

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

Ming Lin ^{*1}, Pichao Wang ^{†1}, Zhenhong Sun ^{‡2}, Heseng Chen ^{§2}, Xiuyu Sun ^{¶2}, Qi Qian ^{||1},
Hao Li ^{**2}, and Rong Jin ^{††1}

¹Alibaba Group, Bellevue, Washington, USA

²Alibaba Group, Hangzhou, Zhejiang, China

February 2, 2021

Abstract

A key component in Neural Architecture Search (NAS) is an accuracy predictor which asserts the accuracy of a queried architecture. To build a high quality accuracy predictor, conventional NAS algorithms rely on training a mass of architectures or a big supernet. This step often consumes hundreds to thousands of GPU days, dominating the total search cost. To address this issue, we propose to replace the accuracy predictor with a novel model-complexity index named Zen-score. Instead of predicting model accuracy, Zen-score directly asserts the model complexity of a network without training its parameters. This is inspired by recent advances in deep learning theories which show that model complexity of a network positively correlates to its accuracy on the target dataset. The computation of Zen-score only takes a few forward inferences through a randomly initialized network using random Gaussian input. It is applicable to any Vanilla Convolutional Neural Networks (VCN-networks) or compatible variants, covering a majority of networks popular in real-world applications. When combining Zen-score with Evolutionary Algorithm, we obtain a novel Zero-Shot NAS algorithm named Zen-NAS. We conduct extensive experiments on CIFAR10/CIFAR100 and ImageNet. In summary, Zen-NAS is able to design high performance architectures in less than half GPU day (12 GPU hours). The resultant networks, named ZenNets, achieve up to 83.0% top-1 accuracy on ImageNet. Comparing to EfficientNets-B3/B5 of the same or better accuracies, ZenNets are up to 5.6 times faster on NVIDIA V100, 11 times faster on NVIDIA T4, 2.6 times faster on Google Pixel2 and uses 50% less FLOPs. Our source code and pre-trained models are released on <https://github.com/idstcv/ZenNAS>.

^{*}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

^{††}jinsong.jr@alibaba-inc.com

1 Introduction

Designing high performance deep neural networks which have high prediction accuracy and low inference cost is a challenging task. The emerging Neural Architecture Search (NAS) methods are developed to facilitate this progress. In most existing NAS algorithms, there are two key components, an architecture generator and an accuracy predictor. The generator proposes potential high-performance networks and asks the predictor to predict their accuracies. Popular generators include uniform sampling [Guo et al., 2020], evolutionary algorithm [Real et al., 2019] and reinforcement learning [Luo et al., 2018]. The accuracy predictor is even more critical than the generator. A low quality predictor might drive the search to a wrong direction. Therefore, nearly all previous NAS algorithms spend abundant computations in building a high quality accuracy predictor.

In the early researches of NAS, people used brute-force algorithms as accuracy predictors [Real et al., 2019, 2017, Xie and Yuille, 2017, Baker et al., 2017]. These methods directly train the queried network to obtain its accuracy. Clearly, the brute-force methods are prohibitive, taking hundreds to thousands of GPU days in one search. To address this issue, more efficient methods were developed. One approach is to parameterize the design space such that each network in the design space is encoded by a vector. Then a number of landmark networks are trained on the target dataset to obtain their accuracies. The accuracy predictor is then learned by fitting the accuracy-structure pairs [Luo et al., 2018, 2020]. However, these methods still need to train considerable number of networks to high enough precision.

Another approach is to construct a supernet which contains all possible sub-networks in design space, such as DART [Liu et al., 2019, Xu et al., 2019], OFANet [Cai et al., 2020] and many others [Zhou et al., 2020, Yang et al., 2020, Xie and Yuille, 2017, Xie et al., 2018, Cai et al., 2019, Zhang et al., 2020b, Wan et al., 2020]. These methods are known as **One-Shot** methods as they optimize the supernet via gradient descent. To answer a query of a given sub-network, the accuracy predictor evaluates the sub-network using supernet parameters as initialization. This parameter-sharing approach reduces the searching cost to a few hundred or thousand GPU hours while achieving competitive accuracy as those generated by brute-force methods.

Despite the great success of the above one-shot supernet approach, training supernet itself is still expensive. In order to include all architectures in the search space, a supernet is often 20~30 times larger than the target network. An annoy side effect of weight-sharing is model interfering [Cai et al., 2020, Ying et al., 2019] which results in degraded accuracy predictor [Sciuto et al., 2019]. In addition, since the supernet must be much larger than the target network, it is difficult to search high-accuracy large-capacity target networks.

In this work, we consider a fundamentally different approach. Since learning or building an accuracy predictor is expensive, we propose to replace the accuracy predictor with some easy-to-compute index. Our approach is inspired by recent advances in deep learning theories [Daniely et al., 2016, Lu et al., 2017, Levine et al., 2020, Fan et al., 2020] which show that the accuracy of a deep model is closely related to its model complexity: larger model complexity indicates smaller generalization error when providing a large enough training dataset [Koltchinskii, 2011]. This theoretical result seems well-aligned with large-scale deep learning practice, especially for very large and deep models [Nguyen et al., 2021, Dosovitskiy et al., 2021].

Based on the above observation, we design a novel index name Zen-score to replace the accuracy predictor in NAS. Zen-score measures the model complexity of a deep neural network instead of predicting its accuracy. It is applicable to any *Vanilla Convolutional Neural Network* (VCN-network). The computation of Zen-score only requires a few forward inferences on randomly initialized network using random Gaussian input, making it extremely fast and lightweight. Notably, Zen-score is immune to the phenomenon we called *model-collapse* caused by Batch Normalization (BN) layer [Ioffe and Szegedy, 2015]. Combining

Zen-score with Evolutionary Algorithm, we obtain a novel NAS algorithm named Zen-NAS. We categorize Zen-NAS as a **Zero-Shot** method since it does not optimize network parameters during search¹. We apply Zen-NAS to search optimal networks under various inference budgets, including inference latency, FLOPs (Floating Point Operations) and model size. The resultant networks, called ZenNets, achieve the state-of-the-art (SOTA) performance on CIFAR10/CIFAR100/ImageNet, outperforming previous human-designed and NAS-designed models by a large margin.

To measure the model complexity of a network, we require the network to be a VCN-network. The VCN-network space covers many previous manually-designed or NAS-designed networks, including ResNet [He et al., 2016], MobileNet [Pham et al., 2018], FBNet [Wan et al., 2020] and many more [Levine et al., 2020, Cai et al., 2020, Hu et al., 2018]. However, it does not cover multi-branch networks such as DenseNet [Huang et al., 2017], or networks with irregular cell-structures which are often preferred in supernet-based methods, such as design space used in [Ying et al., 2019]. This limitation is mostly due to the theoretical difficulty of measuring model-complexity in irregular design space. In other words, the main goal of Zen-NAS is to find efficient structures in a regular and general-enough search space, rather than in irregular and heterogeneous search space. Empirically, although our ZenNets are searched within the VCN-network space, it outperforms most supernet-based NAS methods which are searched in irregular design spaces.

We summarize our main contributions as follows:

- We propose a novel network performance index, Zen-score, to replace accuracy predictor in NAS.
- Based on Zen-score, we developed a novel NAS algorithm named Zen-NAS for large-scale evolutionary architecture search.
- Within half GPU day, the ZenNets automatically designed by Zen-NAS achieve up to 83.1% top-1 accuracy on ImageNet, which is as accurate as EfficientNet-B5. The inference speed of Zen-Net is several times faster than EfficientNets of the same or better accuracies on multiple GPU platforms and mobile devices.
- The largest searchable model in ZenNAS is unbounded above, which means that it can easily design much larger and better networks than the one (ZenNet-0.8ms) given in this work within 1 GPU day. In Figure 6 in the experiment section, there is still no sign of performance saturation at ZenNet-0.8ms. We did not explore even larger models because the final training of ZenNet-0.8ms already takes one week on 8 V100 GPU workstation. In other words, when using ZenNAS to design high-performance models, the real bottleneck is the training of final founded architecture rather than ZenNAS itself. This is very different to conventional NAS methods where the NAS step is the major cost.
- To our best knowledge, Zen-Net is the only network that is comparable to EfficientNet-B5 in top-1 accuracy on ImageNet while being 11 times faster on NVIDIA T4 GPU.

2 Related Work

In this section, we brief review closely related works. Our review is not intent to be comprehensive. For a comprehensive review of NAS, we refer to the monograph [Ren et al., 2020].

