# Zen-NAS: A Zero-Shot NAS for High-Performance Image Recognition

Ming Lin [*1], Pichao Wang [†1], Zhenhong Sun [‡2], Hesen Chen [§2], Xiuyu Sun [¶2], Qi Qian [‖1], Hao Li [**2], and Rong Jin [††1]

[1]Alibaba Group, Bellevue, Washington, USA
[2]Alibaba Group, Hangzhou, Zhejiang, China

March 29, 2021

**Abstract**

Accuracy predictor is a key component in Neural Architecture Search (NAS) for ranking architectures. Building a high-quality accuracy predictor usually costs enormous computation. To address this issue, instead of using an accuracy predictor, we propose a novel zero-shot index dubbed Zen-Score to rank the architectures. The Zen-Score represents the network expressivity and positively correlates with the model accuracy. The calculation of Zen-Score only takes a few forward inferences through a randomly initialized network, without training network parameters. Built upon the Zen-Score, we further propose a new NAS algorithm, termed as Zen-NAS, by maximizing the Zen-Score of the target network under given inference budgets. Within less than half GPU day, Zen-NAS is able to directly search high performance architectures in a data-free style. Comparing with previous NAS methods, the proposed Zen-NAS is magnitude times faster on multiple server-side and mobile-side GPU platforms with state-of-the-art accuracy on ImageNet. Our source code and pre-trained models are released on `https://github.com/idstcv/ZenNAS`.

## 1 Introduction

The design of high-performance deep neural networks is a challenging task. Neural Architecture Search (NAS) methods facilitate this progress. There are mainly two key components, architecture generator and accuracy predictor, in existing NAS algorithms. The generator proposes potential high-performance networks and the predictor predicts their accuracies. Popular generators include uniform sampling [13], evolutionary algorithm [39] and reinforcement learning [29]. The accuracy predictors include brute-force methods [40, 55, 3, 39], predictor-based methods [29, 54, 28] and one-shot methods [25, 58, 66, 59, 55, 56, 6, 63, 52, 5].

[*]ming.l@alibaba-inc.com, https://minglin-home.github.io
[†]pichao.wang@alibaba-inc.com
[‡]zhenhong.szh@alibaba-inc.com
[§]hesen.chs@alibaba-inc.com
[¶]xiuyu.sxy@alibaba-inc.com
[‖]qi.qian@alibaba-inc.com
[**]lihao.lh@alibaba-inc.com
[††]jinrong.jr@alibaba-inc.com

A major challenge of building a high-quality accuracy predictor is the enormous computational cost. Both brute-forced methods and predictor-based methods require to train considerable number of networks. The one-shot methods reduce the training cost via parameter sharing. Albeit being more efficient than brute-forced methods, the one-shot methods still need to train a huge supernet which is still computationally expensive. Recent studies also find that nearly all supernet-based methods suffer from model interfering [5, 60] which degrades the quality of accuracy predictor [44]. In addition, since the supernet must be much larger than the target network, it is difficult to search large target networks under limited resources. These issues make the one-shot methods struggling in designing high-performance networks.

To solve these problems, instead of using an expensive accuracy predictor, we propose an almost zero-cost proxy, dubbed Zen-Score, for efficient NAS. The Zen-Score measures the expressivity [37, 30] of a deep neural network and positively correlates with the model accuracy. The computation of Zen-Score only takes a few forward inferences on randomly initialized network using random Gaussian inputs, making it extremely fast, lightweight and data-free. Moreover, Zen-Score deals with the scale-sensitive problem caused by Batch Normalization (BN)[4, 33], making it widely applicable to real-world problems.

Based on Zen-Score, we design a novel Zen-NAS algorithm. It maximizes the Zen-Score of the target network within inference budgets. Zen-NAS is a **Zero-Shot** method since it does not optimize network parameters during search [1]. We apply Zen-NAS to search optimal networks under various inference budgets, including inference latency, FLOPs (Floating Point Operations) and model size, and achieve the state-of-the-art (SOTA) performance on CIFAR10/CIFAR100/ImageNet, outperforming previous human-designed and NAS-designed models by a large margin. Zen-NAS is the first zero-shot method that achieves SOTA results on large-scale full-resolution ImageNet-1k dataset [12] by the time of writing this work [31, 1, 7].

Our approach is inspired by recent advances in deep learning studies [32, 11, 23, 37, 9, 27, 30, 42, 45, 14, 57] which show that deep models are superior than shallow ones since deep models are more expressive under the same number of neurons. According to the bias-variance trade-off in statistical learning theory [19], increasing the expressivity of a deep network implies smaller bias error. When the size $n$ of training dataset is large enough, the variance error will diminish as $\mathcal{O}(1/\sqrt{n}) \to 0$. This means that the generalization error is dominated by the bias error which could be reduced by more expressive networks. These theoretical results are well-aligned with large-scale deep learning practices[34, 50, 35].

We summarize our main contributions as follows:

- We propose a novel zero-shot proxy Zen-Score for NAS. The proposed Zen-Score is computationally efficient and is proved to be scale-insensitive in present of BN.
- A novel NAS algorithm termed Zen-NAS is proposed to search for networks with maximal Zen-Score in the searching space.
- Within half GPU day, the ZenNets designed by Zen-NAS achieve up to $83.6\%$ top-1 accuracy on ImageNet that is as accurate as EfficientNet-B5. The inference speed of ZenNets is magnitude times faster than EfficientNets of the same or comparable accuracies on multiple hardware platforms.
- To our best knowledge, Zen-NAS is the first zero-shot method that outperforms non-zero-shot methods on ImageNet. ZenNet-1.2ms is the first machine-designed network with accuracy comparable to EfficientNet-B5 on ImageNet while being 10 times faster on NVIDIA T4 GPU.

## 2 Related Work

We briefly review the related works, and for comprehensive review of NAS, the monograph [41] is referred to.

[1] Obviously, the final searched architecture must be trained on the target dataset before deployment.

In the early days of NAS, brute-force methods are adopted to search architectures by directly training the network to obtain its accuracy. For example, the AmoebaNet [39] conducts structure search on CIFAR10 using Evolutionary Algorithm (EA) [20] and then transfers the structure to ImageNet. It takes about 3150 GPU days of searching and achieves $74.5\%$ top-1 accuracy on ImageNet. Inspired by the success of AmoebaNet, many EA-based NAS algorithms are proposed to improve searching efficiency, such as EcoNAS [66], CARS [59], GeNet [55] and PNAS [24]. These methods search on down-sampled images or reduce the number of queries. Reinforced Learning is another popular generator (sampler) in NAS, including NASNet [67], Mnasnet [47] and MetaQNN [3].

Both EA and RL based methods require lots of network training. To address this problem, the predictor-based methods encode architectures into high dimensional vectors. A number of architectures are trained to obtain their accuracies [29, 28] and then are used as training data for learning accuracy predictor. The one-shot methods further reduce the training cost by training a big supernet. This framework is widely applied in many efficient NAS methods, including DARTS [25], SNAS [56], PC-DARTS [58], ProxylessNAS [6], GDAS [63], FBNetV2 [52], DNANet [21], Single-Path One-Shot NAS [13].

Although the above efforts have greatly reduced the searching cost, their top-1 accuracies on ImageNet are below $80.0\%$. The authors of OFANet [5] noted that weight-sharing suffers from model interfering. They propose a progressive-shrinking strategy to address this issue. The resultant OFANet achieves $80.1\%$ accuracy after searching for 51.6 GPU days. EfficientNet [48] is another high precision network designed by NAS. It takes about 3800 GPU days to search EfficientNet-B7 whose accuracy is $84.4\%$. In comparison, Zen-NAS achieves $83.6\%$ accuracy while using magnitude times fewer resources.

A few on-going works are actively exploring zero-shot proxies for efficient NAS. However, these efforts have not delivered the SOTA results. In a recent empirical study, [1] evaluates the performance of six zero-shot pruning proxies on NAS benchmark datasets. The synflow [49] achieves best resluts in their experiments. We compare synflow with Zen-Score under fair settings and show that Zen-Score achieves $+1.1\%$ better accuracy on CIFAR10 and $+8.2\%$ better accuracy on CIFAR100. The concurrent work TE-NAS [7] uses a combination of NTK-score and network expressivity as NAS proxy. Specifically, the TE-NAS estimates the expressivity by directly counting the number of active regions $R_N$ on randomly sampled images. In comparison, Zen-score not only considers the distribution of linear regions but also considers the Gaussian complexity of linear classifier in each linear region, giving a more accurate estimation of network expressivity. The computation of Zen-Score is 80 to 180 times faster than TE-NAS score. In terms of performance, TE-NAS achieves $74.1\%$ top-1 accuracy on ImageNet, lagging behind SOTA baselines. Zen-NAS achieves $+9.5\%$ better accuracy within similar searching cost and outperforms previous SOTA methods.

# 3 Expressivity of Vanilla Network

In this section, we discuss how to measure the expressivity of *vanilla convolutional neural network* (VCNN) family, an ideal prototype for theoretical studies. We show that the expressivity of a network could be efficiently measured by its expected Gaussian complexity, or $\Phi$-score for short. In the next section, we further show that for very deep networks, directly computing $\Phi$-score incurs numerical overflow. This overflow could be addressed by adding BN layers and then re-scaling the $\Phi$-score by a constant. This new score is named as Zen-Score in Section 4.

### 3.1 Notations

An $L$-layer neural network is formulated as a function $f : \mathbb{R}^{m_0} \to \mathbb{R}^{m_L}$ where $m_0$ is the input dimension and $m_L$ is the output dimension. $\boldsymbol{x}_0 \in \mathbb{R}^{m_0}$ denotes the input image. Correspondingly, the output feature map of the $t$-th layer is denoted by $\boldsymbol{x}_t$. The $t$-th layer has $m_{t-1}$ input channels and $m_t$ output channels. The convolutional kernel is $\boldsymbol{\theta}_t \in \mathbb{R}^{m_t \times m_{t-1} \times k \times k}$. The image resolution is $H \times W$. The mini-batch size is $B$. The Gaussian distribution of mean $\mu$ and variance $\sigma^2$ is denoted by $\mathcal{N}(\mu, \sigma)$.

### 3.2 Vanilla Convolutional Neural Network

The **vanilla convolutional neural network** (VCNN) is a widely used prototype in theoretical studies [37, 45, 14]. The main body of a vanilla network is stacked by multiple convolutional layers. Each layer consists of one convolutional operator followed by RELU activation functions. All other components are removed from backbone, including residual link and Batch Normalization. After the main body, global average pool layer (GAP) reduces the feature map resolution to 1x1, followed by a fully-connected layer. At the end a softmax operation converts the network output to label prediction. Given the input $\boldsymbol{x}$ and network parameters $\boldsymbol{\theta}$, $f(\boldsymbol{x}|\boldsymbol{\theta})$ refers to the output of the main body of the network, that is the feature map before the GAP layer (pre-GAP layer). We measure the network expressivity with respect to pre-GAP because it contains most of the information we need.

Modern networks use auxiliary structures such as residual link , Batch Normalization and self-attention block [16]. These structures will not significantly affect the representation power of networks. Therefore, these structures are temporarily removed when measuring network expressivity and then added back in training and testing stages. For non-RELU activation functions, they are replaced by RELU in a similar way. These simple modifications make our method applicable to a majority of non-VCNN models widely used in practice. In fact, nearly all single-branch networks can be converted to vanilla network by the aforementioned modifications.

### 3.3 $\Phi$-Score as Proxy of Expressivity

Given a VCNN $f(\boldsymbol{x}|\boldsymbol{\theta})$, we propose a novel numerical index $\Phi$-score as a proxy of its expressivity. The definition of $\Phi$-score is inspired by recent theoretical studies on deep network expressivity [45, 57]. A key observation in these studies is that a vanilla network can be decomposed into piece-wise linear functions conditioned on activation patterns [32]:

**Lemma 1** ([32, 30])**.** *Denote the activation pattern of the $t$-th layer as $\mathcal{A}_t(\boldsymbol{x})$. Then for any vanilla network $f(\cdot)$,*

$$f(\boldsymbol{x}|\boldsymbol{\theta}) = \sum_{\mathcal{S}_i \in \mathcal{S}} \mathcal{I}_{\boldsymbol{x}}(\mathcal{S}_i) \boldsymbol{W}_{\mathcal{S}_i} \boldsymbol{x} \tag{1}$$

*where $\mathcal{S}_i$ is a convex polytope depending on $\{\mathcal{A}_1(\boldsymbol{x}), \mathcal{A}_2(\boldsymbol{x}), \cdots, \mathcal{A}_L(\boldsymbol{x})\}$; $\mathcal{S}$ is a finite set of convex polytopes in $\mathbb{R}^{m_0}$; $\mathcal{I}_{\boldsymbol{x}}(\mathcal{S}_i) = 1$ if $\boldsymbol{x} \in \mathcal{S}_i$ otherwise zero; $\boldsymbol{W}_{\mathcal{S}_i}$ is a coefficient matrix of size $\mathbb{R}^{m_L \times m_0}$.*

According to Lemma 1, any vanilla network is an ensemble of piece-wise linear functions segmented by convex polytopes $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, \cdots, \mathcal{S}_{|\mathcal{S}|}\}$ where $|\mathcal{S}|$ is the number of linear-regions. Figure 1 illustrates such linear-region convex polytopes of a simple vanilla network with tow-dimensional input and randomly initialized parameters.

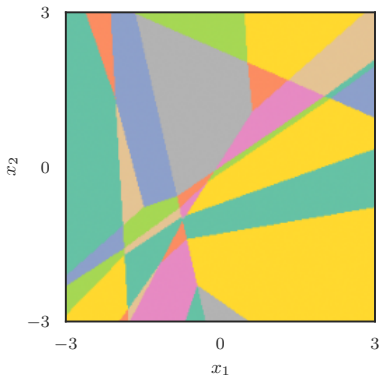



Figure 1: Illustration of linear regions of a fully-connected vanilla network. The input of the network is a two-dimensional vector $\boldsymbol{x} = [x_1, x_2]$. Different color represents different activation pattern. The network has 4 layers and each layer has 4 channels. All neurons are initialized by standard Gaussian distribution. See Figure 2 in [14] for more details.

