Rethinking Channel Dimensions for Efficient Model Design

Dongyoon Han   Sangdoo Yun   Byeongho Heo   YoungJoon Yoo

NAVER AI Lab

Abstract

Designing an efficient model within the limited computational cost is challenging. We argue the accuracy of a lightweight model has been further limited by the design convention: a stage-wise configuration of the channel dimensions, which looks like a piecewise linear function of the network stage. In this paper, we study an effective channel dimension configuration towards better performance than the convention. To this end, we empirically study how to design a single layer properly by analyzing the rank of the output feature. We then investigate the channel configuration of a model by searching network architectures concerning the channel configuration under the computational cost restriction. Based on the investigation, we propose a simple yet effective channel configuration that can be parameterized by the layer index. As a result, our proposed model following the channel parameterization achieves remarkable performance on ImageNet classification and transfer learning tasks including COCO object detection, COCO instance segmentation, and fine-grained classifications. Code and ImageNet pretrained models are available at https://github.com/clovaai/rexnet.

1. Introduction

Designing a lightweight network architecture is crucial for both researchers and practitioners. Popular networks [47, 15, 20, 46] designed for ImageNet classification share a similar design convention where a low dimensional input channel is expanded by a few channel expansion layers towards surpassing the number of classes. Lightweight models [20, 46, 19, 59, 35, 49, 54, 3, 50] also follow this configuration but further shrinks some channels for computational efficiency, which leads to the promising trade-offs between the computational cost and accuracy. In other words, the degree of channel expansion at layers is quite different, where earlier layers have a smaller channel dimension; the penultimate layer that largely expands dimension above the number of classes. This is to realize flop-efficiency by narrow channel dimensions at earlier layers;

to get model expressiveness with sufficient channel dimension at the final feature (see Table 1).

This channel configuration was firstly introduced by MobileNetV2 [46] and became the design convention of configuring channel dimensions in lightweight networks, but how to adjust the channel dimensions towards the optimal under the restricted computational cost has not been profoundly studied. As shown in Table 1, even network architecture search (NAS)-based models [54, 3, 19, 49, 51, 50, 38, 6, 7, 30] were designed upon the convention or little more exploration within few options near the configuration [11, 53] and focused on searching building blocks. Take one step further from the design convention, we hypothesize that the compact models designed by the conventional channel configuration may be limited in the expressive power due to mainly focusing on flop-efficiency; there would exist a more effective configuration over the traditional one.

In this paper, we investigate an effective channel configuration of a lightweight network with additional accuracy gain. Inspired by the works [56, 60], we conjecture the expressiveness of a layer can be estimated by the matrix rank of the output feature. Technically, we study with the averaged rank computed from the output feature of a bunch of networks that are randomly generated with random sizes to reveal the proper range of expansion ratio at an expansion layer and make a link with a rough design principle. Based on the principle, we move forward to find out an overall channel configuration in a network. Specifically, we search network architectures to identify the channel configuration yielding a better accuracy over the aforementioned convention. It turns out that the best channel configuration is parameterized as a linear function by the block index in a network. This parameterization is similar to the configuration used in the works [22, 13], and we reveal the parameterization is also effective in designing a lightweight model.

Based on the investigation, we propose a new model upon the searched channel parameterization. It turns out that a simple modification upon MobileNetV2 could show remarkable improvement in performance on ImageNet classification. Only with the new channel configuration, our models outperform the state-of-the-art networks such as Effi-

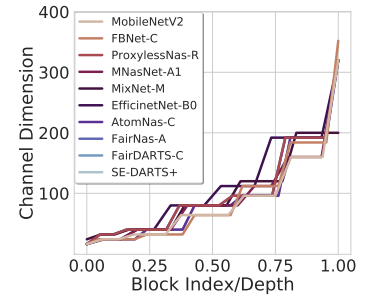| Network	Stem / Building blocks' output channel dimensions	Top-1	Params	Flops
MobileNetV2 [46]	32 / 16(×1)-24(×2)-32(×3)-64(×4)-96(×3)-160(×3)-320(×1)	72.0%	3.4M	0.30B
FBNet-C [54]	16 / 16(×1)-24(×4)-32(×4)-64(×4)-112(×4)-184(×4)-352(×1)	74.9%	5.5M	0.38B
ProxylessNas-R [3]	32 / 16(×1)-32(×2)-40(×4)-80(×4)-96(×4)-192(×4)-320(×1)	74.6%	4.1M	0.32B
MNasNet-A1 [49]	32 / 16(×1)-24(×2)-40(×3)-80(×4)-112(×2)-160(×3)-320(×1)	75.2%	3.9M	0.31B
MixNet-M [51]	24 / 24(×1)-32(×2)-40(×4)-80(×4)-120(×4)-200(×4)	77.0%	5.0M	0.36B
EfficinetNet-B0 [50]	32 / 16(×1)-24(×2)-40(×2)-80(×3)-112(×3)-192(×4)-320(×1)	77.3%	4.8M	0.39B
AtomNas-C [38]	32 / 16(×1)-24(×4)-40(×4)-80(×4)-96(×4)-192(×4)-320(×1)	77.6%	5.9M	0.36B
FairNas-A [6]	32 / 16(×1)-32(×2)-40(×4)-80(×4)-96(×4)-192(×4)-320(×1)	77.5%	5.9M	0.39B
FairDARTS-C [7]	32 / 16(×1)-32(×2)-40(×4)-80(×5)-96(×3)-192(×4)-320(×1)	77.2%	5.3M	0.39B
SE-DARTS+ [30]	32 / 16(×1)-24(×4)-40(×4)-80(×4)-96(×4)-192(×4)-320(×1)	77.5%	6.1M	0.59B

Table 1. **Channel configurations in lightweight models.** We present the output channel dimensions of the stem's 3×3 convolution (e.g., 16, 24, 32) and the building blocks with the number of repeated layers (e.g., ×1, ×2, ×3, ×4, ×5) in block index order. All the models after MobileNetV2 have similar channel configurations to that of MobileNetV2, which have repeated channel dimensions for each stage.

cientNets [50] whose architectures were found by the compound scaling with TPUs. This stresses the effectiveness of our channel configuration over the convention and may encourage the researchers in the NAS field to adopt our channel configuration into the network search space for further performance boosts. The performance improvement of ImageNet classification is well transferred to the object detection and instance segmentation on the COCO dataset [32] and the various fine-grained classification tasks. This indicates our backbones work as strong feature extractors.

