Restructurable Activation Networks

Kartikeya Bhardwaj[1], James Ward[1], Caleb Tung[2,*,†], Dibakar Gope[1,*], Lingchuan Meng[3,†],
Igor Fedorov[4,†], Alex Chalfin[1], Paul Whatmough[1], and Danny Loh[1]

[1]Arm Inc., [2]Purdue University, [3]Amazon, [4]Meta

`kartikeya.bhardwaj@arm.com, dibakar.gope@arm.com`

Abstract

Is it possible to restructure the non-linear activation functions in a deep network to create hardware-efficient models? To address this question, we propose a new paradigm called Restructurable Activation Networks (RANs) that manipulate the amount of non-linearity in models to improve their hardware-awareness and efficiency. First, we propose RAN-explicit (RAN-e) – a new hardware-aware search space and a semi-automatic search algorithm – to replace inefficient blocks with hardware-aware blocks. Next, we propose a training-free model scaling method called RAN-implicit (RAN-i) where we theoretically prove the link between network topology and its expressivity in terms of number of non-linear units. We demonstrate that our networks achieve state-of-the-art results on ImageNet at different scales and for several types of hardware. For example, compared to EfficientNet-Lite-B0, RAN-e achieves a similar accuracy while improving Frames-Per-Second (FPS) by $1.5\times$ on Arm micro-NPUs. On the other hand, RAN-i demonstrates up to $2\times$ reduction in #MACs over ConvNexts with a similar or better accuracy. We also show that RAN-i achieves nearly 40% higher FPS than ConvNext on Arm-based datacenter CPUs. Finally, RAN-i based object detection networks achieve a similar or higher mAP and up to 33% higher FPS on datacenter CPUs compared to ConvNext based models[1].

1. Introduction

Tremendous progress has been made towards building efficient deep networks using either model compression [26,28,33], manual model design [39,45], or automatic Neural Architecture Search (NAS)-based techniques [41, 51]. Despite these advances, significant challenges remain in (1) hardware-aware model design especially for AI accelerators like Neural Processing Units (NPUs) [1, 2],

---

[1]The code to train and evaluate RANs and the pretrained networks are available at this Github repository.

and (2) cost of finding optimized models for a given #MACs/#parameters constraint once a good base model is known. We discuss both of these challenges in detail below.

The first challenge relates to the lack of good hardware-aware building blocks. Specifically, even though excellent NAS methods [4, 10, 32, 34, 41, 51–55, 58] exist to find highly efficient deep networks within large search spaces, there has been limited focus on building a *hardware-aware search space* itself, particularly for AI accelerators. Most search spaces for computer vision tasks rely on Inverted Bottleneck (IBN) blocks – the main building blocks used in MobileNet-V2 [45] and EfficientNet [51] – since they result in highly accurate yet compact models. Existing NAS works search over number of channels, expansion ratio, and kernel sizes for IBN blocks [10, 51]. However, it has been established that while IBN blocks are great for generic processors like phone CPUs, they are not always well-suited for AI accelerators due to poor utilization of the accelerator hardware [58, 63]. To address this, recent NAS techniques use fused convolutions [58, 63] which combine the first two layers of the IBN to form a large, regular convolution. This leads to layers that do not present hardware utilization issues but are computationally very expensive in terms of #MACs/#parameters. Hence, a new search space is needed which contains blocks that (a) enable hardware-aware (i.e., high hardware utilization) models for AI accelerators with low #MACs while achieving high accuracy, and (b) are accompanied by a simple search algorithm.

The second challenge relates to the fact that even if a good model has been designed (either using NAS or manually), it is still very costly to scale it up or down to satisfy various #MACs/#parameters constraints. For example, some existing works perform model scaling using ad-hoc methods, which can result in suboptimal networks (e.g., in ConvNexts [39], the number of blocks is scaled from [3,3,9,3] for ConvNext-Tiny to [3,3,27,3] for ConvNext-Small model without an explanation). Other methods rely on extremely costly EfficientNet-like NAS [51] to find optimal width and depth scaling. Therefore, more focus is required on inexpensive model scaling techniques that result

in high accuracy.