In the early days of NAS, people adopted brute-force methods to find new structures. The AmoebaNet [Real et al., 2019] used Evolutionary Algorithm (EA) to propose potential good structures. Each proposal is then trained from scratch to obtain its accuracy. Since this process is too expensive, AmoebaNet only did structural search on CIFAR10 [Krizhevsky, 2009] and then transferred the structure to ImageNet. It takes about 3150 GPU days of searching and achieves 74.5% top-1 accuracy on ImageNet, comparable to human-designed networks. Inspired by the success of AmoebaNet, many EA-based NAS algorithms were

¹Obviously, the final founded architecture must be trained on target dataset before deployment.

proposed to improve searching efficiency, such as EcoNAS [Zhou et al., 2020], CARS [Yang et al., 2020] and GeNet [Xie and Yuille, 2017]. These methods search on down-sampled images or reduce the number of queries. Reinforced Learning is another popular framework in NAS, including NASNet [Zoph et al., 2018], Mnasnet [Tan et al., 2019] and MetaQNN [Baker et al., 2017]. Both EA and RL require lots of network training, usually taking hundreds even thousands of GPU days. To avoid training large amounts of networks, PNAS [Liu et al., 2018] learns a performance predictor and only selects top- k cells to reduce the number of proposals in each iteration. However, PNAS still needs 224 GPU days to find a network of 74.2% accuracy.

To further reduce the training cost, weight-sharing based methods are proposed and become popular. The overall idea of weight-sharing is to construct a supernet to include all possible cells in the design space. Then the supernet is optimized via gradient descent (GD). A sub-network is then sampled from the supernet to test its accuracy, using supernet parameters as initialization. This framework is widely applied in many differentiable NAS methods, including DARTS [Liu et al., 2019], SNAS [Xie et al., 2018], PC-DARTS [Xu et al., 2019], ProxylessNAS [Cai et al., 2019], GDAS [Zhang et al., 2020b] and FBNetV2 [Wan et al., 2020], as well as other One-Shot NAS methods, such as DNANet [Li et al., 2020], Single-Path One-Shot NAS [Guo et al., 2020]. Very recently, [Luo et al., 2020] combines supernet with semi-supervised learning to progressively learn a better accuracy predictor. They successfully found 76.5% top-1 accuracy network within 4 GPU days.

Although the above efforts have greatly reduced the searching cost, their top-1 accuracies on ImageNet are still below 80.0%. The authors of OFANet [Cai et al., 2020] noted that weight-sharing suffers from model interfering. They propose a progressive-shrinking strategy to address the issue. The resultant OFANet achieves 80.1% accuracy after searching for 51.6 GPU days. EfficientNet [Tan and Le, 2019] 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 achieve 83.0% accuracy within half GPU day.

3 Vanilla Convolutional Neural Networks

In this section, we first introduce our notations. Then we define the Vanilla Convolutional Neural Network (VCN-network) family which forms our design space. Finally we discuss how to convert compatible non-VCN networks to VCN-network in order to compute their Zen-scores.

We denote an RGB image of width w and height h by a vector $\mathbf{x}_0 \in \mathbb{R}^{3 \times w \times h}$. We use $C_k(n, m, s)$ to denote a convolutional layer (Conv) of kernel size k , n input channels, m output channels and stride s . We omit s if $s = 1$. We also omit n if the number of input channels can be inferred from context. We denote BN layer as ‘B’, RELU layer as ‘R’. The residual block is denoted by $I(\cdot)$. By default, we always use BN+RELU after convolution and always connect residual link before RELU, which is consistent with ResNet. An L -layer VCN-network is stacked by L convolutional layers denoted by $\{\mathcal{F}_0, \mathcal{F}_1, \dots, \mathcal{F}_{L-1}\}$, where $\mathcal{F}_i(\cdot) = \text{RELU}(\text{BN}(\text{Conv}(\cdot)))$. Denote the feature map output of the i -th layer as $\mathbf{x}_{i+1} = \mathcal{F}_i(\mathbf{x}_i)$. The random Gaussian distribution of mean μ and deviation σ is denoted by $\mathcal{N}(\mu, \sigma^2)$.

Following previous works [He et al., 2016, Sandler et al., 2018, Pham et al., 2018], we generalize their design space and define our VCN-network family as follows. A VCN-network is a single-branch convolutional neural network, meaning that its main body is stacked by multiple layers of convolutional unit immediately followed by BN layer and RELU [Agarap, 2019] layer. At the end of the main body, global average pool layer (GAP) reduces the feature map resolution to 1x1, followed by a fully-connected layer. Finally a soft-max operation converts the network output to probability prediction.

Although the VCN-network family already covers a majority of classical networks, several popular architectures in NAS literatures are not covered yet. Some of these architectures could be slightly modified to

satisfy the VCN-network regularization, so that their Zen-score is computable. We call these architectures *compatible* to VCN-network. For example:

- Some networks use non-RELU activation function. For these networks, simply replacing non-RELU activation function with RELU before computing Zen-score;
- Some networks use small decoration blocks before or after convolutional layers, such as SE-block [Hu et al., 2018]. For these networks, simply removing the decoration blocks before computing Zen-score.

After the above modifications, we are able to compute Zen-score for compatible architectures. Please note that the modification is only valid during the Zen-score computation. The architecture of the network is not affected outside the Zen-score computation.

4 Zen-Score for Measuring Model Complexity

According to the complexity analysis of deep neural networks in previous literatures [Daniely et al., 2016, Lu et al., 2017, Levine et al., 2020], for a VCN-network **without BN layer**, its model complexity can be numerically estimated by the sensitivity of its output with respect to its input. Since in our approach, network parameters are randomly initialized, it is important to choose the most informative feature map as network output. We have three options: (a) the network output after soft-max; (b) feature map after GAP (post-GAP); (c) feature map before GAP (pre-GAP). Clearly, option (a) is not a good choice because the soft-max output is nearly a constant value in a randomly initialized network. Option (b) is better than (a) but it margins out spatial information. We suggest to use option (c) as it contains most information of the mapping function. To this end, we define the sensitivity-score, or Φ -score for short, for a VCN-network:

Definition 1 (Φ -score) *The sensitivity-score $\Phi(\mathcal{F})$ of an L -layer network $\mathcal{F}(\cdot)$ is defined by*

$$\Phi(\mathcal{F}) = \|\mathcal{F}(\epsilon + \alpha\epsilon') - \mathcal{F}(\epsilon)\|_p \quad (1)$$

where α is a small constant; $\|\cdot\|_p$ is ℓ_p -norm; $\epsilon, \epsilon' \sim \mathcal{N}(0, 1)$; $\mathcal{F}(\epsilon)$ is pre-GAP feature map. We find that $\alpha = 0.01, p = 1$ works well in practice.

It is important to distinguish Φ -score from gradient-norm of network parameters. First, gradient-norm requires a scalar loss function while Φ -score does not. Second, gradient-norm uses feature map after soft-max as network output while Φ -score use pre-GAP feature map as network output.

4.1 BN Layer and Model-Collapse

Using Φ -score to measure model complexity requires that the network contains no BN layer. As BN layer is widely used nowadays, it is important to measure model complexity in the present of BN layers. We find that when there are BN layers in a network, directly using Φ -score as model complexity fails to work. The Φ -score of a large network could be nearly the same as the Φ -score of a much smaller network. We name this phenomenon as **model-collapse**.

To demonstrate model-collapse phenomenon, we design two set of toy networks as examples. The first set of toy networks, denoted as P_{BN} networks, have network width 64 in all layers with number of layers varying from 1 to 30. Each layer is in form of $\{C_3(64)\text{BR}\}$. The $P_{\text{w/oBN}}$ are networks with BN removed, that is, $\{C_3(64)\text{R}\}$. The second set of toy networks, denoted as Q_{BN} networks, are in form of

$$\{C_3(3, 64)\text{BR}, C_3(64, m)\text{BR}, C_3(m, 64)\text{BR}\}$$

where m varies from 2 to 64. Similarly, $Q_{\text{w/oBN}}$ denotes networks with BN removed.