The number of linear regions $|\mathcal{S}|$ has been used as expressivity proxy in several theoretical studies [32, 45, 14, 64, 57]. However, directly using $|\mathcal{S}|$ incurs two limitations: a) Counting $|\mathcal{S}|$ for large network is computationally infeasible; b) The representation power of each $W_{\mathcal{S}_i}$ is not considered in the proxy. The first limitation is due to fact that the number of linear regions grow exponentially for large networks [32, 57]. To understand the second limitation, we recall the Gaussian complexity [18] of linear classifiers:

**Lemma 2** ([18]). *For linear function class $\{f : f(X) = WX \text{ s.t. } \|W\|_F \leq G\}$, its Gaussian complexity is upper bounded by $\mathcal{O}(G)$.*

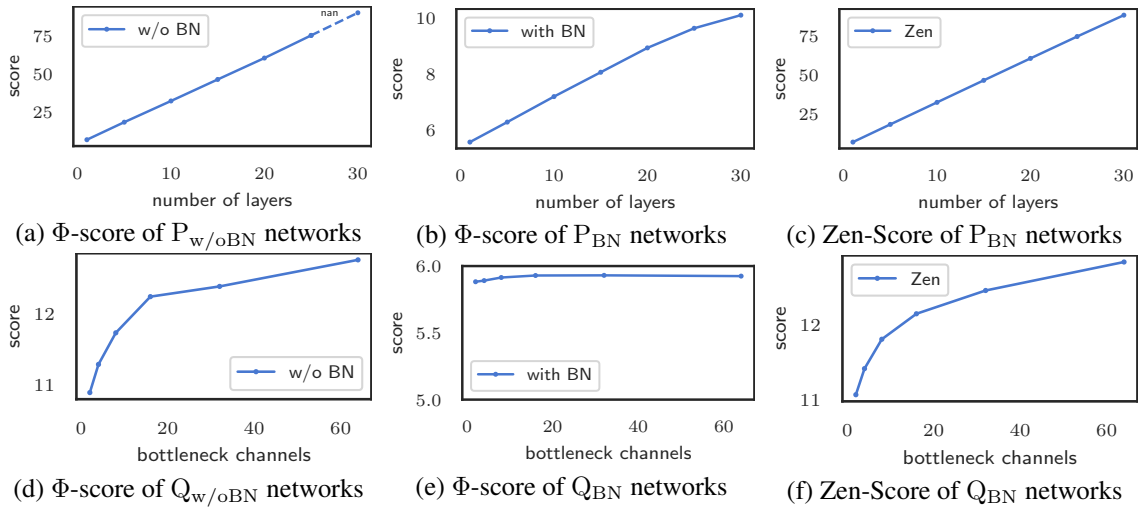
In other words, Lemma 2 says that the expressivity of linear function class, which could be measured by Gaussian complexity, is controlled by the Frobenius norm of its parameter matrix $W$. Inspired by Lemma 1 and Lemma 2, we define the following index for measuring network expressivity :

**Definition 1** ($\Phi$-score for VCNN). *The expected Gaussian complexity for a vanilla network $f(\cdot)$ is defined by*

$$\Phi(f) = \log \mathbb{E}_{\boldsymbol{x},\boldsymbol{\theta}} \left\{ \sum_{\mathcal{S}_i \in \mathcal{S}} \mathcal{I}_{\boldsymbol{x}}(\mathcal{S}_i) \|\boldsymbol{W}_{\mathcal{S}_i}\|_F \right\} \tag{2}$$

$$= \log \mathbb{E}_{\boldsymbol{x},\boldsymbol{\theta}} \|\nabla_{\boldsymbol{x}} f(\boldsymbol{x}|\boldsymbol{\theta})\|_F. \tag{3}$$

In Definition 1, we measure the network expressivity by its expected Gaussian complexity, or $\Phi$-score for short. Since any VCNN is ensemble of linear functions, it is nature to measure its expressivity by averaging the Gaussian complexity of linear function in each linear region. To this end, we randomly sample $\boldsymbol{x}$ and $\boldsymbol{\theta}$ from some prior distributions and then average $\|W_{\mathcal{S}_i}\|_F$. This is equivalent to compute the expected gradient norm of $f$ with respect to input $\boldsymbol{x}$. In our implementation, $\boldsymbol{x}$ and $\boldsymbol{\theta}$ are sampled from standard Gaussian distribution which works well in practice. It is important to note that in $\Phi$-score, only the gradient of $\boldsymbol{x}$ rather than $\boldsymbol{\theta}$ is involved. This is different to zero-cost proxies in [1] which compute gradient of $\boldsymbol{\theta}$ in their formulations. These proxies measure the trainability [53, 7] instead of the expressivity of networks.




(a) Φ-score of $\mathrm{P}_{\mathrm{w/oBN}}$ networks (b) Φ-score of $\mathrm{P}_{\mathrm{BN}}$ networks (c) Zen-Score of $\mathrm{P}_{\mathrm{BN}}$ networks

(d) Φ-score of $\mathrm{Q}_{\mathrm{w/oBN}}$ networks (e) Φ-score of $\mathrm{Q}_{\mathrm{BN}}$ networks (f) Zen-Score of $\mathrm{Q}_{\mathrm{BN}}$ networks

Figure 2: Φ-scores and Zen-Scores of networks, with different depths and bottleneck channels.

# 4 Zen-Score and Zen-NAS

In this section, we show that directly computing Φ-score for very deep networks incurs numerical overflow due to the gradient explosion without BN layers. The gradient explosion could be resolved by adding BN layers back but the Φ-score will be adaptively re-scaled, making it difficult to compare Φ-score between different networks. The same phenomenon has been known as 'scale-sensitive' problem in deep learning complexity analysis [4, 33]. To address this open question, we propose to re-scale the Φ-score one more time by the product of BN layers' variance statistics. This new score is denoted as Zen-Score in order to distinguish from the original Φ-score. The Zen-Score is proven to be scale-insensitive. Finally, we present Zen-NAS algorithm built on Zen-Score and demonstrate its effectiveness in the next section.

## 4.1 Overflow and BN-rescaling

When computing Φ-score for very deep vanilla networks, numerical overflow incurs almost surely. This is because BN layers are removed from the network and the magnitude of network output grows exponentially along depth. To see this, we construct a set of vanilla networks $P_{\mathrm{w/oBN}}$ without BN layers. All networks have the same widths but different depths. Figure 2(a) plots the Φ-scores for $P_{\mathrm{w/oBN}}$. After 30 layers, Φ-score overflows. To address the overflow, we add BN layers back and compute the Φ-scores in Figure 2(b). This time the overflow dismisses but the Φ-scores are scaled-down by a large factor. This phenomenon is termed as BN-rescaling because the Φ-score of a large network is re-scaled to a small value.

To demonstrate that BN-rescaling disturbs architecture ranking, we construct another two set of networks, $Q_{\mathrm{w/oBN}}$ and $Q_{\mathrm{BN}}$, with and without BN respectively. All networks have two layers and have the same number of input and final output channels. The number of bottleneck channels, that is the width of the hidden layer, varies from 2 to 60. The corresponding Φ-score curves are plotted in Figure 2(d) and (e) respectively. When BN layer is presented, the Φ-score becomes nearly constant for all networks. This will confuse the architecture generator and drive the search to a wrong direction.

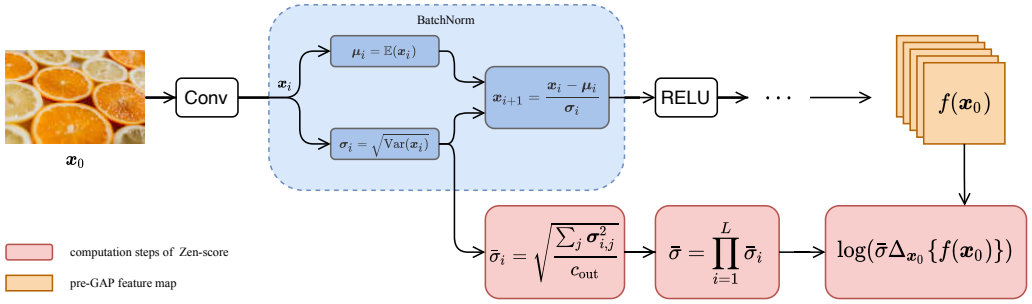



Figure 3: Zen-Score computational graph. $\boldsymbol{x}_0$ is one mini-batch of input images. For each BN layer, we extract its mini-batch deviation parameter $\boldsymbol{\sigma}_i$. $\Delta_{\boldsymbol{x}_0}\{f(\boldsymbol{x}_0)\}$ is the differential of pre-GAP feature map $f(\boldsymbol{x}_0)$ with respect to $\boldsymbol{x}_0$.

## 4.2 From $\Phi$-Score to Zen-Score

**Algorithm 1** Zen-Score

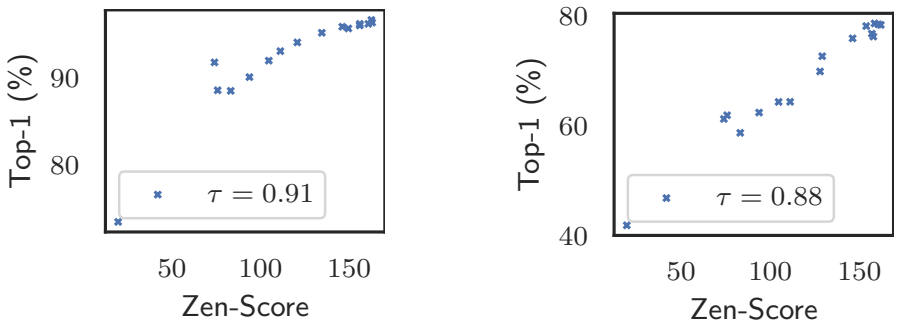
**Require:** Network $\mathcal{F}(\cdot)$ with pre-GAP feature map $f(\cdot)$; $\alpha = 0.01$.
**Ensure:** Zen-Score $\text{Zen}(\mathcal{F})$.

1: Remove all residual links in $\mathcal{F}$.
2: Initialize all neurons in $\mathcal{F}$ by $\mathcal{N}(0, 1)$.
3: Sample $\boldsymbol{x}, \boldsymbol{\epsilon} \sim \mathcal{N}(0, 1)$.
4: Compute $\Delta \triangleq \mathbb{E}_{\boldsymbol{x},\boldsymbol{\epsilon}} \|f(\boldsymbol{x}) - f(\boldsymbol{x} + \alpha\boldsymbol{\epsilon})\|_F$ .
5: For the $i$-th BN layer with $m$ output channels, compute $\bar{\sigma}_i = \sqrt{\sum_j \boldsymbol{\sigma}_{i,j}^2 / m}$ where $\boldsymbol{\sigma}_{i,j}$ is the mini-batch standard deviation statistic of the $j$-th channel in BN.
6: $\text{Zen}(F) \triangleq \log(\Delta) + \sum_i \log(\bar{\sigma}_i)$.

In the above subsection, we showed that BN layer is necessary to prevent numerical overflow when computing $\Phi$-score but comes with the side-effect of re-scaling $\Phi$-score. In this subsection, we design a new Zen-Score which is able to calibrate re-scaling when BN layer is present. The computation of Zen-Score is described in Algorithm 1. Figure 3 visualizes the computational graph of Algorithm 1.

In Algorithm 1, all residual links are removed from the network as pre-processing. Then we randomly sample input vectors and perturb them with Gaussian noise. The perturbation of the pre-GAP feature map is denoted as $\Delta$ in Line 4. This step replaces the gradient of $\boldsymbol{x}$ with finite differential $\Delta$ to avoid backward-propagation. To get Zen-Score, the scaling factor $\bar{\sigma}_i^2$ is averaged from the variance of each channel in BN layer. Finally, the Zen-Score is computed by the log-sum of $\Delta$ and $\bar{\sigma}_i$. The following theorem guarantees that the Zen-Score of network with BN layers approximates the $\Phi$-score of the same network without BN layers. The proof is postponed to Supplementary F.

**Theorem 1.** *Let $\bar{f}(\boldsymbol{x}_0) = \bar{\boldsymbol{x}}_L$ be an $L$-layer vanilla network without BN layers. $f(\boldsymbol{x}_0) = \boldsymbol{x}_L$ is its sister network with BN layers. For some constants $0 < \delta < 1$, $K_0 \leq \mathcal{O}[\sqrt{\log(1/\delta)}]$, when $BHW \geq \mathcal{O}[(LK_0)^2]$*




(a) CIFAR10 (b) CIFAR100

Figure 4: Zen-Score v.s. top-1 accuracy, 16 randomly sampled structures generated from ResNet-50, with Kendall's $\tau$-score between accuracy and Zen-Score.

*is large enough, with probability at least* $1-\delta$*, we have*

$$(1-L\epsilon)^2 \leq \frac{(\prod_{t=1}^{L}\bar{\sigma}_t^2)\mathbb{E}_\theta\{\|\boldsymbol{x}_L\|^2\}}{\mathbb{E}_\theta\|\bar{\boldsymbol{x}}_L\|^2} \leq (1+L\epsilon)^2 \tag{4}$$

*where* $\epsilon \triangleq \mathcal{O}(2K_0/\sqrt{BHW})$.

Informally speaking, Theorem 1 says that to compute $\|\bar{f}(\cdot)\|$, we only need to compute $\|f(\cdot)\|$ then re-scale with $\prod_{t=1}^{L}\bar{\sigma}_t$. The approximation error is bounded by $L\epsilon$. By taking gradient of $\boldsymbol{x}$ on both $\bar{f}(\cdot)$ and $f(\cdot)$, we obtain the desired relationship between Zen-Score and $\Phi$-score.

### 4.3 Effectiveness of Zen-Score

We show that Zen-Score positively correlates with model accuracy. We use ResNet-50 for CIFAR as initial structure. Then we randomly perturb the structure, for example, shrinking or expanding channels, removing or adding layers. In this way, we randomly sample 16 structures and then train them on CIFAR10/CIFAR100. We plot the top-1 accuracy v.s. Zen-Score in Figure 4. The Zen-Scores positively correlate to the network accuracies, especially in high-precision regimes.

### 4.4 Zen-NAS For Maximizing Expressivity

We design Zen-NAS algorithm to maximize the Zen-Score of the target network. The step-by-step description of Zen-NAS is given in Algorithm 2. The Zen-NAS uses Evolutionary Algorithm (EA) as architecture generator. It is possible to choose other generators such as Reinforced Learning or even greedy selection. The choice of EA is due to its simplicity.