Our contributions are 1) a study on designing a single layer (§3); 2) a network architecture exploration concerning the channel configuration towards a simple yet effective parameterization (§4); 3) using our models to achieve remarkable results on ImageNet [45] outperformed recent lightweight models including NAS-based models (§5); 4) revealing the high applicability of our ImageNet-pretrained backbones transferring to several tasks including object detection, instance segmentation and fine-grained classification, which indicate the high expressiveness of our model and the effectiveness of our channel configuration (§5).

2. Related Work

After appearance of AlexNet [29], VGG [47], GoogleNet [48], and ResNet [15] which show significant improvements in ImageNet classification, much lighter models such as [23, 20, 23] have been proposed with lowered computational budgets. Using the new operator depthwise convolution (dwconv) [20], several architectures [5, 46, 59, 35] have been proposed with further efficient architecture designs. Taking advantage of the depthwise convolution could reduce a large amount of learnable parameters, and showed significant FLOPs reduction. Recently, structured network architecture search (NAS) methods have been proposed to yield the lightweight models [54, 11, 3, 19, 49, 51, 50, 53, 38, 6, 7, 30], and EfficientNet [50] which based on compound scaling of width, depth, and resolution, became a de facto state-of-art model. Take one step forward from the existing lightweight models, we focus on finding an effective channel config-

uration for an inverted bottleneck module, which is an alternative to searching building blocks.

3. Designing an Expansion Layer

In this section, we study how to design a layer properly considering the expressiveness, which is essential to design an entire network architecture.

3.1. Preliminary

Estimating the expressiveness. In language modeling, the authors [56] firstly highlighted that the softmax layer may suffer from turning the logits to the entire class probability due to the rank deficiency. This stems from the low input dimensionality of the final classifier and the vanished nonlinearity at the softmax layer when computing the log-probability. The authors proposed a remedy of enhancing the expressiveness, which improved the model accuracy. This implies that a network can be improved by dealing with the lack of expressiveness at certain layers. Estimating the expressiveness was studied in a model compression work [60]. The authors compressed a model at layer-level by a low-rank approximation; investigated the amount of compression by computing the singular values of each feature. Inspired by the works, we conjecture that the rank may be closely related to the expressiveness of a network, and studying it may provide an effective layer design guide.

Layer designs in practice. ResNet families [15, 16, 55] have bottleneck blocks doubling the input channel dimensions (i.e., 64-128-256-512 in order) to make the final dimension (2048) above the number of classes at last. The recent efficient models [54, 11, 3, 19, 49, 51, 50, 53, 38, 6, 7, 30] increase the channel dimensions steadily in inverted bottlenecks; therefore, they commonly involve a large expansion layer at the penultimate layer. The output dimension of the stem is set to 32 which expands the 3-dimensional input. Both of the inverted bottleneck [46] and bottleneck block [15] have the convolutional expansion layer with the predefined expansion ratio (mostly 6 or 4). Are these layers designed correctly and just need to design a new model accordingly?
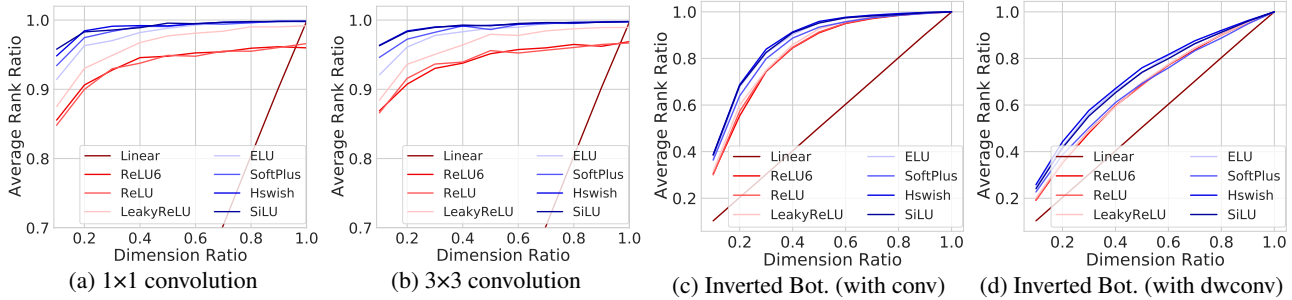
Figure 1. **Visualization of the output rank**. We measure the rank ratio (i.e., rank/output channel dimension) vs. channel dimension ratio (i.e., input channel dimension/output channel dimension) from diverse architectures averaged over 1,000 random-sized networks with various nonlinear functions: (a) A single 1×1 convolution; (b) A single 3×3 convolution; (c) An inverted bottleneck with a 3×3 convolution; (d) An inverted bottleneck with a 3×3 depthwise convolution [46]. We fundamentally observe that all the ranks are expanded above the input channel dimensions by the nonlinear functions with different network architectures.

3.2. Empirical study

Sketch of the study. We aim to study a design guide of a single expansion layer that expands the input dimension. We measure the rank of the output features from the diverse architectures (over 1,000 random-sized networks) and see the trend as varying the input dimensions towards the output dimensions. The rank is originally bounded to the input dimension, but the subsequent nonlinear function will increase the rank above the input dimension [1, 56]. However, a certain network fails to expand the rank close to the output dimension, and the feature will not be fully utilized. We statistically verify the way of avoiding failure when designing the network. The study further uncovers the effect of complicated nonlinear functions such as ELU [8] or SiLU (Swish-1) [17, 42] and where to use them when designing lightweight models.

Materials. We generate a bunch of networks with the building blocks consists of 1) a single 1×1 or 3×3 convolution layer; 2) an inverted bottleneck block [46] with a 3×3 convolution or 3x3 depthwise convolution inside. We have the layer output (i.e., feature) $f(\mathbf{WX})$ with $\mathbf{W} \in \mathbb{R}^{d_{out} \times d_{in}}$ and the input $\mathbf{X} \in \mathbb{R}^{d_{in} \times N}$, where f denotes a nonlinear function[1] with the normalization (we use a BN [24] here). d_{out} is randomly sampled to realize a random-sized network, and d_{in} is proportionally adjusted for each *channel dimension ratio* (i.e., d_{in}/d_{out}) in the range [0.1, 1.0]. N denotes the batch size, where we set $N > d_{out} > d_{in}$. We compute *rank ratio* (i.e., $\mathrm{rank}(f(\mathbf{WX}))/d_{out}$) for each model and average them. An inverted bottleneck block is similarly handled as a single convolution layer to compute rank[2].

