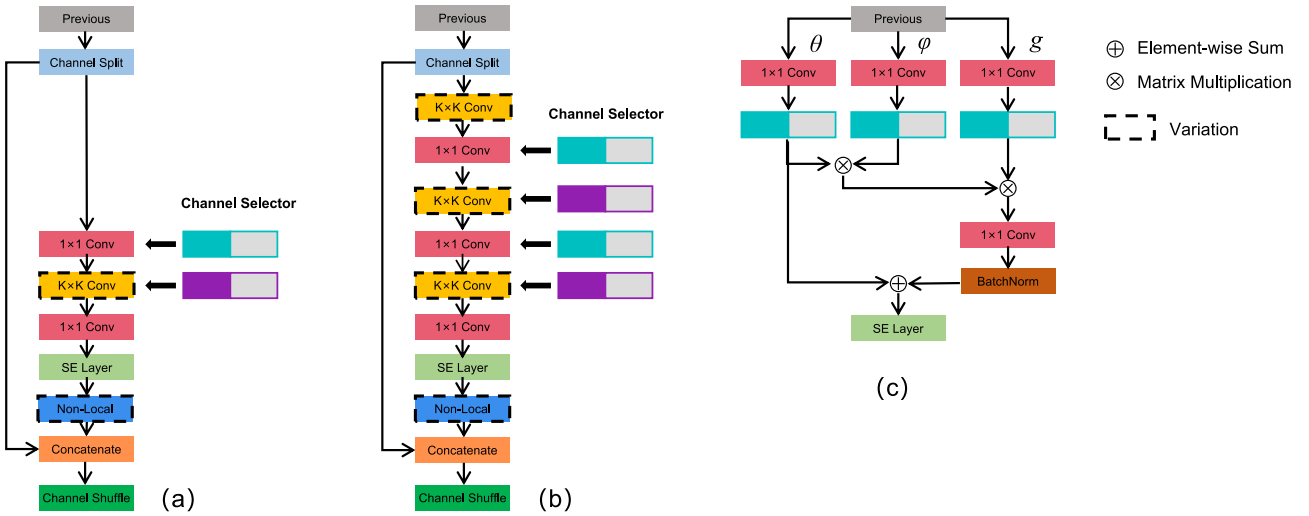


Fig. 4 **a–c** How we add the channel selector into the tiny robust blocks and the pure robust blocks, respectively. The kernel size is set to 3, 5, and 7, respectively. The dashed line indicates the internal search space for the tiny robust blocks. Therefore, if we remove the non-local layer from the main branch of (**a**) and (**b**), **a** and **b** will denote the pure tiny blocks

layer of the non-local block, since it has been found [18] that the self-attention mechanism could make the neural network more robust. Figure 3 shows two non-local blocks, which are referred to as Embedded-Gaussian version and Gaussian version [18]. The non-local block is only one of the choice blocks in the stride-1 layer of the supernet, because its output feature map size is not compatible with the stride-1 layer. Figure 3b shows the internal architecture of the pure robust blocks.

(3) Tiny Robust Blocks: In addition, we borrow the idea from Refs. [15, 16, 38] to design shufflev2 and shufflev2-xception blocks. To make them more robust, we try to add the non-local block and the SE layer into the main branch of the original shufflev2 and shufflev2-xception block. Figures 3a and b show the internal architecture of the tiny robust block. The kernel sizes of the depth-wise convolutional layer are 3, 5, and 7, respectively. Furthermore, since the non-local layer will add more parameters for our shufflev2 or shufflev2-xception block, we set the non-local layer of shufflev2 and shufflev2-xception block as an optional choice, which means that it is also another new search space. When we remove the non-local layer

and SE layer in tiny robust blocks, they will work as pure tiny blocks. In Fig. 3, the dashed line indicates the internal search space for each block. The encoding method of choice blocks is to assign an order from 0 to 21 for each block.

Algorithm 1: Supernet training

```
foreach New Candidate Network ∈ Block Sampling do
    while epoch ≤ 20 do
        Trades-YOPO-m-n;
        Update the corresponding weight of Supernet;
    end
end
foreach New Candidate Network ∈ Block and Channel Jointly Sampling do
    while epoch ≤ 20 do
        Trades-YOPO-m-n;
        Update the corresponding weight of Supernet;
    end
end
Return Supernet
```

Channel search spaces

The channel number plays an essential role in the neural network's efficiency and computational cost [29]. Apart from the adversarial and clean accuracy, we select the total number of parameters as the third objective. We only search the intermediate channel number of each block, including pure robust blocks, pure tiny blocks, and tiny robust blocks. Heuristically, the reduction of intermediate channel numbers will not give rise to the deterioration of adversarial accuracy and clean accuracy, which is well suited for the balance between the performance of neural networks and their model size. Specifically, Fig. 4 shows how we add a channel selector into the intermediate part of pure tiny blocks, pure robust blocks, and tiny robust blocks. The channel selector ratio ranges from 0 to 2, and its interval is 0.2. The encoding method of choice channels is to assign an order from 0 to 10 for each channel selector ratio.

Uniform sampling

Our supernet sampling strategy is to sample choice blocks at first and then jointly sample choice blocks and choice channels once the warm-up training of the supernet is completed. We find that it is challenging for our supernet to converge when we jointly sample block choices and channel choices in the beginning. We build up a parameter table in advance to speed up the sampling procedure. We will give more details about block sampling and channel sampling.

Block sampling

We investigate many network architectures on the CIFAR10 and SVHN dataset and our pilot studies suggest that the tiny supernet size ranges from 1.5 to 4 M. In the block sampling phase, we only search block choices for the supernet architecture and set a constraint that the number of the supernet parameters range from 1.823 to 2.375 M. In all experiments, we train the supernet for 500 epochs in block sampling phases and sample a new architecture every 20 epochs.

Block and channel jointly sampling

The supernet jointly searches block and channel choices after the phase of block sampling. We refer to this phase as block and channel jointly sampling. We set another constraint that the number of the supernet parameters should range from 1.61 to 2.37 M in this phase. In all experiments, we train the supernet for 500 epochs in block and channel jointly sampling phase and sample a new architecture every 20 epochs.

Adversarial training

We aim to explore the influence of network architecture on its robustness against adversarial attacks. Therefore, we focus on white-box adversarial attacks bounded by l_∞. As we all know, PGD [3] adversarial training is computationally expensive and hard to converge. We follow [21, 23] and adopt the TRADES-YOPO-m-n algorithm [21] to speed up our adversarial training. This work adopts the loss function in Ref. [23], which is described as below

$$\min \mathbb{E}\left\{\mathcal{L}(f_\theta(x), y) + \max_{\|\eta\le\epsilon\|} \mathcal{L}(f_\theta(x), f_\theta(x+\eta))/\lambda\right\}, \quad (6)$$

where $\mathcal{L}(.,.)$ denotes the cross-entropy loss function; $f_\theta(x)$ denotes the output vector of the neural network, which is parameterized by θ. y is the label-indicator vector; η denotes the image noise (perturbation); λ is a balancing hyperparameter. $f_\theta(x+\eta)$ denotes the output vector of the neural network where its input has been added into the perturbation η.

This loss function reduces the gap between adversarial and non-adversarial examples when the model undertakes the classification task, i.e., make the classification boundary more smooth. YOPO-m-n borrows the idea from Pontryagin's Maximum Principle [39] to approximate the back-propagation. One of YOPO's assumptions is that the adversarial perturbation only affects the first layer's weights. TRADES-YOPO-m-n is to perform n times gradient descent to update the weights of the first layer and iteratively runs m times for each datum. Zhang et al. [21] state that $m \times n$ should be larger than the number of attack iterations, so that TRADES-YOPO-m-n could achieve a competitive result. In

our experimental setting, the number of outer loops m is set to 5, and the number of inner loops n is set to 3 if we desire to attack the models in ten iterations by PGD [3]. The training time for each epoch is 2.5 min running in the training platform equipped with a single GPU V100. The supernet adversarial training algorithm is presented in Algorithm 3.2.1.