In this paper, we propose a new paradigm to create hardware-efficient deep networks. Specifically, we show that manipulating the amount of non-linearity in deep networks can be a new way to achieve hardware-awareness and/or significant reduction in computational cost. Non-linear activation functions have been a fundamental part of deep neural networks since their inception. While many advanced activation functions have been proposed in literature [27, 29, 43], several important questions remain unaddressed. For instance, how much non-linearity can we remove from a network without significant accuracy loss? Activation functions have been viewed as cheap operations in deep learning from a computational cost standpoint. Consequently, they have not been used to build efficient deep networks in prior work. In view of the above challenges, we ask the following **key questions** in this paper:

1. Is it possible to manipulate the non-linearities in a deep network to create accelerator-hardware-aware models?

2. Given a good base model, can changing the amount of non-linearity in a network allow us to quickly scale it up or down to any target resource constraints in a *training-free* way to obtain highly accurate models?

To address the above questions, we propose *Restructurable Activation Networks* (RANs). Our approach is based on *explicit* and *implicit* restructuring of the non-linear activation functions to improve the model efficiency while still achieving high accuracy. Specifically, for the first question, we propose a search space that contains new building blocks that can *restructure* IBN blocks into small, regular convolutions to generate hardware-aware networks. This is highly useful to improve the hardware utilization (e.g., for AI accelerators) without increasing #MACs/#parameters significantly. Since the amount of non-linearity and structure of the model explicitly changes with this technique, we call these models RAN-e (RAN-explicit).

For the second question, we look into a recent study that theoretically analyzes the topological properties of deep networks and shows how accuracy of different models is related to their structural characteristics (e.g., presence of skip connections, etc.) [7]. Since no training is required to evaluate these topological properties, we use this method to scale a given base model in a *training-free* way. We also show that for a certain class of networks, such topological properties are related to the total number of non-linear units in a network. Therefore, changing the topological structure of networks also impacts the amount of non-linearity and, thus, affects the expressivity of deep networks. Hence, we exploit the metrics in [7] to scale ConvNext class of models and show that they can be scaled in a significantly better way than the ad-hoc method used in ConvNext [39].

Since our method results in an implicit restructuring of non-linearity, we call these models RAN-i (RAN-implicit).

We emphasize that our work is *not* a full-blown NAS. The objective of this paper is to (a) demonstrate the power of manipulating the amount of non-linearity in networks to create hardware-efficient models, and (b) highlight the effectiveness of a new search space that comes with its own lightweight search algorithm towards building accelerator hardware-aware networks. As such, unlike the majority of the NAS literature, we do *not* focus on building the most effective search algorithm. That is, our lightweight search technique is limited only to the proposed blocks within our new search space, is often semi- (and not fully-) automatic, and does *not* search over important factors like number of channels, number of blocks, kernel sizes, expansion ratios, etc. Nevertheless, we demonstrate that our search space and preliminary algorithm result in highly accurate models that perform extremely well on NPUs. Hence, the scope of this work is to design only the new search space that can result in more hardware-aware models. Integrating this search space into a full-blown NAS is left as a future work.

We make the following **key contributions** in this work:

1. We propose Restructurable Activation Networks (RANs), a new paradigm that improves hardware efficiency of deep networks by manipulating the amount of non-linearity in the models.

2. We first create RAN-explicit (RAN-e) models that rely on a new search space and result in high accuracy and significantly improved accelerator hardware utilization without increasing MACs. We then create RAN-implicit (RAN-i) models that scale existing base models like ConvNexts in a training-free way to satisfy certain #MACs/#parameters. We also present an initial attempt at co-designing a restructurable block with its own activation function for ConvNext networks.