Observations. Figure 1 shows how the rank changes with respect to the input channel dimension on average. Dimension ratio (d_{in}/d_{out}) on x-axis denotes the reciprocal of the

expansion ratio [46]. We observe the followings:

(i) *Drastic channel expansion harms the rank.* This holds for a single convolution layer and an inverted bottleneck both as shown in Figure 1. The impact gets bigger with 1) an inverted bottleneck (see Figure 1c); 2) a depthwise convolution (see Figure 1d)[3];

(ii) *Nonlinearities expand rank.* Figure 1 shows averaged rank is expanded above the input channel dimension by nonlinear functions. They expand rank more at smaller dimension ratio, and complicated ones such as ELU, or SiLU do more;

(iii) *Nonlinearities are critical for convolutions.* Nonlinearities expand the rank of 1×1 and 3×3 single convolutions more than an inverted bottleneck (see Figure 1a and 1b vs. Figure 1c and 1d).

What we learn from the observations. We learned the followings: 1) an inverted bottleneck is needed to design with the expansion ratio of 6 or smaller values at the first 1×1 convolution; 2) each inverted bottleneck with a depthwise convolution in a lightweight model needs a higher channel dimension ratio; 3) a complicated nonlinearity such as ELU and SiLU needs to be placed after 1×1 convolutions or 3×3 convolutions (not depthwise convolutions). Based on the knowledge, in the following section, we perform channel dimension searches to find an effective channel configuration for entire channel dimensions. This is to uncover whether the conventional way of configuring channels shown in Table 1 is optimal or not, albeit the models have worked well with high accuracy.

Verification of the study. We finally provide an experimental backup to make sure what we have learned contributes to improving accuracy. We train the models consist-

---

[1]We use ReLU [39], ReLU6 [46], LeakyReLU [36], ELU [8], Soft-Plus [12], Hard Swish [19], and SiLU (Swish-1) [17, 42].

[2]We denote the input and output of an inverted bottleneck as the input of the first 1×1 convolution; the output after the addition operation of the shortcut and the bottleneck, respectively.

[3]Actually, many recent lightweight models avoid increasing the channel dimension drastically (i.e., dimension ratio is usually higher than 0.5) at an inverted bottleneck; the models have inverted bottlenecks where the first 1×1 convolution has the expansion ratio 3, 4, or 6. Figure 1a tells us the design factors are tenable; the architects may know this empirically, but we believe that we have provided an underlying reason through the study.

Network	FLOPs	Top-1	Nuc. norm
Baseline	103M	48.0%	5997.5
+ Increase DR of 1×1 conv (1/20→1/6)	99M	52.1%	6655.9
+ Increase DR of IB (0.22→0.8)	105M	53.8%	6703.2
+ Replace ReLU6 with SiLU	105M	54.6%	6895.9

Table 2. **Factor analysis of the study**. We use four different models with similar computational complexity and report the accuracy and rank represented by the nuclear norm of the final feature. We average all the numbers over three models trained on CIFAR-100 [28]. Each factor successively improves the accuracy and expands the rank without extra computational cost.

ing of two inverted bottlenecks (IBs)[4] to adjust the channel dimension ratio (DR) of IBs and the first 1×1 convolutions in each IB. Starting from the baseline with the low DR 1/20, we successively study through the picked models with 1) increasing DR of the first 1×1 conv to 1/6; 2) increasing DR at every IB from 0.22 to 0.8; 3) replacing the first ReLU6 with SiLU in each IB. Table 2 shows each factor works well, and the rank and accuracy increase together.

4. Designing with Channel Configuration

4.1. Problem Formulation

Our goal is to reveal an effective channel configuration of designing a network under the computational demands. We formulate the following problem:

$$\max_{c_i, i=1...d} \text{Acc}(N(c_1, \ldots c_d))$$
$$\text{s.t.} \quad c_1 \leq c_2 \leq \cdots \leq c_{d-1} \leq c_d,$$
$$\text{Params}(N) \leq P, \ \text{FLOPs}(N) \leq F, \quad (1)$$

where Acc denotes the top-1 accuracy of the model; c_i denotes output channel of i-th block among d building blocks; P and F denote the target parameter size and FLOPs. We involve the monotonic increasing of c_i because this contains the channel configurations shown in Table 1; we note that the opposite case, channel dimension consistently decreasing, requires a hard-to-use amount of FLOPs. Here, we are not targeting a hardware-specific model, so we concern with FLOPs rather than inference latency. Notice that many NAS methods [54, 3, 19, 49, 51, 50, 38, 6, 7, 30] search the network N with fixing c_i with based on the predefined channel configurations as shown in Table 1, but on the other hand we search c_i while fixing the network N.

4.2. Searching with channel parameterization

We observe a general trend of architectural shapes for diverse computational demands through searches. Alternative to optimizing eq.(1) directly, we represent the channel dimensions at each building block with a piecewise linear function, which can reduce the search space. We parameterize the channel dimensions as $c_i = af(i) + b$, where

---

[4]The models with a single IB cannot be similar in computational complexity with the fixed stem, so the models should contain at least two IBs.

a and b are to be searched; let $f(i)$ as a piecewise linear function by picking a subset of $f(i)$ up from $1 \ldots d$. In this fashion, the channel parameterizations contain the conventional MobileNetV2-based channel configurations (i.e., stage-wise channel configurations) shown in Table 1. We adopt the CIFAR-10 and CIFAR-100 datasets [28] as done in NAS methods [34, 4, 30] to search the parameterization.

To control the other variables, we set all the networks that have the fixed channel dimension at the stem 3×3 convolution of 16 followed by a BN [24] with a ReLU and have the large expansion layer at the penultimate layer. We use the original inverted bottleneck (expansion ratio of 6) [46] as our building blocks, which is a fundamental block of lightweight NAS methods, so we do not search the building blocks' expansion ratio. The chosen elements are based on the above investigation of single-layer design. Optimization is done alternatively by searching and training a network. We train each model for 30 epochs for faster training [41] and the early stopping strategy [30]. Each training is repeated three times for averaging the accuracies to reduce the accuracy fluctuation caused by random initialization.