Algorithm 2: Multi-objective search

Input : population size N, crossover probability p_c, mutation probability p_m, number of generations G, supernet S_w,, clean accuracy predictor S_c, adversarial accuracy S_a, parameters calculator S_{cp}, P^0 the first generated population, P_g the gth generated population, F^0 all non-dominated fronts of first generated population, F_g all non-dominated fronts of gth generated population, F_g^0 the first non-dominated front of gth generated population, F_g^i the ith non-dominated front of gth generated population. Q_0 the first offspring produced from the first population P_0, Q_g the gth offspring produced from the gth population P_g

Output: F_G^0 The first non-dominated front on Gth generation

$g \leftarrow 0$ // initialize a generation counter
$P_0 \leftarrow$ Initialize Population N from Supernet S_w
$f_{p_0}^{clean} \leftarrow S_c(P_0)$ // compute clean accuracy for the first population
$f_{p_0}^{adv} \leftarrow S_a(P_0)$ // compute adversarial accuracy for the first population
$f_{p_0}^{params} \leftarrow S_{cp}(P_0)$ // calculate the size of models for the first population
$F^0 \leftarrow$ Fast Nondominated Sort(P_0)
$F^0 \leftarrow$ Crowding Distance Assignment(F^0)
$P_0 \leftarrow$ Binary Tournament Selection(P_0) // choose parents through tournament selection for mating
$Q_0 \leftarrow$ Crossover(P_0, p_c) // create offspring population by crossover between parents
$Q_0 \leftarrow$ Mutation(Q_0, p_m) // induce randomness to offspring population through mutation
$f_{q_0}^{clean} \leftarrow S_c(Q_0)$ // compute clean accuracy for the first offspring
$f_{q_0}^{adv} \leftarrow S_a(Q_0)$ // compute adversarial accuracy for the first offspring
$f_{q_0}^{params} \leftarrow S_{cp}(Q_0)$ // calculate the size of models for the first offspring
while $g < G$ **do**
  $R_g \leftarrow P_g \cup Q_g$ // merge parent and offspring population
  $F_g \leftarrow$ Fast Nondominated Sort(F_g)
  $P_{g+1} \leftarrow 0$
  $i \leftarrow 1$
  while $P_{g+1} + F_g^i \leq N$ **do**
    $F_g^i \leftarrow$ Crowding Distance Assignment(F_g^i)
    $P_{g+1} \leftarrow P_{g+1} \cup F_g^i$
    $i = i + 1$
  end
  Sort(F_g^i) // sort in descending order by the crowding distance operator
  $P_{g+1} \leftarrow P_{g+1} \cup F_g^i[1 : (N - P_{g+1})]$ // choose the first $(N - P_{g+1})$ elements of F_g^i
  $P_{g+1} \leftarrow$ Binary Tournament Selection(P_{g+1})
  $Q_{g+1} \leftarrow$ Crossover(P_{g+1}, p_c)
  $Q_{g+1} \leftarrow$ Mutation(Q_{g+1}, p_m)
  $f_{q_{g+1}}^{clean} \leftarrow S_c(Q_{g+1})$ // compute clean accuracy for the $g+1$ offspring
  $f_{q_{g+1}}^{adv} \leftarrow S_a(Q_{g+1})$ // compute adversarial accuracy for the $g+1$ offspring
  $f_{q_{g+1}}^{params} \leftarrow S_{cp}(Q_{g+1})$ // calculate the size of models for the $g+1$ offspring
end
Return F_G^0

Multi-objective search

We use NSGA-II [32] as the multi-objective search algorithm. The multi-objective search algorithm is presented in Algorithm 3.4. The first objective is the clean accuracy. It evaluates the performance of the model on the clean training data. The second objective is the adversarial accuracy. It evaluates the performance of a model under a white-box PGD attack bounded by l_∞. The third objective is to evaluate the number of parameters of our searched subnet. Moreover, the weights of each searched subnet are cloned from the corresponding part of the supernet. Therefore, there is no need to train any searched subnet in the whole search process. We are able to get the first objective value and second objective value after the inference of each searched subnet, and we can quickly obtain the number of parameters for each searched subnet by checking our parameters table. In the initialization of NSGA-II (Lines 1–13 in Algorithm 2), we randomly initialize the parent population P_0 at the beginning, where each individual (subnet) of the population is evaluated with three fitness values: the adversarial error, clean error, and the number of parameters. The weights of each subnet are cloned from those of the corresponding part

of the supernet. Therefore, the adversarial error and clean error of each subnet can be calculated by the inference with the weight from the supernet, while the model size of each subnet is closely related to its network architecture. Then, P_0 is sorted based on the non-dominated sorting. From Line 9 to Line 10 in Algorithm 2, we employ the tournament scheme and mutation operator to generate the offspring population Q_0. During the iteration of optimization (Lines 14–32), a combined population $R_t = P_t \cup Q_t$ is formed. Then, R_t will be sorted into fronts of individuals in an ascending order according to the non-dominated sorting, as described in Line 16. The new population P_{t+1} is achieved from R_t by selecting the elite solutions front by front according to their front number in an ascending order, as shown in Lines 19–23. The selection continues until F_g^i, from which only a subset of the solutions are selected according to the crowding distance values in a descending order. We will present the details of crossover and mutation operators in the next.

Crossover

Before crossover, our multi-objective search adopts the tournament selection for choosing two parents. In the experimental settings, the number of individuals that participate in the tournament scheme is set to 10. Our crossover operator inherits and recombines the block or channel from the two parents to generate a new subnet. To solve the channel dimensional mismatch problems, we aim to pre-allocate the weight matrix for the convolutional kernels, i.e., `max_output_channel`, `max_input_channel`, and `kernel_size`. Our maximum channel dimension is two times as much as the original dimension. After crossover, the dimension of the weight matrix for the current batch is still unchanged. However, we only keep the value of the weight matrix for the current input and output channel `[: c_out, : c_in, : ]`. Moreover, the value of other channels in the weight matrix is forced to be zero. In this case, we not only solve the channel dimension inconsistency problem but also implement channel crossover and mutation. Moreover, we have to move the non-local block out of the search space in the stride-2 layer, since the non-local block is not able to present in the stride-2 layer of the new subnet, since the size of the output feature map of the non-local block cannot match the input size of the stride-1 layer.

Mutation

The mutation operator is to re-assign each block or channel selector ratio of subnets from the search space. The mutation operator will be triggered if the randomized probability is larger than the pre-defined mutation probability. Our block encoding scheme is from 0 to 21. The block that contains the non-local layer is from 12 to 21. Most blocks can be arbitrarily mutated in each layer except for the stride-2 layer. Because the non-local block is not able to present in the stride-2 layer of new subnet, since the size of the output feature map of the non-local block cannot match the input size of the stride-1 layer. To enhance the diversity of the population and prohibit creating completely different network architectures, we set the mutation probability to 0.1, which means that the subnet has 10% opportunity to change the block or the channel number.

Training from scratch or fine-tuning

After finishing the multi-objective search, we obtain one set of non-dominated architectures. Generally speaking, there are two ways to deal with each searched subnet on the non-dominated front. One is to inherit the weights from the supernet and fine-tune them, and the other is to train each subnet from scratch. We also use TRADES-YOPO-m-n for our adversarial training algorithm. We will examine the differences between these two approaches in the next section.

Experimental result and analysis

In this section, we introduce our experimental settings for the overall framework, including the supernet training, NSGA-II search, and training from scratch or fine-tuning. In addition, following our search results, we try to give a guideline for how to devise neural network architectures to defend against adversarial attacks. We conduct several ablation studies to reveal how the network architecture influences the robustness against adversarial attacks. We perform extensive studies on CIFAR-10 [34] and SVHN [35] to validate the effectiveness of our overall framework. On CIFAR-10, we do the zero-padding with the 4 pixels on each side and randomly crop back into the original size. Then, we randomly flip the images horizontally and normalize them into [0, 1] for CIFAR and SVHN datasets. To better investigate the influence of the network architecture on robustness under adversarial attacks, we assume that the adversary has a complete access to a neural network, including the architecture and all parameters. That is why, we focus on white-box attacks on different architectures of neural networks.

Experimental settings

Supernet

According to Algorithm 3.2.1, our supernet first enters into the block sampling phase, and the maximum number of training epochs is set to 500. We provide 22 different blocks for the block sampling search space. Figure 3 gives the detail of our choices block. We use the stochastic gradient descent

method (SGD) as our optimizer. We use a batch size of 512, a momentum of 0.9, and a weight decay of $5e-4$. The initial learning rate in block sampling is set to 0.1 and is lowered by ten times at epochs 200, 400, and 450. The number of supernet parameters ranges from 1.823 to 2.375 M. After that, we jointly sample the blocks and channels of each layer in our supernet. We provide ten channel selectors for each block, and Fig. 4 presents the details of the channel selector choices for different types of blocks. We increase the number of epochs to 1000 with a batch size of 512, a momentum of 0.9 and a weight decay of $5e-4$. Specifically, we gradually enlarge the channel selector search space in every 20 epochs, which can help the supernet quickly converge. For example, we set our channel selector ratio to 1.8 and 2.0 before epoch 520 and add one more channel selector ratio of 1.6 at epoch 540. The channel selector ratio search space ranges from 0.2 to 2.0, and the interval is 0.2. The initial learning rate in block and channel jointly sampling is set to 0.1 and is lowered by ten times at epochs 600, 700, and 800. As for PGD attacks, the size of perturbations is set to $\epsilon = 8/255$ in an infinite norm. The outer loops m and the inner loops n of our adversarial training algorithm YOPO-m-n are set to 5 and 3, respectively. λ in Eq. 6 is set to 1, which means that we try to balance the model performance on the adversarial and non-adversarial examples.