3. While RAN-e results in explicit changes in model structure with direct non-linear unit manipulation, RAN-i modifies the topology (depth/width) of the base model and, hence, implicitly also changes the amount of non-linearity. Towards this implicit non-linearity manipulation, we theoretically prove the link between the topological metric in [7] and expressivity of deep networks like ResNets and ConvNexts.

4. Finally, we achieve state-of-the-art results on ImageNet at several #MACs/#parameters scales and for multiple types of hardware ranging from micro-NPUs to datacenter CPUs. RAN-e leads to $1.5\times$ higher FPS than EfficientNet-Lite-B0 on an Arm micro-NPU with a similar accuracy. Also, RAN-i outperform ConvNexts by nearly $2\times$ fewer MACs with a minor drop in accuracy ($\sim 0.2\%$). We also achieve up to 40% higher
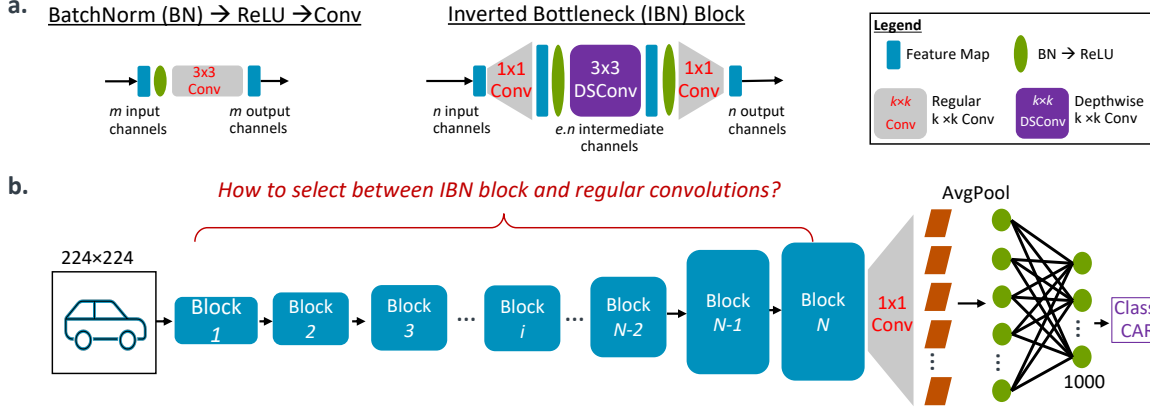
Figure 1. (a) Two blocks: BatchNorm (BN)→ReLU→Conv and a standard Inverted Bottleneck (IBN) block. Even if the #MACs and #parameters are similar between these two blocks, the IBN block has many more non-linear units than BN→ReLU→Conv. Indeed, the IBN block is much more expressive than a regular convolution layer which is also clear from the high accuracy achieved by the IBN-based models. (b) We formulate the problem of how much non-linearity we can remove from a network as a search between the blocks in (a).

FPS than ConvNext on an Arm-based datacenter CPU. When used as backbones in object detection, RAN-i achieve a similar or higher mAP with 33% higher FPS on datacenter CPUs compared to ConvNexts.

The paper is organized as follows. The RAN-e models are proposed in Section 2 along with their results. Section 3 proposes the RAN-i model scaling and shows its effectiveness. Section 4 demonstrates an initial attempt towards co-designing the restructurable blocks with a new activation function. After some discussion on future directions in Section 5, we review the related work in Section 6. Finally, we conclude the paper in Section 7.

2. RAN-explicit and New Search Space

In this section, we propose RAN-explicit, a new class of models whose architecture can be restructured by manipulating the amount of non-linearity in the network. We accomplish this by proposing a search space that contains new blocks. We start by formulating the problem below.