4.3. Search Results

We perform individual searches under different constraints of computational costs to provide reliable and generalizable search results. We assign four search constraints to aim to search across different target model sizes. After each search, we collect top-10%, middle-10% (i.e., the models between top-50% and 60%), and bottom-10% models in terms of the model accuracy from 200 searched models to analyze them. To this end, we first visualize the collected models' channel configuration in Figure 2 of each search; we then report the detailed performance statistics of the models and the best/worst-performing models' channel configuration in Table 3.

Figure 2 illustrates the clear trends in which channel configuration is more effective in terms of accuracy. We observe that the *linear parameterizations* by the block index as colored with red enjoy higher accuracies while maintaining similar computational costs. This parameterization is regularly found throughout searching in diverse environments as shown in the figure. Note that the best models in Table 3 have the channels configured with linearly increasing that is identical to the linear parameterization. The models in green have highly reduced the input-side channels and therefore, most of the weight parameters are placed at the output-side resulting in the loss of accuracy. In addition, intriguingly, blue represents the models at the middle 10%-accuracy, which looks similar to the channel configurations in Table 1. Their conventional configurations are designed to attain a flop-efficiency by limiting the channels at earlier layers and giving more channels close to the output. Therefore, we may safely suggest that we need to change the con-
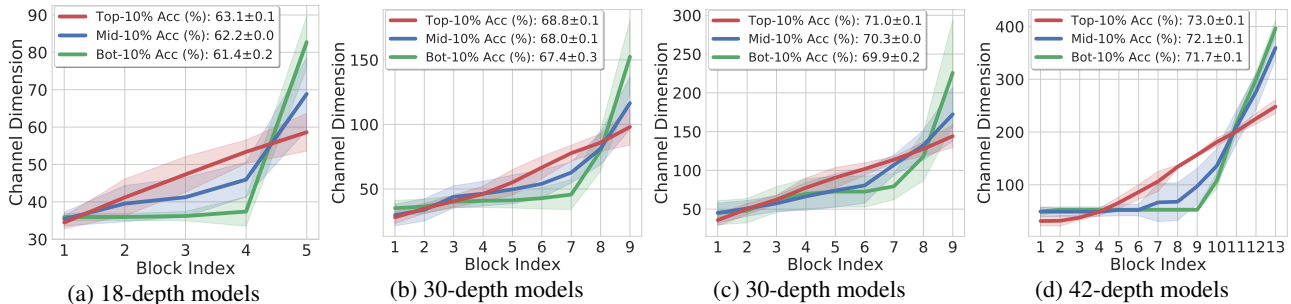
Figure 2. **Visualization of the searched models' channel dimensions vs. block index**. Red: top-10%; blue: middle-10%; green: bottom-10% accuracy models; we plot the averaged channel configurations with the 1-sigma range over each searched candidate.

Search constraints	Ranking	Acc (%)	Params (M)	FLOPs (M)	Best and worst models among searched configurations
(a) Models with 5 inverted bot., # Params≃0.2M, FLOPs≃30M	Top-10% Mid-10% Bot-10%	63.1±0.1 62.2±0.0 61.4±0.2	0.2±0.0 0.2±0.0 0.2±0.0	30±1 30±1 30±1	Best: 34-34-45-55-66 (Acc: 63.4%) Worst: 36-36-36-36-83 (Acc: 61.1%)
(b) Models with 9 inverted bot., # Params≃0.5M, FLOPs≃100M	Top-10% Mid-10% Bot-10%	68.8±0.1 68.0±0.1 67.4±0.3	0.5±0.0 0.5±0.0 0.5±0.0	101±5 100±6 97±8	Best: 24-33-42-50-59-68-77-85-94 (Acc: 68.9%) Worst: 39-39-39-39-39-39-39-87-158 (Acc: 67.6%)
(c) Models with 9 inverted bot., # Params≃1.0M, FLOPs≃200M	Top-10% Mid-10% Bot-10%	71.0±0.1 70.3±0.0 69.9±0.2	1.0±0.0 1.0±0.0 1.0±0.0	210±15 198±18 200±15	Best: 30-45-59-74-88-103-117-132-146 (Acc: 71.1%) Worst: 47-47-70-70-70-70-70-70-364 (Acc: 69.6%)
(d) Models with 13 inverted bot., # Params≃3.0M, FLOPs≃300M	Top-10% Mid-10% Bot-10%	73.0±0.1 72.1±0.1 71.7±0.1	3.0±0.0 3.0±0.0 3.0±0.0	351±5 351±5 351±6	Best: 34-34-34-40-64-88-112-136-160-184-208-232-256 (Acc: 73.2%) Worst: 52-52-52-52-52-52-52-52-52-115-263-263-412 (Acc: 71.6%)

Table 3. **Detailed searched channel configurations.** We provide the detailed individual searched results under the different search constraints. Along with Figure 2, we report the detailed numbers including the averaged accuracy, # of parameters, FLOPs over top-10%, midddle-10%, and bottom-10% accuracy models. We further present each of the best and the worst models' channel configurations.

vention (blue) towards the new channel configuration (red) that can achieve additional accuracy gains.

We have additional search results shown in Figure 4 when fixing the network depth and searching under the different computational costs; we found identical linear parameterizations. Furthermore, we perform the same experiment on CIFAR-10 with the constraints of computational costs and found the same trends in Figure A1. As mentioned earlier, we believe that the success of the works [13, 22] may stem from similar parameterizations. In the case of training the 18-depth models, which have about 30 MFLOPs, it takes 1.5 GPU days for searching and training about 100 models for individual 30 epochs training with 3 runs for training each model. The training cost for the entire search is $30\,\text{MFLOPs} \times 30\,\text{epochs} \times 3\,\text{runs} \times 100\,\text{models} = 2.7 \times 100\,\text{GFLOPs}$ epochs, and this is smaller than training a single ImageNet model ResNet50 for 100 epochs ($4.0\,\text{GFLOPs} \times 100$ epochs). Although the actual training time is less than that of ResNet50, comparing directly with the ImageNet model may be inappropriate. We provide a rough estimate of the amount of computation in practice.