NSGA-II

In our experiment settings, our total population size is 100. Then, P_0 is sorted based on the non-dominated sorting, and the size of P_0 is 50. From Line 9 to Line 10 in Algorithm 2, we employ the tournament scheme and mutation operator discussed in Sects. "Crossover" and "Mutation" to generate the offspring population Q_0 of size 50. The mutation probability is set to 0.1, which means that we have 10% to trigger the mutation operator. The number of individuals that takes part in the tournament scheme is set to 10. During the iteration of optimization (Lines 14-32), a combined population $R_t = P_t \cup Q_t$ is formed, and the size of R_t is 100. Note that the population sizes of P_{t+1} and Q_{t+1} are both 50. The number of generations is set to 20. We use the hypervolume (HV) to indicate whether our search algorithm has converged or not. Most of our experiments indicate that our multi-objective search algorithm has converged around the 18th generation.

Training from scratch

According to Fig. 1, we can obtain one set of non-dominated subnet architectures. We randomly initialize each subnet's weights and set the training epoch for every subnet to 100. However, we find that most of the subnets have converged at epoch 40. The initial learning rate of each subnet is 0.1 and is lowered by ten times at epochs 20, 40, and 80. The

Table 2 Supernet transferability on number of attack steps with the term of accuracy on CIFAR-10 (%)

Supernet training	Subnet model size	Clean Acc	$\mathbf{P}^{10}_{2/255}$	$\mathbf{P}^{10}_{4/255}$	$\mathbf{P}^{10}_{6/255}$	$\mathbf{P}^{10}_{8/255}$	$\mathbf{P}^{30}_{2/255}$	$\mathbf{P}^{30}_{4/255}$	$\mathbf{P}^{30}_{6/255}$	$\mathbf{P}^{30}_{8/255}$	$\mathbf{P}^{50}_{2/255}$	$\mathbf{P}^{50}_{4/255}$	$\mathbf{P}^{50}_{6/255}$	$\mathbf{P}^{50}_{8/255}$
$\mathbf{S}^{10}_{8/255}$	1.7453M	73.18	62.62	52.07	40.91	31.01	62.60	52.05	40.83	30.68	62.60	52.04	40.77	30.65
$\mathbf{S}^{1}_{8/255}$	1.7574M	73.19	62.53	51.23	39.83	29.35	62.55	51.23	39.77	29.01	62.52	51.20	39.77	28.92
$\mathbf{S}^{2}_{8/255}$	**1.6822M**	**76.54**	**66.64**	**55.11**	**42.90**	**31.83**	**66.65**	**55.08**	**42.73**	**31.27**	**66.61**	**55.10**	**42.71**	**31.26**

Table 3 Supernet transferability on number of attack steps with the term of accuracy on SVHN (%)

Supernet training	Subnet model size	Clean Acc	$\mathbf{P}^{10}_{2/255}$	$\mathbf{P}^{10}_{4/255}$	$\mathbf{P}^{10}_{6/255}$	$\mathbf{P}^{10}_{8/255}$	$\mathbf{P}^{30}_{2/255}$	$\mathbf{P}^{30}_{4/255}$	$\mathbf{P}^{30}_{6/255}$	$\mathbf{P}^{30}_{8/255}$	$\mathbf{P}^{50}_{2/255}$	$\mathbf{P}^{50}_{4/255}$	$\mathbf{P}^{50}_{6/255}$	$\mathbf{P}^{50}_{8/255}$
$\mathbf{S}^{10}_{8/255}$	1.7657M	87.26	72.65	68.17	60.82	50.91	72.27	68.09	60.81	50.64	72.21	68.04	50.35	50.24
$\mathbf{S}^{1}_{8/255}$	1.7428M	88.34	71.53	66.45	59.83	49.53	71.52	66.31	58.80	48.56	71.51	66.21	58.70	49.51
$\mathbf{S}^{5}_{8/255}$	**1.6836M**	**90.54**	**76.28**	**70.16**	**62.73**	**54.70**	**76.15**	**70.15**	**62.11**	**54.69**	**76.10**	**70.15**	**62.09**	**54.68**

Table 4 Supernet transferability on Epsilon size with the term of accuracy on CIFAR-10 (%)

Supernet training	Subnet model size	Clean Acc	$\mathbf{P}^{10}_{2/255}$	$\mathbf{P}^{10}_{4/255}$	$\mathbf{P}^{10}_{6/255}$	$\mathbf{P}^{10}_{8/255}$	$\mathbf{P}^{30}_{2/255}$	$\mathbf{P}^{30}_{4/255}$	$\mathbf{P}^{30}_{6/255}$	$\mathbf{P}^{30}_{8/255}$	$\mathbf{P}^{50}_{2/255}$	$\mathbf{P}^{50}_{4/255}$	$\mathbf{P}^{50}_{6/255}$	$\mathbf{P}^{50}_{8/255}$
$\mathbf{S}^{10}_{8/255}$	1.7453M	73.18	62.62	52.07	40.91	**31.01**	62.60	52.05	40.83	**30.68**	62.60	52.04	40.77	**30.65**
$\mathbf{S}^{10}_{6/255}$	1.6840M	78.63	**67.79**	**54.85**	**41.584**	29.03	**67.76**	**54.80**	**41.42**	28.19	**67.78**	**54.80**	**41.32**	28.09
$\mathbf{S}^{10}_{4/255}$	1.6775M	75.82	62.63	46.87	31.87	20.27	62.65	46.85	31.65	19.62	62.62	46.86	31.65	19.55
$\mathbf{S}^{10}_{2/255}$	**1.6632M**	**81.95**	62.14	38.52	20.29	8.823	62.15	38.21	19.60	7.675	62.13	38.18	19.54	7.606

Table 5 Supernet transferability on Epsilon size with the term of accuracy on SVHN (%)

Supernet training	Subnet model size	Clean Acc	$\mathbf{P}^{10}_{2/255}$	$\mathbf{P}^{10}_{4/255}$	$\mathbf{P}^{10}_{6/255}$	$\mathbf{P}^{10}_{8/255}$	$\mathbf{P}^{30}_{2/255}$	$\mathbf{P}^{30}_{4/255}$	$\mathbf{P}^{30}_{6/255}$	$\mathbf{P}^{30}_{8/255}$	$\mathbf{P}^{50}_{2/255}$	$\mathbf{P}^{50}_{4/255}$	$\mathbf{P}^{50}_{6/255}$	$\mathbf{P}^{50}_{8/255}$
$\mathbf{S}^{10}_{8/255}$	1.7657M	87.26	72.65	68.17	60.82	**50.91**	72.27	68.09	60.81	**50.64**	72.21	68.04	60.35	**50.24**
$\mathbf{S}^{10}_{6/255}$	1.6947M	88.45	**72.37**	**69.46**	**61.47**	49.87	**72.26**	**69.32**	**61.42**	49.82	**72.15**	**69.27**	**61.32**	49.85
$\mathbf{S}^{10}_{4/255}$	1.6873M	85.82	62.37	58.47	52.43	42.70	62.35	58.37	51.49	41.50	62.30	58.21	51.37	40.61
$\mathbf{S}^{10}_{2/255}$	**1.6712M**	**91.14**	60.14	50.66	39.38	21.50	59.88	49.53	38.27	20.17	58.96	48.18	37.12	20.54

Table 6 Supernet transferability on model size with the term of accuracy on CIFAR-10 (%)

Supernet training	Subnet model size	Clean Acc	$\mathbf{P}^{10}_{2/255}$	$\mathbf{P}^{10}_{4/255}$	$\mathbf{P}^{10}_{6/255}$	$\mathbf{P}^{10}_{8/255}$	$\mathbf{P}^{30}_{2/255}$	$\mathbf{P}^{30}_{4/255}$	$\mathbf{P}^{30}_{6/255}$	$\mathbf{P}^{30}_{8/255}$	$\mathbf{P}^{50}_{2/255}$	$\mathbf{P}^{50}_{4/255}$	$\mathbf{P}^{50}_{6/255}$	$\mathbf{P}^{50}_{8/255}$
$\mathbf{S}^{10}_{8/255}$	1.7453M	73.18	62.62	52.07	40.91	31.01	62.60	52.05	40.83	30.68	62.60	52.04	40.77	30.65
$\mathbf{S}^{10}_{6/255}$	1.6840M	**78.63**	**67.79**	54.85	41.584	29.03	**67.76**	54.80	41.42	28.19	**67.78**	54.80	41.32	28.09
$\mathbf{S}^{2}_{8/255}$	**1.6822M**	76.54	66.64	**55.11**	**42.90**	**31.83**	66.65	**55.08**	**42.73**	**31.27**	66.61	**55.10**	**42.71**	**31.26**