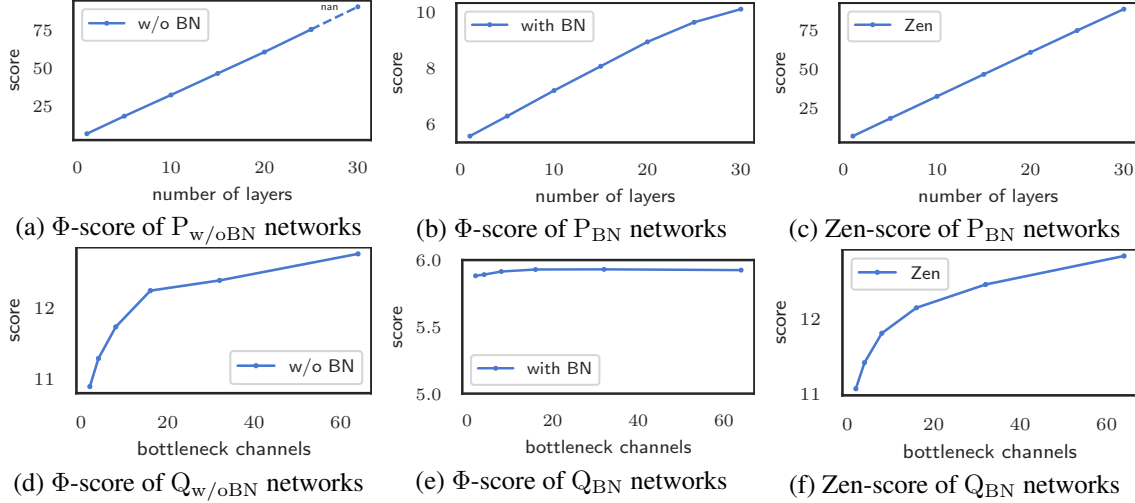


Figure 1: Φ -scores and Zen-scores of networks, with different depths and bottleneck channels.

Now, let us compute the Φ -score for $P_{w/oBN}$ against different number of layers. The curve is shown in Figure 1(a). We note that the Φ -score overflows the float32 maximum range after 30 layers. Therefore, in practice we cannot remove BN when computing Φ -score, especially for deep networks. For P_{BN} , the Φ -score is plotted in Figure 1(b). The numerical overflow problem is well-addressed thanks to the BN layer. In this figure, BN layers cause model-collapse where Φ -scores are scaled down by a constant.

In the second toy example, we plot Φ -score of Q networks with or without BN in Figure 1(d) and (e). Clearly, when BN layer is present, the Φ -score becomes nearly constant. This will confuse the architecture generator and drive the search to a wrong direction.

4.1.1 BN with Residual Link

In addition to BN layer, residual link also interferes model-complexity measurement when BN layer is present. To see this, we recall the toy example $P_{w/oBN}$ defined above. Now we add residual link in form of $\{I(C_3(64))R\}$. We compute Φ -score of $P_{w/oBN}$ with and without residual link respectively in Figure 2(a). The Φ -scores of two networks equal to each other. This is not surprise because the residual link cannot increase model complexity. In Figure 2(b), we additionally insert BN layer after each convolutional layer. We still compute Φ -score directly. This time the Φ -score no longer increases linearly against the number of layers and is even decayed for deeper networks. This phenomenon is caused by using residual link with BN layer together. Based on the above observation, it is a must to remove all residual links before numerically measuring the model complexity. This step does not alternate the ground-truth model complexity under measurement.

We design a novel index named Zen-score to measure model complexity when BN layer is present in the architecture. The correctness of Zen-score is guaranteed by the following two mathematical facts:

Fact-1 RELU is transparent to scaling:

$$\text{RELU}(t/\sigma) = \text{RELU}(t)/\sigma \quad \forall \sigma > 0.$$

Fact-2 For a given x and $W \sim \mathcal{N}(0, 1)$, $\mathbb{E}\{Wx\} = 0$.

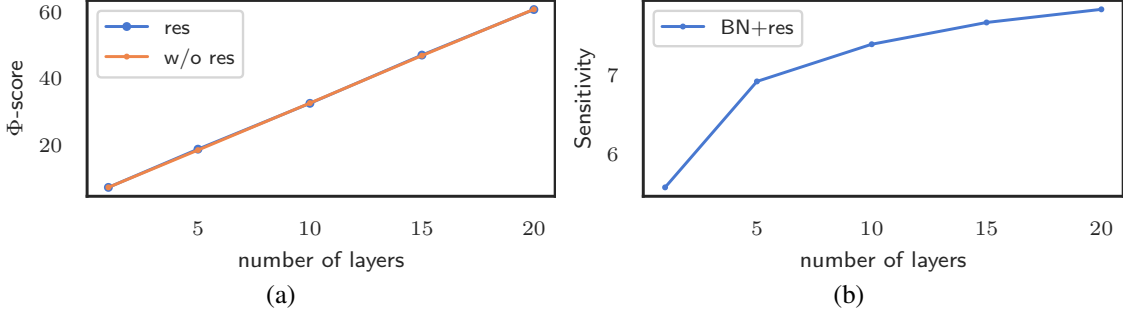


Figure 2: (a) Φ -score of P_{woBN} , with or w/o residual link; (b) Φ -score of P_{BN} + residual link.

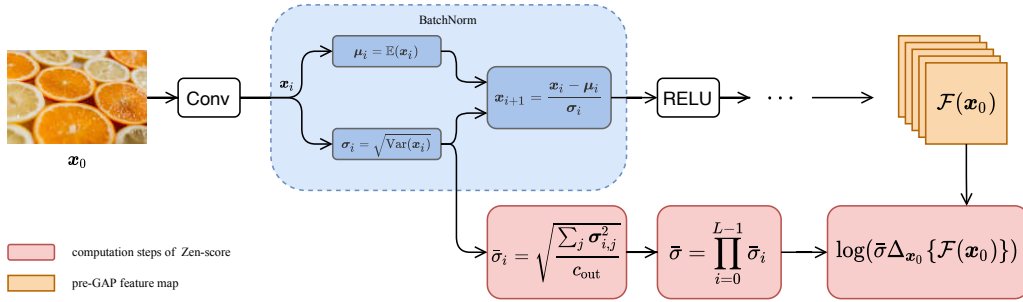


Figure 3: Zen-score computation steps. x_0 is one mini-batch of input images. For each BN layer, we extract its mini-batch deviation parameter σ_i . $\Delta_{x_0} \{\mathcal{F}(x_0)\}$ is the differential of pre-GAP feature map $\mathcal{F}(x_0)$ with respect to x_0 .

Combining the above two facts, we observe that when the batch size is large enough, the output of BN+RELU is almost equal to using RELU alone and then multiplying a scaling constant $1/\sigma$, where σ is the standard deviation of the feature map distribution. In most implementation, BN layer normalizes each channel independently. To address this issue, we average over each channel's σ . That is, we define the standard deviation $\bar{\sigma}_i$ of the i -th BN layer with m output channels as $\bar{\sigma}_i = \sqrt{\sum_{j=1}^m \sigma_{i,j}^2 / m}$.

4.2 Zen-Score against Model-Collapse

The computation of Zen-score is detailed in Algorithm 1. Figure 3 illustrates key computation steps. In Algorithm 1, we first remove all residual links in the network. Then we randomly sample many random input vectors and perturb them with Gaussian noise. After that we compute the perturbation of the pre-GAP feature map, denoted as Δ . To get Zen-score, the scaling factor $\bar{\sigma}_i$ of each BN layer is averaged from the mini-batch standard deviation statistic of each channel. Finally the Zen-score is computed by the log-sum of Δ and $\bar{\sigma}_i$.

To validate that Zen-score does not suffer from model-collapse, we plot Zen-scores of P_{BN} and Q_{BN} networks in Figure 1 (c)(f) respectively. In these figures, the Zen-score perfectly approximates the Φ -scores, showing that Zen-score is immune to model-collapse.

Algorithm 1 Zen-Score

Require: Network $F(\cdot)$ with pre-GAP feature map $f(\cdot)$; $\alpha = 0.01, p = 1$.

Ensure: Zen-score $\text{Zen}(F)$.

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

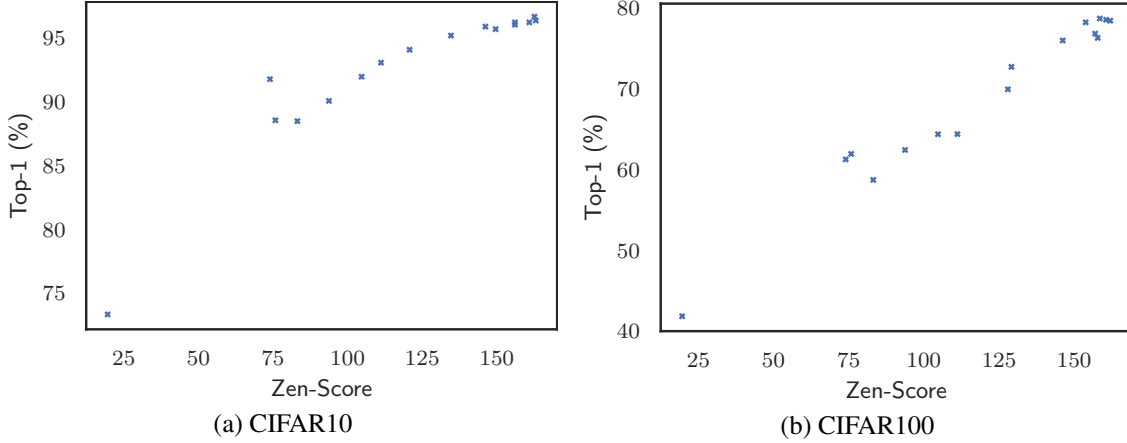


Figure 4: Zen-score v.s. top-1 accuracy, 16 randomly sampled structures of same model size equal to ResNet-50.