4.4. Network upgrade

We rebuild the existing model based on the investigations in practice. From the baseline MobileNetV2 [46] which introduced the convention of channel configuration, we only reassign each output channel dimension of inverted bottlenecks by following the parameterization. We use the identical setting of the stem (i.e., 3×3 convolution with BN and ReLU6) and the inverted bottleneck with the expansion ratio 6. We use the same large expansion layer at the penultimate layer. To fairly compare with the aforementioned lightweight models MobileNetV3 [19], MixNet [51], EfficientNet [50], AtomNas [38], FairNAS [6], FairDARTS [7], DART+ [30], we further replace ReLU6 with SiLU [17, 42] and adopt SE [21] in the inverted bottlenecks.

Based on the investigation in §3, we replace ReLU6 only after the first 1×1 convolution in each inverted bottleneck because we observed the layer with a smaller dimension ratio needs to be more addressed; the second depthwise convolution has the channel dimension ratio of 1, so we do not replace ReLU6 here. This can further realize the simplicity of model design and benefit from faster training speeds since ReLU6s remain. Using other nonlinear functions such as ELU shows similar accuracy gains (see Table A2, but we use SiLU (Swish-1) for a fair comparison with the aforementioned lightweight models which use SiLU.

Note that only with these simple modifications, our model outperforms NAS-based methods in many experiments (§5), which signifies the importance of the channel configuration. We call our model Rank Expansion Networks (ReXNet) as observed the actual rank expansion in §6. Additionally, we build another model with the linear parameterization upon MobileNetV1 [20], which shows a large accuracy improvement (+2.3pp) over the baseline 72.5%. We call this model ReXNet (plain). The detailed model specification of ReXNets is provided in Appendix B.

Network	Top-1	Top-5	FLOPs	Params
MobileNetV1 [20]	70.6%	89.5%	0.56B	4.2M
MobileNetV2 [46]	72.0%	91.0%	0.30B	3.5M
MobileNetV2 (x1.4) [46]	74.7%	92.5%	0.59B	6.9M
ShuffleNetV1 (x2) [59]	73.7%	-	0.52B	-
ShuffleNetV2 (x2) [35]	75.4%	-	0.60B	-
NASNet-A [61]	74.0%	91.7%	0.56B	5.3M
AmoebaNet-A [43]	75.5%	92.0%	0.56B	5.1M
PNASNet [33]	74.2%	91.9%	0.59B	5.1M
DARTS [34]	73.1%	91.0%	0.60B	4.9M
FBNet-C [54]	74.9%	-	0.38B	5.5M
Proxyless-R [3]	74.6%	92.2%	0.32B	4.1M
P-DARTS [4]	75.6%	92.6%	0.56B	4.9M
MobileNetV3-Large† [19]	75.2%	-	0.22B	5.4M
MnasNet-A3† [49]	76.7%	93.3%	0.40B	5.2M
MixNet-M† [51]	77.0%	93.3%	0.36B	5.0M
EfficientNet-B0† [50]	76.7%	-	0.39B	5.3M
AtomNAS-C+† [38]	77.6%	93.6%	0.36B	5.9M
EfficientNet-B0†* [50]	77.3%	-	0.39B	5.3M
FairNAS-A†* [6]	77.5%	93.7%	0.39B	5.3M
FairDARTS-C†* [7]	77.2%	93.5%	0.39B	5.3M
SE-DARTS+†* [30]	77.5%	93.6%	0.59B	6.1M
ReXNet (×1.0)†	77.3%	93.4%	0.40B	4.8M
ReXNet (×1.0)†*	**77.9%**	**93.9%**	0.40B	4.8M

Table 4. **Comparison of ImageNet performance.** We compare ReXNet (×1.0) with other lightweight models including structured NAS-based models. We report the accuracy at the final epoch of training. †: used SE and SiLU ; *: used additional techniques such as AutoAug, RandAug, or Mixup.

5. Experiment

5.1. ImageNet Classification

Training on ImageNet. We train our model on the ImageNet dataset [45] using the standard data augmentation [48] with stochastic gradient descent (SGD) and minibatch size of 512 on four GPUs. Learning rate is initially set to 0.5 and is scheduled by cosine learning rate scheduling. Weight decay is set to 1e-5. Table 4 show the performance comparison with popular lightweight models where all the reported models are trained and evaluated with 224×224 image size. Comparing with the models [19, 49, 51, 50] with SE [21] and SiLU[5] [17, 42], our model outperforms the most of the models searched by NAS including MobileNetV3-Large, MNasNet-A3, MixNet-M, and EfficientNet-B0 (without AutoAug [9]) with at least **+0.3pp** accuracy improvement. Our model outperforms MixNet-M and AtomNas-C+ which use mixed-kernel operations under similar computational costs.

Additionally, we train our model with RandAug [10] to compare it fairly with the NAS-based models [50, 6, 7, 30] using additional regularizations (denoted by * in Table 4) such as Mixup [58], AutoAug [9], and RandAug [10], our model improves all the models including EfficientNet-B0 (with AutoAug), FairNas-A, FairDARTS-C, and SE-
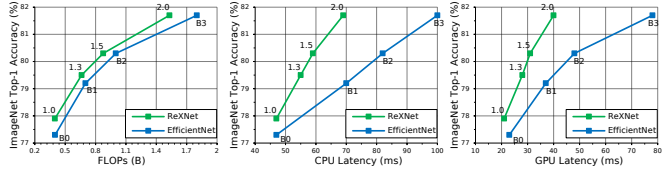

Figure 3. **ImageNet accuracy vs. FLOPs and latencies**. We visualize the numbers of ReXNets (×1.0, ×1.3, ×1.5, and ×2.0) and EfficientNets-B0, B1, B2, and B3 in Table 5. We observe ReXNets outperforms each of the EfficientNet counterpart in practice.

Network	Top-1	Top-5	FLOPs	Params	CPU	GPU
ReXNet (×0.9)	**77.2%**	**93.5%**	0.35B	4.1M	45ms	20ms
Eff-B0 [50]	77.3%	93.5%	0.39B	5.3M	47ms	23ms
ReXNet (×1.0)	**77.9%**	**93.9%**	0.40B	4.8M	47ms	21ms
Eff-B1 [50]	79.2%	94.5%	0.70B	7.8M	70ms	37ms
ReXNet (×1.3)	**79.5%**	**94.7%**	0.66B	7.6M	55ms	28ms
Eff-B2 [50]	80.3%	95.0%	1.0B	9.2M	77ms	48ms
ReXNet (×1.5)	80.3%	**95.2%**	0.9B	9.7M	59ms	31ms
Eff-B3 [50]	**81.7%**	95.6%	1.8B	12M	100ms	78ms
ReXNet (x2.0)	81.6%	**95.7%**	1.5B	16M	69ms	40ms

Table 5. **Scalablity of our models**. We adjust ReXNet (×1.0) via width multipliers to compare with EfficientNets [50] on ImageNet dataset. We report the overall performances with CPU and GPU latencies in practice.