Table 7 Supernet transferability on model size with the term of accuracy on SVHN (%)

Supernet training	Subnet model size	Clean Acc	$\mathbf{P}^{10}_{2/255}$	$\mathbf{P}^{10}_{4/255}$	$\mathbf{P}^{10}_{6/255}$	$\mathbf{P}^{10}_{8/255}$	$\mathbf{P}^{30}_{2/255}$	$\mathbf{P}^{30}_{4/255}$	$\mathbf{P}^{30}_{6/255}$	$\mathbf{P}^{30}_{8/255}$	$\mathbf{P}^{50}_{2/255}$	$\mathbf{P}^{50}_{4/255}$	$\mathbf{P}^{50}_{6/255}$	$\mathbf{P}^{50}_{8/255}$
$\mathbf{S}^{10}_{8/255}$	1.7453M	73.18	62.62	52.07	40.91	31.01	62.60	52.05	40.83	30.68	62.60	52.04	40.77	30.65
$\mathbf{S}^{10}_{6/255}$	1.6947M	88.45	72.37	69.46	61.47	49.87	72.26	69.32	61.42	49.82	72.15	69.27	61.32	49.85
$\mathbf{S}^{5}_{8/255}$	**1.6836M**	**90.54**	**76.28**	**70.16**	**62.73**	**54.70**	**76.15**	**70.15**	**62.11**	**54.69**	**76.10**	**70.15**	**62.09**	**54.68**

Table 8 The width impact on supernet with the term of accuracy on CIFAR-10 (%)

Supernet training	Subnet model size	Clean Acc	$\mathbf{P}^{10}_{2/255}$	$\mathbf{P}^{10}_{4/255}$	$\mathbf{P}^{10}_{6/255}$	$\mathbf{P}^{10}_{8/255}$	$\mathbf{P}^{30}_{2/255}$	$\mathbf{P}^{30}_{4/255}$	$\mathbf{P}^{30}_{6/255}$	$\mathbf{P}^{30}_{8/255}$	$\mathbf{P}^{50}_{2/255}$	$\mathbf{P}^{50}_{4/255}$	$\mathbf{P}^{50}_{6/255}$	$\mathbf{P}^{50}_{8/255}$
$\mathbf{S}^{1}_{8/255}$	**1.7574M**	73.19	62.53	51.23	39.83	29.35	62.55	51.23	39.77	29.01	62.52	51.20	39.77	28.92
$\mathbf{2S}^{1}_{8/255}$	2.1878M	**79.87**	**69.38**	**56.81**	**43.61**	**31.73**	**69.38**	**56.71**	**43.51**	**31.12**	**69.37**	**56.76**	**43.44**	**31.09**
$\mathbf{4S}^{10}_{8/255}$	–	–	–	–	–	–	–	–	–	–	–	–	–	–
$\mathbf{8S}^{10}_{8/255}$	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Table 9 The width impact on supernet with the term of accuracy on SVHN (%)

Supernet training	Subnet model size	Clean Acc	$\mathbf{P}^{10}_{2/255}$	$\mathbf{P}^{10}_{4/255}$	$\mathbf{P}^{10}_{6/255}$	$\mathbf{P}^{10}_{8/255}$	$\mathbf{P}^{30}_{2/255}$	$\mathbf{P}^{30}_{4/255}$	$\mathbf{P}^{30}_{6/255}$	$\mathbf{P}^{30}_{8/255}$	$\mathbf{P}^{50}_{2/255}$	$\mathbf{P}^{50}_{4/255}$	$\mathbf{P}^{50}_{6/255}$	$\mathbf{P}^{50}_{8/255}$
$\mathbf{S}^{5}_{8/255}$	**1.6836M**	90.54	76.28	70.16	62.73	54.70	76.15	70.15	62.11	54.69	76.10	70.15	62.09	54.68
$\mathbf{2S}^{1}_{8/255}$	2.1676M	**91.87**	**79.80**	**73.45**	**65.31**	**57.43**	**79.71**	**73.42**	**65.27**	**57.38**	**79.70**	**73.40**	**65.18**	**57.32**
$\mathbf{4S}^{10}_{8/255}$	–	–	–	–	–	–	–	–	–	–	–	–	–	–
$\mathbf{8S}^{10}_{8/255}$	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Table 10 Difference between training from scratch and fine-tuning with this term of accuracy

Supernet training	Clean Acc			$\mathbf{P}^{10}_{2/255}$			$\mathbf{P}^{10}_{4/255}$			$\mathbf{P}^{10}_{6/255}$			$\mathbf{P}^{10}_{8/255}$		
	avg	min	max	avg	min	max	avg	min	max	avg	min	max	avg	min	max
$\mathbf{S}^{10}_{8/255}$	11.50	6.907	15.33	12.20	8.273	16.32	10.12	6.787	14.92	9.472	5.235	13.65	8.414	2.011	12.55
$\mathbf{S}^{10}_{4/255}$	**9.614**	4.589	18.12	**10.46**	5.839	18.64	**9.746**	3.423	13.12	**7.412**	2.663	10.082	**5.391**	1.849	8.910
$\mathbf{S}^{2}_{8/255}$	17.08	9.782	**25.94**	17.04	10.15	**25.35**	15.02	6.464	**21.09**	12.16	3.429	**18.83**	9.044	0.338	**16.32**

optimizer we use here is SGD. We use a batch size of 512, a momentum of 0.9, and a weight decay of $5e-3$. We evaluate our model on white-box l_{inf} bounded PGD attack with different epsilon values and step-sizes. The epsilon for evaluation ranges from $2/255$ to $8/255$, and its interval is $2/255$. The number of PGD attack steps for evaluation ranges from 10 to 50. The hyperparameter of the fine-tuning method is the same as mentioned above. Fine-tuning is to inherit the weights from the supernet for each subnet as initialization, while training from scratch is to initialize the weights of each subnet randomly.

Supernet transferability

In this section, we aim to understand whether the use of a weaker PGD attack for adversarial training of the supernet will considerably deteriorate the adversarial learning performance of subnets. Specifically, we adjust the degree of the PGD attack by changing the number of attack steps and the epsilon size. To begin with, we build up a baseline for our best subnet in the CIFAR-10 and SVHN dataset, which is presented in the first row of Tables 2 and 3. The first column denotes how we train the supernet. For instance, the subscript of $\mathbf{S}^{10}_{8/255}$ is the epsilon size of the PGD attack for the supernet, which is set to $8/255$. And the superscript is the number of attack steps, which is set to 10. The last column denotes the best adversarial accuracy of the subnet model under different degrees of attacks. For example, $\mathbf{P}^{10}_{8/255}$ means the subnet is under the PGD attack with an epsilon size of $8/255$ and ten attack steps. Since we focus on the network architecture under adversarial attacks, the subnet presented in the following tables is the non-dominated architecture that achieves the best adversarial performance after trained from scratch.

Number of attack steps

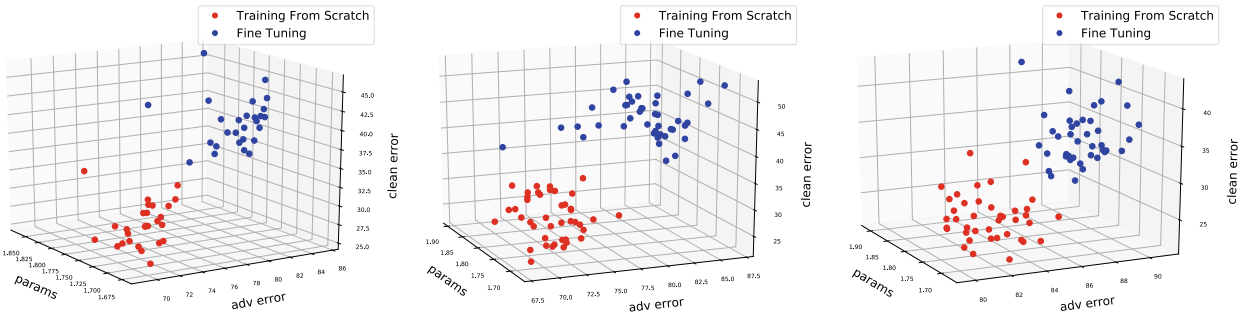
Table 2 indicates that the subnet performs better even if the supernet is under fewer attack steps during the adversarial training. The third row $S^{2}_{8/255}$ denotes the supernet is under white-box PGD attack with two steps in adversarial training, and the second row $S^{1}_{8/255}$ denotes the supernet is under white-box PGD attack with 1 step in adversarial training. In comparison with $S^{10}_{8/255}$, $S^{2}_{8/255}$ helps to increase the adversarial accuracy and clean accuracy by 6.4% and 4.5%, respectively, and reduce its subnet model size by 3.6%. In addition, the gap between $S^{1}_{8/255}$ and our baseline $S^{10}_{8/255}$ is very tiny for different objectives. Therefore, we surmise that the supernet will have strong transferability even if we reduce the attack steps for its adversarial training. Moreover, we can easily observe that the adversarial accuracy of subnets is strongly affected by the epsilon size but not the number of attack steps. For example, the difference of the adversarial accuracy $P^{10}_{2/255}$, $P^{30}_{2/255}$ and $P_{2/255}$ is very small regardless how the supernet is trained. It meets the same conclusion in Table 3. Therefore, we think that this observation not only helps us to save much more training time by reducing the number of attack steps but also makes it possible for the subnets to obtain stronger adversarial defensive ability.