4.3 Effectiveness of Zen-Score

We show that Zen-score effectively indicates the model accuracy. We use ResNet-50 for CIFAR as initial structure. Then we randomly perturb the structure, for example, shrinking high-level channels and expanding low-level channels, removing layers. After random perturbation, we uniformly scale-up or scale-down the network width such that the number of network parameters is equal to that of ResNet-50. In this way, we random 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. There are large variance in the low-precision regimes. This is because these poor structures are far from reasonable ones and little could be expected in these outlier areas.

4.4 Zen-NAS via Large-Scale Evolution

The computation of Zen-score is extremely fast. Based on Zen-score, we design a novel Zen-NAS algorithm for large-scale network architecture evolution. The step-by-step description of Zen-NAS is given in Algorithm 2.

In Algorithm 2, we randomly generate T structures via evolutionary algorithm. At each iteration step

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^* .

```
1: Initialize population  $\mathcal{P} = \{F_0\}$ .
2: for  $t = 1, 2, \dots, T$  do
3:   if  $t < T/2$  then
4:      $\lambda = (T - 2t)/T, \rho = \lambda + 0.1(1 - \lambda)$ .
5:   else
6:      $\rho = 0.1$ .
7:   end if
8:   Randomly select  $F_t \in \mathcal{P}$ .
9:   Mutate  $\hat{F}_t = \text{MUTATE}(F_t, \rho, \mathcal{S})$ 
10:  if  $\hat{F}_t$  exceeds inference budget then
11:    Do nothing.
12:  else
13:    Get Zen-score  $z = \text{Zen}(\hat{F}_t)$ .
14:    Append  $\hat{F}_t$  to  $\mathcal{P}$ .
15:  end if
16:  Remove network of the smallest Zen-score if the size of  $\mathcal{P}$  exceeds  $B$ .
17: end for
18: Return  $F^*$ , the network of the highest Zen-score in  $\mathcal{P}$ .
```

Algorithm 3 MUTATE

Require: Structure F_t , probability ρ of mutation, search space \mathcal{S} .

Ensure: Randomly mutated structure \hat{F}_t .

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

t , we randomly draw a structure in the population \mathcal{P} and mutate it. The mutation algorithm is presented in Algorithm 3. With probability ρ , we update the basic block type of a randomly selected super-block. We encourage the random exploration in the first half $T/2$ iterations by setting ρ decaying from 1 to 0.1. Then the width and depth of the selected super-block 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 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 the smallest Zen-score items. 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.

model	# params	FLOPs	Top-1 Accuracy	
			CIFAR10	CIFAR100
ZenNet-0.5M	0.5 M	140 M	96.2%	79.9%
ZenNet-1.0M	1.0 M	162 M	96.2%	80.1%
ZenNet-2.0M	2.0 M	487 M	97.5%	84.4%

Table 1: ZenNet-0.5M/1.0M/2.0M on CIFAR10/CIFAR100.

5 Experiments

In this section, we design numerical experiments to validate the superiority of Zen-NAS in real-world problems.

Dataset We conduct experiments on CIFAR10/CIFAR100 [Krizhevsky, 2009] and ImageNet-1k [Deng et al., 2009]. 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 [Pham et al., 2018]: mix-up [Zhang et al., 2018], label-smoothing [Szegedy et al., 2016], random erasing [Zhong et al., 2020], random crop/resize/flip/lighting and AutoAugment [Cubuk et al., 2019].

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 [Loshchilov and Hutter, 2017]. We train models up to 1440 epochs in CIFAR10/100, 480 epochs in ImageNet. Following previous works [Aguilar et al., 2020, Li et al., 2020, Cai et al., 2020]. We use ResNet-152 as teacher network to train the target network.

Search Space To align with previous works, we consider two search spaces respectively:

- **Search Space I** Following [He et al., 2016, Radosavovic et al., 2020], this search space consists of Residual Blocks and Bottleneck Blocks defined in ResNet. The number of bottleneck channels is searchable.
- **Search Space II** Following [Sandler et al., 2018, Pham et al., 2018], this search space consists of MobileNet Blocks. The depth-wise expansion ratio is searched in set $\{1, 2, 4, 6\}$.

In each Zen-NAS searching 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 fixed to be four for CIFAR10/CIFAR100 and five for ImageNet. In Zen-NAS (Algorithm 2), we run evolutionary algorithm for $T = 48,000$ iterations. The evolutionary population size is 256. The running time for each search is around $8 \sim 12$ GPU hours.

5.1 Zen-NAS on CIFAR10/CIFAR100

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 $\{0.5 \text{ M}, 1.0 \text{ M}, 2.0 \text{ M}\}$. The resultant networks are labeled as ZenNet-0.5M/1.0M/2.0M. Table 1 summarized our results on CIFAR10/CIFAR100. We compare several popular NAS-designed models for CIFAR10/CIFAR100 in Figure 5, including AmoebaNet [Real et al., 2019], DARTS [Liu et al., 2019], P-DARTS [Chen et al.,

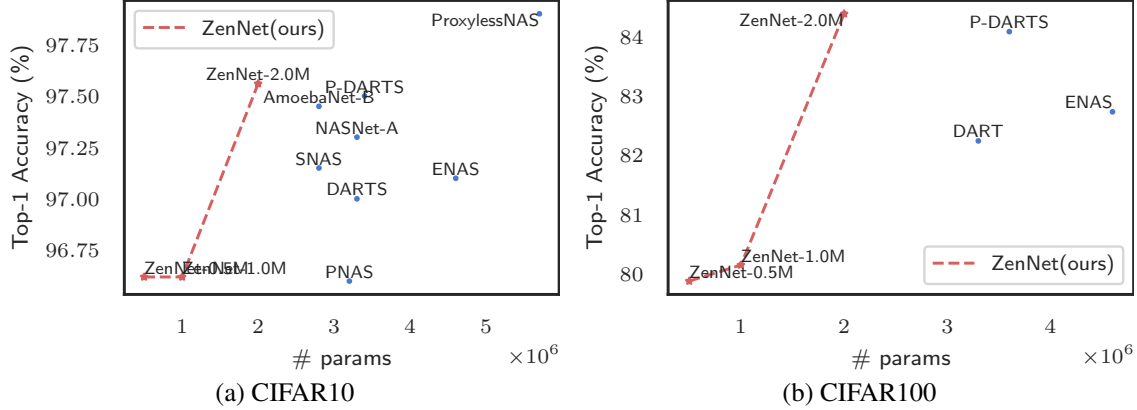


Figure 5: ZenNet accuracy v.s. model size (# params).

2019], SNAS [Xie et al., 2018], NASNet-A [Zoph et al., 2018], ENAS [Pham et al., 2018], PNAS [Liu et al., 2018], ProxylessNAS [Cai et al., 2019]. ZenNets outperform baseline methods by 30% ~ 50% parameter reduction while achieving the same accuracies.

5.2 Zen-NAS on ImageNet

We use Zen-NAS to search for efficient ZenNets on ImageNet. We consider the following popular networks as baselines: (a) manually-designed networks, including ResNet [He et al., 2016], DenseNet [Huang et al., 2017], ResNeSt [Zhang et al., 2020a]; MobileNet-V2 [Sandler et al., 2018] (b) NAS-designed networks for fast inference on GPU, including OFANet-9ms/11ms [Cai et al., 2020], DFNet [Li et al., 2019], RegNet [Radosavovic et al., 2020]; (c) NAS-designed networks optimized for FLOPs, including OFANet-389M/482M/595M [Cai et al., 2020], DNANet [Li et al., 2020], EfficientNet [Tan and Le, 2019], Mnasnet [Tan et al., 2019], DFNet [Li et al., 2019].

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~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 [Zhang et al., 2020a, Cai et al., 2020, Radosavovic et al., 2020].