2.1. Problem Formulation

How much non-linearity can we remove from a deep network without losing significant accuracy? To address this question, we first create a setup that can allow us to systematically experiment with the amount of non-linearity in a given model. To this end, consider the two standard blocks shown in Fig. 1(a): BN→ReLU→Conv and IBN. Assuming both blocks receive a feature map of same height and width ($H \times W$), the number of MACs for BN→ReLU→Conv = $H \times W \times 3 \times 3 \times m \times m = 9m^2HW$. For the IBN blocks, ignoring the MACs in depthwise layers[2] and assuming the

expansion ratio $e = 6$ (similar to [45, 51]), the number of MACs for IBN = $H \times W \times [1 \times 1 \times n \times 6n + 1 \times 1 \times 6n \times n] = 12n^2HW$. With a simple calculation, it is easy to see that if $m = (2/\sqrt{3})n \approx 1.155n$, #MACs/#parameters for BN→ReLU→Conv and IBNs are equal.

Even if a regular BN→ReLU→Conv layer has similar #MACs/#parameters as an IBN block, it is well known that IBN blocks achieve significantly higher accuracy [45]. This is because the IBN blocks have a much higher *expressivity* than the regular convolution layers due to a large number of non-linear units in IBN [40, 42]. Specifically, for our example above (see Fig. 1(a)), the total number of non-linear ReLU units in BN→ReLU→Conv = m, whereas the IBN has $6n+6n = 12n$ ReLU units. Hence, even if $m = 1.155n$ (so that both IBN and regular convolution layer have similar #MACs/#parameters), the IBN has more than $10\times$ higher number of non-linear units than BN→ReLU→Conv. This results in better expressivity of IBN-based models. Note that, for AI accelerators, a regular convolution executes much faster than an IBN block due to a significantly better hardware utilization [58, 63], especially if they have similar #MACs/#parameters. Therefore, it is still preferable to have some regular convolution layers in the model.

Based on the above observations, we formulate the task of how much non-linearity can be removed from a network as a novel search problem that chooses between low non-linearity blocks like regular convolutions (e.g., BN→ReLU→Conv) and high non-linearity blocks like IBNs (see Fig. 1(b)). Hence, our search problem is:

$$\min_{\boldsymbol{\theta}, \boldsymbol{\alpha}} \mathcal{L}(y, f(\boldsymbol{\theta}, \boldsymbol{\alpha})) + \lambda ||\mathbb{I}(\boldsymbol{\alpha})||_0, \tag{1}$$

where, $\mathcal{L}$ is the cross-entropy loss, y is the true label for the given classification task, f is the function represented by the *SuperNet* created using the search space, $\boldsymbol{\theta}$ are the

---

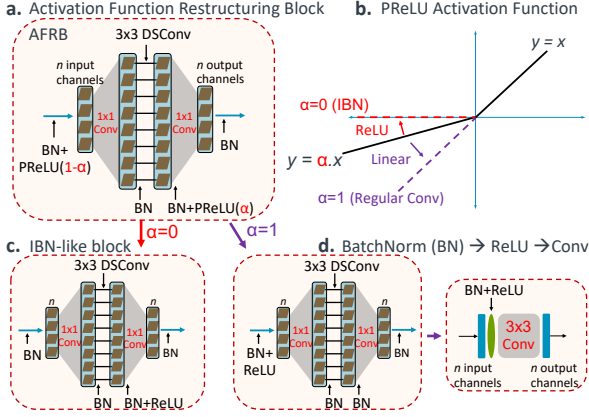[2]Depthwise layer has far fewer MACs than pointwise 1×1 layers.

Figure 2. (a) Proposed Activation Function Restructuring Block (AFRB) consists of an IBN structure with PReLU activation functions. The first PReLU uses $(1 - \alpha)$ and the second one uses α parameter (same α is shared between the two PReLUs). (b) PReLU activation function: if $\alpha = 0$, it becomes ReLU, and if $\alpha = 1$, the function becomes linear (i.e., no activation). (c, d) For $\alpha = 0$ ($\alpha = 1$), AFRB becomes an IBN-like (a BN→ReLU→Conv) block. Thus, AFRB can restructure an IBN block into a regular convolution by removing non-linear units from the network. This is accomplished using a single scalar trainable α parameter.

weight parameters of the SuperNet, and $\boldsymbol{\alpha}$ are the parameters that select between low and high non-linearity blocks inside the SuperNet (e.g., BN→ReLU→Conv vs. IBNs). The indicator function $\mathbb{I}$ produces a vector whose elements are 1 if a non-linear unit is present, and 0 otherwise; thus, the l_0 norm of this indicator function quantifies the number of non-linear units in the model. Finally, λ is a hyperparameter that controls the contribution of the second loss. Therefore, the goal of the above problem is to minimize the cross-entropy loss during training and also reduce the total number of non-linear units in the network.