In Algorithm 2, we randomly generate $N$ structures. At each iteration step $t$, we randomly select a structure in the population $\mathcal{P}$ and mutate it. The mutation algorithm is presented in Algorithm 3. The search space has several types of blocks. With probability $\rho$, we randomly select a block in the structure and change the type of this block. We encourage the random exploration in the first half $T/2$ iterations by setting $\rho$ decaying from 1 to 0.1. Then the width and depth of the selected layer is mutated in a given range. We choose $[0.5, 2.0]$ as the mutation range in this work, that is, within half or double of the current value. The

**Algorithm 2** Zen-NAS

**Require:** Search space $\mathcal{S}$, inference budget $B$, total number of iterations $T$, evolutionary population size $N$, initial structure $F_0$.
**Ensure:** NAS-designed ZenNet $F^*$.

Initialize population $\mathcal{P} = \{F_0\}$.
**for** $t = 1, 2, \cdots, T$ **do**
  **if** $t < T/2$ **then**
    $\lambda = (T - 2t)/T$, $\rho = \lambda + 0.1(1 - \lambda)$.
  **else**
    $\rho = 0.1$.
  **end if**
  Randomly select $F_t \in \mathcal{P}$.
  Mutate $\hat{F}_t = \text{MUTATE}(F_t, \rho, \mathcal{S})$
  **if** $\hat{F}_t$ exceeds inference budget **then**
    Do nothing.
  **else**
    Get Zen-Score $z = \text{Zen}(\hat{F}_t)$.
    Append $\hat{F}_t$ to $\mathcal{P}$.
  **end if**
  Remove network of the smallest Zen-Score if the size of $\mathcal{P}$ exceeds $B$.
**end for**
Return $F^*$, the network of the highest Zen-Score in $\mathcal{P}$.

**Algorithm 3** MUTATE

**Require:** Structure $F_t$, probability $\rho$ of mutation, search space $\mathcal{S}$.
**Ensure:** Randomly mutated structure $\hat{F}_t$.

Uniformly select a block $h$ in $F_t$.
Draw $u$ from $[0, 1]$-uniform distribution.
**if** $u < \rho$ **then**
  Uniformly select a basic block type from $\mathcal{S}$.
  Replace $h$ with new block type.
**end if**
Uniformly alternate the width and depth of $h$ within some range.
Return the mutated structure $\hat{F}_t$.

new structure $\hat{F}_t$ is appended to the population if its inference cost does not exceed the budget. Finally, we maintain the population size by removing networks with smallest Zen-Scores. After $T$ iterations, the network with the largest Zen-Score is returned as the output of Zen-NAS. We name the resulting architectures as ZenNets.

# 5 Experiments

In this section, experiments on CIFAR10/CIFAR100 [20] and ImageNet-1k [12] are conducted to validate the superiority of Zen-NAS. We first compare Zen-Score to several zero-shot proxies on CIFAR10 and

| proxy | CIFAR10 | CIFAR100 |
|---|---|---|
| Zen-Score | **96.2%** | **80.1%** |
| FLOPs | 93.1% | 64.7% |
| grad | 92.8% | 65.4% |
| synflow | 95.1% | 75.9% |
| TE-Score | 96.1% | 77.2% |
| Random | 93.5±0.7% | 71.1±3.1% |

Table 1: Top-1 accuracies on CIFAR10/CIFAR100 for five zero-shot proxies. Budget: model size $N \leq 1\,\mathrm{M}$. 'Random': average accuracy $\pm$ std for random search.

CIFAR100, using the same search space, search policy and training settings. Then we compare Zen-NAS to the state-of-the-art methods on ImageNet. More SOTA models on CIFAR10/CIFAR100 can be found in Supplementary D. Finally, we compare the searching cost of Zen-NAS with SOTA methods in subsection 5.3.

The inference speed on NVIDIA T4 and Google Pixel2 is reported in Supplementary C. The Zen-Scores of ResNets and accuracies under fair training settings are reported in Supplementary E. We enclose one big performance table of networks on ImageNet in Supplementary G.

To align with previous works, we consider the following two search spaces:

- **Search Space I** Following [15, 38], this search space consists of residual blocks and bottleneck blocks defined in ResNet. The number of bottleneck channels is searchable.
- **Search Space II** Following [43, 36], this search space consists of MobileNet blocks. The depth-wise expansion ratio is searched in set $\{1, 2, 4, 6\}$.

Please see Supplementary A for datasets description and detail experiment settings.

In each trial, the initial structure is a randomly selected small network which is guaranteed to satisfy the inference budget. The kernel size is searched in set $\{3, 5, 7\}$. Following conventional designs, the number of stages is three for CIFAR10/CIFAR100 and five for ImageNet. The evolutionary population size is 256, number of evolutionary iterations $T = 96,000$. The resolution is 32x32 for CIFAR10/CIFAR100 and 224x224 for ImageNet.

## 5.1 Zen-Score v.s. Other Zero-Shot Proxies

Following [1, 7], we compare Zen-Score to four zero-shot proxies: FLOPs, gradient-norm (grad) of network parameters, synflow [49] and TE-NAS score (TE-Score) [7]. For each proxy, we replace Zen-Score by that proxy in Algorithm 2 and then run Algorithm 2 for $T = 96,000$ iterations to ensure convergence. Since synflow is the smaller the better, we use its negative value in Algorithm 2. Following convention, we search for best network on CIFAR10/CIFAR100 within model size $N \leq 1\,\mathrm{M}$. The convergence curves are plotted in Supplementary C, Figure 9, 10, 11, 12, 13. In these figures, all five scores improves monotonically along iterations.

After the above NAS step, we train the network of the best score for each proxy under the same training setting. To provide a random baseline, we randomly generate networks. The width of the layer varies in range $[4, 512]$, and the depth of each stage varies in range $[1, 10]$. If the network size is larger than $1\,\mathrm{M}$, we

| proxy | model | N | time | speed-up |
|---|---|---|---|---|
| TE-Score | ResNet18 | 16 | 4.38 | 1.0x |
| | ResNet50 | 16 | 20.5 | 1.0x |
| Zen-Score | ResNet18 | 16 | 0.05 | 87.6x |
| | ResNet50 | 16 | 0.15 | 136.6x |

Table 2: Time cost of computing Zen/TE-Score for ResNet18/50 at resolution 224x224. The statistical error is within $5\%$. 'time': time for computing Zen/TE-score for $N$ images, measured in seconds, averaged over 100 trials. 'speed-up': speed-up rate of TE-Score v.s. Zen-Score.

shrink its width by factor $0.75$ each time until it satisfies the budget. 32 random networks are generated and trained in total.

The top-1 accuracy is reported in Table 1. Zen-Score significantly outperforms the other four proxies on both CIFAR10 and CIFAR100. TE-Score is the runner-up proxy, followed by synflow. It is not surprise to see that naive proxies, such as FLOPs and gradient-norm, perform poorly, even worse than random search.

To compare the computational efficiency of Zen-Score and TE-score, we compute two scores for ResNet18 and ResNet50 at 224x224 resolution. The expected time cost is averaged over 100 trials. We find that averaging Zen/TE-Score over $N = 16$ random images is sufficient to reduce the statistical error below $5\%$. The results are reported in Table 2. The computation of Zen-Score is $87 \sim 136$ times faster than TE-Score. The speed-up ratio is getting higher for bigger models.

## 5.2 Zen-NAS on ImageNet

We use Zen-NAS to search efficient network (ZenNet) on ImageNet. We consider the following popular networks as baselines: (a) manually-designed networks, including ResNet [15], DenseNet [17], ResNeSt [61], MobileNet-V2 [43] (b) NAS-designed networks for fast inference on GPU, including OFANet-9ms/11ms [5], DFNet [22], RegNet [38]; (c) NAS-designed networks optimized for FLOPs, including OFANet-389M/ 482M/ 595M [5], DNANet [21], EfficientNet [48], Mnasnet [47].

Among these networks, EfficientNet is a popular baseline in NAS-related works. EfficientNet-B0/B1 are suitable for mobile device for their small FLOPs and model size. EfficientNet-B3$\sim$B7 are large models that are best to be deployed on a high-end GPU. Although EfficientNet is optimized for FLOPs, its inference speed on GPU is within top-tier ones. Many previous works compare to EfficientNet by inference speed on GPU [61, 5, 38].

**Searching Low Latency Networks** Following previous works [5, 22, 38], we use Zen-NAS to optimize network inference speed on NVIDIA V100 GPU. We use Search Space I in this experiment. The inference speed is tested at batch size 64, half precision (float16). We search for networks of inference latency within 0.1/0.2/0.3/0.5/0.8/1.2 milliseconds (ms) per image. For testing inference latency, we set batch-size=64 and do mini-batch inference 30 times. The averaged inference latency is recorded. The top-1 accuracy on ImageNet v.s. inference latency is plotted in Figure 5. Clearly, ZenNets outperform baseline models in both accuracy and inference speed by a large margin. The largest model ZenNet-1.2ms achieves $83.6\%$ top-1 accuracy which is between EfficientNet-B5 and B6. It is about $4.9$x faster than EfficientNet at the same

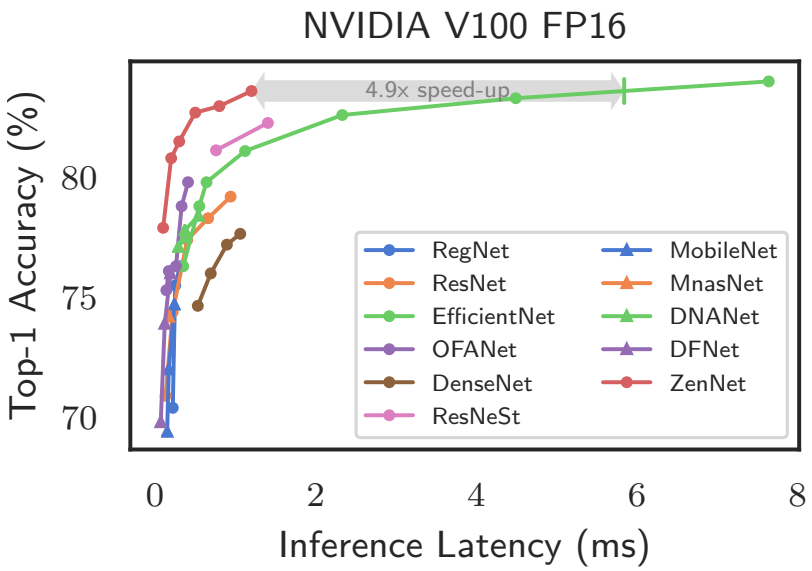



Figure 5: ZenNets top-1 accuracy v.s. inference latency (milliseconds per image) on ImageNet. Benchmarked on NVIDIA V100 GPU, half precision (FP16), batch size 64, searching cost 0.5 GPU day.

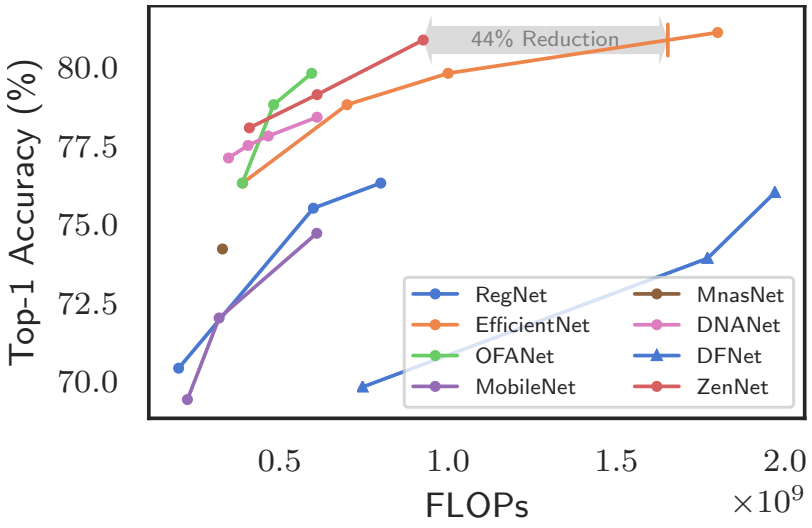



Figure 6: ZenNets optimized for FLOPs.

accuracy level.

**Searching Lightweight Networks** Following previous works [5, 48], we use Zen-NAS to search lightweight networks with small FLOPs. We use Search Space II in this experiment. We search for networks of com-

| NAS | Method | Top-1 (%) | GPU Day |
|---|---|---|---|
| AmoebaNet-A [39] | EA | 74.5 | 3150† |
| EcoNAS [66] | EA | 74.8 | 8 |
| CARS-I [59] | EA | 75.2 | 0.4 |
| GeNet [55] | EA | 72.1 | 17 |
| DARTS [25] | GD | 73.1 | 4 |
| SNAS [56] | GD | 72.7 | 1.5 |
| PC-DARTS [58] | GD | 75.8 | 3.8 |
| ProxylessNAS [6] | GD | 75.1 | 8.3 |
| GDAS [63] | GD | 74 | 0.8 |
| FBNetV2-L1 [52] | GD | 77.2 | 25 |
| NASNet-A [67] | RL | 74 | 1800 |
| Mnasnet-A [47] | RL | 75.2 | - |
| MetaQNN [3] | RL | 77.4 | 96 |
| PNAS [24] | SMBO | 74.2 | 224 |
| SemiNAS [28] | SSL | 76.5 | 4 |
| TE-NAS [7] | ZS | 74.1 | 0.2 |
| OFANet [5] | PS | 80.1 | 51.6 |
| EfficientNet-B7 [48] | Scaling | 84.4 | 3800‡ |
| Zen-NAS | ZS | 83.6 | 0.5 |

Table 3: NAS searching cost comparison. 'Top-1': top-1 accuracy on ImageNet-1k. 'Method': 'EA' is short for Evolutionary Algorithm; 'GD' is short for Gradient Descent; 'RL' is short for reinforcement Learning; 'ZS' is short for Zero-shot; 'SMBO', 'SSL', 'PS' and 'Scaling' are special searching methods/frameworks. †: Running on TPU; ‡: The cost is estimated by [52];

putational cost within 400/600/900 M FLOPs. Similar to OFANet and EfficientNet, we add SE-blocks after convolutional layers. The top-1 accuracy v.s. FLOPs is plotted in Figure 6. Again, ZenNets outperform most models by a large margin. ZenNet-900M-SE achieves $80.8\%$ top-1 accuracy which is comparable to EfficientNet-B3 with $43\%$ fewer FLOPs. The runner-up is OFANet whose efficiency is similar to ZenNet.