Epsilon size

Table 4 indicates that the supernet can improve its subnets' representation abilities if it is not overloaded with the epsilon size. First, the subnet of $S^{10}_{2/255}$ achieves the highest clean accuracy up to 81.95%, but the adversarial accuracy of its subnet under $P^{10}_{8/255}$, $P^{30}_{8/255}$, $P^{50}_{8/255}$ attacks is unable to surpass 9%. Our assumption is that the subnet has not fully developed its resilience ability, since its supernet is incapable of learning by generating adversarial examples with a larger epsilon size. The assumption has been verified that the subnet of $S^{10}_{4/255}$ and $S^{10}_{6/255}$ hugely increase the adversarial accuracy when it is under $P^{10}_{8/255}$ attack. Specifically, when in comparison $S^{10}_{4/255}$ with $S^{10}_{2/255}$, we find that the adversarial performance of their subnets under $P^{10}_{8/255}$, $P^{30}_{8/255}$, $P^{50}_{8/255}$ increase by 146.2%, 156.4%, and 157.0%. Another assumption is that if the epsilon size exceeds the supernet's workload, it will reduce both the clean accuracy and adversarial accuracy of subnets. We can easily observe that the subnet of $S^{10}_{6/255}$ obtains the best adversarial performance. In comparison, $S^{10}_{8/255}$ largely weakens its subnet's performance regardless of clean accuracy, or under $P^{10}_{8/255}$, $P^{10}_{6/255}$, $P^{10}_{4/255}$ attacks. Specifically, when comparing the subnet of $S^{10}_{6/255}$ with $S^{10}_{8/255}$, the clean accuracy of subnet is increased by 7.447% and adversarial accuracy by 1.63%, 5.338%, and 8.256% under $P^{10}_{6/255}$, $P^{10}_{4/255}$, and $P^{10}_{2/255}$ attacks. However, when the epsilon size exceeds $6/255$ in our case, the subnet performance begins to decline gradually. It meets the same conclusion in Table 5.

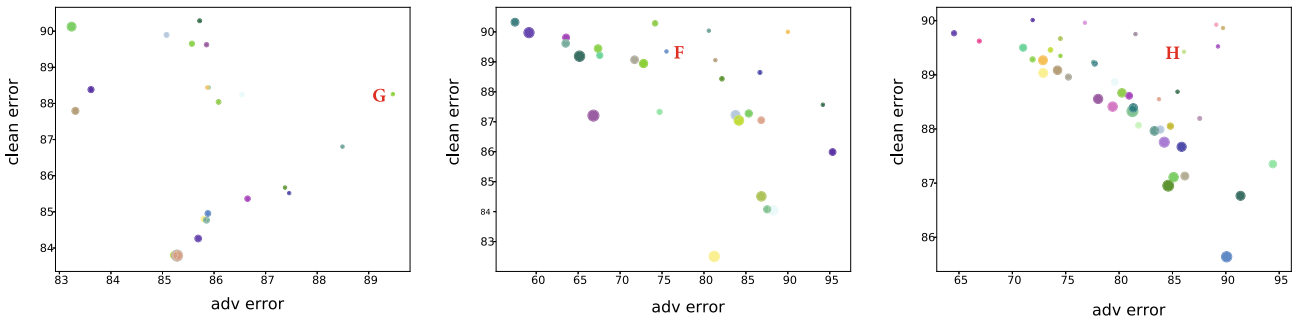
Model size

Table 6 shows that the adversarial performance of the model may not be closely correlated with the model size, which is inconsistent with the conclusion in Ref. [3]. When comparing $S^{2}_{8/255}$ with our baseline model $S^{10}_{8/255}$, the subnet model size drops by 3.751% but the clean accuracy and adversarial accuracy in $P^{10}_{8/255}$, $P^{10}_{6/255}$, $P^{10}_{4/255}$, and $P^{10}_{2/255}$ separately increase by 4.591%, 2.642%, 4.864%, 5.838%, and 6.420%. When comparing with the subnet of $S^{10}_{6/255}$ and $S^{10}_{8/255}$, we find that the best subnet size drops by 3.640%, but the clean accuracy and the adversarial accuracy in $P^{10}_{6/255}$, $P^{10}_{4/255}$, and $P^{10}_{2/255}$ separately increase by 7.447%, 1.63%, 5.338%, and 8.256%. We can achieve the same result in Table 7. Therefore, we think that the performance of the subnet, including



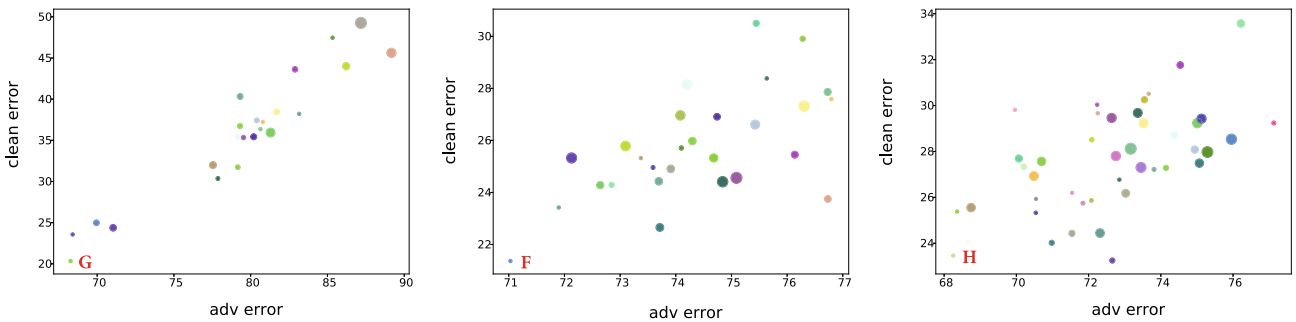
(a) $S^{10}_{8/255}$ (b) $S^{2}_{8/255}$ (c) $S^{10}_{4/255}$

Fig. 5 Difference between training from scratch and fine-tuning. The red points represent the solutions on the non-dominated front obtained by NSGA-II, where the weights for subnets are randomly initialized. By contrast, the blue points are different solutions achieved by the same way, but the weights of the subnets are inherited from supernet



(a) $2S^{1}_{8/255}$ (b) $S^{10}_{6/255}$ (c) $S^{2}_{8/255}$

Fig. 6 The first non-dominated front before training from scratch among the adversarial error, clean error, and number of parameters of subnets on CIFAR-10. The size of circle indicates the number of parameter of subnets. The vertical axis represents the clean error of the subnets, while the horizontal axis represents the adversarial error of the subnets



(a) $2S^{1}_{8/255}$ (b) $S^{10}_{6/255}$ (c) $S^{2}_{8/255}$

Fig. 7 The first non-dominated front after trained from scratch on CIFAR-10. The size of circle indicates the number of parameters of the subnets. The vertical axis represents the clean error of the subnets, while the horizontal axis represents the adversarial error of the subnets. The same color of circles in Fig. 7 means that they hold the same network architecture

the clean accuracy and adversarial accuracy, has a strong correlation with the training mode of its supernet but not the mode size of itself.