A standard way to solve problem (1) can be using any NAS algorithm including differentiable NAS methods like DARTS [37]. Towards this, a traditional SuperNet to select between IBN and regular convolution layer would involve putting IBNs and regular convolutions as branches and then an α parameter can select among those options. However, this SuperNet is computationally expensive and requires high GPU memory due to multiple branches. To overcome these issues, we introduce new *Activation Function Restructuring Blocks* (AFRB) that do not require multiple branches (and, hence, are low cost), and enable us to study the non-linearity problem in a very systematic way.

2.2. New Activation Function Restructuring Blocks

In this section, we propose our Activation Function Restructuring Blocks (AFRB) that automatically restructure IBNs into small, regular 3×3 convolutions. Fig. 2(a) illustrates the proposed AFRB that consists of an IBN-like

structure with two PReLUs. For simplicity, we remove the non-linear activation function after the first 1×1 (expansion) layer. Also, the first PReLU appears before the first 1×1 convolution and uses $(1 - \alpha)$ as its trainable parameter. On the other hand, the second PReLU uses (α) parameter and appears after the depthwise separable convolution (DSConv). Both PReLUs share the same α value.

The PReLU activation function is very interesting because of its trainable α parameter. Specifically, as shown in Fig. 2(b), if $\alpha = 0$, it behaves as a ReLU and if $\alpha = 1$, it becomes linear ($y = x$), i.e., no activation. Therefore, if we control the trainable parameter α for the PReLU, we can use it to systematically remove the non-linear units from a network and analyze its impact on accuracy. In other words, using AFRB in our search space allows us to *prune* out the non-linear units from a network in a fully trainable way.

Let us now examine how AFRB enables an explicit restructuring of the blocks. In Fig. 2(a), if $\alpha = 0$, the first PReLU becomes linear and gets removed, while the second PReLU becomes a ReLU that appears after the DSConv layer. Clearly, this resembles an IBN except for the missing ReLU after the expansion layer (see Fig. 2(c)); we can easily bring back that ReLU once the search phase is over, i.e., during the final training of the searched subnetwork. In contrast, if $\alpha = 1$ in Fig. 2(a), the first PReLU becomes a ReLU and the second PReLU becomes linear (see Fig. 2(d)). In this case, after the very first ReLU, there are no more non-linear units in the block. That is, the three layers (1×1 expansion layer, DSConv, 1×1 projection layer, along with their BatchNorms) are all linear operations. We know from prior art [8, 22], these linear layers can be *analytically* collapsed into a single regular convolution. Hence, for $\alpha = 1$, the AFRB block restructures into a BN→ReLU→Conv block.

Therefore, AFRB is a unique building block that promotes an *explicit trainable restructuring* of entire operations and directly chooses between an IBN block or a BN→ReLU→Conv. Hence, an AFRB-based search space encourages the discovery of completely new kinds of deep networks. We next discuss the direct implications of our block from a hardware perspective.

