공학석사학위논문

해상도 기반의 CNN 네트워크의 점진적 스케일링 기법

Resolution based Incremental Scaling Methodology for CNNs

2023년 2월

서울대학교 대학원
인공지능협동과정
임 정 섭

해상도 기반의 CNN 네트워크의 점진적 스케일링 기법

Resolution based Incremental Scaling Methodology for CNNs

지도교수 하 순 회

이 논문을 공학석사 학위논문으로 제출함

2022년 12월

서울대학교 대학원

인공지능협동과정

임 정 섭

임정섭의 공학석사 학위논문을 인준함

2022년 12월

위 원 장 유 승 주 (인)

부위원장 하 순 회 (인)

위 원 이 영 기 (인)

학위논문 원문 서비스에 대한 동의서

본인의 학위논문에 대하여 서울대학교가 아래와 같이 학위논문 저작물을 제공하는 것에 동의합니다.

1. 동의사항

   ① 본인의 논문을 보존이나 인터넷 등을 통한 온라인 서비스 목적으로 복제할 경우 저작물의 내용을 변경하지 않는 범위 내에서의 복제를 허용합니다.

   ② 본인의 논문을 디지털화하여 인터넷 등 정보통신망을 통한 논문의 일부 또는 전부의 복제배포 및 전송 시 무료로 제공하는 것에 동의합니다.

2. 개인(저작자)의 의무

   본 논문의 저작권을 타인에게 양도하거나 또는 출판을 허락하는 등 동의 내용을 변경하고자 할 때는 소속대학(원)에 공개의 유보 또는 해지를 즉시 통보하겠습니다.

3. 서울대학교의 의무

   ① 서울대학교는 본 논문을 외부에 제공할 경우 저작권 보호장치(DRM)를 사용하여야 합니다.

   ② 서울대학교는 본 논문에 대한 공개의 유보나 해지 신청 시 즉시 처리해야 합니다.

논문 제목 : 해상도 기반의 CNN 네트워크의 점진적 스케일링 기법

학위구분 : **석사**
학　　과 : 인공지능협동과정
학　　번 : 2021-24200
연 락 처 : 010-4172-3778
저 작 자 : 임정섭　(인)

제 출 일 : 2022년 12월 06일

서울대학교총장 귀하

Abstract

Resolution based Incremental Scaling Methodology for CNNs

Jungsub Lim
Interdisciplinary Program in Artificial Intelligence
College of Engineering
The Graduate School
Seoul National University

Designing an optimal CNN for each embedded device with a different resource budget would be time-consuming and inefficient. Network scaling provides a viable solution to tackle this challenge, In this work, we propose a novel network scaling strategy called RBIS(resolution-based incremental scaling). Unlike the previous works that consider the width, depth, and input resolution together, we first find the input resolution candidates on a given hardware platform. For each resolution candidate, we incrementally scale the depth and width of each stage up to the available resource. Comparison with other scaling methods proves the superiority of the proposed scaling methodology. Codes are available at https://anonymous.4open.science/r/RBIS-Resolution-based-Incremental-Scaling-A661/

Keywords : Neural Architecture Search, Network Scaling, EfficientNet

Student Number : 2021-24200

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

As convolutional neural networks (CNNs) deliver state-of-the-art accuracy in many computer vision tasks such as image classification, object detection, and segmentation, there is a growing need to run CNN applications in embedded devices with limited computational resources. When performing CNN applications in embedded devices, we want to maximize the accuracy under the resource budget of the device. Designing an optimal CNN for each device with a different resource budget would be time-consuming and inefficient. Network scaling has been proven to be an effective way to tackle this challenge. After we optimally design a baseline CNN for a device with a tight resource constraint, we can scale the network for a more powerful device in three dimensions: width, depth, and input resolution.

Many researchers have dealt with the problem of effectively scaling networks. An early example is a series of ResNet[4] from ResNet-18 to ResNet-152. Meanwhile, WideResNet[5] increased the width of their models. Another approach is to increase the depth of the model([4, 6]) or the resolution of the model[7]. EfficientNet[1] initiated a study on how to scale up the network systematically and efficiently. Fig. 1.1 shows the empirical results presented in their work. Since the accuracy quickly saturates as only one dimension is scaled, as shown in this figure, they proposed a novel scaling method called *compound scaling*, in which they scale up the network width, depth, and resolution in a balanced

way. The compound scaling method finds a scaling factor of each dimension (α for depth,

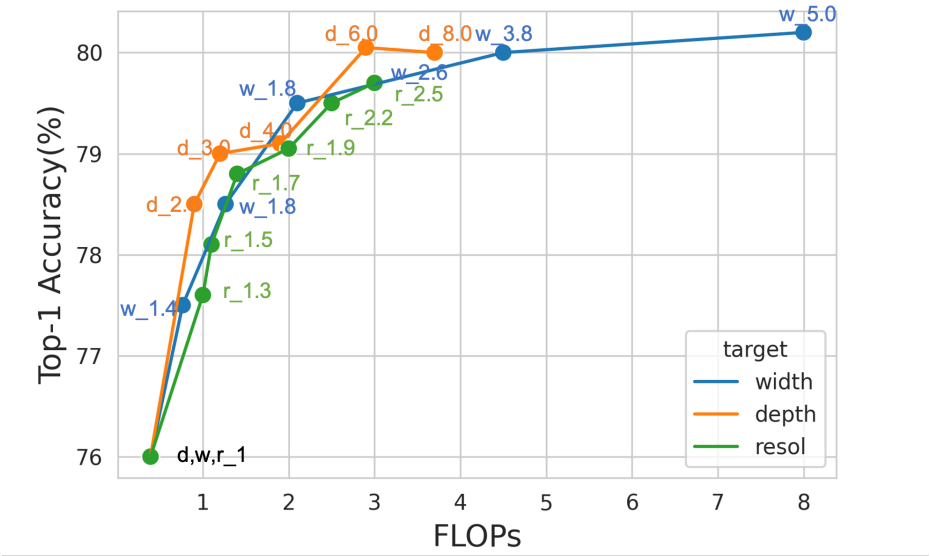

Figure 1.1: Performance saturates as d, w, or r increases alone. This figure is a merger of three separate graphs in Fig. 3 of EfficientNet[1]

.