5.2.1 Searching Low Inference Latency Networks for NVIDIA V100 GPU

Following previous works [Cai et al., 2020, Radosavovic et al., 2020, Li et al., 2019], 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 benchmarked at batch size 64, half precision (float16). We search for networks with inference latency within 0.1/0.2/0.3/0.5/0.8 milliseconds (ms) per image. For benchmarking inference latency, we set batch size=64 and do mini-batch inference 30 times. The averaged inference latency is recorded. The resolution is 160/192/192/224/224 for 0.1/0.2/0.3/0.5/0.8 ms respectively. The resultant networks are labeled as ZenNet-0.1/0.2/0.3/0.5/0.8ms. The top-1 accuracy on ImageNet v.s. inference latency is plotted in Figure 6.

Clearly, ZenNets outperform baseline models in both accuracy and inference speed by a large margin.

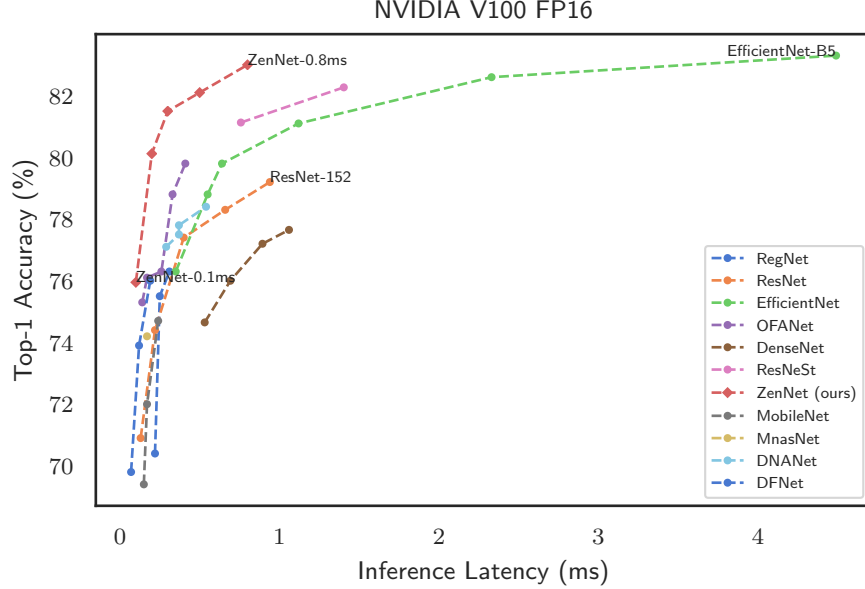


Figure 6: ZenNets top-1 accuracy v.s. inference latency (milliseconds per image) on ImageNet. Benchmarked on NVIDIA V100 GPU, half precision (FP16), batch size 64, searching cost 0.5 GPU day.

ZenNet-0.8ms achieves 83.0% top-1 accuracy on ImageNet, as good as EfficientNet-B5 while its inference speed is 5.6 times faster than EfficientNet-B5. The OFANets are the runner-up efficient models among NAS-designed models. Since OFANet is based on supernet framework, it is difficult to search for very large target networks. The best model released in OFANet paper is around 80% top-1 accuracy, comparable to EfficientNet-B2. Among manually designed networks, ResNeSt achieves the best accuracy-speed trade-off. The top-1 accuracy of ResNeSt-101 is between EfficientNet-B4 to EfficientNet-B5 while being 3.2 times faster than EfficientNet-B5.

5.2.2 Searching Lightweight Networks

Following previous works, we use Zen-NAS to search lightweight networks with small FLOPs. We use Search Space II in this experiment. We search for networks with inference latency within 400/600/900 M FLOPs. Similar to OFANet and EfficientNet, we add SE-blocks after convolutional layers. The resultant networks are labeled as ZenNet-300/400/900M-SE with input image resolution 192/256/224 respectively. The top-1 accuracy v.s. FLOPs is plotted in Figure 7. Again, ZenNets outperform most FLOP-efficient models by a large margin. ZenNet-900M-SE achieves 80.8% top-1 accuracy which is comparable to EfficientNet-B3 with 50% less FLOPs. The runner-up is OFANet whose efficiency is similar to ZenNet. It is unclear how OFANet performs at 900 MFLOPs since the original authors did not release better model.

5.2.3 ZenNet Inference Speed on NVIDIA T4 GPU and Google Pixel2 Mobile GPU

We demonstrate that ZenNets are able to outperform baseline models on various heterogeneous hardware platforms. To this end, we benchmark inference speed of ZenNet on NVIDIA T4 GPU and Google Pixel2 mobile GPU. The NVIDIA T4 GPU is highly optimized for deep neural network inference and widely used

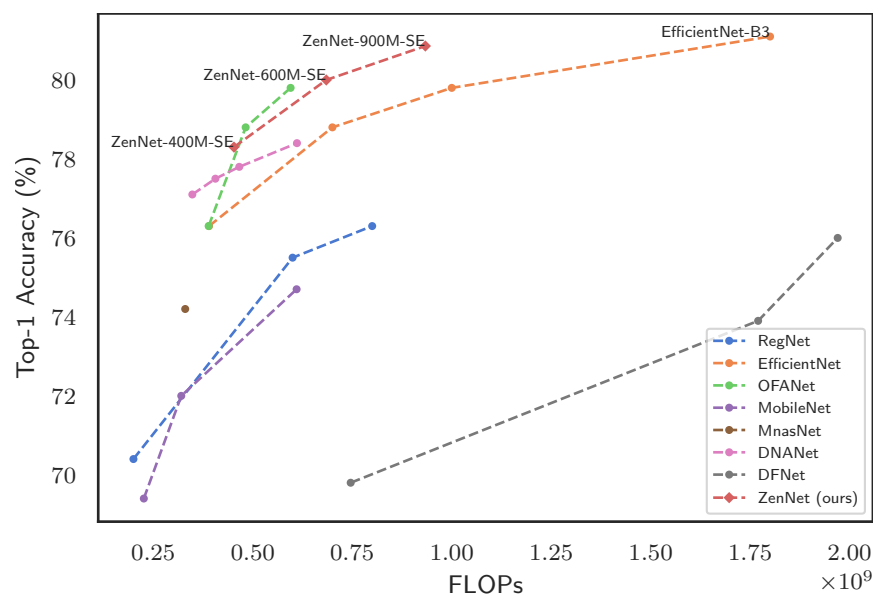


Figure 7: ZenNets optimized for FLOPs.

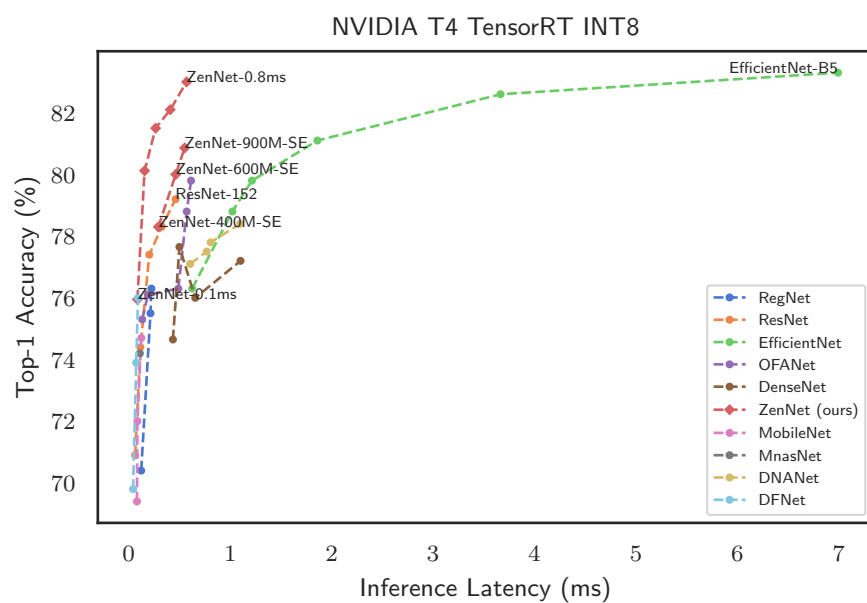


Figure 8: ZenNets top-1 accuracy on ImageNet-1k v.s. inference latency (milliseconds per image) on NVIDIA T4, TensorRT INT8, batch size 64.