Hardware Advantages. The restructuring of AFRB from IBN to BN→ReLU→Conv reduces the computational and memory costs and also improves the hardware utilization on AI accelerators. Specifically, as mentioned earlier, the #MACs for IBN in Fig. 2(c) = $12n^2HW$, whereas that for BN→ReLU→Conv in Fig. 2(d) = $9n^2HW$. Hence, the restructuring directly results in 25% savings in #MACs/#parameters. Since the regular convolutions do not have utilization issues on AI accelerators, these lower MACs execute at much higher rate on the accelerator, thereby significantly boosting the hardware performance.
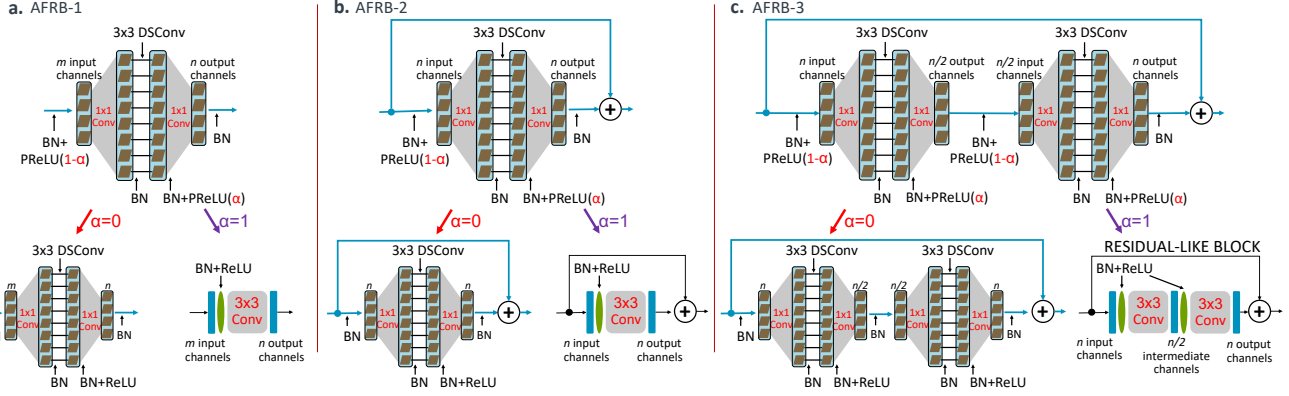
Figure 3. Proposed search space: (a) AFRB-1 is used when residual is not present, i.e, if there is a feature map downsampling with stride>1 or if #input and #output channels are different. (b) AFRB-2 is used when a residual can be present (#input channels = #output channels and stride=1). (c) AFRB-3 can collapse into a residual like block (with half intermediate channels, unlike standard residual blocks). This makes the #MACs/#parameters for AFRB-3 same as AFRB-1 but with additional expressivity (i.e., more non-linear units than AFRB-1).

2.3. Proposed Hardware-Aware Search Space

We now exploit AFRB to create a novel search space that will be used to generate accelerator hardware-aware models. Fig. 3 shows different blocks used to create our Super-Net. As evident, we use three kinds of blocks: (1) AFRB-1 is used when #input channels are different from #output channels or if there is a stride to downsample feature maps; (2) AFRB-2 is used when a valid residual skip connection can be added to the block (i.e., #input channels = #output channels and stride = 1); (3) AFRB-3 can collapse into a residual-like block if $\alpha = 1$. Also, AFRB-3 reduces the number of intermediate channels to half the input channels. The idea here is to increase the number of non-linear units while still keeping #MACs/#parameters the same as AFRB-1. A search over the SuperNet containing the above blocks results in accelerator hardware-aware deep networks.

2.4. Lightweight Search Algorithm

Let us now explain how to search using our proposed blocks. We first create a SuperNet using AFRBs where each block i (see Fig. 1(b)) has its own α_i parameter. Starting with problem (1), it is easy to see that the trainable α_i parameters make it possible to pick between low non-linearity blocks (BN→ReLU→Conv) and high non-linearity blocks (IBN). The search problem now becomes:

$$\min_{\boldsymbol{\theta},\boldsymbol{\alpha}} \mathcal{L}(y, f(\boldsymbol{\theta},\boldsymbol{\alpha})) + \lambda||\mathbb{I}(\boldsymbol{\alpha} \in \{0,1\})||_0, \qquad (2)$$