## 5.3 Searching Cost of Zen-NAS v.s. SOTA

The major time cost of Zen-NAS is the computation of Zen-Score. The network latency is predicted by an in-house latency predictor whose time cost is nearly zero. According to Table 2, the computation of Zen-Score for ResNet-50 only takes $0.15$ second. This means that scoring 96,000 networks similar to ResNet-50 only takes $4$ GPU hours, or $0.17$ GPU day.

We compare Zen-NAS searching cost to SOTA NAS methods in Table 3. Since every NAS method uses different settings, it is difficult to make a fair comparison that everyone agrees with. Nevertheless, we only concern about the best model reported in each paper and the corresponding searching cost. This gives us a rough impression of the efficiency of these methods and their practical ability of designing high-performance models.

From Table 3, for conventional NAS methods, it takes hundreds to thousands GPU days to find a good structure of accuracy better than $78.0\%$. Many one-shot methods are very fast. For most one-shot methods,

the best accuracy is below $80\%$. In comparison, Zen-NAS achieves $83.6\%$ top-1 accuracy within 0.5 GPU day. Among methods achieving above $80.0\%$ top-1 accuracy in Table 3, the searching speed of Zen-NAS is nearly 100 times faster than OFANet and 7800 times faster than EfficientNet. TE-NAS uses less GPU day than Zen-NAS in Table 3. This does not conflict with Table 2 because the total number of networks evaluated by the two methods are different.

# 6 Conclusion

We proposed Zen-NAS, a zero-shot neural architecture search framework for designing high performance deep image recognition networks. Without optimizing network parameters, Zen-NAS rank networks via network expressivity which can be numerically measured by Zen-Score. The searching speed of Zen-NAS is dramatically faster than previous SOTA methods. The ZenNets automatically designed by Zen-NAS are significantly more efficient in terms of inference latency, FLOPs and model size, in multiple recognition tasks. We wish the elegance of Zen-NAS will inspire more theoretical researches towards a deeper understanding of efficient network design.

# References


[1] Mohamed S. Abdelfattah, Abhinav Mehrotra, Łukasz Dudziak, and Nicholas D. Lane. Zero-Cost Proxies for Lightweight NAS. In *ICLR*, 2021. 2, 3, 5, 10

[2] Gustavo Aguilar, Yuan Ling, Yu Zhang, Benjamin Yao, Xing Fan, and Chenlei Guo. Knowledge Distillation from Internal Representations. In *AAAI*, 2020. 18

[3] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing Neural Network Architectures using Reinforcement Learning. In *ICLR*, 2017. 1, 3, 13

[4] Peter L. Bartlett, Dylan J. Foster, and Matus J. Telgarsky. Spectrally-normalized margin bounds for neural networks. *Advances in Neural Information Processing Systems*, 30, 2017. 2, 6

[5] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-All: Train One Network and Specialize it for Efficient Deployment on Diverse Hardware Platforms. In *ICLR*, 2020. 1, 2, 3, 11, 12, 13, 18

[6] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In *ICLR*, 2019. 1, 3, 13, 20

[7] Wuyang Chen, Xinyu Gong, and Zhangyang Wang. Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective. In *ICLR*, 2021. 2, 3, 5, 10, 13, 21

[8] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive DARTS: Bridging the Optimization Gap for NAS in the Wild. In *ICCV*, 2019. 20

[9] Nadav Cohen and Amnon Shashua. Inductive Bias of Deep Convolutional Networks through Pooling Geometry. In *ICLR*, 2017. 2

[10] Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. AutoAugment: Learning Augmentation Policies from Data. In *CVPR*, pages 113–123, 2019. 18

[11] Amit Daniely, Roy Frostig, and Yoram Singer. Toward Deeper Understanding of Neural Networks: The Power of Initialization and a Dual View on Expressivity. In *NIPS*, volume 29, pages 2253–2261, 2016. 2

[12] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009. 2, 9

[13] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single Path One-Shot Neural Architecture Search with Uniform Sampling. In *ECCV*, 2020. 1, 3

[14] Boris Hanin and David Rolnick. Complexity of Linear Regions in Deep Networks. In *ICML*, 2019. 2, 4, 5

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *CVPR*, pages 770–778, 2016. 10, 11, 23

[16] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. Squeeze-and-Excitation Networks. In *CVPR*, 2018. 4

[17] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely Connected Convolutional Networks. In *CVPR*, pages 2261–2269, 2017. 11

[18] Sham M. Kakade, Karthik Sridharan, and Ambuj Tewari. On the Complexity of Linear Prediction: Risk Bounds, Margin Bounds, and Regularization. *Advances in Neural Information Processing Systems*, 21, 2008. 5

[19] Vladimir Koltchinskii. *Oracle Inequalities in Empirical Risk Minimization and Sparse Recovery Problems*, volume 2033. Springer Science & Business Media, 2011. 2

[20] Alex Krizhevsky. Learning multiple layers of features from tiny images, 2009. 3, 9

[21] Changlin Li, Jiefeng Peng, Liuchun Yuan, Guangrun Wang, Xiaodan Liang, Liang Lin, and Xiaojun Chang. Block-wisely Supervised Neural Architecture Search with Knowledge Distillation. In *CVPR*, 2020. 3, 11, 18

[22] Xin Li, Yiming Zhou, Zheng Pan, and Jiashi Feng. Partial Order Pruning: For Best Speed/Accuracy Trade-off in Neural Architecture Search. In *CVPR*, 2019. 11

[23] Shiyu Liang and R. Srikant. Why Deep Neural Networks for Function Approximation? In *ICLR*, 2016. 2

[24] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Li Fei-Fei, Alan L. Yuille, Jonathan Huang, and Kevin Murphy. Progressive Neural Architecture Search. In *ECCV*, pages 19–35, 2018. 3, 13, 20

[25] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable Architecture Search. In *ICLR*, 2019. 1, 3, 13, 20

[26] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic Gradient Descent with Warm Restarts. In *ICLR*, 2017. 18

[27] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The Expressive Power of Neural Networks: A View from the Width. In *NIPS*, volume 30, pages 6231–6239, 2017. 2

[28] Renqian Luo, Xu Tan, Rui Wang, Tao Qin, Enhong Chen, and Tie-Yan Liu. Semi-Supervised Neural Architecture Search. In *NIPS*, 2020. 1, 3, 13

[29] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural Architecture Optimization. In *NIPS*, pages 7827–7838, 2018. 1, 3

[30] Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. On the Expressive Power of Deep Neural Networks. In *ICML*, pages 2847–2854, 2017. 2, 4

[31] Joseph Mellor, Jack Turner, Amos Storkey, and Elliot J. Crowley. Neural Architecture Search without Training. *arXiv:2006.04647 [cs, stat]*, 2021. 2

[32] Guido Montúfar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the Number of Linear Regions of Deep Neural Networks. In *NIPS*, 2014. 2, 4, 5

[33] Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. The role of over-parametrization in generalization of neural networks. In *International Conference on Learning Representations*, 2018. 2, 6

[34] Thao Nguyen, Maithra Raghu, and Simon Kornblith. Do Wide and Deep Networks Learn the Same Things? Uncovering How Neural Network Representations Vary with Width and Depth. In *ICLR*, 2021. 2

[35] Hieu Pham, Zihang Dai, Qizhe Xie, Minh-Thang Luong, and Quoc V. Le. Meta Pseudo Labels. *arXiv:2003.10580 [cs, stat]*, 2021. 2

[36] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient Neural Architecture Search via Parameters Sharing. In *ICML*, pages 4095–4104. PMLR, 2018. 10, 18, 20

[37] Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos. In *NIPS*, volume 29, 2016. 2, 4

[38] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing Network Design Spaces. In *CVPR*, 2020. 10, 11

[39] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized Evolution for Image Classifier Architecture Search. In *AAAI*, 2019. 1, 3, 13, 20

[40] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Quoc V. Le, and Alex Kurakin. Large-Scale Evolution of Image Classifiers. In *ICML*, pages 2902–2911, 2017. 1

[41] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions, 2020. 2

[42] David Rolnick and Max Tegmark. The power of deeper networks for expressing natural functions. In *ICLR*, 2018. 2

[43] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *CVPR*, 2018. 10, 11

[44] Christian Sciuto, Kaicheng Yu, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the Search Phase of Neural Architecture Search. *arXiv:1902.08142 [cs, stat]*, 2019. 2

[45] Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. Bounding and Counting Linear Regions of Deep Neural Networks. In *ICML*, 2018. 2, 4, 5

[46] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. In *CVPR*, pages 2818–2826, 2016. 18

[47] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V. Le. MnasNet: Platform-Aware Neural Architecture Search for Mobile. In *CVPR*, 2019. 3, 11, 13

[48] Mingxing Tan and Quoc Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *ICML*, pages 6105–6114, 2019. 3, 11, 12, 13

[49] Hidenori Tanaka, Daniel Kunin, Daniel L. K. Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. In *NIPS*, 2020. 3, 10

[50] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. *arXiv:2012.12877 [cs]*, 2021. 2

[51] Roman Vershynin. *High-Dimensional Probability: An Introduction with Applications in Data Science*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2018. 24

[52] Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, Peter Vajda, and Joseph E. Gonzalez. FBNetV2: Differentiable Neural Architecture Search for Spatial and Channel Dimensions. In *CVPR*, pages 12965–12974, 2020. 1, 3, 13

[53] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking Winning Tickets Before Training by Preserving Gradient Flow. In *International Conference on Learning Representations*, 2019. 5

[54] Wei Wen, Hanxiao Liu, Hai Li, Yiran Chen, Gabriel Bender, and Pieter-Jan Kindermans. Neural Predictor for Neural Architecture Search. In *ECCV*, 2020. 1

[55] Lingxi Xie and Alan Yuille. Genetic CNN. In *ICCV*, 2017. 1, 3, 13

[56] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: Stochastic neural architecture search. In *ICLR*, 2018. 1, 3, 13, 20

[57] H. Xiong, L. Huang, M. Yu, L. Liu, F. Zhu, and L. Shao. On the Number of Linear Regions of Convolutional Neural Networks. In *ICML*, 2020. 2, 4, 5

[58] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. PC-DARTS: Partial Channel Connections for Memory-Efficient Architecture Search. In *ICLR*, 2019. 1, 3, 13

[59] Zhaohui Yang, Yunhe Wang, Xinghao Chen, Boxin Shi, Chao Xu, Chunjing Xu, Qi Tian, and Chang Xu. CARS: Continuous Evolution for Efficient Neural Architecture Search. In *CVPR*, pages 1829–1838, 2020. 1, 3, 13

[60] Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. NAS-Bench-101: Towards Reproducible Neural Architecture Search. In *ICML*, 2019. 2

[61] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Zhi Zhang, Haibin Lin, Yue Sun, Tong He, Jonas Mueller, R. Manmatha, Mu Li, and Alexander Smola. ResNeSt: Split-Attention Networks, 2020. 11

[62] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. Mixup: Beyond Empirical Risk Minimization. In *ICLR*, 2018. 18

[63] Miao Zhang, Huiqi Li, Shirui Pan, Xiaojun Chang, and Steven Su. Overcoming Multi-Model Forgetting in One-Shot NAS With Diversity Maximization. In *CVPR*, pages 7806–7815, 2020. 1, 3, 13

[64] Xiao Zhang and Dongrui Wu. Empirical Studies on the Properties of Linear Regions in Deep Neural Networks. In *ICLR*, 2019. 5

[65] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random Erasing Data Augmentation. In *AAAI*, 2020. 18

[66] Dongzhan Zhou, Xinchi Zhou, Wenwei Zhang, Chen Change Loy, Shuai Yi, Xuesen Zhang, and Wanli Ouyang. EcoNAS: Finding Proxies for Economical Neural Architecture Search. In *CVPR*, pages 11396–11404, 2020. 1, 3, 13

[67] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning Transferable Architectures for Scalable Image Recognition. In *CVPR*, 2018. 3, 13, 20

# A Datasets and Experiment Settings

**Dataset** CIFAR10 has 50 thousand training images and 10 thousand testing images in 10 classes with resolution 32x32. CIFAR100 has the same number of training/testing images but in 100 classes. ImageNet-1k has over 1.2 million training images and 50 thousand validation images in 1000 classes. We use the official training/validation split in our experiments.
**Augmentation** We use the following augmentations as in [36]: mix-up [62], label-smoothing [46], random erasing [65], random crop/resize/flip/lighting and AutoAugment [10].
**Optimizer** For all experiments, we use SGD optimizer with momentum 0.9; weight decay 5e-4 for CIFAR10/100, 4e-5 for ImageNet; initial learning rate 0.1 with batch size 256; cosine learning rate decay [26]. We train models up to 1440 epochs in CIFAR10/100, 480 epochs in ImageNet. Following previous works [2, 21, 5]. We use ResNet-152 as teacher network to train the target network.

# B Implementation

Our code is implemented in PyTorch. The synflow implementation is available at `https://github.com/mohsaied/zero-cost-nas/blob/main/foresight/pruners/measures/synflow.py`. The TE-NAS score implementation is available at `https://github.com/VITA-Group/TENAS/blob/main/lib/procedures`.

# C Additional Figures

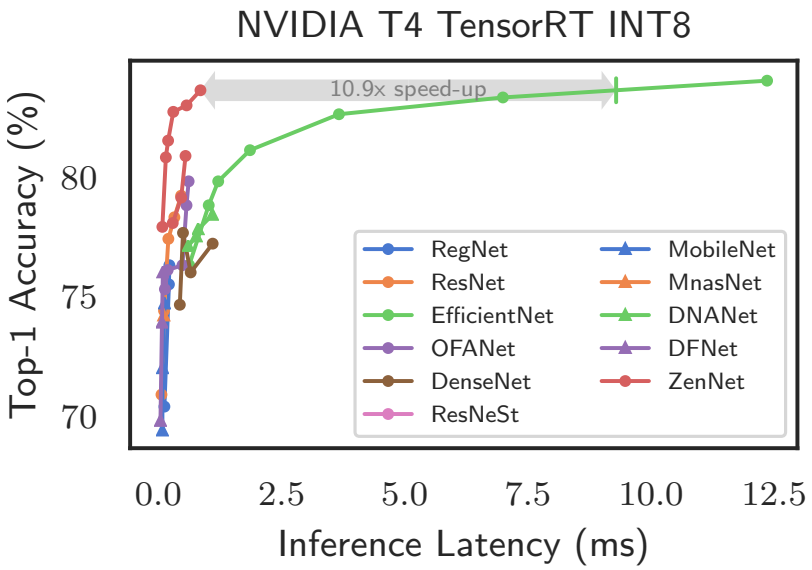



Figure 7: ZenNets top-1 accuracy on ImageNet-1k v.s. inference latency (milliseconds per image) on NVIDIA T4, TensorRT INT8, batch size 64. ZenNet-0.8ms∼1.2ms and ZenNet-400M-SE∼900M-SE are plotted as two separated curves.