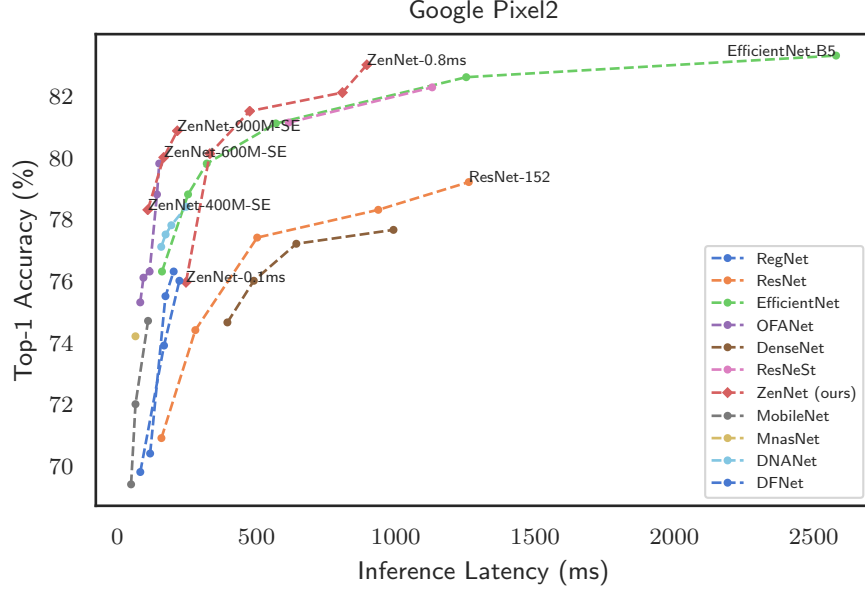


Figure 9: ZenNets top-1 accuracy on ImageNet-1k v.s. inference latency (milliseconds per image) on Google Pixel2, single image.

model	repeat	rel-err(%)	time
ZenNet-2.0M	4	0.48%	0.20 s
	8	0.34%	0.41 s
	16	0.24%	0.81 s
	32	0.17%	1.62 s
ZenNet-0.8ms	4	0.26%	0.85 s
	8	0.18%	1.68 s
	16	0.13%	3.36 s
	32	0.09%	6.72 s

Table 2: Zen-score computational cost. 95% confidence interval.

in industrial online services. Google Pixel2 is a cell-phone with a moderate mobile GPU. The inference latency of ZenNets and baseline models are plotted in Figure 8 and Figure 9. In both figures, ZenNets outperform baseline models significantly. On T4 GPU, ZenNet-0.8ms is more than 11 times faster than EfficientNet-B5. It is also 2.0 times faster than EfficientNet-B5 on Pixel2. Among lightweight models, ZenNet-900M-SE is 2.6 times faster than EfficientNet-B3.

5.3 Zen-NAS Searching Speed

The major time cost of Zen-NAS is the computation of Zen-score. In Table 2, we report the computational cost of Zen-score. We set batch size 64 and then compute the averaged Zen-score over 64 mini-batch

Architecture	Framework	Top-1 (%)	GPU Day
AmoebaNet-A [Real et al., 2019]	EA	74.5	3150†
EcoNAS [Zhou et al., 2020]	EA	74.8	8
CARS-I [Yang et al., 2020]	EA	75.2	0.4
GeNet [Xie and Yuille, 2017]	EA	72.1	17
DARTS [Liu et al., 2019]	GD	73.1	4
SNAS [Xie et al., 2018]	GD	72.7	1.5
PC-DARTS [Xu et al., 2019]	GD	75.8	3.8
ProxylessNAS [Cai et al., 2019]	GD	75.1	8.3
GDAS [Zhang et al., 2020b]	GD	74	0.8
FBNetV2-L1 [Wan et al., 2020]	GD	77.2	25
NASNet-A [Zoph et al., 2018]	RL	74	1800
Mnasnet-A [Tan et al., 2019]	RL	75.2	-
MetaQNN [Baker et al., 2017]	RL	77.4	96
PNAS [Liu et al., 2018]	SMBO	74.2	224
SemiNAS [Luo et al., 2020]	SSL	76.5	4
OFANet [Cai et al., 2020]	PS	80.1	51.6
EfficientNet-B7 [Tan and Le, 2019]	Scaling	84.4	3800‡
ZenNet-0.8ms	EA	83.0	0.5

Table 3: NAS searching cost comparison. 'Top-1': top-1 accuracy on ImageNet-1k. 'Framework': 'EA' is short for Evolutionary Algorithm; 'GD' is short for Gradient Descent; 'RL' is short for reinforcement Learning; 'SMBO', 'SSL', 'PS' and 'Scaling' are special searching methods/frameworks. †: Running on TPU; ‡: The cost is estimated by [Wan et al., 2020];

instances. This process is repeated several times (2nd column). We report the relative statistical error of Zen-score in the 3rd column. The statistical error is estimated from the 95% confidence interval of random repeats. The time cost is reported in the 4th column. We take ZenNet-2.0M and ZenNet-0.8ms as example. The other models have similar computational cost. The Zen-score must be computed in full-precision (FP32) therefore the inference speed in Table 2 is different from the half-precision speed.

From Table 2, it is sufficient to get better than 0.5% relative error with probability at least 95% by 4 times forward inferences of mini-batch 64. This means that scoring 48,000 networks similar to ZenNet-0.8ms only takes 12 GPU hours, or 0.5 GPU day.

When we have latency constraint, a major factor that slows down the searching is the latency benchmark. To address this issue, we learn a latency prediction model. First, we randomly generate a large number of architectures and measure their latencies. Then we encode each architecture into a vector. A small 10-layer perceptron network is trained to predict the latency. The prediction error is within 5% for 99% architectures. In this way, we could nearly ignore the time cost of latency prediction in Zen-NAS.

5.3.1 Searching Cost of Zen-NAS v.s. SOTA

We compare the Zen-NAS searching cost to SOTA NAS methods in Table 3. Since each NAS method use different settings, including different search spaces, training methods, total number of networks quired during search, it is difficult to make a fair comparison that everyone agrees with. Nevertheless, in Table 3

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 4: Zen-scores of ResNets.

we only concern about the best model obtained in each NAS method and the corresponding searching cost. This would give us a rough estimation of the efficiency of each NAS method and their maximal ability to design high-performance models.

From Table 3, we could see that for conventional NAS methods, it takes hundreds to thousands GPU days to find a good structure of accuracy better than 78.0%. Many supernet-based methods are very fast. Some supernet-based methods could run within 1 GPU day. This is because these methods usually search on small datasets such as CIFAR and then transfer structures to ImageNet. Some directly search on ImageNet but use down-sampled resolution or early stopping. However, these acceleration tricks all incur side effects, leading to less optimal structures. The best model searchable by supernet-based methods is limited by the size of the supernet, which is often 10 times smaller than supernet. As a consequence, all supernet-based methods struggle to find better than 80% top-1 accuracy models on ImageNet.

In comparison, Zen-NAS achieves 83.0% top-1 accuracy with searching cost less than 0.5 GPU day. Among methods achieving above 80.0% top-1 accuracy in Table 3, the searching speed of Zen-NAS is 100 times faster than OFANet and 7200 times faster than EfficientNet.

5.4 Zen-Scores of ResNets

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 4.

Next we show that the Zen-scores are positively correlated to the top-1 accuracies. We consider two baselines in Table 5. The 2nd column reports the top-1 accuracies obtained in the ResNet original paper [He et al., 2016]. 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.

In Figure 10, we plot the Zen-score against top-1 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.

6 Conclusion

We proposed Zen-NAS, a zero-shot neural architecture search method for deep image recognition. Without optimizing network parameters, Zen-NAS is able to identify good network structures via measuring a model-

Model	Top-1 [He et al., 2016]	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 5: Top-1 accuracies of ResNet. Reported by [He et al., 2016] and using enhanced training methods we used in this paper.