DARTS+ by at least **+0.4pp**[6]. Strikingly, our model does not require further searches, but it either outperforms or is comparable to NAS-based models.

Comparison with Efficientnets. We compare ReXNets with EfficientNets [50] about model scalability with the performances. To this end, we adopt the width multiplier concept [20, 46, 59, 35, 19] for scaling. Note that all our models are trained with the fixed resolution 224×224 unlike EfficientNets trained with the resolutions from 224×224 to 600×600. We do not use the method such as FixResNet [52] which performs additional training.

Table 5 shows our models adjusted by the multiplier from ×0.9 to ×2.0 with remarkable accuracy increments (see more models in Appendix B.3). We measure CPU and GPU inference speeds to show the efficiency; we average the latencies over 1,000 runs with the batch size 1 on an Intel Xeon CPU E5-2630 and the batch size 64 on a V100 GPU, respectively. Figure 3 visualizes our models' computational efficiency compared with EfficientNets; as the model size is larger, we observe that our models are much faster than EfficientNets. Notice that ReXNet (×2.0) is about **1.4× and 2.0× faster** than EfficientNet-B3 on CPU and GPU, respectively with almost the same accuracy and improves **+2.5pp** top-1 accuracy on EfficientNet-B1 at a similar speed. This benefit may come from the fixed resolution and network depth for training and inference over all the models which can reduce the memory access time.

---

[5]MobileNetV3 uses Hard Swish [19], and MnasNet does not use SiLU.

[6]ReXNet (×1.0) has been improved to **78.1**% trained by the novel optimizer AdamP [18] replacing SGD and further improved to **78.4**% trained with the new training method ReLabel [57], respectively.

Model	Input Size	Bbox AP at IOU			Params	FLOPs
		AP	AP_{50}	AP_{75}		
MobileNetV1 [20] + SSDLite	320×320	22.2	-	-	5.1M	1.31B
MobileNetV2 [46] + SSDLite	320×320	22.1	-	-	4.3M	0.79B
MobileNetV3 [19] + SSDLite	320×320	22.0	-	-	5.0M	0.62B
MnasNet-A1 [49] + SSDLite	320×320	23.0	-	-	4.9M	0.84B
EfficientNet-B0 [50] + SSDLite[†]	320×320	23.5	39.5	23.8	6.2M	0.97B
ReXNet (×0.9) + SSDLite	320×320	24.7	41.4	24.9	5.0M	0.88B
ReXNet (×1.0) + SSDLite	320×320	**25.3**	**42.3**	**25.6**	5.7M	1.01B
EfficientNet-B1 [50] + SSDLite[†]	320×320	25.8	42.5	26.2	8.7M	1.35B
EfficientNet-B2 [50] + SSDLite[†]	320×320	26.6	43.6	27.2	10.0M	1.55B
ReXNet (×1.3) + SSDLite	320×320	**27.3**	**45.2**	**27.8**	8.4M	1.60B

Table 6. **COCO object detection results with SSDLite [46]**. We report box AP scores on `testdev2017` of our models with SSDLite comparing with lightweight models (FLOPs≃1.0B). [†]: the model performances are trained by ourselves.

Backbone	Input Size	Bbox AP at IOU			Params	FLOPs
		AP	AP_{50}	AP_{75}		
EfficientNet-B0 [50] + FPN	1200×800	38.0	60.1	40.4	21.0M	123.0B
ReXNet (×0.9) + FPN	1200×800	**38.0**	**60.6**	**40.8**	20.1M	123.0B
ResNet50 [15] + FPN	1200×800	37.6	58.2	40.9	41.8M	202.2B
ReXNet (×2.2) + FPN	1200×800	**41.5**	**64.0**	**44.9**	33.0M	153.8B

Table 7. **COCO object detection results with Faster RCNN [44] and FPN [31]**. We report box APs on `val2017`.

Dataset	Network	Top-1	FLOPs	Params
Food-101 [2]	ResNet50 [15]	87.0%	4.1B	25.6M
	EfficientNet-B0 [50]	87.5%	0.4B	5.3M
	ReXNet (×1.0)	**88.4**%	0.4B	4.8M
Stanford Cars [27]	ResNet50 [15]	**92.6**%	4.1B	25.6M
	EfficientNet-B0 [50]	90.7%	0.4B	5.3M
	ReXNet (×1.0)	91.5%	0.4B	4.8M
Aircraft [37]	ResNet50 [15]	89.4%	4.1B	25.6M
	EfficientNet-B0 [50]	87.1%	0.4B	5.3M
	ReXNet (×1.0)	**89.5**%	0.4B	4.8M
Flowers-102 [40]	ResNet50 [15]	97.7%	4.1B	25.6M
	EfficientNet-B0 [50]	97.3%	0.4B	5.3M
	ReXNet (×1.0)	**97.8**%	0.4B	4.8M

Table 8. **Transfer learning results on fine-graned datasets**. Our ReXNet (×1.0) is more efficient than ResNet50 and EfficientNet-B0, yet transfers well to the various datasets.

5.2. COCO object detection

Training SSDLite. We validate our backbones through object detection on the COCO dataset [32] in SSDLite [46] which has lightweight detection heads suitable for seeing the feature extractor's capability. We follow the identical design elements [46, 19, 49] by building the first head on top of the final expansion layer which has the stride 16 and another head on top of the final layer. We train ReXNets (×0.9, ×1.0, and ×1.3) and EfficientNets-B0, B1, and B2 in SSDLites with the same training settings [46, 19, 49] including 320×320 image resolution for fair comparison.

Table 6 shows ours largely outperform the other backbones with comparable computational costs. ReXNet (×1.0) outperforms EfficientNet-B1 trained with 240×240 by **+1.8pp**, and ReXNet (×1.3) outperforms EfficientNet-B2 trained with 260×260 by **+0.7pp** under similar computational costs. Interestingly, ReXNet (×0.9) achieves **+1.2pp** AP improvement over EfficientNet-B0 with less computa-

tional costs. The large AP improvements indicate our channel dimension configuration can help finetuning as well. We provide more detection results in Appendix C.2. Note that ReXNet (×0.9) is faster than EfficientNet-B0 (75ms vs. 77ms), and ReXNet (×1.3) is faster than EfficientNet-B2 (88ms vs. 101ms) on an Intel Xeon CPU E5-2630.