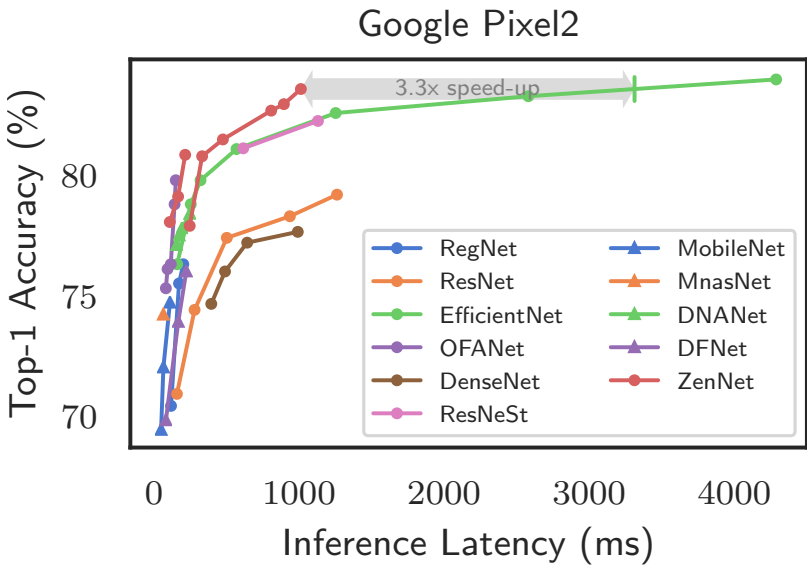



Figure 8: ZenNets top-1 accuracy on ImageNet-1k v.s. inference latency (milliseconds per image) on Google Pixel2, single image. ZenNet-0.8ms∼1.2ms and ZenNet-400M-SE∼900M-SE are plotted as two separated curves.

We test the performance of ZenNets on devices other than NVIDIA V100 GPU. The two hardware platforms are considered. NVIDIA T4 is an industrial level GPU optimized for INT8 inference. All networks are exported to TensorRT engine at precision INT8 to benchmark their inference speed on T4. Google Pixel2 is a modern cell phone with moderate powerful mobile GPU. In Figure 7 and Figure 8, we report the inference speed of ZenNets on T4 and Pixel2 as well as several SOTA models. The best ZenNet-1.2ms is 10.9x times faster than EfficientNet on NVIDIA T4, 1.6x times faster on Pixel2.

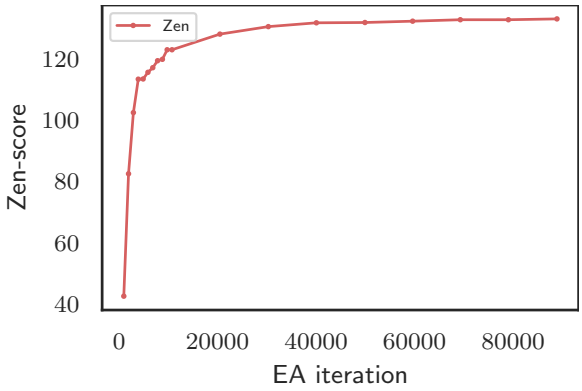



Figure 9: NAS process for maximizing Zen-Score. x-axis: number of evolutionary iterations. y-axis: Largest Zen-Score in the current population.

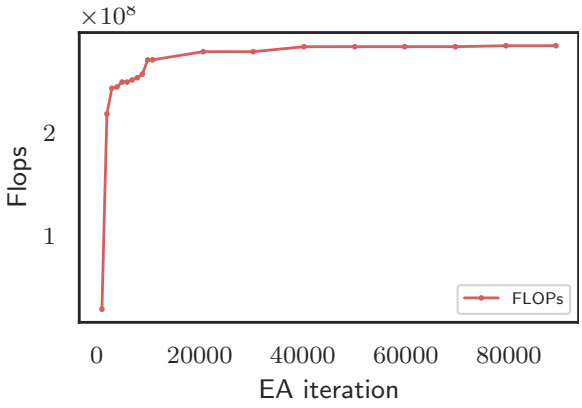



Figure 10: NAS process for maximizing FLOPs. x-axis: number of evolutionary iterations. y-axis: Largest FLOPs in the current population.

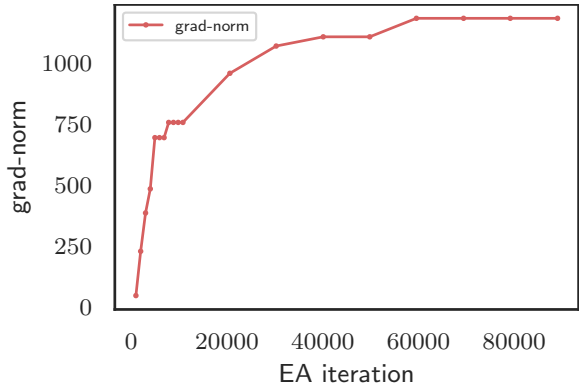



Figure 11: NAS process for maximizing grad-norm. x-axis: number of evolutionary iterations. y-axis: Largest grad-norm in the current population.

# D Zen-NAS on CIFAR

Following previous works, we use Zen-NAS to optimize model size on CIFAR10 and CIFAR100 datasets. We use Search Space I in this experiment. We constrain the number of network parameters within $\{1.0\,\mathrm{M}, 2.0\,\mathrm{M}\}$. The resultant networks are labeled as ZenNet-1.0M/2.0M. Table 4 summarized our results. We compare several popular NAS-designed models for CIFAR10/CIFAR100 in Figure 14, including AmoebaNet [39], DARTS [25], P-DARTS [8], SNAS [56], NASNet-A [67], ENAS[36], PNAS [24], Proxyless-NAS [6]. ZenNets outperform baseline methods by $30\% \sim 50\%$ parameter reduction while achieving the same accuracies.

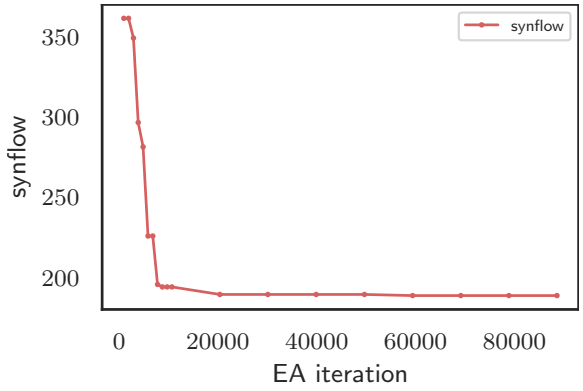



Figure 12: NAS process for maximizing synflow. x-axis: number of evolutionary iterations. y-axis: Smallest synflow in the current population.

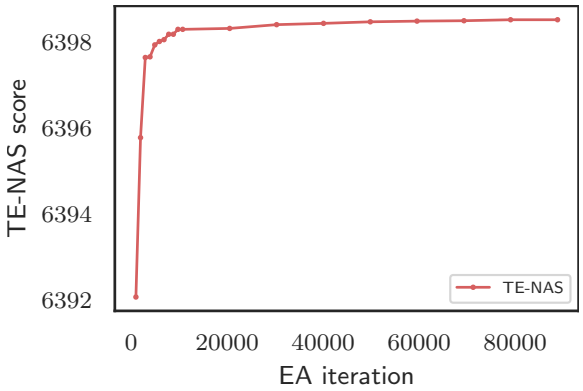



Figure 13: NAS process for maximizing TE-NAS score. x-axis: number of evolutionary iterations. y-axis: Largest TE-NAS score in the current population. The NTK score in TE-NAS is the smaller the better. Therefore we use $R_N$ − NTK as TE-score in EA. This is slightly different from [7] where the rank of NTK is used as score.

| model | # params | FLOPs | C10 | C100 |
|---|---|---|---|---|
| ZenNet-1.0M | 1.0 M | 162 M | 96.5% | 80.1% |
| ZenNet-2.0M | 2.0 M | 487 M | 97.5% | 84.4% |

Table 4: ZenNet-1.0M/2.0M on CIFAR10 (C10) and CIFAR100 (C100).

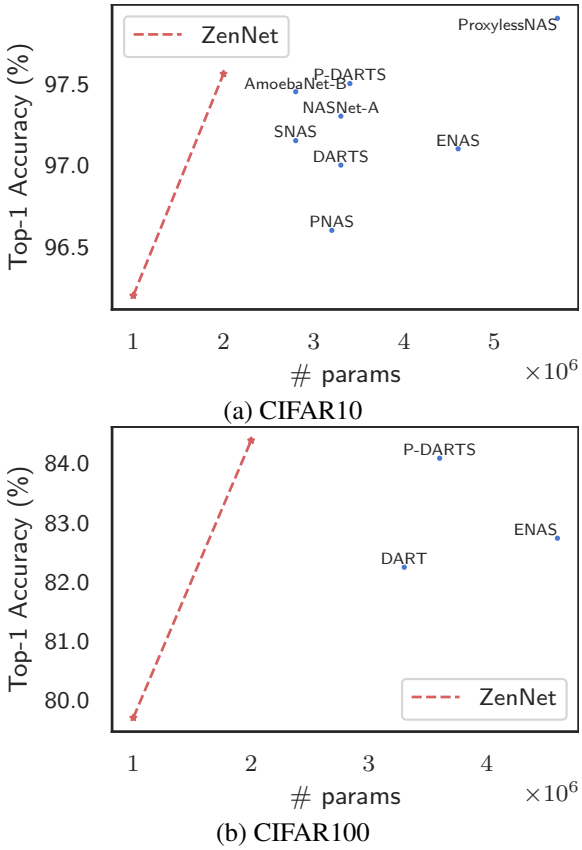



Figure 14: ZenNet accuracy v.s. model size (# params) on CIFAR10 and CIFAR100.

| Model | FLOPs | # Params | Zen-Score |
|---|---|---|---|
| ResNet-18 | 1.82G | 11.7M | 59.53 |
| ResNet-34 | 3.67G | 21.8M | 112.32 |
| ResNet-50 | 4.12G | 25.5M | 140.3 |
| ResNet-101 | 7.85G | 44.5M | 287.87 |
| ResNet-152 | 11.9G | 60.2M | 433.57 |

Table 5: Zen-Scores of ResNets.

# E Zen-Scores and Accuracies of ResNets under Fair Training Setting

ResNets are widely used in computer vision. It is interesting to understand the ResNets via Zen-Score analysis. We report the Zen-Scores of ResNets in Table 5. In Figure 15, we plot the Zen-Score against top-1

| Model | Top-1 [15] | Top-1 (ours) |
|---|---|---|
| ResNet-18 | 70.9% | 72.1% |
| ResNet-34 | 74.4% | 76.3% |
| ResNet-50 | 77.4% | 79.0% |
| ResNet-101 | 78.3% | 81.0% |
| ResNet-152 | 79.2% | 82.3% |

Table 6: Top-1 accuracies of ResNets. Reported by [15] and using enhanced training methods we used in this paper.

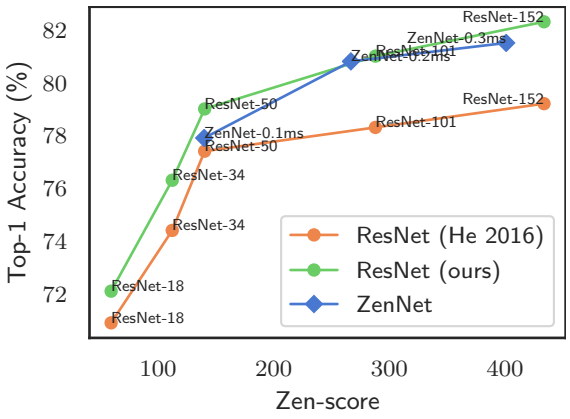



Figure 15: ResNet/ZenNet Zen-Score v.s. top-1 accuracy on ImageNet.

accuracy of ResNet and ZenNet on ImageNet. From the figure, it is clearly that even for the same model, the training method matters a lot. There is considerable performance gain of ResNets after using our enhanced training methods. The Zen-Scores positively correlate to the top-1 accuracies for both ResNet and ZenNets.

Next we show that the Zen-Scores is well-aligned with top-1 accuracies across different models. We consider two baselines in Table 6. The 2nd column reports the top-1 accuracies obtained in the ResNet original paper [15]. We found that these models are under-trained. We use enhanced training methods to train ResNets in the same way as we trained ZenNets. The corresponding top-1 accuracies are reported in the 3rd column.

# F Proof of Theorem 1

We introduce a few more notations for our proof. Suppose the network has $L$ convolutional layers. The $t$-th layer has $m_{t-1}$ input channels and $m_t$ output channels. The convolutional kernel is $\boldsymbol{\theta}_t \in \mathbb{R}^{m_t \times m_{-1t} \times k \times k}$. The image resolution is $H \times W$. The mini-batch size is $B$. The output feature map of the $t$-th layer is $\boldsymbol{x}_t$. We use $\boldsymbol{x}_t^{(i,b,h,w)}$ to denote the pixel of $\boldsymbol{x}_t$ in the $i$-th channel, $b$-th image at cooridinate $(h, w)$. $\mathcal{N}(\mu, \sigma)$ denotes Gaussian distribution with mean $\mu$ and variance $\sigma^2$. For random variables $z, a$ and a constant $b$, the notation $z = a \pm b$ means $|z - a| \leq b$. To avoid notation clutter, we use $C_{\log}^{1/\delta}(\cdot)$ to denote some logarithmic polynomial in $1/\delta$ and some other variables. Since the order of these variables in $C_{\log}^{1/\delta}(\cdot)$ is logarithmic, they do not alternate the polynomial order of our bounds.