β for width, and γ for resolution) and applies a common compound coefficient ϕ to scale them uniformly. The scaled computation complexity becomes $(\alpha \times \beta^2 \times \gamma^2)^\phi$ in terms of FLOPS. They determine the actual scaling factors by a grid search method and apply them to all stages of the network. There are subsequent studies to find a better combination of scaling factors than compound scaling. Even though compound scaling provides a simple and effective way to find how to scale the network, it has several drawbacks that make it sub-optimal. First, they use a hardware-agnostic metric, FLOPs, to balance the scaling factors of three dimensions. It is well understood that having low FLOPs does not necessarily guarantee that the latency of the model is also low. For example, NasNet-A [8] has a similar FLOPs count as MobileNetV1[9], but its complicated and fragmented cell-level structure is not hardware friendly, so the actual latency is slower. For real-life applications on an embedded device, we desire models that show a good trade-off

between accuracy and latency (not FLOPs). Second, determining the scaling factors of three dimensions is somewhat vague or unclear. Compound scaling uses an exhaustive grid search method over possible combinations. Third, in compound scaling, the scaling factors are uniformly applied to all stages. Recognizing this drawback, the same authors proposed a modified version in EfficientNetV2[10], where they stack more layers to later stages. However, they do not precisely state the mechanism of how each stage is differently scaled. In this work, we propose a novel scaling methodology to overcome those drawbacks, based on the following observations from Fig. 1.1 on which the compound scaling is based.

- Since the incremental gain of accuracy over a unit delay varies differently, it might be better to balance the slope, called the cost-performance ratio, among three scaling dimensions.

- Resolution scaling is more slowly saturated than the others. We speculate the reason as follows: While width and depth scaling increases the model parameters for a given input model, resolution scaling increases the incoming data information.

- The graph is drawn after uniform scaling is performed. Width and depth scaling may have a different effect on each stage of the network. While input resolution is applied to all stages, it might be better to scale each stage's width and depth differently.

The proposed scaling methodology is called *RBIS*(Resolution-Based Incremental Scaling). As the name implies, we treat resolution scaling separately from width and depth scaling. First, we prepare a set of resolution candidates. Next, we perform incremental scaling separately for each resolution to increase the accuracy within a given latency constraint. Finally, we choose the best scaled network by comparing the results. The heart of the proposed methodology is incremental scaling. Unlike the previous scaling methods that apply the same scaling factor to all stages, our incremental scaling

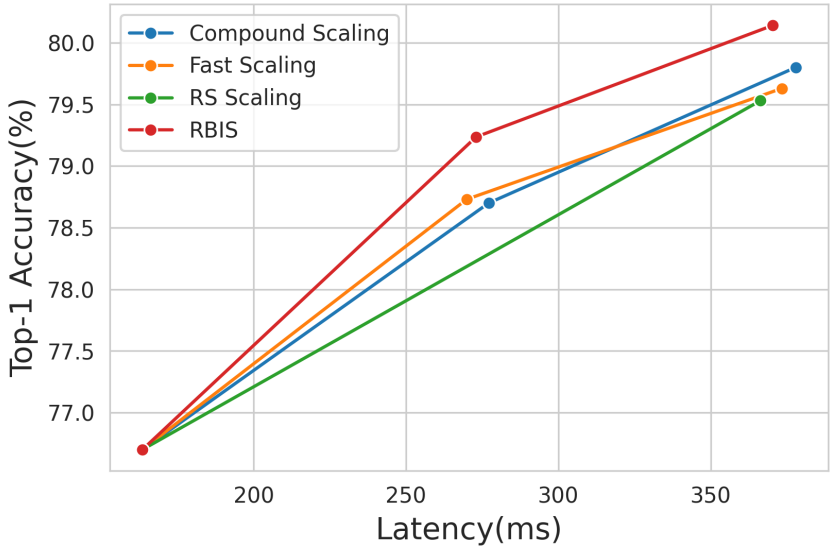

Figure 1.2: Accuracy comparison of the networks scaled by the proposed and compound scaling methods.

method finds the scale factor of each stage differently. We must determine how much width/depth is increased at a time for incremental scaling. Since we consider each stage separately, there is a large number of possible ways of scaling a network. We devise a strategy to make a trade-off between the computation complexity and the performance.

Fig. 1.2 shows the comparison between the proposed and other scaling methods for EfficientNets, including the original compound scaling method of EfficientNets. Fast scaling [2] and ResNet-RS scaling [3] (shortly RS scaling) are two other methods that are compared with the proposed method in experiments. As shown in this figure, RBIS achieves the highest top-1 accuracy on ImageNet among all methods with comparable latency. Comparison with the other scaling methods proves the effectiveness of the proposed scaling methodology.