Width impact of supernet

Table 8 indicates that the width of the supernet makes a large impact on the defensive ability of subnets. However, it does

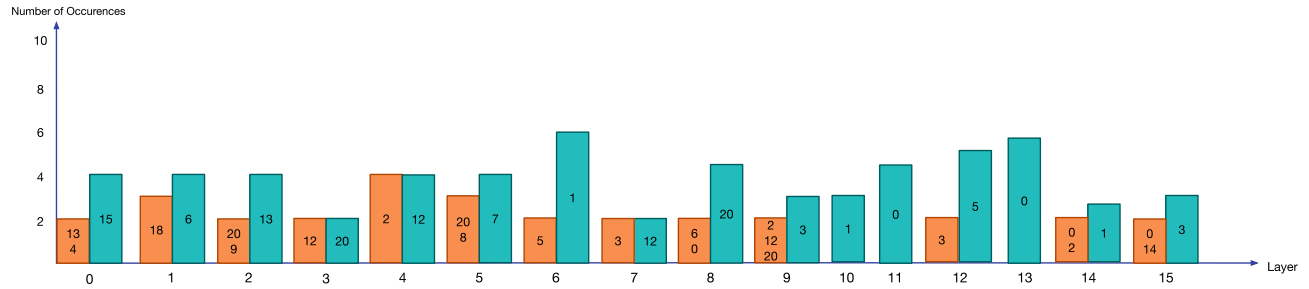


Fig. 8 Block statistics for each layer of the top ten subnets on CIFAR-10. The horizontal axis represents the layer IDs of the subnets. For instance, 0 means the first layer on the subnets. The vertical axis represents which two blocks are the most frequently adopted in a certain layer. For instance, block ID 15 is the most frequently used in the first layer of the top ten subnets. Moreover, block ID 13 and block ID 4 are the second most frequently used in the first layer

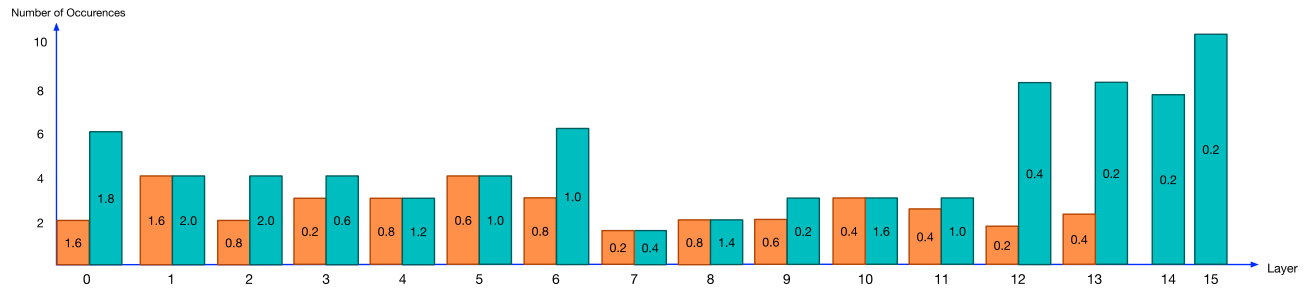


Fig. 9 Channel statistics for each layer of the top ten subnets on CIFAR-10. The horizontal axis represents the layer id of subnets. For instance, 0 means the first layer on the subnets. The vertical axis represents which two-channel expansion ratio is the most frequently used in certain layers. For instance, channel expansion ratio 1.8 is the most frequently used in the first layer of the top ten subnets and channel expansion ratio 1.6 is the second most frequently used in the first layer

not mean that an infinite large width of a neural network can improve the adversarial performance of subnets. Specifically, we try to enlarge the channels of each layer of the supernet, and the expansion ratio ranges from [1, 2, 4, 8], but the channel selection remains the same ranging from 0 to 2, and the interval is 0.2. $2S^1_{8/255}$ denotes that we double the channel size of each layer of our supernet, and we use PGD attack with an epsilon size of 8/255 and 1 step to generate adversarial examples for the supernet training. $2S^1_{8/255}$ increases the adversarial accuracy by 10.9% and clean accuracy by 9.1% in comparison with $S^1_{8/255}$. Even though we double the channel size of each layer of our supernet, theoretically the model size of the subnet of $2S^1_{8/255}$ should be twice as large as the subnet of $S^1_{8/255}$. However, the model size of the subnet of $2S^1_{8/255}$ just increases by 24.4% in comparison with the subnet model size of $S^1_{8/255}$, which proves that our NAS framework is very efficient. In addition, we also try to enlarge the channel size of the neural network by a factor of 4 and 8, but we find that it is too difficult to converge. It meets the same conclusion in Table 9. Therefore, it is hard to say that there is a linear correlation between the width of the supernet and its adversarial performance.

Training from scratch or fine-tuning

We can easily observe there exists a huge gap between training from scratch and fine-tuning from Fig. 5. The three axes for each graph represent the number of parameters, the adversarial error, and the clean error for each subnet, respectively. The subscript for each graph in Fig. 5 denotes the supernet training mode. The red points represent the solutions on the non-dominated front obtained by NSGA-II, where the weights for subnets are randomly initialized. By contrast, the blue points are different solutions achieved in the same way, but the weights of the subnets are inherited from the supernet. For example, Fig. 5a presents the distribution of subnets for training from scratch and fine-tuning. And the PGD attack of their supernet in Fig. 5b is with the epsilon size of 8/255 and ten steps. We can clearly observe that no matter how we train the supernet, the subnets which are trained from scratch perform better on adversarial examples and non-adversarial examples. Table 10 quantifies the gap between training from scratch and fine-tuning under PGD attacks with different epsilon sizes and attack steps. The first column of Table 10 indicates the training mode of the supernet. The second column of Table 10 shows the difference in the subnets' performance on non-adversarial examples, which

Table 11 Block encoding scheme and layer statistics on CIFAR-10

	Architecture																				
Block	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
S	✓	✓	✓				✓	✓	✓				✓	✓	✓							
SX				✓	✓	✓				✓	✓	✓				✓	✓	✓				
NE							✓	✓	✓	✓	✓	✓							✓	✓		
NG													✓	✓	✓	✓	✓	✓			✓	✓
BN																			✓		✓	
K=3	✓			✓			✓			✓			✓			✓						
K=5		✓			✓			✓			✓			✓			✓					
K=7			✓			✓			✓			✓			✓			✓				
Layer	**Statistics**																					
0	1				2						1			2		**4**						
1				2			**4**										1		3			
2								1		2				**4**							2	1
3	1	1			1	1							**2**					1		1	**2**	
4			**4**		2								**4**									
5								**4**	3												3	
6		**4**	1			2				1			1							1		
7	1	1		**3**		1			1				**3**									
8	2						2			1											**5**	
9			2	**3**									2								2	1
10		**3**	1	1			1					1		1					1		1	
11	**4**		1							1			1	1				1				1
12	**5**	1	1	2														1				
13	**6**	1	1	1					1													
14	2	**3**	2									1	1			1						
15	2			**4**				1			1				2							

Shuffle+SE: S, Shuffle-Xception+SE: SX, Non-Local-EmbeddedGaussian+SE: NE, Non-Local-Gaussian+SE: NG, BatchNorm: BN; K: kernel size

adopt training from scratch and fine-tuning, respectively. The minimum of average value is 9.614 in $S^{10}_{4/255}$ and the maximum average value is 25.94 in $S^{2}_{8/255}$. Therefore, it also clearly indicates that training from scratch surpasses fine-tuning by at least 20% in terms of theb clean accuracy. As for the adversarial performance, the minimum average is 5.391 in ($S^{10}_{4/255}$, $P^{10}_{8/255}$). In other words, the model trained from scratch surpasses the model with fine-tuning by 48% for the adversarial accuracy, because the average value of adversarial performance in ($S^{10}_{4/255}$, $P^{10}_{8/255}$) is 11.23. We hypothesize that the role of the supernet in our NAS framework is to find the best architecture for the subnet but not to deliver the best weights to the subnet. Fine-tuning is not always beneficial to the training. The reason may be that the weights of each newly sampled subnet are not good enough, as there are only 20 training epoches. However, we can find the best subnets among them by means of NSGA-II during the optimization in terms of the objectives.

Subnets' analysis

This section analyzes the top ten subnet architectures in terms of the clean accuracy and adversarial accuracy and their corresponding supernet training method on CIFAR-10 and SVHN. Our aim is to gain insights from our top best results and reveal the rule for how to design a more robust tiny neural network.

Adversarial error, clean error, and the size of neural network

Figure 6 shows the non-dominated subsets obtained by NSGA-II on the CIFAR-10 dataset and their corresponding supernets are $2S^{1}_{8/255}$, $S^{10}_{6/255}$, and $S^{2}_{8/255}$, respectively. We use circles to represent the subnets, and the size of the circle indicates its size (number of parameters). In Fig. 6b and c, it can be easily observed that there does exist a trade-off relationship between the adversarial error, clean error, and size of neural networks. Figure 7 shows that the order of non-dominated subnets has greatly changed after trained from

scratch. To clearly illustrate how the order of non-dominated subnets changes after trained from scratch, each subnet (circle) is denoted by a different color. The same color circles in Figs. 6 and 7 indicate that they own the same network architecture. We get an important observation from Figs. 6 and 7 that most tiny neural networks (tiny circles) achieve a significant reduction on both the adversarial error and clean error after trained from scratch. For instance, the G point in Fig. 7a, which owns the lowest clean error and lowest adversarial error, has a larger adversarial error and clean error before training from scratch. It also meets the same observation for the F point when we compare with Figs. 6a and 7b. Hence, we conclude that our pipeline can effectively increase the adversarial accuracy and clean accuracy of tiny neural networks.