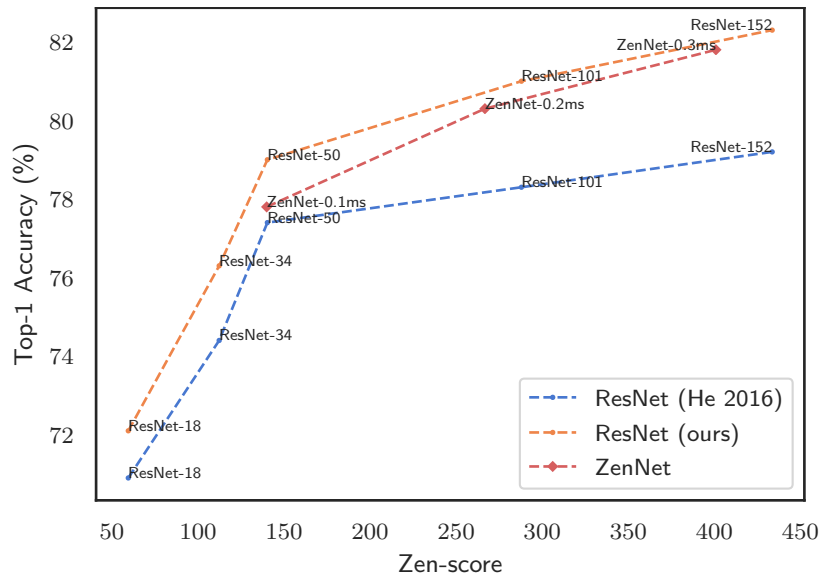


Figure 10: ResNet/ZenNet Zen-score v.s. top-1 accuracy on ImageNet.

complexity index named Zen-score. The searching speed of Zen-NAS is 100 to 7200 times faster than previous SOTA methods, while the resultant networks are significantly more efficient in terms of inference latency, FLOPs and model size, in various vision tasks. We wish the elegance of Zen-NAS will inspire more theoretical researches towards a deeper understanding of efficient network design.

References

- Abien Fred Agarap. Deep Learning using Rectified Linear Units (ReLU). *arXiv:1803.08375 [cs, stat]*, 2019. 4
- Gustavo Aguilar, Yuan Ling, Yu Zhang, Benjamin Yao, Xing Fan, and Chenlei Guo. Knowledge Distillation from Internal Representations. In *AAAI*, 2020. 10

- Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing Neural Network Architectures using Reinforcement Learning. In *ICLR*, 2017. 2, 4, 15
- Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In *ICLR*, 2019. 2, 4, 11, 15
- 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. 2, 3, 4, 10, 11, 15, 25
- Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive DARTS: Bridging the Optimization Gap for NAS in the Wild. In *ICCV*, 2019. 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. 10
- Amit Daniely, Roy Frostig, and Yoram Singer. Toward Deeper Understanding of Neural Networks: The Power of Initialization and a Dual View on Expressivity. *NIPS*, 29:2253–2261, 2016. 2, 5
- 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. 10
- Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR*, 2021. 2
- Feng-Lei Fan, Rongjie Lai, and Ge Wang. Quasi-Equivalence of Width and Depth of Neural Networks. *arXiv:2002.02515 [cs, stat]*, 2020. 2
- 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. 2, 4
- K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *CVPR*, pages 770–778, 2016. 3, 4, 10, 11, 16, 17
- Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. Squeeze-and-Excitation Networks. In *CVPR*, 2018. 3, 5
- Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely Connected Convolutional Networks. In *CVPR*, pages 2261–2269, 2017. 3, 11
- Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *ICML*, 2015. 2
- Vladimir Koltchinskii. *Oracle Inequalities in Empirical Risk Minimization and Sparse Recovery Problems*, volume 2033. Springer Science & Business Media, 2011. 2
- Alex Krizhevsky. Learning multiple layers of features from tiny images, 2009. 3, 10
- Yoav Levine, Noam Wies, Or Sharir, Hofit Bata, and Amnon Shashua. Limits to Depth Efficiencies of Self-Attention. In *NIPS*, volume 33, 2020. 2, 3, 5
- Changlin Li, Jiefeng Peng, Liuchun Yuan, Guangrun Wang, Xiaodan Liang, Liang Lin, and Xiaojun Chang. Blockwisely Supervised Neural Architecture Search with Knowledge Distillation. In *CVPR*, 2020. 4, 10, 11
- 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

- 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. 4, 11, 15
- Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable Architecture Search. In *ICLR*, 2019. 2, 4, 10, 15
- Ilya Loshchilov and Frank Hutter. SGDR: Stochastic Gradient Descent with Warm Restarts. In *ICLR*, 2017. 10
- Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The Expressive Power of Neural Networks: A View from the Width. *NIPS*, 30:6231–6239, 2017. 2, 5
- Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural Architecture Optimization. In *NIPS*, pages 7827–7838, 2018. 2
- Renqian Luo, Xu Tan, Rui Wang, Tao Qin, Enhong Chen, and Tie-Yan Liu. Semi-Supervised Neural Architecture Search. In *NIPS*, 2020. 2, 4, 15
- 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
- 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. 3, 4, 10, 11
- Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing Network Design Spaces. In *CVPR*, 2020. 10, 11, 25
- 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. 2
- Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized Evolution for Image Classifier Architecture Search. In *AAAI*, 2019. 2, 3, 10, 15
- 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. 3
- Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *CVPR*, 2018. 4, 10, 11
- 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
- Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. In *CVPR*, pages 2818–2826, 2016. 10
- Mingxing Tan and Quoc Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *ICML*, pages 6105–6114, 2019. 4, 11, 15
- Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V. Le. MnasNet: Platform-Aware Neural Architecture Search for Mobile. In *CVPR*, 2019. 4, 11, 15
- 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. 2, 3, 4, 15
- Lingxi Xie and Alan Yuille. Genetic CNN. In *ICCV*, 2017. 2, 4, 15

- Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: Stochastic neural architecture search. In *ICLR*, 2018. 2, 4, 11, 15
- 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. 2, 4, 15
- 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. 2, 4, 15
- 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, 3
- 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, 2020a. 11
- Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. Mixup: Beyond Empirical Risk Minimization. In *ICLR*, 2018. 10
- 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, 2020b. 2, 4, 15
- Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random Erasing Data Augmentation. In *AAAI*, 2020. 10
- Dongzhan Zhou, Xinchu 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. 2, 4, 15
- Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning Transferable Architectures for Scalable Image Recognition. In *CVPR*, 2018. 4, 11, 15

A One Big Table of All Networks on ImageNet

model	resolution	# params/M	FLOPs/M	Top-1 Acc	latency(ms)		
					V100	T4	Pixel2
RegNetY-200MF	224	3.2	200	70.4%	0.22	0.12	118.17
RegNetY-400MF	224	4.3	400	74.1%	0.44	0.17	181.09
RegNetY-600MF	224	6.1	600	75.5%	0.25	0.21	173.19
RegNetY-800MF	224	6.3	800	76.3%	0.31	0.22	202.66
ResNet-18	224	11.7	1822	70.9%	0.13	0.06	158.70
ResNet-34	224	21.8	3670	74.4%	0.22	0.11	280.44
ResNet-50	224	25.6	4120	77.4%	0.40	0.20	502.43
ResNet-101	224	44.5	7800	78.3%	0.66	0.32	937.11
ResNet-152	224	60.2	11580	79.2%	0.94	0.46	1261.97
EfficientNet-B0	224	5.3	390	76.3%	0.35	0.62	160.72
EfficientNet-B1	240	7.8	700	78.8%	0.55	1.02	254.26
EfficientNet-B2	260	9.2	1000	79.8%	0.64	1.21	321.45
EfficientNet-B3	300	12.0	1800	81.1%	1.12	1.86	569.30
EfficientNet-B4	380	19.0	4200	82.6%	2.33	3.66	1252.79
EfficientNet-B5	456	30.0	9900	83.3%	4.49	6.99	2580.25
EfficientNet-B6	528	43.0	19000	84.0%	7.64	12.36	4287.81
EfficientNet-B7	600	66.0	37000	84.4%	13.73	†	8615.92
MobileNetV2-0.25	224	1.5	44	51.8%	0.08	0.04	16.71
MobileNetV2-0.5	224	2.0	108	64.4%	0.10	0.05	26.99
MobileNetV2-0.75	224	2.6	226	69.4%	0.15	0.08	49.78
MobileNetV2-1.0	224	3.5	320	72.0%	0.17	0.08	65.59
MobileNetV2-1.4	224	6.1	610	74.7%	0.24	0.12	110.70
MnasNet-1.0	224	4.4	330	74.2%	0.17	0.11	65.50
DNANet-a	224	4.2	348	77.1%	0.29	0.60	157.94
DNANet-b	224	4.9	406	77.5%	0.37	0.77	173.66
DNANet-c	224	5.3	466	77.8%	0.37	0.81	194.27
DNANet-d	224	6.4	611	78.4%	0.54	1.10	248.08
DFNet-1	224	8.5	746	69.8%	0.07	0.04	82.87