Chapter 2

Related Work

A number of works propose to scale models by depth, width, or input image resolution. While ResNets use depth scaling, WideResnet[5] and MobileNets scale the network by width scaling. It is also well-known that a bigger input image size will help accuracy. Unlike previous studies that focused on only one of depth, width, and resolution, EfficientNet proposed a method to effectively increase the performance of a CNN network by simultaneously increasing the network's depth, width, and resolution, as mentioned above. Compound scaling is a general scaling method that could be effectively applied to ResNets and MobileNets. But they used their own NAS framework called MNasNet[11] to develop the baseline model and applied compound scheduling to obtain EfficientNets that showed state-of-the-art performance at that time.

There are subsequent studies to find a better combination of scaling factors than compound scaling. Fast scaling[2] is a method that puts more weight on width, observing that the activation map size matters more than FLOPs in terms of latency. ResNet-RS[3] de-emphasizes resolution scaling by slowing down resolution scaling. Their scaling heuristic is to scale the depth in regimes where overfitting occurs by width scaling. These methods apply a unified principle to scale the whole model, ignoring the stage-wise differences.

On the other hand, recent studies show that applying the same rules to all stages is

not optimal for performance improvement. Since the size of the activation map is reduced by half each time it goes through a stage and each stage has a different number of parameters, stages are likely to have different characteristics in terms of cost-performance ratio. EfficientNetV2[10] modifies the EfficientNet architecture admitting that uniform scaling strategy is sub-optimal. Their modified scaling strategy is gradually adding more layers to later stages and restricting the maximum input size to a smaller value. RegNet[12] also gave many observations of good architecture by experimenting with thousands of architectures to find a relationship between widths and depths of good networks. However, RegNet and EfficientNetV2 do not present a generalized scaling rule like compound scaling. Many NAS(Neural Architecture Search) methodologies have been published, along with scaling performance as well as how to find the base net. S3NAS[13] proposes a method to find a sub-optimal neural architecture for various hardware by modifying the single-path NAS[14] method. After finding the baseline network through S3NAS, they apply the compound scaling method to scale up the network. This network is selected as a benchmark to show that the proposed scaling method works better than compound scaling on an arbitrary CNN network.

The most closely related work is the work in greedy-network enlarging[15]. Similarly to the proposed method, they proposed a kind of incremental scaling method, applying a different scaling ratio for each stage. However, they do not differentiate resolution scaling from width/depth scaling. They use a proxy method for performance estimation to accelerate the scaling process, using a subset of the ImageNet dataset for fast training. Their claimed performance is not reproducible as their code is not open to the public.

Chapter 3

Network Scaling Problem

The modern CNN backbone architectures usually consist of a stem layer, network body, and head. The main burden of FLOPs and parameter size lies in the network body, as typically, the stem layer is a convolutional layer and the head is a fully-connected layer. Thus, this paper focuses on the scaling of the network body. The network body consists of several stages, defined as a sequence of layers or blocks with the same spatial size. For example, EfficientNet-B0 body is composed of 7 stages.

Scaling up convolutional neural networks is widely used to achieve better accuracy. If we increase the depth of a stage, we could capture richer and more complex features, which is likely to improve the accuracy in contrast to a shallow network. If we increase the width of a stage, more fine-grained features can be learned with more kernels. While having more kernels is beneficial for accuracy improvement, it is a common practice to reduce the number of kernels by network pruning to reduce the resource requirement in embedded systems. Since higher input resolution provides more input information, it is surely helpful to increase the accuracy. To match the high resolution, however, a deeper and wider network is needed to acquire large receptive fields to capture richer fine-grained features. It means that the three dimensions of network scaling are not independent. Suppose that the network body of a baseline network, N, consists of L stages. We let w_i and d_i be the width and the depth of the i-th stage, respectively where $i \in \{1, 2, \ldots, L\}$. Unlike

the compound scaling method that applies uniform scaling, we treat each stage separately. Thus the network scaling problem addressed in this work is defined as follows;

Input: Baseline network, the target embedded device, and the latency constraint, T.

Output: All values of $\{w_i\}$ and $\{d_i\}$ as well as input resolution, r, scaling the baseline network.

Objective: Maximize the accuracy under a given latency constraint.

Since there are $2 \times L + 1$ variables to determine, the number of combinations to meet the latency constraint is too large to explore exhaustively. We propose a novel systematic solution to this problem, which is explained below.

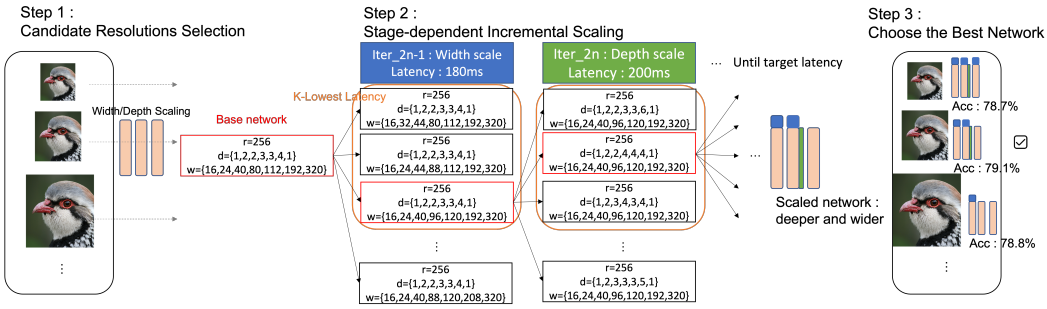

Figure 3.1: Overview of the proposed RBIS scaling algorithm that consists of three steps: Candidate resolution selection, stage-dependent incremental scaling, and best network selection.

.