Block and channel analysis

We make another statistics analysis of which block and channel expansion ratio are used most frequently in the top ten subnets. The best trade-off tiny neural network architectures can be viewed as a combinatorial optimization problem. Namely, each layer of network architecture could be viewed as the combination of specific blocks and channels. The optimum network architecture could be viewed as different layers of neural network combinatorial optimization problems.

Table 11 shows our block encoding scheme and layer statistics. The upper part of Table 11 explains how to build up a certain block for each block identifier. For instance, Block 0 comprises the ShuffleV2 block and the SE layer. The kernel size of Block 0 is 3. The specific block internal composition can be referred to in Fig. 3a. The lower part of Table 11 shows the number of occurrences for a certain block in each layer. For example, the 15th block represents four times in the first layer (layer 0) of our top ten subnets, while the 13th and 2nd blocks represent 2 times in the first layer (layer 0). In addition, Fig. 8 shows which two blocks are the most frequently adopted in a certain layer. The data in Fig. 8 come from Table 11. For our block encoding scheme, blocks with fewer than eight are denoted as pure tiny blocks, i.e., they are meant to make our neural network tinier. Block identifiers between 9 and 17 are denoted as tiny robust blocks and used to enhance the robustness of the tiny blocks. Block identifiers larger than 18 denote pure robust blocks, i.e., they are meant to increase the adversarial performance of neural networks. From Fig. 8, we can observe the trend that the top trade-off tiny neural networks prefer to use robust or tiny robust blocks in the first four layers, while the last eight layers prefer to adopt pure tiny blocks. We surmise that since the PGD attack mainly focuses on pixel-wise perturbations, the robust blocks can mitigate the attack effect in the first several layers. The tiny blocks can help the neural network to keep a balance between the clean performance and the number of parameters.

Table 12 Channel encoding scheme and layer statistics on CIFAR-10

	Architecture									
Channel	0	1	2	3	4	5	6	7	8	9
0.2	✓									
0.4		✓								
0.6			✓							
0.8				✓						
1.0					✓					
1.2						✓				
1.4							✓			
1.6								✓		
1.8									✓	
2.0										✓
Layer	**Statistics**									
0	1					1		2	6	
1			1			1		4		4
2		1		2	1		1		1	4
3	3	1	6							
4	1		1	3		2			1	
5	1		4		4		1			
6			5	4	1					
7	2	2	1		1	2	1	1		
8	1		1	3		2	3			
9	4	1	3			1			1	
10	1	4			1			4		
11	1	3	1	1	3		1			
12	2	8								
13	6	3	1							
14	7	1	1	1						
15	10									

Table 12 shows our channel expansion ratio encoding scheme and the layer statistics. The upper part of Table 12 introduces the relationship between the encoding channel identifier and the channel expansion ratio. The lower part of Table 12 shows the number of occurrences for a certain channel expansion ratio. For example, the channel expansion ratios 1.8 and 1.6 represent six times and two times in the first layer of the top 10 subnets, respectively. Figure 9 shows which two channel expansion ratios are used most frequently in a certain layer. From Fig. 9, we can observe that the top trade-off tiny neural networks prefer to adopt larger channels in the first three layers, and then, the channel number for the rest of the layers gradually declines. The top ten tiny subnets use the smallest channel number in the last three layers. Our explanation is that since the PGD attack mainly focuses on pixel-wise perturbations, the wider channel in the first sev-

Table 13 The Lego subnet experiments on CIFAR-10 (%)

Supernet training	Subnet model size	Clean Acc	$\mathbf{P}^{10}_{2/255}$	$\mathbf{P}^{10}_{4/255}$	$\mathbf{P}^{10}_{6/255}$	$\mathbf{P}^{10}_{8/255}$	$\mathbf{P}^{30}_{2/255}$	$\mathbf{P}^{30}_{4/255}$	$\mathbf{P}^{30}_{6/255}$	$\mathbf{P}^{30}_{8/255}$	$\mathbf{P}^{50}_{2/255}$	$\mathbf{P}^{50}_{4/255}$	$\mathbf{P}^{50}_{6/255}$	$\mathbf{P}^{50}_{8/255}$
LEGO $\mathbf{2S}^{l}_{8/255}$	**2.1546M**	**80.495**	**70.64**	**59.16**	**46.70**	**35.02**	**70.65**	**59.14**	**46.60**	**34.51**	**70.67**	**59.12**	**46.61**	**34.42**
$\mathbf{2S}^{l}_{8/255}$	2.1878M	79.87	69.38	56.81	43.61	31.73	69.38	56.71	43.51	31.12	69.37	56.76	43.44	31.09

Table 14 The Lego subnet experiments on SVHN (%)

Supernet training	Subnet model size	Clean Acc	$\mathbf{P}^{10}_{2/255}$	$\mathbf{P}^{10}_{4/255}$	$\mathbf{P}^{10}_{6/255}$	$\mathbf{P}^{10}_{8/255}$	$\mathbf{P}^{30}_{2/255}$	$\mathbf{P}^{30}_{4/255}$	$\mathbf{P}^{30}_{6/255}$	$\mathbf{P}^{30}_{8/255}$	$\mathbf{P}^{50}_{2/255}$	$\mathbf{P}^{50}_{4/255}$	$\mathbf{P}^{50}_{6/255}$	$\mathbf{P}^{50}_{8/255}$
LEGO $\mathbf{2S}^{5}_{8/255}$	**2.1593M**	**92.81**	**82.81**	**76.54**	**68.81**	**61.88**	**82.79**	**76.51**	**68.75**	**61.55**	**82.75**	**76.48**	**68.73**	**61.53**
$\mathbf{2S}^{l}_{8/255}$	2.1676M	91.87	79.80	73.45	65.31	57.43	79.71	73.42	65.27	57.38	79.70	73.40	65.18	57.32

Table 15 Benchmark on CIFAR-10 (%)

Supernet training	Subnet model size	Clean Acc	$\mathbf{P}^{20}_{8/255}$	$\mathbf{P}^{100}_{8/255}$	$\mathbf{P}^{100}_{2/255}$
RobNet-small [30]	4.41M	78.05	**48.32**	**48.07**	–
PreAct ResNet-18 [40]	3.42M	60.78	28.02	27.69	28.01
MORNAS [9]	3.56M	59.98	34.56	34.50	–
LEGO $\mathbf{2S}^{1}_{8/255}$	2.1546M	**80.50**	35.01	34.41	**70.66**
$\mathbf{2S}^{1}_{8/255}$	2.1878M	79.87	31.73	31.09	69.35
$\mathbf{S}^{10}_{6/255}$	1.6840M	78.63	29.02	28.07	67.75
$\mathbf{S}^{2}_{8/255}$	**1.6822M**	76.54	31.80	31.25	66.35

eral layers can help mitigate the adversarial attacks, and the gradually declining channel number can reduce the size of the neural network.

Assumption verification

To verify our assumption, the most frequently used blocks and channels of each layer in Figs. 8 and 9 are selected as the block and channel choices, respectively, in our model. We call this new subnet architecture as Lego-Net. Table 13 shows that Lego-Net performs better than our state-of-the-art subnet $2S^{1}_{8/255}$, especially in adversarial performance. Specifically, Lego-Net can increase adversarial accuracy by 10.36% in $P^{10}_{8/255}$ and clean accuracy by 0.78%, while the size of Lego-Net drops by 1.52%. We also achieve the same result on the SVHN dataset, as shown in Table 14. To conclude, we should put pure robust or tiny robust blocks in the shallow layers and pure tiny blocks in the rest of the layers to build up tiny robust neural networks. In terms of channel design, we should put wider intermediate channels in the shallow layers and gradually reduce the intermediate channels in the rest of the layers.