The input image $\boldsymbol{x}_0$ are sampled from $\mathcal{N}(0, 1)$. In a vanilla network without BN layer, the feature map $\bar{\boldsymbol{x}}_t$ is generated by the following forward inference process:

$$\begin{aligned}\bar{\boldsymbol{x}}_0 =& \boldsymbol{x}_0\\ \bar{\boldsymbol{x}}_t =& [\boldsymbol{\theta}_t * \bar{\boldsymbol{x}}_{t-1}]_+\end{aligned}$$

where $*$ is the convolutional operator, $[z]_+ = \max(z, 0)$.

In Zen-score computation, BN layer is inserted after every convolutional operator. The forward inference now becomes:

$$\boldsymbol{g}_t = \boldsymbol{\theta}_t * \boldsymbol{x}_{t-1} \tag{5}$$

$$[\boldsymbol{\sigma}_t^{(i)}]^2 = \frac{1}{BHW} \sum_{b,h,w} [\boldsymbol{g}_t^{(i,b,h,w)}]^2 \tag{6}$$

$$\bar{\sigma}_t^2 = \frac{1}{m_t} \sum_{i=1}^{m_t} [\boldsymbol{\sigma}_t^{(i)}]^2 \tag{7}$$

$$\boldsymbol{x}_t^{(i)} = \left[\frac{\boldsymbol{g}_t^{(i)}}{\boldsymbol{\sigma}_t^{(i)}}\right]_+ = \frac{1}{\boldsymbol{\sigma}_t^{(i)}} [\boldsymbol{g}_t^{(i)}]_+ \,. \tag{8}$$

Please note that in Eq. (8), we use a modified BN layer instead of the standard BN, where we do not subtract mean value in the normalization step. This will greatly simply the proof. If the reader is concerned about this, it is straightforward to replace all BN layers with our modified BN layers so that the computational process exactly follows our proof. In practice, we did not observe noticable difference by switching between two BN layers because the mean value of mini-batch is very close to zero.

To show that the Zen-score computed on BN-enabled network $f(\boldsymbol{x}_0) = \boldsymbol{x}_L$ approximates the $\Phi$-score computed on BN-free network $\bar{f}(\boldsymbol{x}_0) = \bar{\boldsymbol{x}}_L$, we only need to prove

$$(\prod_{t=1}^{L} \bar{\sigma}_t)^2 \mathbb{E}_\theta \|\boldsymbol{x}_L\|^2 \approx \mathbb{E}_\theta \|\bar{\boldsymbol{x}}_L\|^2 \,. \tag{9}$$

In deed, when Eq. (9) holds true, by taking gradient w.r.t. $\boldsymbol{x}$ on both side, the proof is then completed. To prove Eq. (9), we need the following theorems and lemmas.

## F.1 Useful Theorems and Lemmas

The first theorem is Bernstein's inequality. It can be found in many statistical textbooks, such as [51, Theorem 2.8.1].

**Theorem 2** (Bernstein's inequality)**.** *Let $x_1, x_2, \cdots, x_N$ be independent bounded random variables of mean zero, variance $\sigma$. $|x_i| \leq K$ for all $i \in \{1, 2, \cdots, N\}$. $\boldsymbol{a} = [a_1, a_2, \cdots, a_N]$ is a fixed $N$-dimensional vector. Then $\forall t \geq 0$,*

$$\mathbb{P}(\left|\sum_{i=1}^{N} a_i x_i\right| \geq t) \leq 2\exp\left[-c\min\left(\frac{t^2}{\sigma^2\|\boldsymbol{a}\|_2^2}, \frac{t}{K\|\boldsymbol{a}\|_\infty}\right)\right] .$$

A direct corollary gives the upper bound of sum of random variables.

**Corollary 1.** *Under the same setting of Theorem 2, with probability at least $1-\delta$,*

$$\left|\sum_{i=1}^{N} a_i x_i\right| \leq C_{\log}^{1/\delta}(\cdot)\sigma\|\boldsymbol{a}\|_2 .$$

*Proof.* Let

$$\begin{aligned}\delta &\triangleq 2\exp\left[-c\min\left(\frac{t^2}{\sigma^2\|\boldsymbol{a}\|_2^2}, \frac{t}{K\|\boldsymbol{a}\|_\infty}\right)\right] \\ &= \max\left\{2\exp\left[-c\frac{t^2}{\sigma^2\|\boldsymbol{a}\|_2^2}\right], 2\exp\left[-c\frac{t}{K\|\boldsymbol{a}\|_\infty}\right]\right\} .\end{aligned}$$

That is,

$$\begin{aligned}&\delta \geq 2\exp\left[-c\frac{t^2}{\sigma^2\|\boldsymbol{a}\|_2^2}\right] \\ \Leftrightarrow &t \leq \sqrt{\frac{1}{c}\log(2/\delta)}\sigma\|\boldsymbol{a}\|_2 = C_{\log}^{1/\delta}(\cdot)\sigma\|\boldsymbol{a}\|_2 ,\end{aligned}$$

and

$$\begin{aligned}&\delta \geq 2\exp\left[-c\frac{t}{K\|\boldsymbol{a}\|_\infty}\right] \\ \Leftrightarrow &t \leq \frac{1}{c}\log(2/\delta)K\|\boldsymbol{a}\|_\infty = C_{\log}^{1/\delta}(\cdot)K\|\boldsymbol{a}\|_\infty .\end{aligned}$$

Therefore, with probability at least $1-\delta$,

$$\begin{aligned}\left|\sum_{i=1}^{N} a_i x_i\right| &\leq \min\{C_{\log}^{1/\delta}(\cdot)\sigma\|\boldsymbol{a}\|_2, C_{\log}^{1/\delta}(\cdot)K\|\boldsymbol{a}\|_\infty\} \\ &\leq C_{\log}^{1/\delta}(\cdot)\min\{\sigma\|\boldsymbol{a}\|_2, K\|\boldsymbol{a}\|_\infty\} .\end{aligned}$$

That is,

$$\left|\sum_{i=1}^{N} a_i x_i\right| \leq C_{\log}^{1/\delta}(\cdot)\sigma\|\boldsymbol{a}\|_2 .$$

□

When the random variables are sampled from Gaussian distribution, it is more convenient to use the following tighter bound.

**Theorem 3.** *Let $x_1, x_2, \cdots, x_N$ be sampled from $\mathcal{N}(0, \sigma)$, $\boldsymbol{a} \in \mathbb{R}^N$ be a fixed a vector. Then $\forall t \geq 0$,*

$$\mathbb{P}(\left|\sum_{i=1}^{N} a_i x_i\right| > t) \leq \exp\left[-\frac{t^2}{2\sigma^2\|a\|_2^2}\right] .$$

**Corollary 2.** *With probability at least $1-\delta$,*

$$\left|\sum_{i=1}^{N} a_i x_i\right| \leq \sqrt{2\log(1/\delta)}\sigma\|\boldsymbol{a}\|_2 = C_{\log}^{1/\delta}(\cdot)\sigma\|\boldsymbol{a}\|_2 .$$

The proof is simple since the sum of Gaussian random variables is still Gaussian random variables.

The following two lemmas are critical in our lower bound analysis. The proof is straightforward once using the symmetric property of random variable distribution.

**Lemma 3.** *Suppose $x \in \mathbb{R}$ is a mean zero, variance $\sigma^2$ random variable with symmetric distribution. Then $\mathbb{E}[x]_+^2 = 4\sigma^2/4$.*

**Lemma 4.** *Suppose $\theta_i \sim \mathcal{N}(0,1)$. $\|\boldsymbol{x}\| = \|\boldsymbol{y}\|$ are two fixed vectors. Then*

$$\mathbb{E}_\theta[\sum_i \theta_i x_i]_+^2 = \frac{1}{2}\mathbb{E}_\theta[\sum_i \theta_i x_i]^2 = \mathbb{E}_\theta[\sum_i \theta_i y_i]_+^2 .$$

## F.2 Proof of Eq. (9)

Since $\boldsymbol{x}_0 \sim \mathcal{N}(0,1)$, with probability at least $1-\delta$, $\|\boldsymbol{x}_0\|_\infty \leq C_{\log}^{1/\delta}(\cdot) \triangleq K_0$ for some constant $K_0$. Now suppose at layer $t$, $\|\boldsymbol{x}_{t-1}\|_\infty \leq K_{t-1}$. The following lemma shows that after convolution, $\|\boldsymbol{g}_t\|_\infty$ is also bounded with high probability.

**Lemma 5.** *Let $\boldsymbol{\theta}^{(i,b,h,w)} \sim \mathcal{N}(0,1)$, $\boldsymbol{\theta}_t \in \mathbb{R}^{m_t \times m_{t-1} \times k \times k}$. For fixed $\boldsymbol{x}_{t-1} \in \mathbb{R}^{m_{t-1} \times B \times H \times W}$, $\boldsymbol{g}_t \triangleq \boldsymbol{\theta}_t * \boldsymbol{x}_{t-1}$. Then with probability at least $1-\delta$,*

$$\|\boldsymbol{g}_t\|_\infty \leq C_{\log}^{1/\delta}(\cdot)^2 k\sqrt{m_{t-1}}K_{t-1} .$$

*Proof.* Let us consider $\boldsymbol{g}_t^{(j,b,\alpha,\beta)}$, that is, the $j$-th channel, $b$-th image, at pixel $(\alpha, \beta)$. For any $1 \leq j \leq m_t$, $1 \leq \alpha \leq H$, $1 \leq \beta \leq W$,

$$\boldsymbol{g}_t^{(j,b,\alpha,\beta)} = \sum_{i=1}^{m_{-1t}} \sum_{p=-\frac{k-1}{2}}^{\frac{k-1}{2}} \sum_{q=-\frac{k-1}{2}}^{\frac{k-1}{2}} \boldsymbol{\theta}_t^{(j,i,p,q)} \boldsymbol{x}_{t-1}^{(i,b,\alpha+p,\beta+p)}$$

Clearly,

$$\mathbb{E}_\theta \boldsymbol{g}_t^{(j,b,\alpha,\beta)} = 0 .$$

According to Corollary 2,

$$\begin{aligned} |\boldsymbol{g}_t^{(j,b,\alpha,\beta)}| &\leq C_{\log}^{1/\delta}(\cdot) C_{\log}^{1/\delta}(\cdot) K_{t-1}\sqrt{m_{t-1}}k \\ &\leq C_{\log}^{1/\delta}(\cdot)^2 k\sqrt{m_{t-1}}K_{t-1} . \end{aligned}$$

□

The variance of $\boldsymbol{g}_t$ is bounded with high probability too.

**Lemma 6.** *With probability at least* $1-\delta$*,*

$$\begin{aligned}\mathbb{E}[\boldsymbol{g}_t^{(j,b,\alpha,\beta)}]^2 =&\sigma_t^* \pm C_{\log}^{1/\delta}(\cdot)k\sqrt{m_{t-1}}K_{t-1}\\ \sigma_t^{*2} \triangleq&\frac{1}{4}m_{t-1}k^2\ .\end{aligned}$$

*Proof.* By definition,

$$\boldsymbol{g}_t^{(j,b,\alpha,\beta)} = \sum_{i=1}^{m_{-1t}}\sum_{p=-\frac{k-1}{2}}^{\frac{k-1}{2}}\sum_{q=-\frac{k-1}{2}}^{\frac{k-1}{2}}\boldsymbol{\theta}_t^{(j,i,p,q)}\boldsymbol{x}_{t-1}^{(i,b,\alpha+p,\beta+p)}$$

Clearly, $\boldsymbol{g}_t^{(j,b,\alpha,\beta)}$ is Gaussian random variable with zero-mean.

$$\mathbb{E}[\boldsymbol{g}_t^{(j,b,\alpha,\beta)}]^2 = \sum_{i=1}^{m_{t-1}}\sum_{p=-\frac{k-1}{2}}^{\frac{k-1}{2}}\sum_{q=-\frac{k-1}{2}}^{\frac{k-1}{2}}[\boldsymbol{x}_{t-1}^{(i,b,\alpha+p,\beta+p)}]^2\ .$$

By Lemma 3,

$$\mathbb{E}[\boldsymbol{x}_{t-1}^{(i,b,\alpha+p,\beta+p)}]^2 =\frac{1}{4}\ .$$

Therefore,

$$\begin{aligned}&|\mathbb{E}[\boldsymbol{g}_t^{(j,b,\alpha,\beta)}]^2 - \frac{1}{4}m_{t-1}k^2|\\ \le&C_{\log}^{1/\delta}(\cdot)k\sqrt{m_{t-1}}K_{t-1}\ .\end{aligned}$$

Define $\sigma_t^{*2} \triangleq \frac{1}{4}m_{t-1}k^2$, the proof is completed. □

Next we show that both $\sigma_t^{(i)}$ and $\bar{\sigma}_t$ concentrate around $\sigma^*$.

**Lemma 7.** *With probability* $1-\delta$*,*

$$\begin{aligned}[\boldsymbol{\sigma}_t^{(i)}]^2 =&(1\pm\epsilon_t)[\sigma_t^*]^2\\ \bar{\sigma}_t =&(1\pm\frac{\epsilon_t}{\sqrt{m_t}})[\sigma_t^*]^2\end{aligned}$$

*where*

$$\epsilon_t \triangleq 4C_{\log}^{1/\delta}(\cdot)^5\frac{1}{\sqrt{BHW}}K_{t-1}^2$$

*Proof.* Directly apply Corollary 1,

$$\begin{aligned}[\boldsymbol{\sigma}_t^{(i)}]^2 =&\mathbb{E}[\boldsymbol{g}_t^{(j,b,\alpha,\beta)}]^2 \pm C_{\log}^{1/\delta}(\cdot)\frac{1}{\sqrt{BHW}}\max\{[\boldsymbol{g}_t^{(j,b,\alpha,\beta)}]^2\}\\ =&\mathbb{E}[\boldsymbol{g}_t^{(j,b,\alpha,\beta)}]^2 \pm C_{\log}^{1/\delta}(\cdot)\frac{1}{\sqrt{BHW}}C_{\log}^{1/\delta}(\cdot)^4m_{t-1}k^2K_{t-1}^2\\ =&[\sigma_t^*]^2 \pm \frac{1}{\sqrt{BHW}}C_{\log}^{1/\delta}(\cdot)^5m_{t-1}k^2K_{t-1}^2\ .\end{aligned}$$

Similary,

$$\bar{\sigma}_t^2 = [\sigma_t^*]^2 \pm \frac{1}{\sqrt{m_t BHW}} C_{\log}^{1/\delta}(\cdot)^5 m_{t-1} k^2 K_{t-1}^2 \,.$$

Define

$$\begin{aligned}\epsilon_t \triangleq & \frac{1}{[\sigma_t^*]^2} \frac{1}{\sqrt{BHW}} C_{\log}^{1/\delta}(\cdot)^5 m_{t-1} k^2 K_{t-1}^2 \\ = & \frac{4}{m_{t-1}k^2} C_{\log}^{1/\delta}(\cdot)^5 \frac{1}{\sqrt{BHW}} m_{t-1} k^2 K_{t-1}^2 \\ = & 4 C_{\log}^{1/\delta}(\cdot)^5 \frac{1}{\sqrt{BHW}} K_{t-1}^2\end{aligned}$$

Then we have

$$\begin{aligned}[\boldsymbol{\sigma}_t^{(i)}]^2 = & (1 \pm \epsilon_t)[\sigma_t^*]^2 \\ \bar{\sigma}_t = & (1 \pm \frac{\epsilon_t}{\sqrt{m_t}})[\sigma_t^*]^2\end{aligned}$$

□

Next is our main lemma.