Chapter 4

Proposed Scaling Methodology: RBIS

Figure. 3.1 shows the overall flow of the proposed methodology which consists of three steps. In the first step, we select a set of resolutions to explore. In the proposed method, the resolution scaling exploration differs from the width and depth scaling. It is different from other approaches that mostly de-emphasize the resolution scaling.

In the second step, we find all values $\{w_i\}$ and $\{d_i\}$ by incremental scaling for each input resolution. From the baseline model, we incrementally increase each stage's width and depth in an iterative fashion. There are several parameters to choose from in this step: Which stages to consider first, which one to increase between width and depth, and how much to increase. After the best combination of $\{w_i\}$ and $\{d_i\}$ is found for each resolution from the second step, we choose the best scaled network that gives the highest accuracy by comparing the results from the second step.

4.1 Step 1: Candidate Resolutions Selection

In a preliminary experiment, a relation between the latency and input resolution is obtained by running the baseline model on various hardware platforms. We run EfficientNet-B0 on Pixel2 CPU, RTX3090 GPU, NVIDIA Jetson AGX Xavier, varying batch sizes (BS) reflecting the computing power of the hardware platform. Fig. 4.1 shows the measured results after gradually increasing the resolution by 4. TRT and FP16 denote Ten-

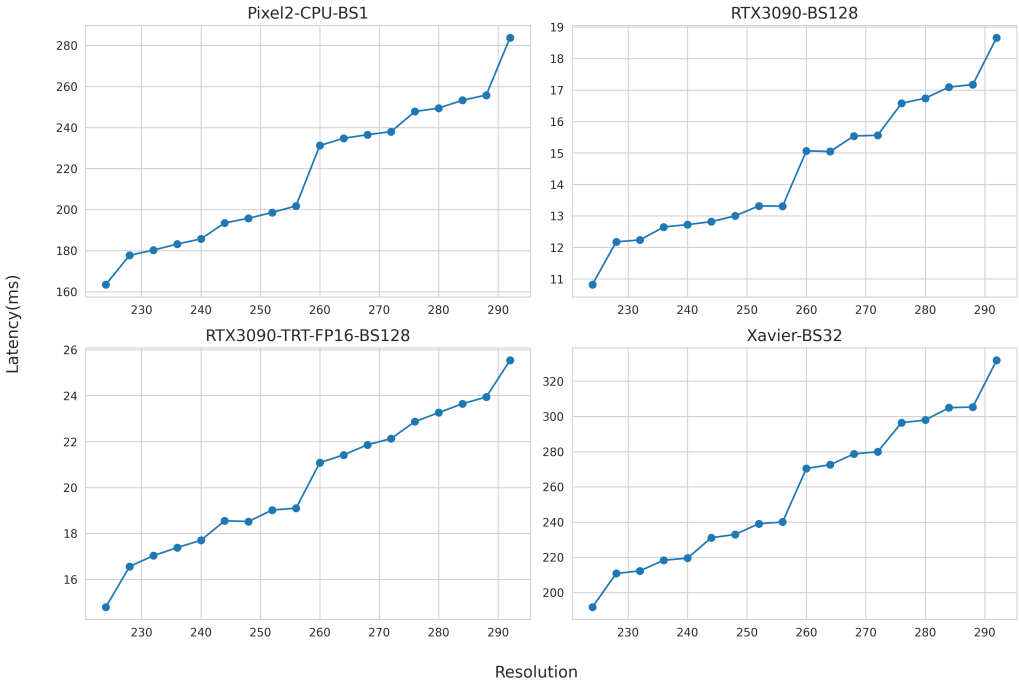

Figure 4.1: Non-linear relationship between input resolution and the measured latency on various hardware platforms: Pixel2 CPU, RTX3090 GPU, and AGX Xavier. BS means the batch size.

sorRT conversion and half-precision usage for inference optimization. As shown in the figure, in all hardware platforms, it is observed that the relationship is not linear but piece-wise linear, meaning that there are some jumps of rapid latency increment. When selecting the candidate input resolutions, we skip those resolutions after the jump, assuming that these resolutions are not optimal. It is empirically found that the best resolution is one of the resolutions just before the jump. A possible interpretation is that we could deliver the richest possible features to the model while minimizing the latency increment. Therefore, in this work, we include those resolutions and their neighbors in the candidate resolution pool.

4.2 Step 2: Stage-dependent Incremental Scaling

As mentioned in the introduction, incremental scaling is motivated by our observations from Fig. 1.1. We aim to scale widths and depths in the decreasing order of cost-performance ratio, or *score* that is defined as follows:

$$Score = \frac{\max\{Acc - ParentAcc, 0\}}{Latency - ParentLatency} \tag{4.1}$$

ParentAcc and *ParentLatency* refer to the accuracy and latency of the network before scaling, respectively. *Acc* and *Latency* refer to the accuracy and latency after incremental scaling. Max operator is required in case scaled network has a lower accuracy than the base network.

For width and depth scaling, we need to determine which stages to scale and how much. Since scaling one dimension only is saturated fast, we scale the width and depth alternately. Width scaling is performed at the first and odd-numbered iterations, while depth scaling is done in even-numbered iterations. For width scaling, we increase the width of a chosen stage by 10 percent and set it to the nearest multiple of 4 by rounding. Such increment is determined empirically; more iterations will be necessary if we reduce the increment. If we increase more, the search space of width scaling will be reduced. We increase the depth of chosen stages by one block for depth scaling.