Training Faster RCNN. We adopt Faster RCNN [44] to explore the maximal performance of ReXNets. We plug ReXNets (×0.9) and (×2.2), EfficientNet-B0, and ResNet50 into FPN [31] and train with the image size of 1200×800 following the original setting [44, 31] such as freezing all BNs. Table 7 shows ReXNets' superiority over others; ReXNet (×0.9) and ReXNet (×2.2) improves EfficientNet-B0 and ResNet50 by **+0.5pp** and **+3.9pp** APs, respectively with smaller computational costs; ReXNet (×2.2) achieves **41.5** AP only with the standard Faster RCNN framework without any bells and whistles.

5.3. Fine-grained classifications

We finetune the ImageNet-pretrained models on the datasets Food-101 [2], Stanford Cars [27], FGVC Aircraft [37], and Oxford Flowers-102 [40] to verify the transferability. We compare ReXNet (×1.0) with ResNet50 [15] and EfficientNet-B0 [50] on each dataset. We exhaustively search the hyper-parameters including learning rate and weight decay for the best results as done in the work [26] for each model. We train all the layers using SGD with the same learning rates without using additional training techniques. Training and evaluation are done with 224×224 image size; we use center-cropped images from the resized images with the shorter side of 256 for evaluation. Table 8 shows ReXNet (×1.0) outperforms EfficientNet-B0 for all the datasets with large margins and mostly surpasses

Backbone	Input Size	Mask AP at IOU			Bbox AP at IOU			Params	FLOPs
		AP	AP_{50}	AP_{75}	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}		
EfficientNet-B0 [50] + FPN	1200×800	34.8	56.8	36.6	38.4	60.2	40.8	23.7M	123.0B
ReXNet (×0.9) + FPN	1200×800	**35.2**	**57.4**	**37.1**	**38.7**	**60.8**	**41.6**	22.8M	123.0B
ResNet50 [15] + FPN	1200×800	34.6	55.9	36.8	38.5	59.0	41.6	44.2M	206.5B
ReXNet (×2.2) + FPN	1200×800	**37.8**	**61.0**	**40.2**	**42.0**	**64.5**	**45.6**	35.6M	153.8B

Table 9. **COCO instance segmentation results with Mask RCNN [14] and FPN [31]**. We report box and mask APs on `val2017`.
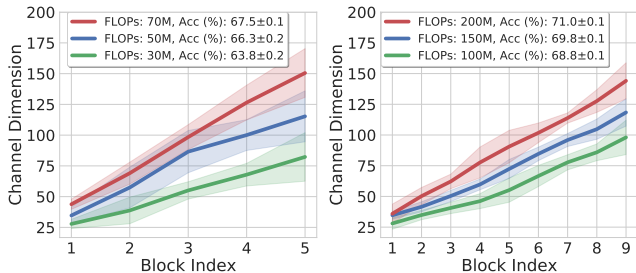
(a) Top-10% models (18-depth)   (b) Top-10% models (30-depth)

Figure 4. **Searched configuration under fixed depth**. We similarly plot the searched channel dimensions of top-10% models from the searched candidates with fixed depths. The search constraints (FLOPs) are Red: 70M (left) 200M (right); Blue: 50M (left) and 150M (right); Green: 30M (left) and 100M (right).

ResNet50 which has **5×** more parameters. This indicates our backbone can perform as a more generalizable feature extractor than other models even with fewer parameters and may reflect the effectiveness of our channel configuration.

5.4. COCO Instance Segmentation

We use Mask RCNN [14] to validate the performance of ReXNets on instance segmentation. We train the models with the identical setting in the Faster RCNN training in Table 7. Table 9 shows our backbones' efficiency; ReXNet (×0.9) outperforms EfficientNet-B0 by **+0.6pp** mask AP and **+0.5pp** bbox AP with fewer parameters; ReXNet (×2.2) gains **+3.2pp** mask AP and **+3.5pp** bbox AP over ResNet50 with much less computational complexity.

6. Discussion

Fixing network depth and searching models. We further verify the linear channel parameterization by searching for new models under different constraints. We fix the network depth as 18 and 30 and give the constraints with FLOPs of 30M, 50M, and 70M for 18-depth models; 100M, 150M, and 200M for 30-depth models, respectively. Eventually, Figure 4 shows all the top 10% models have identical shapes that are linear functions but have different slopes due to the different FLOPs. This shows that linear channel configurations outperform the conventional configuration for various computational demands.

Rank visualization. We study the model expressiveness by analyzing the rank of trained models. This analysis is to see how the linear parameterization affects the rank. We visualize the rank computed from the trained models in two
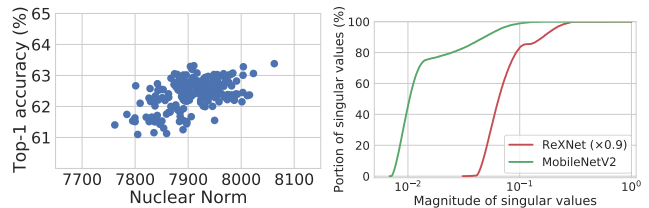

Figure 5. **Visualization of rank**. The models with higher accuracy have higher nuclear norms (left); our ReXNet has larger singular values than MobileNetV2 (right).

manners; we show the distribution of accuracy vs. rank (represented by the nuclear norm) from 18-depth models in §4; we then compare MobileNetV2 with ReXNet (×0.9) by visualizing the cumulative distribution of the singular values that are normalized to [0, 1] computed from the features of the images in the ImageNet validation set. As shown in Figure 5, we observe 1) a higher-accuracy model has a higher rank; 2) ReXNet clearly expands the rank over the baseline.

7. Conclusion

In this work, we have studied a new approach to designing lightweight models. We have conjectured that the conventional stage-wise channel configurations which have widely used in NAS methods after proposed by MobileNetV2 have limited the model accuracy; if we find a more effective channel configuration, the models would have accuracy gains. We first studied an appropriate way of designing a single building block and a layer targeted to lightweight models; we then proposed a search method for a channel configuration via piece-wise linear functions of block index. The search space contains the conventions, and we found an effective channel configuration which can be parameterized by a linear function of the block index. Based on the parameterization, we have achieved a new model which could outperform the recent lightweight models including NAS-based models on ImageNet dataset. Our ReXNets further showed remarkable finetuning performances on COCO object detection, instance segmentation, and fine-grained classifications. Consequently, we believe our work highlighted a new perspective of architecture search that helps to search with.