Main comparisons

We discuss the difference between Guo et al. [30] and our work. First, our backbone supernet uses ShuffleNetV2 [15], which is one of the typical tiny neural networks, while Guo et al. [30] employ ResNet [5] as the backbone of supernet. Moreover, the authors [30] design the connection of blocks as their search space, and there are only three choices (3×3 separable convolution, identity, and zero) for their candidate blocks. However, our work re-designs the block search spaces, i.e., we try to incorporate the attention mechanism into tiny blocks. We not only provide 22 block choices, including pure tiny blocks, pure robust blocks, and tiny robust blocks, but also design the new channel search space for each block. We aim to find the optimum blocks and channels for each layer of tiny neural network. Our work pays more attention to balancing the clean error, adversarial error, and the neural network size when we design our search space and the supernet. Second, although the authors [30] claim that their NAS algorithm is one-shot, they use different computational budgets: small, medium, and large to search different sizes of neural networks. In other words, they at least searched three times to get different sizes of neural networks. Our work is a real one-shot NAS algorithm, as the multi-objective optimization employed in our method is able to generate diverse models with different structures in the one-shot. We have considered the balance between the clean error, the adversarial error, and the neural network size in our search process.

From Table 15, we can clearly see that the clean accuracy of LEGO $2S^{1}_{8/255}$ increases by 3.1% and the size of the neural network drops by 104.7%. Although our LegoNet cannot compete with RobNet-small for adversarial performance, we surmise that increasing the size of neural networks would improve the robustness. Furthermore, we hypothesize that the epsilon size (the pixel perturbation range) in reality is not as high as 8 pixels, and the most common attack should be light-weight perturbation. Therefore, we try to reduce the epsilon size from 8/255 to 2/255, and we find that the gap between the adversarial performance and clean performance for our LegoNet is only 14%. Therefore, we conclude that our LegoNet can keep the balance between adversarial accuracy, clean accuracy, and neural network size.

Conclusion

We propose a tiny adversarial multi-objective one-shot neural network search framework, which aims to find the best trade-off networks in terms of the adversarial error, the clean error, and the size of the neural network. Our study reveal several observations on how the adversarial training method of the supernet will affect the subnets' adversarial performance. We also hint at how to design tiny robust neural networks based on our block and channel statistics. Furthermore, we conduct experiments to empirically verify our hypothesis on improving the robustness of neural networks without significantly reducing the clean accuracy and enlarging the neural network size. However, the proposed TAM-NAS framework still has a clear limitation,, i.e., the performance of the subnets heavily relies on the network architecture of the supernets. Therefore, in the future, we will develop a co-evolutionary multi-objective NAS framework, where the architecture of the supernet will also evolve during the search process.

Acknowledgements This work is supported by the National Natural Science Foundation of China under Grant No. 62136003, 61972188, 62122035, 62206122, and 62103150, and the China Postdoctoral Science Foundation under Grant No. 2021M691012. Y. Jin is funded by an

Alexander von Humboldt Professorship for AI endowed by the German Federal Ministry of Education and Research.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Christian S, Zaremba W, Sutskever I, Joan B, Dumitru E, Ian G, Rob F (2014) Intriguing properties of neural networks
2. Goodfellow IJ, Shlens J, Szegedy C (2015) Explaining and harnessing adversarial examples
3. Madry A, Makelov A, Schmidt L, Tsipras D, Vladu A (2018) Towards deep learning models resistant to adversarial attacks
4. Cubuk ED, Zoph B, Schoenholz SS, Le Quoc V (2018) Intriguing properties of adversarial examples
5. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 770–778
6. Su D, Zhang H, Chen H, Yi J, Chen P-Y, Gao Y (2018) Is robustness the cost of accuracy? A comprehensive study on the robustness of 18 deep image classification models. In: Proceedings of the European conference on computer vision (ECCV), pp 631–648
7. Xie C, Yuille A (2020) Intriguing properties of adversarial training
8. Huang H, Wang Y, Erfani SM, Quanquan G, James B, Xingjun M (2021) Exploring architectural ingredients of adversarially robust deep neural networks. In: Neural information processing systems
9. Liu J, Jin Y (2021) Multi-objective search of robust neural architectures against multiple types of adversarial attacks. Neurocomputing 453:73–84
10. Großhans M, Sawade C, Brückner M, Scheffer T (2013) Bayesian games for adversarial regression problems. In: International conference on machine learning, pp 55–63
11. Tsipras D, Santurkar S, Engstrom L, Turner A, Madry A (2019) Robustness may be at odds with accuracy
12. Jin Y, Sendhoff B (2008) Pareto-based multiobjective machine learning: an overview and case studies. In: IEEE transactions on systems, man, and cybernetics, Part C (applications and reviews) 38(3):397–415
13. Lu Z, Whalen I, Boddeti VN, Dhebar YD, Deb K, Goodman ED, Banzhaf W (2018) Nsga-net: neural architecture search using multi-objective genetic algorithm. In: Proceedings of the genetic and evolutionary computation conference
14. Lu Z, Whalen I, Dhebar YD, Deb K, Goodman ED, Wolfgang B, Vishnu NB (2019) Multiobjective evolutionary design of deep convolutional neural networks for image classification. IEEE Trans Evol Comput 25:277–291
15. Ma N, Zhang X, Zheng H-T, Sun J (2018) Shufflenet v2: practical guidelines for efficient cnn architecture design. In: Proceedings of the European conference on computer vision (ECCV), pp 116–131
16. Chollet F (2017) Xception: deep learning with depthwise separable convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 1251–1258
17. Hu J, Shen L, Sun G (2018) Squeeze-and-excitation networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 7132–7141
18. Wang X, Girshick R, Gupta A, He K (2018) Non-local neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 7794–7803
19. Myerson RB (2013) Game theory. Harvard University Press, Cambridge
20. Daskalakis C, Goldberg PW, Papadimitriou CH (2009) The complexity of computing a nash equilibrium. SIAM J Comput 39(1):195–259
21. Zhang D, Zhang T, Lu Y, Zhu Z, Dong B (2019) You only propagate once: accelerating adversarial training via maximal principle. 32
22. Kannan H, Kurakin A, Goodfellow I (2018) Adversarial logit pairing. arXiv preprint arXiv:1803.06373
23. Zhang H, Yu Y, Jiao J, Xing EP, Ghaoui LE, Jordan MI (2019) Theoretically principled trade-off between robustness and accuracy
24. Xie L, Yuille A (2017) Genetic cnn. In: Proceedings of the IEEE international conference on computer vision, pp 1379–1388
25. Real E, Aggarwal A, Huang Y, Le QV (2019) Regularized evolution for image classifier architecture search. Proc AAAI Conf Artif Intell 33:4780–4789
26. Zoph B, Vasudevan V, Shlens J, Le Quoc V (2018) Learning transferable architectures for scalable image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 8697–8710
27. Pham H, Guan MY, Zoph B, Le QV, Dean J (2018) Efficient neural architecture search via parameter sharing. 80:4092–4101
28. Liu H, Simonyan K, Yiming Y (2019) Darts: differentiable architecture search
29. Dong X, Yang Y (2019) One-shot neural architecture search via self-evaluated template network. In: Proceedings of the IEEE international conference on computer vision, pp 3681–3690
30. Guo M, Yang Y, Xu R, Liu Z, Lin D (2020) When nas meets robustness: In search of robust architectures against adversarial attacks. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp 631–640
31. Bender G, Kindermans P-J, Zoph B, Vasudevan V, Le Q (2018) Understanding and simplifying one-shot architecture search. In: International conference on machine learning, pp 550–559
32. Deb K, Pratap A, Agarwal S, Meyarivan TAMT (2002) A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Trans Evol Comput 6(2):182–197
33. Huang S, Lu Z, Deb K, Boddeti VN (2022) Revisiting residual networks for adversarial robustness: an architectural perspective. arXiv preprint arXiv:2212.11005
34. Krizhevsky A, Hinton G et al (2009) Learning multiple layers of features from tiny images. Master's thesis, Department of Computer Science, University of Toronto
35. Netzer Y, Wang T, Coates A, Bissacco A, Wu B, Ng AY (2011) Reading digits in natural images with unsupervised feature learning. NIPS workshop on deep learning and unsupervised feature learning
36. Russakovsky O, Deng J, Hao S, Krause J, Satheesh S, Ma S, Huang Z, Karpathy A, Khosla A, Bernstein M, Berg AC, Fei-Fei L (2015) ImageNet large scale visual recognition challenge. Int J Comput Vis (IJCV) 115(3):211–252
37. Buades A, Coll B, Morel J-M (2005) A non-local algorithm for image denoising. In: 2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05), vol 2, pp 60–65. IEEE

38. Guo Z, Zhang X, Haoyuan M, Heng W, Liu Z, Wei Y, Sun J (2020) Single path one-shot neural architecture search with uniform sampling. 12361:544–560
39. Kopp RE (1962) Pontryagin maximum principle. In: Mathematics in science and engineering, vol 5, pp 255–279. Elsevier
40. He K, Zhang X, Ren S, Sun J (2016) Identity mappings in deep residual networks, pp 116–131