At each iteration, we decided to scale two stages at a time. By scaling two at a time, target latency can be reached with fewer iterations than scaling one stage only. Since there are L stages, the number of combinations for width and depth scaling becomes ${}_LH_2$ each. With 10 TPUs available to us, we could evaluate 10 combinations concurrently and choose the best one at each iteration. As we scale 5 stages from EfficientNet, we need to sample 10 combinations out of ${}_5H_2 = 15$ combinations. We choose 10 combinations in the increasing order of latency, covering 10 out of 15 of all possible combinations. From preliminary experiments, it is observed that combinations with higher latency are

less likely to be selected since their accuracy improvement does not outweigh the latency increase. If we have more TPUs available, we may scale more stages at each iteration. In the last iteration, we may decrease the number of stages to one in order to meet the latency constraints.

We find the scaling decision with the highest score at each iteration and reset the base network according to equation (4.1). It makes the decision of each iteration depend only on the current state. This iterative process is repeated until we reach the target latency. The chosen network of the last iteration stage becomes the network with the best accuracy with the given resolution input.

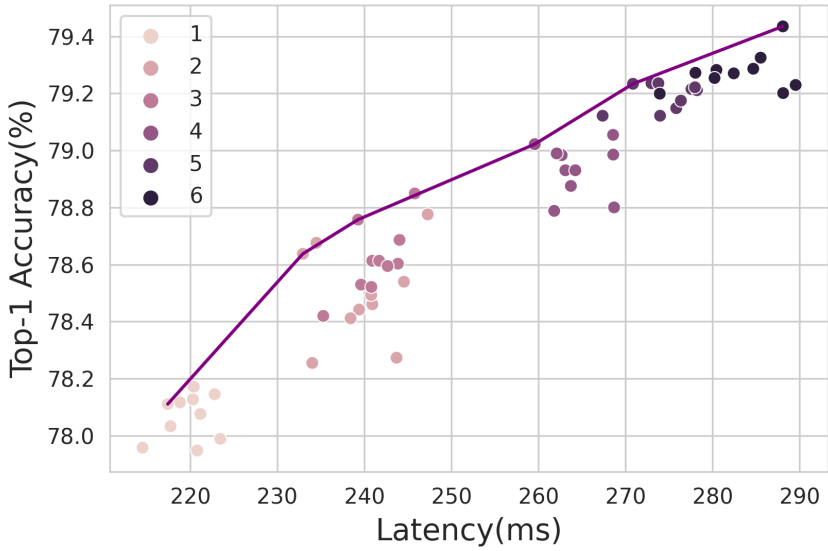

Figure 4.2: Illustration of the proposed incremental scaling process from EfficientNet-B0 until the latency of EfficientNet-B1 is reached

The process of scaling EfficientNet-B0 to EfficientNet-B1 is demonstrated in Fig. 4.2, assuming that the input resolution is 256. Note that it took us 6 iterations to reach EfficientNet-B1 latency and achieved 0.5% higher accuracy gain than EfficientNet-B1 without any training strategy added. The last iteration is not included in the final result because including it will exceed the latency constraint.

There are other possible ways to perform incremental scaling. Instead of alternating the width and depth scaling, one method is to compare both directions at each iteration, which is the first attempt we tried. In this method, we choose one stage for width scaling and one stage for depth scaling separately to have $2 \times L$ scaling candidates and choose the best one at each iteration. It turns out that a 10% increment of width results in a noticeably shorter latency increment than adding one more block in a stage. On the other hand, the accuracy difference between the width and depth scaling is not as large as the latency difference. In other words, this method favors width scaling more than depth scaling. It also explains why we separate resolution scaling from width and depth scaling. Since incremental resolution scaling tends to give a lower score than width or depth scaling, resolution scaling would only be taken after the depth-width scaling is saturated, which explains why the previous work usually de-emphasizes resolution scaling.

4.3 Step3: Choose the Best Network

After we find the scaled network with the highest accuracy for each resolution candidate, we compare the found networks in terms of the accuracy to choose the best network. Table 4.1 shows the accuracy obtained for each resolution candidate when we scale EfficientNet-B0 with the latency constraint set to the latency of EfficientNet-B1. As shown in the table, the accuracy is highest when the input resolution is increased from 224 to 256. The accuracy obtained from the proposed methodology is 79.23, which is higher than that of EfficientNet-B1 (78.7).

Table 4.1: Best accuracy for each input resolution.

Resolution	240	256	272	288
Accuracy(%)	79.16	79.23	78.9	78.6

The proposed RBIS is summarized in Algorithm 1 and Algorithm 2. In Algorithm 1, we perform incremental scaling iteratively, alternating width scaling and depth scal-

Algorithm 1 RBIS

Input: Base Architecture $\mathbf{N_0}$
Output: Scaled Architecture
R = *Select candidate resolutions* as in Fig.4.1
$NET = [\,]$, $ACC = [\,]$;
for $resol \in \boldsymbol{R}$ **do**
    $t, a = compute\ latency\ and\ accuracy\ of\ \mathbf{N_0}\ at\ resol$
    $i = 0$;
    while $t <$ **T do**
        if $i\%2 == 0$ **then**
            $N_{i+1}, a_{i+1}, t = width\ scaling(N_i, a_i, t)$ as in Algorithm 2
        else
            $N_{i+1}, a_{i+1}, t = depth\ scaling(N_i, a_i, t)$ as in Algorithm 2
        end
        i++;
    end
    append N_i to NET;
    append a_i to ACC;
end
$k = \arg\max ACC$
return $NET[k]$

ing for each resolution candidate. We save the best network model and the associated latency for each resolution candidate. This iteration process continues until the measured latency becomes larger than the given latency constraint. After all resolution candidates are covered, we choose the best network.