DFNet-2	224	18.0	1770	73.9%	0.12	0.07	168.04
DFNet-2a	224	18.1	1970	76.0%	0.19	0.09	223.20
OFANet-9ms	118	5.2	313	75.3%	0.14	0.13	82.69
OFANet-11ms	192	6.2	352	76.1%	0.17	0.19	94.17
OFANet-389M(+)	224	8.4	389	79.1%	0.26	0.49	116.34
OFANet-482M(+)	224	9.1	482	79.6%	0.33	0.57	142.76
OFANet-595M(+)	236	9.1	595	80.0%	0.41	0.61	150.83
OFANet-389M*	224	8.4	389	76.3%	0.26	0.49	116.34
OFANet-482M*	224	9.1	482	78.8%	0.33	0.57	142.76
OFANet-595M*	236	9.1	595	79.8%	0.41	0.61	150.83
DenseNet-121	224	8.0	2883	74.7%	0.53	0.43	395.51
DenseNet-161	224	28.7	7818	77.7%	1.06	0.50	991.61
DenseNet-169	224	14.1	3418	76.0%	0.69	0.65	490.24
DenseNet-201	224	20.0	4367	77.2%	0.89	1.10	642.98
ResNeSt-50	224	27.5	5390	81.1%	0.76	‡	615.77
ResNeSt-101	224	48.3	10200	82.3%	1.40	‡	1130.59
ZenNet-0.1ms	192	34.5	1799	77.0%	0.10	0.08	246.87
ZenNet-0.2ms	192	44.6	2993	80.1%	0.20	0.16	332.54
ZenNet-0.3ms	224	86.5	5127	81.5%	0.30	0.26	475.64
ZenNet-0.5ms	224	148.3	8194	82.1%	0.50	0.41	808.28
ZenNet-0.8ms	224	105.1	9440	83.0%	0.80	0.57	895.98
ZenNet-400M-SE	192	12.4	453	78.3%	0.29	0.29	109.73
ZenNet-600M-SE	256	12.6	685	80.0%	0.49	0.46	166.94
ZenNet-900M-SE	224	19.4	934	80.8%	0.55	0.55	215.68

Table 6: 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.

B Detail Structure of ZenNets

We list detail structure in Table 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17.

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 $k \times k$ full convolution layer in 'Btn' block is replaced by depth-wise convolution in 'MB' block. 'kernel' is the kernel size of $k \times k$ 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	16	2	24	1
Res	5	16	40	2	48	4
Res	5	40	240	2	64	4
Res	5	240	256	1	64	3
Res	5	256	336	2	248	3
Res	3	336	408	1	256	4
Conv	1	480	6576	1	-	1

Table 7: ZenNet-0.1ms

block	kernel	in	out	stride	bottleneck	# layers
Conv	3	3	8	2	-	1
Res	3	8	96	2	56	1
Res	5	96	192	2	80	2
Res	5	192	120	2	56	5
Btn	5	120	416	1	128	4
Btn	3	416	536	2	320	5
Btn	5	536	648	1	320	3
Conv	1	648	3768	1	-	1

Table 8: ZenNet-0.2ms

block	kernel	in	out	stride	bottleneck	# layers
Conv	3	3	8	2	-	1
Res	3	8	32	2	72	1
Res	3	32	80	2	112	4
Btn	5	80	344	2	32	3
Btn	5	344	184	1	120	3
Btn	5	184	576	2	320	3
Btn	5	576	384	1	320	3
Btn	5	384	824	1	304	3
Btn	5	824	656	1	312	5
Conv	1	656	1368	1	-	1

Table 9: ZenNet-0.3ms

block	kernel	in	out	stride	bottleneck	# layers
Conv	3	3	24	2	-	1
Res	3	24	48	2	120	1
Res	3	48	48	2	208	1
Btn	3	48	288	2	72	5
Btn	5	288	512	1	128	3
Btn	5	512	368	2	1	3
Btn	5	368	392	1	320	3
Btn	5	392	304	1	112	5
Btn	5	304	832	1	384	4
Btn	3	832	664	1	312	2
Btn	5	664	568	1	256	4
Btn	5	568	304	1	256	5
Res	5	304	304	1	320	4
Btn	3	304	784	1	384	5
Conv	1	784	2048	1	-	1

Table 10: ZenNet-0.5ms

block	kernel	in	out	stride	bottleneck	# layers
Conv	3	3	32	2	-	1
Res	3	32	128	2	88	1
Btn	3	128	128	2	112	3
Btn	3	128	88	1	112	4
Btn	3	88	416	2	128	5
Btn	5	426	184	1	48	3
Btn	5	184	184	1	48	3
Res	5	184	224	1	32	5
Btn	5	224	176	1	72	4
Btn	5	176	232	1	128	4
Btn	5	232	344	1	128	3
Btn	3	644	128	1	48	3
Btn	3	128	128	1	64	5
Res	3	128	112	1	32	3
Res	3	112	56	1	64	5
Btn	5	56	128	1	96	4
Btn	5	128	1232	2	256	5
Btn	5	1232	1280	1	320	3
Btn	5	1280	824	1	304	4
Btn	5	824	1232	1	240	3
Conv	1	1232	2128	1	-	1

Table 11: ZenNet-0.8ms

C Speed v.s. FLOPs

A majority of NAS-designed networks use depth-wise convolution as building blocks. The depth-wise convolution is suitable for mobile device where the model size and FLOPs are the major concerns. However, when we aim to optimize the inference speed on sever-side GPU, smaller FLOPs may not indicate faster inference speed. To show this, we plot the latency v.s. FLOPs in Figure 11. Similar phenomenon was observed in several previous works [Cai et al., 2020, Radosavovic et al., 2020]. The reason behind this phenomenon is that the GPU latency consists of two parts: the actual FLOPs computation and the feature map I/O. Since CUDA cores in GPU are very power, the feature map I/O dominates the total inference time. Therefore, on server-side GPU, a fast structure should use compact feature map and relax the FLOPs constraint.

block	kernel	in	out	stride	bottleneck	expansion	# layers
Conv	3	3	32	2	-	-	1
MB	7	32	48	2	40	1	1
MB	7	48	88	2	96	1	1
MB	7	88	144	2	176	2	3
MB	7	144	248	2	224	4	5
Conv	1	248	1204	1	-	-	1

Table 12: ZenNet-400M-SE

block	kernel	in	out	stride	bottleneck	expansion	# layers
Conv	3	3	16	2	-	-	1
MB	7	16	32	2	32	1	1
MB	7	32	72	2	96	1	1
MB	7	72	120	2	128	2	4
MB	7	120	200	2	176	4	3
MB	7	200	168	1	192	4	5
Conv	1	168	156	1	-	-	1

Table 13: 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 14: ZenNet-900M-SE

block	kernel	in	out	stride	bottleneck	# layers
Conv	3	3	64	1	-	1
Btn	5	64	168	1	16	3
Btn	3	168	80	2	32	4
Btn	5	80	112	2	16	3
Btn	5	112	144	1	24	3
Btn	3	144	32	2	40	1
Conv	1	32	512	1	-	1

Table 15: ZenNet-0.5M for CIFAR10/CIFAR100

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 16: 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 17: ZenNet-2.0M for CIFAR10/CIFAR100

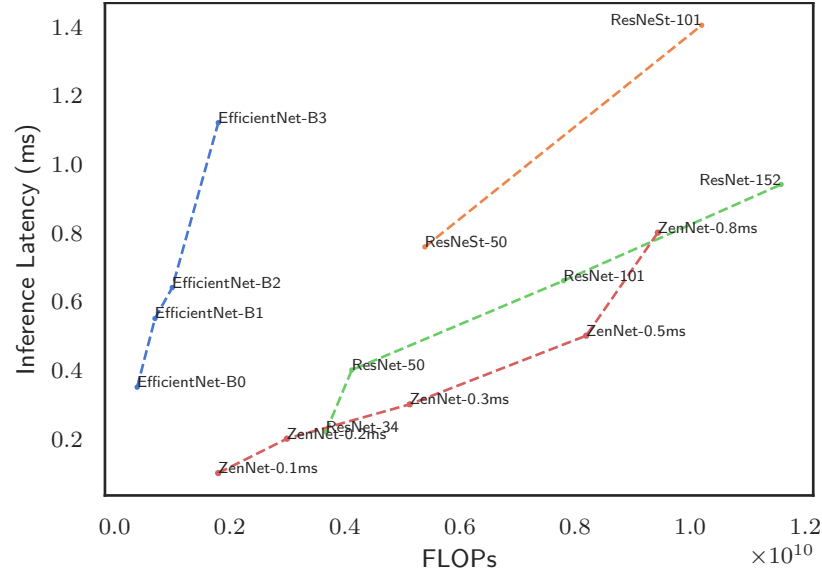


Figure 11: FLOPs v.s. Latency, V100 GPU, batch size 64, half precision (FP16) .