**Lemma 8.** *Under the same setting of Lemma 7, with probability* $1 - \delta$*,*

$$(\sigma_t^*)^2 \|\boldsymbol{x}_t\|^2 = \frac{1}{1 \pm \epsilon_t} \| [\boldsymbol{g}_t]_+ \|^2 \,.$$

*Proof.* By definition,

$$\begin{aligned}\|\boldsymbol{x}_t\|^2 = & \sum_i \left[\frac{1}{\boldsymbol{\sigma}_t^{(i)}}\right]^2 \left[\boldsymbol{g}_t^{(i)}\right]_+^2 \\ = & \sum_i \left[\frac{1}{(1 \pm \epsilon_t)\sigma_t^*}\right]^2 \left[\boldsymbol{g}_t^{(i)}\right]_+^2\end{aligned}$$

Then

$$\begin{aligned}& \frac{(\sigma_t^*)^2 \|\boldsymbol{x}_t\|^2}{\sum_i \left[\boldsymbol{g}_t^{(i)}\right]_+^2} \\ = & \frac{1}{\sum_i \left[\boldsymbol{g}_t^{(i)}\right]_+^2} \sum_i \left[\frac{\sigma_t^*}{\boldsymbol{\sigma}_t^{(i)}}\right]^2 \left[\boldsymbol{g}_t^{(i)}\right]_+^2\end{aligned}$$

By Lemma 7, we have

$$\frac{1}{1 + \epsilon_t} \leq \frac{\sigma_t^*}{\boldsymbol{\sigma}_t^{(i)}} \leq \frac{1}{1 - \epsilon_t}$$

□

Finally, we inductively bound $|\boldsymbol{x}_t^{(i,b,h,w)}|$.

**Lemma 9.** *With probability at least* $1-\delta$*,*

$$|\boldsymbol{x}_t^{(i,b,h,w)}| \le \frac{C_{\log}^{1/\delta}(\cdot)^2}{\sqrt{(1-\epsilon_t)}} K_{t-1}$$
$$K_t \le C_{\log}^{1/\delta}(\cdot)^{2t} \prod_{j=1}^{t} (1-\epsilon_j)^{-1/2} K_0 \ .$$

*Proof.* By definition,

$$\boldsymbol{x}_t^{(i,b,h,w)} = \frac{1}{\boldsymbol{\sigma}_t^{(i)}} [\boldsymbol{g}_t^{(i,b,h,w)}]_+$$

From Lemma 5,

$$[\boldsymbol{g}_t^{(i,b,h,w)}]_+ \le C_{\log}^{1/\delta}(\cdot)^2 K_{t-1} \sqrt{m_{t-1}} k$$

From Lemma 7,

$$\begin{aligned}[\boldsymbol{\sigma}_t^{(i)}]^2 &= (1 \pm \epsilon_t)[\sigma_t^*]^2 \\ &= \frac{1}{4}(1 \pm \epsilon_t) m_{t-1} k^2\end{aligned}$$

Then

$$\begin{aligned}|\boldsymbol{x}_t^{(i,b,h,w)}| \le& \frac{C_{\log}^{1/\delta}(\cdot)^2 K_{t-1} \sqrt{m_{t-1}} k}{\sqrt{\frac{1}{4}(1 \pm \epsilon_t) m_{t-1} k^2}} \\ \le& \frac{C_{\log}^{1/\delta}(\cdot)^2 K_{t-1} \sqrt{m_{t-1}} k}{\sqrt{\frac{1}{4}(1 - \epsilon_t) m_{t-1} k^2}} \\ \le& 2 \frac{C_{\log}^{1/\delta}(\cdot)^2 K_{t-1}}{\sqrt{(1-\epsilon_t)}} \\ \to& \frac{C_{\log}^{1/\delta}(\cdot)^2 K_{t-1}}{\sqrt{(1-\epsilon_t)}} \quad \text{absorb 2 into } C_{\log}^{1/\delta}(\cdot)\end{aligned}$$

Therefore,

$$\begin{aligned}K_t &\triangleq \frac{C_{\log}^{1/\delta}(\cdot)^2 K_{t-1}}{\sqrt{(1-\epsilon_t)}} \\ \Rightarrow K_t &= C_{\log}^{1/\delta}(\cdot)^{2t} \prod_{j=1}^{t} (1-\epsilon_j)^{-1/2} K_0\end{aligned}$$

□

Combining all above together, we are now ready to prove Eq. (9).

Denote $z_0 = 1$. It is trivial to see that $z_0\|\boldsymbol{x}_0\|^2 = z_0\|\bar{\boldsymbol{x}}_t\|^2$. By induction, suppose at layer $t$, we already have $z_{t-1}\|\boldsymbol{x}_{t-1}\|^2 = \|\bar{\boldsymbol{x}}_{t-1}\|^2$. Using Lemma 4,

$$\begin{aligned}\mathbb{E}_\theta\|\bar{\boldsymbol{x}}_t\|^2 =&\mathbb{E}_\theta\|\left[\boldsymbol{\theta}_t * \bar{\boldsymbol{x}}_{t-1}\right]_+\|^2\\ =&\mathbb{E}_\theta\|\left[\boldsymbol{\theta}_t * z_{t-1}\boldsymbol{x}_{t-1}\right]_+\|^2\\ =&z_{t-1}\mathbb{E}_\theta\|\left[\boldsymbol{\theta}_t * \boldsymbol{x}_{t-1}\right]_+\|^2\\ =&z_{t-1}\mathbb{E}_\theta\|\left[\boldsymbol{g}_t\right]_+\|^2\end{aligned}$$

On the other hand, from Lemma 8,

$$\begin{aligned}\bar{\sigma}_t^2 z_{t-1}\|\boldsymbol{x}_t\|^2 =&z_{t-1}\frac{\bar{\sigma}_t^2}{(\sigma_t^*)^2}(\sigma_t^*)^2\|\boldsymbol{x}_t\|^2\\ =&z_{t-1}(1\pm\frac{\epsilon_t}{\sqrt{m_t}})(\sigma_t^*)^2\|\boldsymbol{x}_t\|^2 \quad \text{Lemma [lem:sigma-i-concentration]}\\ =&z_{t-1}(1\pm\frac{\epsilon_t}{\sqrt{m_t}})\frac{1}{1\pm\epsilon_t}\|\left[\boldsymbol{g}_t\right]_+\|^2\,.\end{aligned}$$

Therefore,

$$\mathbb{E}_\theta\{\bar{\sigma}_t^2 z_{t-1}\|\boldsymbol{x}_t\|^2\} = (1\pm\frac{\epsilon_t}{\sqrt{m_t}})\frac{1}{1\pm\epsilon_t}\mathbb{E}_\theta\|\bar{\boldsymbol{x}}_t\|^2$$

By taking

$$z_t \triangleq \bar{\sigma}_t^2 z_{t-1}/[(1\pm\frac{\epsilon_t}{\sqrt{m_t}})\frac{1}{1\pm\epsilon_t}]\,,$$

we complete the induction of $z_t\|\boldsymbol{x}_t\|^2 = \|\bar{\boldsymbol{x}}_t\|^2$ for all $t$.

Chaining $t = \{1, 2, \cdots, L\}$, we get

$$\mathbb{E}_\theta\{(\prod_{t=1}^L \bar{\sigma}_t^2)\|\boldsymbol{x}_L\|^2\} = \prod_{t=1}^L\left[(1\pm\frac{\epsilon_t}{\sqrt{m_t}})\frac{1}{1\pm\epsilon_t}\right]\mathbb{E}_\theta\|\bar{\boldsymbol{x}}_L\|^2\,,$$

where

$$\begin{aligned}\epsilon_t \triangleq&4C_{\log}^{1/\delta}(\cdot)^5\frac{1}{\sqrt{BHW}}K_{t-1}^2\\ K_t \triangleq&C_{\log}^{1/\delta}(\cdot)^{2t}\prod_{j=1}^t(1-\epsilon_j)^{-1/2}K_0\,.\end{aligned}$$

Finally, integrate everything together, we have proved that, with probability at least $1-\delta$,

$$(\prod_{t=1}^L \bar{\sigma}_t^2)\mathbb{E}_\theta\{\|\boldsymbol{x}_L\|^2\} = \prod_{t=1}^L\left[(1\pm\frac{\epsilon_t}{\sqrt{m_t}})\frac{1}{1\pm\epsilon_t}\right]\mathbb{E}_\theta\|\bar{\boldsymbol{x}}_L\|^2\,.$$

That is,

$$\prod_{t=1}^{L}\left[(1-\frac{\epsilon_t}{\sqrt{m_t}})\frac{1}{1+\epsilon}\right] \leq \frac{(\prod_{t=1}^{L}\bar{\sigma}_t^2)\mathbb{E}_\theta\{\|\boldsymbol{x}_L\|^2\}}{\mathbb{E}_\theta\|\bar{\boldsymbol{x}}_L\|^2} \leq \prod_{t=1}^{L}\left[(1+\frac{\epsilon_t}{\sqrt{m_t}})\frac{1}{1-\epsilon_t}\right] .$$

To further simply the above results, we consider the asymptotic case where $BHW$ is large enough. Then $\epsilon_t$ will be a small number. By first order approximation of binomial expansion, $(1+\epsilon)^L \approx 1+L\epsilon+\mathcal{O}(\epsilon^2)$. To see that $\epsilon_t$ is bounded by a small constant, we denote $\gamma_t \triangleq \max_{j\in[1,t]}\epsilon_j$. Then

$$\begin{aligned} K_t \leq & \mathcal{O}[(1+\frac{t-1}{2}\gamma_{t-1})K_0] \\ \gamma_t \leq & \mathcal{O}\{\frac{K_0}{\sqrt{BHW}}[(1+\frac{(t-1)}{2}\gamma_{t-1}]\} . \end{aligned} \tag{10}$$

By the recursive equation Eq. (10), when $\gamma_{t-1} \leq \frac{2}{L-1}$,

$$\begin{aligned} \gamma_t \leq & \mathcal{O}\{\frac{K_0}{\sqrt{BHW}}[(1+\frac{(t-1)}{2}\gamma_{t-1}]\} \\ \leq & \mathcal{O}\{\frac{2K_0}{\sqrt{BHW}}\} . \end{aligned}$$

Therefore, by taking $\frac{2K_0}{\sqrt{BHW}} \leq \frac{2}{L}$, that is $BHW \geq \mathcal{O}\{L^2K_0^2\}$, we have

$$\epsilon = \max\epsilon_t \leq \mathcal{O}\{\frac{2K_0}{\sqrt{BHW}}\}$$

to be a small number.

When $\epsilon$ is a small number,

$$\begin{aligned} \frac{(\prod_{t=1}^{L}\bar{\sigma}_t^2)\mathbb{E}_\theta\{\|\boldsymbol{x}_L\|^2\}}{\mathbb{E}_\theta\|\bar{\boldsymbol{x}}_L\|^2} \leq & \prod_{t=1}^{L}\left[(1+\frac{\epsilon_t}{\sqrt{m_t}})\frac{1}{1-\epsilon_t}\right] \\ \leq & (1+\epsilon)^L(1-\epsilon_t)^{-L} \\ \approx & (1+L\epsilon)^2 . \end{aligned}$$

Similarly,

$$\frac{(\prod_{t=1}^{L}\bar{\sigma}_t^2)\mathbb{E}_\theta\{\|\boldsymbol{x}_L\|^2\}}{\mathbb{E}_\theta\|\bar{\boldsymbol{x}}_L\|^2} \geq (1-L\epsilon)^2 .$$