Algorithm 2 Width/Depth incremental scaling

Input: Base Architecture **N**, Parent accuracy/latency *a*/*t*
Output: Best scaled architecture N in current iteration
P=[K-lowest latency networks among ${}_LH_2$ scaled networks];
for $net \in \boldsymbol{P}$ **do**
    $score = \frac{Acc_{\mathbf{net}} - a}{L_{\mathbf{net}} - t}$
    if $score > curmax$ **then**
        $curmax = score$
        $N = net$
    end
end
return N

4.4 Performance Estimation

To guide the search process, we have to estimate the accuracy of a given architecture. Many proxy methods are suggested including shorter training times [12], training on a subset of the data [16], on proxy data [8] or using network transformations [17]. Unfortunately, we could not get satisfactory results from those proxy methods. Even though those low-fidelity approximations reduce the training time, they introduce bias in the accuracy estimation which leads to poor rank preservation.

As a result, we decided to use a more accurate method: train each candidate with the entire training dataset and evaluate the accuracy with the validation data. The execution time of the proposed technique depends on the following hyper-parameters: (1) the number of resolutions to be searched, (2) the number of steps for incremental scaling, (3) how many candidates to examine for each step, and (4) how many epochs to train for each step. Surely there is a trade-off between the performance and the execution time. Even though we could obtain similar results by reducing the number of training epochs for each step, we deliberately perform full training in the current implementation to verify that the proposed incremental scaling method is valid paying the cost of long training time. By selecting two stages per iteration and deciding a proper amount of scaling, the number of iterations could be reduced to 6, as illustrated in Fig. 4.2. It took us about a week to scale EfficietNet-B0 to EfficientNet-B1. We believe such a long time is tolerable since the network scaling is performed offline only once for each target hardware platform. Nonetheless, it is left as a future research topic to find a faster method of ranking the candidates of incremental scaling at each iteration.

Chapter 5

Experiments

In this section, we show how our scaling method works and evaluate the effectiveness of our method by comparing it with other methods. The hardware platform used to measure latency is Pixel2, and latency measurement uses the benchmark binary for *TFlite* provided by Tensorflow. We didn't use multi-cpu nor gpu mode for the sake of simplicity. We use ImageNet dataset to test our results. As for our computational budget, we use 11 v2-8 TPU pods to do our experiments.

First, we introduce the baseline networks we use for the experiments and the hyperparameters used with those networks. Next, we show how to apply RBIS to scale networks. Lastly, we compare the scaled networks with networks that are scaled using other scaling methods. In all experiments, we scale our networks up to the latency of EfficientNet-B2, as our methodology targets embedded and mobile devices.

5.1 Benchmark Networks

EfficientNet is a de-facto network model to evaluate any scaling method for two reasons. First, it applies to embedded devices and shows reasonably good performance with relatively small latency. Second, since many scaling methods are applied to EfficientNets, it is easy for us to compare our method with others. As for hyper-parameters, we use the same setting as the original EfficientNet code in our experiments. As scaling proceeds,

we change the number of filters and the number of blocks per stage.

While EfficientNet is the most widely used network, we also want to show that our method works well on another network. We select S3NAS[13] as another benchmark since its scaling result is available. It is a family of convolutional neural networks generated from a Neural Architecture Search method. S3NAS also uses MBconv blocks[18] like EfficientNet, but the kernel size or expansion ratio of blocks may vary within a stage, and the accumulated number of blocks linearly increases like regnet[12]. It uses more parameters and higher FLOPs than other networks with similar latency but achieves better accuracy. After a small baseline network is searched, it also uses compound scaling to scale up the base network. We scale the found baseline network with several scaling methods and compare the results. Like the case of EfficientNet, we use hyper-parameters from the official code repository. Other eminent networks such as ResNets were not used in our scaling experiments as their latency is much higher than these two lightweight networks that are developed for embedded devices.

Table 5.1: Scaled Architecture of EfficientNet-B0 using RBIS

	Channels			**Blocks**		
Stage	Eff-B0	Eff-B1	RBIS	Eff-B0	Eff-B1	RBIS
Stem	32	32	32	1	1	1
0	16	16	16	1	2	1
1	24	24	24	2	3	2
2	40	40	40	2	3	2
3	80	80	104	3	4	5
4	112	112	112	3	4	3
5	192	192	192	4	5	7
6	320	320	384	1	2	1
Head	1280	1280	1408	1	1	1

Table 5.2: Comparison of scaled network models obtained by the proposed and other methods (*dwr* [1], *dWr* [2], and *rs* [3])

Base Network	Accuracy	Latency(ms)
EfficientNet-B0	76.7	164.978
EfficientNet-B1-dwr	78.7	277.396
EfficientNet-B1-dWr	78.73	269.912
EfficientNet-B1-RBIS	**79.23**	272.98
EfficientNet-B2-dwr	79.8	378.36
EfficientNet-B2-dWr	79.63	373.352
EfficientNet-B2-rs	79.53	366.262
EfficientNet-B2-RBIS	**80.1**	370.341
S3NAS-B0	77.4	158.2
S3NAS-B1-dwr	78.95	274.8
S3NAS-B1-dWr	78.4	262.9
S3NAS-B1-RBIS	**79.34**	275.7
S3NAS-B2-dwr	79.55	377.9
S3NAS-B2-dWr	79.76	378.4
S3NAS-B2-RBIS	**80.22**	371.4

5.2 Scaling EfficientNet

In the first step of the proposed methodology, we select four different resolutions positioned just before the steep increase of latency as shown in Fig. 4.1: 240, 256, 272. 288. For each resolution, we perform incremental scaling until the target latency is reached. An example of iterative incremental scaling process is displayed in Fig. 4.2. Table 5.1 shows the scaled network produced by RBIS. Note that unlike EfficientNet, the network obtained by RBIS scales each stage differently. And it is noted that identical stages are chosen multiple times. This is consistent with the observation from EfficientNetV2 that it is better to stack more layers in the later stages of the network.