References

[1] Arash Amini, Amin Karbasi, and Farokh Marvasti. Low-rank matrix approximation using point-wise operators. *IEEE Transactions on Information Theory*, 58(1):302–310, 2011. 3

[2] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101–mining discriminative components with random forests. In *European conference on computer vision*, pages 446–461. Springer, 2014. 7

[3] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *ICLR*, 2018. 1, 2, 4, 6

[4] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *ICCV*, 2019. 4, 6

[5] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *CVPR*, 2017. 2

[6] Xiangxiang Chu, Bo Zhang, Ruijun Xu, and Jixiang Li. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. *arXiv preprint arXiv:1907.01845*, 2019. 1, 2, 4, 5, 6

[7] Xiangxiang Chu, Tianbao Zhou, Bo Zhang, and Jixiang Li. Fair darts: Eliminating unfair advantages in differentiable architecture search. In *ECCV*, 2020. 1, 2, 4, 5, 6

[8] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In *ICLR*, 2016. 3

[9] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *CVPR*, 2019. 6

[10] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical data augmentation with no separate search. *arXiv preprint arXiv:1909.13719*, 2019. 6

[11] Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, et al. Chamnet: Towards efficient network design through platform-aware model adaptation. In *CVPR*, 2019. 1, 2

[12] Charles Dugas, Yoshua Bengio, François Bélisle, Claude Nadeau, and René Garcia. Incorporating second-order functional knowledge for better option pricing. In *NIPS*, 2001. 3

[13] Dongyoon Han, Jiwhan Kim, and Junmo Kim. Deep pyramidal residual networks. In *CVPR*, 2017. 1, 5

[14] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *ICCV*, 2017. 8

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 2, 7, 8

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *ECCV*, 2016. 2

[17] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016. 3, 5, 6

[18] Byeongho Heo, Sanghyuk Chun, Seong Joon Oh, Dongyoon Han, Sangdoo Yun, Gyuwan Kim, Youngjung Uh, and Jung-Woo Ha. Adamp: Slowing down the slowdown for momentum optimizers on scale-invariant weights. *arXiv: 2006.08217*, 2020. 6

[19] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *ICCV*, 2019. 1, 2, 3, 4, 5, 6, 7

[20] 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. *arXiv preprint arXiv:1704.04861*, 2017. 1, 2, 5, 6, 7

[21] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *arXiv:1709.01507*, 2017. 5, 6

[22] Gao Huang, Zhuang Liu, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, 2017. 1, 5

[23] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016. 2

[24] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 3, 4

[25] Hanjoo Kim, Minkyu Kim, Dongjoo Seo, Jinwoong Kim, Heungseok Park, Soeun Park, Hyunwoo Jo, KyungHyun Kim, Youngil Yang, Youngkwan Kim, et al. NSML: Meet the MLaaS platform with a real-world case study. *arXiv preprint arXiv:1810.09957*, 2018. 8

[26] Simon Kornblith, Jonathon Shlens, and Quoc V Le. Do better imagenet models transfer better? In *CVPR*, pages 2661–2671, 2019. 7

[27] Jonathan Krause, Jia Deng, Michael Stark, and Li Fei-Fei. Collecting a large-scale dataset of fine-grained cars. *In Second Workshop on Fine-Grained Visual Categorization*, 2013. 7

[28] A. Krizhevsky. Learning multiple layers of features from tiny images. In *Tech Report*, 2009. 4

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 2

[30] Hanwen Liang, Shifeng Zhang, Jiacheng Sun, Xingqiu He, Weiran Huang, Kechen Zhuang, and Zhenguo Li. Darts+: Improved differentiable architecture search with early stopping. *arXiv preprint arXiv:1909.06035*, 2019. 1, 2, 4, 5, 6

[31] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *ICCV*, 2017. 7, 8

[32] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014. 2, 7

[33] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *ECCV*, 2018. 6

[34] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *ICLR*, 2019. 4, 6

[35] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *ECCV*, 2018. 1, 2, 6

[36] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013. 3

[37] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151*, 2013. 7

[38] Jieru Mei, Yingwei Li, Xiaochen Lian, Xiaojie Jin, Linjie Yang, Alan Yuille, and Jianchao Yang. Atomnas: Fine-grained end-to-end neural architecture search. In *ICLR*, 2020. 1, 2, 4, 5, 6

[39] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010. 3

[40] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729. IEEE, 2008. 7

[41] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *CVPR*, 2020. 4

[42] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017. 3, 5, 6

[43] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI*, 2019. 6

[44] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, 2015. 7

[45] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 2, 6

[46] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018. 1, 2, 3, 4, 5, 6, 7

[47] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 1, 2

[48] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015. 2, 6

[49] 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 *CVPR*, 2019. 1, 2, 4, 6, 7

[50] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019. 1, 2, 4, 5, 6, 7, 8

[51] Mingxing Tan and Quoc V Le. Mixconv: Mixed depthwise convolutional kernels. In *BMVC*, 2019. 1, 2, 4, 5, 6

[52] Hugo Touvron, Andrea Vedaldi, Matthijs Douze, and Hervé Jégou. Fixing the train-test resolution discrepancy. In *NIPS*, 2019. 6

[53] Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, et al. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In *CVPR*, 2020. 1, 2

[54] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*, 2019. 1, 2, 4, 6

[55] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017. 2

[56] Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W Cohen. Breaking the softmax bottleneck: A high-rank rnn language model. In *ICLR*, 2018. 1, 2, 3

[57] Sangdoo Yun, Seong Joon Oh, Byeongho Heo, Dongyoon Han, Junsuk Choe, and Sanghyuk Chun. Re-labeling imagenet: from single to multi-labels, from global to localized labels. *arXiv preprint arXiv:2101.05022*, 2021. 6

[58] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*, 2018. 6

[59] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, 2018. 1, 2, 6

[60] Xiangyu Zhang, Jianhua Zou, Xiang Ming, Kaiming He, and Jian Sun. Efficient and accurate approximations of nonlinear convolutional networks. In *CVPR*, 2015. 1, 2

[61] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018. 6