# G One Big Table of Networks on ImageNet

| model | resolution | # params | FLOPs | Top-1 Acc | latency(ms) | | |
|---|---|---|---|---|---|---|---|
| | | | | | V100 | T4 | Pixel2 |
| RegNetY-200MF | 224 | 3.2 M | 200 M | 70.4% | 0.22 | 0.12 | 118.17 |
| RegNetY-400MF | 224 | 4.3 M | 400 M | 74.1% | 0.44 | 0.17 | 181.09 |
| RegNetY-600MF | 224 | 6.1 M | 600 M | 75.5% | 0.25 | 0.21 | 173.19 |
| RegNetY-800MF | 224 | 6.3 M | 800 M | 76.3% | 0.31 | 0.22 | 202.66 |
| ResNet-18 | 224 | 11.7 M | 1.8 G | 70.9% | 0.13 | 0.06 | 158.70 |
| ResNet-34 | 224 | 21.8 M | 3.6 G | 74.4% | 0.22 | 0.11 | 280.44 |
| ResNet-50 | 224 | 25.6 M | 4.1 G | 77.4% | 0.40 | 0.20 | 502.43 |
| ResNet-101 | 224 | 44.5 M | 7.8 G | 78.3% | 0.66 | 0.32 | 937.11 |
| ResNet-152 | 224 | 60.2 M | 11.5 G | 79.2% | 0.94 | 0.46 | 1261.97 |
| EfficientNet-B0 | 224 | 5.3 M | 390 M | 76.3% | 0.35 | 0.62 | 160.72 |
| EfficientNet-B1 | 240 | 7.8 M | 700 M | 78.8% | 0.55 | 1.02 | 254.26 |
| EfficientNet-B2 | 260 | 9.2 M | 1.0 G | 79.8% | 0.64 | 1.21 | 321.45 |
| EfficientNet-B3 | 300 | 12.0 M | 1.8 G | 81.1% | 1.12 | 1.86 | 569.30 |
| EfficientNet-B4 | 380 | 19.0 M | 4.2 G | 82.6% | 2.33 | 3.66 | 1252.79 |
| EfficientNet-B5 | 456 | 30.0 M | 9.9 G | 83.3% | 4.49 | 6.99 | 2580.25 |
| EfficientNet-B6 | 528 | 43.0 M | 19.0 G | 84.0% | 7.64 | 12.36 | 4287.81 |
| EfficientNet-B7 | 600 | 66.0 M | 37.0 G | 84.4% | 13.73 | † | 8615.92 |
| MobileNetV2-0.25 | 224 | 1.5 M | 44 M | 51.8% | 0.08 | 0.04 | 16.71 |
| MobileNetV2-0.5 | 224 | 2.0 M | 108 M | 64.4% | 0.10 | 0.05 | 26.99 |
| MobileNetV2-0.75 | 224 | 2.6 M | 226 M | 69.4% | 0.15 | 0.08 | 49.78 |
| MobileNetV2-1.0 | 224 | 3.5 M | 320 M | 72.0% | 0.17 | 0.08 | 65.59 |
| MobileNetV2-1.4 | 224 | 6.1 M | 610 M | 74.7% | 0.24 | 0.12 | 110.70 |
| MnasNet-1.0 | 224 | 4.4 M | 330 M | 74.2% | 0.17 | 0.11 | 65.50 |
| DNANet-a | 224 | 4.2 M | 348 M | 77.1% | 0.29 | 0.60 | 157.94 |
| DNANet-b | 224 | 4.9 M | 406 M | 77.5% | 0.37 | 0.77 | 173.66 |
| DNANet-c | 224 | 5.3 M | 466 M | 77.8% | 0.37 | 0.81 | 194.27 |
| DNANet-d | 224 | 6.4 M | 611 M | 78.4% | 0.54 | 1.10 | 248.08 |
| DFNet-1 | 224 | 8.5 M | 746 M | 69.8% | 0.07 | 0.04 | 82.87 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| DFNet-2 | 224 | 18.0 M | 1.8 G | 73.9% | 0.12 | 0.07 | 168.04 |
| DFNet-2a | 224 | 18.1 M | 2.0 G | 76.0% | 0.19 | 0.09 | 223.20 |
| OFANet-9ms | 118 | 5.2 M | 313 M | 75.3% | 0.14 | 0.13 | 82.69 |
| OFANet-11ms | 192 | 6.2 M | 352 M | 76.1% | 0.17 | 0.19 | 94.17 |
| OFANet-389M(+) | 224 | 8.4 M | 389 M | 79.1% | 0.26 | 0.49 | 116.34 |
| OFANet-482M(+) | 224 | 9.1 M | 482 M | 79.6% | 0.33 | 0.57 | 142.76 |
| OFANet-595M(+) | 236 | 9.1 M | 595 M | 80.0% | 0.41 | 0.61 | 150.83 |
| OFANet-389M* | 224 | 8.4 M | 389 M | 76.3% | 0.26 | 0.49 | 116.34 |
| OFANet-482M* | 224 | 9.1 M | 482 M | 78.8% | 0.33 | 0.57 | 142.76 |
| OFANet-595M* | 236 | 9.1 M | 595 M | 79.8% | 0.41 | 0.61 | 150.83 |
| DenseNet-121 | 224 | 8.0 M | 2.9 G | 74.7% | 0.53 | 0.43 | 395.51 |
| DenseNet-161 | 224 | 28.7 M | 7.8 G | 77.7% | 1.06 | 0.50 | 991.61 |
| DenseNet-169 | 224 | 14.1 M | 3.4 G | 76.0% | 0.69 | 0.65 | 490.24 |
| DenseNet-201 | 224 | 20.0 M | 4.3 G | 77.2% | 0.89 | 1.10 | 642.98 |
| ResNeSt-50 | 224 | 27.5 M | 5.4 G | 81.1% | 0.76 | ‡ | 615.77 |
| ResNeSt-101 | 224 | 48.3 M | 10.2 G | 82.3% | 1.40 | ‡ | 1130.59 |
| ZenNet-0.1ms | 224 | 30.1 M | 1.7 G | 77.8% | 0.10 | 0.08 | 181.7 |
| ZenNet-0.2ms | 224 | 49.7 M | 3.4 G | 80.8% | 0.20 | 0.16 | 357.4 |
| ZenNet-0.3ms | 224 | 85.4 M | 4.9 G | 81.5% | 0.30 | 0.26 | 517.0 |
| ZenNet-0.5ms | 224 | 118 M | 8.3 G | 82.7% | 0.50 | 0.41 | 798.7 |
| ZenNet-0.8ms | 224 | 183 M | 13.9 G | 83.0% | 0.80 | 0.57 | 1365.0 |
| ZenNet-1.2ms | 224 | 180 M | 22.0 G | 83.6% | 1.20 | 0.85 | 2051.1 |
| ZenNet-400M-SE | 224 | 5.7 M | 410 M | 78.0% | 0.248 | 0.39 | 87.9 |
| ZenNet-600M-SE | 224 | 7.1 M | 611 M | 79.1% | 0.358 | 0.52 | 128.6 |
| ZenNet-900M-SE | 224 | 13.3 M | 926 M | 80.8% | 0.55 | 0.55 | 215.68 |

Table 7: One big table of all networks referred in this work.
$+$: OFANet trained using supernet parameters as initialization.
$*$: OFANet trained from scratch. We adopt this setting for fair comparison.
†: fail to run due to out of memory.
‡: official model implementation not supported by TensorRT.

# H Detail Structure of ZenNets

We list detail structure in Table 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18.

The 'block' column is the block type. 'Conv' is the standard convolution layer followed by BN and RELU. 'Res' is the residual block used in ResNet-18. 'Btn' is the residual bottleneck block used in ResNet-50. 'MB' is the MobileBlock used in MobileNet and EfficientNet. To be consistent with 'Btn' block, each 'MB' block is stacked by two MobileBlocks. That is, the kxk full convolutional layer in 'Btn' block is replaced by depth-wise convolution in 'MB' block. 'kernel' is the kernel size of kxk convolution layer in each block. 'in', 'out' and 'bottleneck' are numbers of input channels, output channels and bottleneck channels respectively. 'stride' is the stride of current block. '# layers' is the number of duplication of current block type.

| block | kernel | in | out | stride | bottleneck | # layers |
|---|---|---|---|---|---|---|
| Conv | 3 | 3 | 24 | 2 | - | 1 |
| Res | 3 | 24 | 32 | 2 | 64 | 1 |
| Res | 5 | 32 | 64 | 2 | 32 | 1 |
| Res | 5 | 64 | 168 | 2 | 96 | 1 |
| Btn | 5 | 168 | 320 | 1 | 120 | 1 |
| Btn | 5 | 320 | 640 | 2 | 304 | 3 |
| Btn | 5 | 640 | 512 | 1 | 384 | 1 |
| Conv | 1 | 512 | 2384 | 1 | - | 1 |

Table 8: ZenNet-0.1ms

| block | kernel | in | out | stride | bottleneck | # layers |
|---|---|---|---|---|---|---|
| Conv | 3 | 3 | 24 | 2 | - | 1 |
| Btn | 5 | 24 | 32 | 2 | 32 | 1 |
| Btn | 7 | 32 | 104 | 2 | 64 | 1 |
| Btn | 5 | 104 | 512 | 2 | 160 | 1 |
| Btn | 5 | 512 | 344 | 1 | 192 | 1 |
| Btn | 5 | 344 | 688 | 2 | 320 | 4 |
| Btn | 5 | 688 | 680 | 1 | 304 | 3 |
| Conv | 1 | 680 | 2552 | 1 | - | 1 |

Table 9: ZenNet-0.2ms

| block | kernel | in | out | stride | bottleneck | # layers |
|---|---|---|---|---|---|---|
| Conv | 3 | 3 | 24 | 2 | - | 1 |
| Btn | 5 | 24 | 64 | 2 | 32 | 1 |
| Btn | 3 | 64 | 128 | 2 | 128 | 1 |
| Btn | 7 | 128 | 432 | 2 | 128 | 1 |
| Btn | 5 | 432 | 272 | 1 | 160 | 1 |
| Btn | 5 | 272 | 848 | 2 | 384 | 4 |
| Btn | 5 | 848 | 848 | 1 | 320 | 3 |
| Btn | 5 | 848 | 456 | 1 | 320 | 3 |
| Conv | 1 | 456 | 6704 | 1 | - | 1 |

Table 10: ZenNet-0.3ms

| block | kernel | in | out | stride | bottleneck | # layers |
|---|---|---|---|---|---|---|
| Conv | 3 | 3 | 8 | 2 | - | 1 |
| Btn | 7 | 8 | 64 | 2 | 32 | 1 |
| Btn | 3 | 64 | 152 | 2 | 128 | 1 |
| Btn | 5 | 152 | 640 | 2 | 192 | 4 |
| Btn | 5 | 640 | 640 | 1 | 192 | 2 |
| Btn | 5 | 640 | 1536 | 2 | 384 | 4 |
| Btn | 5 | 1536 | 816 | 1 | 384 | 3 |
| Btn | 5 | 816 | 816 | 1 | 384 | 3 |
| Conv | 1 | 816 | 5304 | 1 | - | 1 |

Table 11: ZenNet-0.5ms

| block | kernel | in | out | stride | bottleneck | # layers |
|---|---|---|---|---|---|---|
| Conv | 3 | 3 | 16 | 2 | - | 1 |
| Btn | 5 | 16 | 64 | 2 | 64 | 1 |
| Btn | 3 | 64 | 240 | 2 | 128 | 2 |
| Btn | 7 | 240 | 640 | 2 | 160 | 3 |
| Btn | 7 | 640 | 768 | 1 | 192 | 4 |
| Btn | 5 | 768 | 1536 | 2 | 384 | 5 |
| Btn | 5 | 1536 | 1536 | 1 | 384 | 3 |
| Btn | 5 | 1536 | 2304 | 1 | 384 | 5 |
| Conv | 1 | 2304 | 4912 | 1 | - | 1 |

Table 12: ZenNet-0.8ms

| block | kernel | in | out | stride | bottleneck | # layers |
|---|---|---|---|---|---|---|
| Conv | 3 | 3 | 32 | 2 | - | 1 |
| Btn | 5 | 32 | 80 | 2 | 32 | 1 |
| Btn | 7 | 80 | 432 | 2 | 128 | 5 |
| Btn | 7 | 432 | 640 | 2 | 192 | 3 |
| Btn | 7 | 640 | 1008 | 1 | 160 | 5 |
| Btn | 7 | 1008 | 976 | 1 | 160 | 4 |
| Btn | 5 | 976 | 2304 | 2 | 384 | 5 |
| Btn | 5 | 2304 | 2496 | 1 | 384 | 5 |
| Conv | 1 | 2496 | 3072 | 1 | - | 1 |

Table 13: ZenNet-1.2ms

| block | kernel | in | out | stride | bottleneck | expansion | # layers |
|---|---|---|---|---|---|---|---|
| Conv | 3 | 3 | 16 | 2 | - | - | 1 |
| MB | 7 | 16 | 40 | 2 | 40 | 1 | 1 |
| MB | 7 | 40 | 64 | 2 | 64 | 1 | 1 |
| MB | 7 | 64 | 96 | 2 | 96 | 4 | 5 |
| MB | 7 | 96 | 224 | 2 | 224 | 2 | 5 |
| Conv | 1 | 224 | 2048 | 1 | - | - | 1 |

Table 14: ZenNet-400M-SE

| block | kernel | in | out | stride | bottleneck | expansion | # layers |
|---|---|---|---|---|---|---|---|
| Conv | 3 | 3 | 24 | 2 | - | - | 1 |
| MB | 7 | 24 | 48 | 2 | 48 | 1 | 1 |
| MB | 7 | 48 | 72 | 2 | 72 | 2 | 1 |
| MB | 7 | 72 | 96 | 2 | 88 | 6 | 5 |
| MB | 7 | 96 | 192 | 2 | 168 | 4 | 5 |
| Conv | 1 | 192 | 2048 | 1 | - | - | 1 |

Table 15: ZenNet-600M-SE

| block | kernel | in | out | stride | bottleneck | expansion | # layers |
|---|---|---|---|---|---|---|---|
| Conv | 3 | 3 | 16 | 2 | - | - | 1 |
| MB | 7 | 16 | 48 | 2 | 72 | 1 | 1 |
| MB | 7 | 48 | 72 | 2 | 64 | 2 | 3 |
| MB | 7 | 72 | 152 | 2 | 144 | 2 | 3 |
| MB | 7 | 152 | 360 | 2 | 352 | 2 | 4 |
| MB | 7 | 360 | 288 | 1 | 264 | 4 | 3 |
| Conv | 1 | 288 | 2048 | 1 | - | - | 1 |

Table 16: ZenNet-900M-SE

| block | kernel | in | out | stride | bottleneck | # layers |
|---|---|---|---|---|---|---|
| Conv | 3 | 3 | 88 | 1 | - | 1 |
| Btn | 7 | 88 | 120 | 1 | 16 | 1 |
| Btn | 7 | 120 | 192 | 2 | 16 | 3 |
| Btn | 5 | 192 | 224 | 1 | 24 | 4 |
| Btn | 5 | 224 | 96 | 2 | 24 | 2 |
| Btn | 3 | 96 | 168 | 2 | 40 | 3 |
| Btn | 3 | 168 | 112 | 1 | 48 | 3 |
| Conv | 1 | 112 | 512 | 1 | - | 1 |

Table 17: ZenNet-1.0M for CIFAR10/CIFAR100

| block | kernel | in | out | stride | bottleneck | # layers |
|---|---|---|---|---|---|---|
| Conv | 3 | 3 | 32 | 1 | - | 1 |
| Btn | 5 | 32 | 120 | 1 | 40 | 1 |
| Btn | 5 | 120 | 176 | 2 | 32 | 3 |
| Btn | 7 | 176 | 272 | 1 | 24 | 3 |
| Btn | 3 | 272 | 176 | 1 | 56 | 3 |
| Btn | 3 | 176 | 176 | 1 | 64 | 4 |
| Btn | 5 | 176 | 216 | 2 | 40 | 2 |
| Btn | 3 | 216 | 72 | 2 | 56 | 2 |
| Conv | 1 | 72 | 512 | 1 | - | 1 |

Table 18: ZenNet-2.0M for CIFAR10/CIFAR100