Table 5.2 shows the latency and accuracy of networks obtained when scaled by compound scaling[dwr] and our method[RBIS]. The best performance could be obtained when the input resolution is 256. This is larger than the resolution used in the origi-

nal EfficientNet-B1 architecture which is 240. When scaled further up to the latency of EfficientNet-B2, we could achieve 0.3 % higher accuracy than EfficientNet-B2 when the resolution is 272.

In addition, the proposed scaling method is compared with two other methods, fast scaling method [2], and ResNet-RS method [3]. It can be seen from Table 5.2 that our scaling method performs the best. The fast scaling method denoted as dWr in Table 3 shows 78.73 % and 79.63 % accuracy when it is scaled to comparable latency with EfficientNet-B1 and EfficinetNetB2, respectively. The accuracy results may be different from the paper as we use the hyper-parameters from the original EfficientNet code instead of their settings for fair comparison among all methods. EfficientNet-RS represents the network obtained by the ResNet-RS method. When scaled to latency comparable to EfficientNet-B2, it shows lower accuracy with slightly smaller latency compared to EfficientNet-B2.

5.3 Scaling S3NAS

As shown in Table 5.2, our scaling method shows a better latency-accuracy trade-off than compound scaling and fast scaling for S3NAS. In this experiment, we borrow hyper-parameters for each scaling method from the corresponding paper, which may not be optimal for the S3NAS base network. We admit that skipping hyperparameter search may hinder fair comparison. However, it highlights the merit of our method that the proposed scaling method can be applied without hyper-parameter search. We use the following scaling factors in terms of (d, w, r) notation: (1.16, 1.08, 1.13)[dwr], (1.06, 1.27, 1.03)[dWr] for EfficientNet-B1 and (1.25, 1.12, 1.18)[dwr], (1.09, 1.4, 1.04)[dWr] for EfficientNet-B2. While compound scaling and fast scaling achieve 78.95%, and 78.4% in accuracy, respectively, RBIS achieves 79.34% within 276ms target latency. With the target latency of 377ms, compound scaling and fast-accurate scaling achieve 79.55% and 79.76%, respectively, while RBIS achieves 80.22%.

There are three interesting observations we can make from the experimental results. First, the scaled networks from S3NAS show higher accuracy than those from EfficientNet. It implies that the baseline model affects the performance more than the scaling method. Second, the performance gap between the proposed method and compound scaling is more significant from B1 to B2 scaling than from B0 to B1 scaling, which is the opposite of EfficientNet. It implies that compound scaling may not be suitable for S3NAS even though it works well for EfficientNet. On the other hand, the proposed scaling method works well for S3NAS, which confirms the generality of the proposed scaling method in contrast to the other methods specific to a certain network type.

Last but not least, the best-scaled model from S3NAS requires a resolution of 288, which is much bigger than EfficientNet-B1 and our scaled model from EfficientNet. The recommended input resolution from compound scaling is 256. When the resolution is 288, depth scaling is not performed, as it causes the latency of the network to exceed the target latency. As for width, the 3rd stage out of 5 increased by 10 percent. It clearly proves that no preference for one dimension should be made, which contradicts the claim of previous works that prefer width or depth scaling to input resolution. And it confirms that it is better to treat resolution differently from the width and depth scaling. A possible reason for this unexpected result for S3NAS is that S3NAS has more parameters than EfficientNet-B0 since it is wider and deeper. Hence, increasing the input resolution is more effective than having more parameters.

When we scale again towards EfficientNet-B2 latency($377ms$), candidate resolutions become $\{288, 320\}$. For two candidate resolutions, 5 and 2 iterations were needed to reach the target latency, respectively. The final scaling result obtained by RBIS is denoted by width[3344] and depth[4555] at resolution 288: it means that the width scaling is applied to the 3rd and the 4-th layers twice while the depth scaling is applied to the 4-th layer once and to the 5-th layer three times.

Chapter 6

Conclusion

In this paper, we propose RBIS, a novel neural network scaling method that helps find networks with a better latency-accuracy trade-off than the popular compound scaling. Unlike the previous works, resolution scaling is treated separately from the width and depth scaling. Since there is a non-linear relation between the latency and the input resolution, we select some candidate resolutions in the first step. For each resolution, we apply incremental width and depth scaling alternately until the latency constraint is met. There are several hyper-parameters to determine in this step: how many stages to consider, how much to increase, and how many combinations to evaluate. Even though we decided on those hyper-parameters empirically, there is room for improvement, which is left as a future research topic. Experiments with two benchmark networks, EfficientNet and S3NAS, show that the proposed scaling method finds a more accurate model by up to 0.53 percent for EfficientNet-B1 and 0.67 percent for the S3NAS B2 scale. In the current implementation, we use a full train for the performance estimation of each scaling candidate. Reducing the proposed methodology's computation cost is another future research topic.

Chapter 7

Ablation Study

Although RBIS produces good performance through a novel scale method, it takes a lot of time because RBIS requires full train for network evaluation. Here are two methods we tried to save time.

7.1 Network transformation

The first method is Network Transformation. Network transformation is a method first introduced in Net2Net[19] which slightly transforms a part of the network but still maintains the same function. That is, for the same input, both networks before and after transformation produce the same output. When another width of channels or blocks are added, Net2Net adjusts the weight of old parts so that the output value doesn't change. ENAS[20] and EAS[21] use network transformation to execute the architecture search in a short time. In the case of RBIS, the network structure after one step of scaling is quite similar to the base network. From this, we thought Net2Net fits well with RBIS. But there were some issues. First, the ranking evaluated by net2net is different from the ranking evaluated by full train , and the accumulated difference leads to the sub-optimal result. Table 7.1 shows the accuracy rank difference between full-train and net-transformation methods. Second, it performed worse than a scratch train. We want to know how much potential this network has in full train. If you use the net2net method, you can get more

than 75% performance within 10 epochs, but the performance does not rise further. On the other hand, scratch train achieves less than 70% performance at 10 epochs, but close to 80% performance at around 250 epochs. OpenAI has also reported that this weight reuse attempt yields sub-optimal results to scratch train. [22] As a result, it is difficult to evaluate the network using the net2net method as a proxy for the evaluation method we want.

Table 7.1: Ranking Difference between Full-train(FT) and Network-transformation(NT)

	Network	FT	NT
Width	w1	10	7
	w2	1	2
	w3	2	1
	w4	3	3
	w5	7	5
Depth	d1	6	6
	d2	9	8
	d3	5	10
	d4	8	11
	d5	4	4
Resol	r	11	9

7.2 Zero-shot proxy

ZenNAS[23] predicts the performance of given network structures without training. With only one forward call, it measures how much the model differentiates the input from the same input with small noises. The authors of [24] introduce $\phi - score$ to calculate the complexity of vanilla convolution neural network(VCNN). ZenNAS[23] points out $\phi - score$ diverges as the network deepens, and they solve this issue by introducing Batch-Normalization. They name their zero-shot proxy as Zen-Score, and they claim Zen-Score preserves ranking well, recording τ=0.88 on CIFAR-100 data. We tried to integrate Zen-Score in our RBIS framework, replacing the accuracy with Zen-score in incremental

scaling for depth and width increment. With vanilla Zen-Score, the performance of the network obtained was 78.7% in EfficientNet-B1 target, which is not improved from the original EfficientNet-B1. We observed Zen-Score favors depth direction, which shows no saturation. To improve this, we normalized the z-score by dividing Zen-Score by the number of conv blocks used in the network. When using the tuned Zen-score obtained in this way, we obtained 79.34% targeting EfficientNet-B1, resulting in performance as good as RBIS. However, when targeting EfficientNet-B2, we achieved only 79.9%, which fell short of RBIS by 0.2%p.

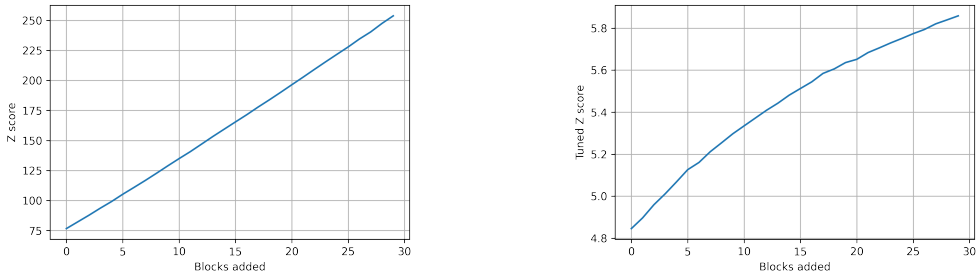

Figure 7.1: Illustration of the original Zen-Score and our tuned Zen-Score when adding conv blocks to the network

Bibliography

[1] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[2] Piotr Dollár, Mannat Singh, and Ross Girshick. Fast and accurate model scaling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 924–932, 2021.

[3] Irwan Bello, William Fedus, Xianzhi Du, Ekin Dogus Cubuk, Aravind Srinivas, Tsung-Yi Lin, Jonathon Shlens, and Barret Zoph. Revisiting resnets: Improved training and scaling strategies. *Advances in Neural Information Processing Systems*, 34:22614–22627, 2021.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[5] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[6] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[7] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

[8] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

[9] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, 2017.

[10] Mingxing Tan and Quoc Le. Efficientnetv2: Smaller models and faster training. In *International Conference on Machine Learning*, pages 10096–10106. PMLR, 2021.

[11] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019.

[12] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollar. Designing Network Design Spaces. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10425–10433, Seattle, WA, USA, 2020. IEEE.

[13] Jaeseong Lee, Jungsub Rhim, Duseok Kang, and Soonhoi Ha. S3NAS: Fast Hardware-aware Neural Architecture Search Methodology. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1, 2021.

[14] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-path nas: Designing hardware-efficient convnets in less than 4 hours. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 481–497. Springer, 2019.

[15] Chuanjian Liu, Kai Han, An Xiao, Yiping Deng, Wei Zhang, Chunjing Xu, and Yunhe Wang. Greedy Network Enlarging. *arXiv:2108.00177 [cs]*, 2021.

[16] Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets. In *Artificial intelligence and statistics*, pages 528–536. PMLR, 2017.

[17] Junran Peng, Ming Sun, ZHAO-XIANG ZHANG, Tieniu Tan, and Junjie Yan. Efficient neural architecture transformation search in channel-level for object detection. *Advances in Neural Information Processing Systems*, 32, 2019.

[18] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[19] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2Net: Accelerating Learning via Knowledge Transfer. 2016.

[20] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pages 4095–4104. PMLR, 2018.

[21] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[22] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[23] Ming Lin, Pichao Wang, Zhenhong Sun, Hesen Chen, Xiuyu Sun, Qi Qian, Hao Li, and Rong Jin. Zen-nas: A zero-shot nas for high-performance image recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 347–356, 2021.

[24] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. *Advances in neural information processing systems*, 27, 2014.