where, the indicator function $\mathbb{I}$ now takes non-zero values only if $\alpha_i = 0$ or $\alpha_i = 1$ for each block i in the network. In practice, we relax this problem further by trying to make $\alpha_i = 1$ where possible and not putting any constraint to make $\alpha_i = 0$. That is, instead of searching between linear (no activation, $\alpha_i = 1$) or ReLU ($\alpha_i = 0$) in problem (2),

we use the following loss to perform a binary search between linear ($\alpha_i = 1$) or non-linear ($\alpha_i \neq 1$):

$$\min_{\boldsymbol{\theta},\boldsymbol{\alpha}} \mathcal{L}(y, f(\boldsymbol{\theta},\boldsymbol{\alpha})) + \lambda||\boldsymbol{\alpha} - \mathbf{1}||_2^2, \qquad (3)$$

where, $\mathbf{1}$ is a vector of all 1's. The above objective function aims to minimize the cross entropy loss while making as many $\alpha_i = 1$ as possible. The blocks where $\alpha_i \neq 1$ are assumed to be high non-linearity blocks like IBNs. Therefore, minimizing problem (3) can directly restructure some of the IBNs into regular convolutions.

We make a clear distinction between our search phase and final finetuning phases of the initial searched model (in terms of final minor changes to the model architecture). Again, since our search problem only focuses on whether we can remove non-linear units from each block or not, we do not conduct a full-blown NAS in this work. Specifically, number of channels, expansion ratios, number of blocks, etc., are all decided manually when designing the SuperNet. *Our goal is to start with this given SuperNet and make it accelerator-hardware-aware without losing significant accuracy* and *not to build the most effective search algorithm over traditional factors like number of channels, expansion ratios, number of blocks, etc.* To this end, our search will be *semi-automatic*, and we will show how each design decision affects the quality of the model. Nevertheless, we will demonstrate the effectiveness of the proposed search space and our lightweight search algorithm in creating highly efficient deep networks. In the next section, we present this semi-automatic process and show the results on ImageNet.

2.5. RAN-e: ImageNet Evaluation

We now exploit our proposed search space to create state-of-the-art networks for the ImageNet image classification task. Fig. 4(a) shows the structure of our SuperNet
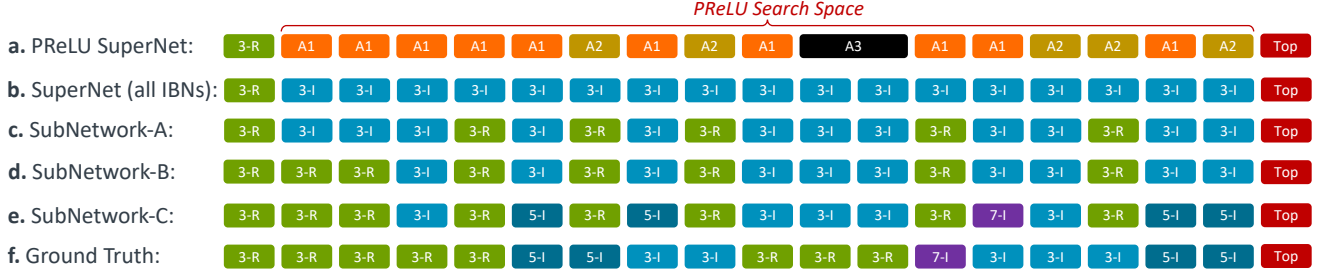
Figure 4. (a) **PReLU SuperNet** shows the SuperNet structure in terms of AFRB blocks. A1, A2, A3 stand for AFRB-1, AFRB-2, and AFRB-3, respectively. The search is conducted over these blocks to see which of them can be restructured into regular convolutions. (b) **SuperNet (all IBNs)** consists of 1 regular stem convolution and 17 IBN blocks with 3×3 kernel sizes ("3" in 3-R or 3-I denotes kernel size). (c) **SubNetwork-A:** The PReLU-based *accuracy-driven* search reveals that 5 out of 17 blocks can be made regular convolutions without significant accuracy loss. (d) **SubNetwork-B:** The first two IBNs are very heavy in MACs due to large feature maps and expansion ratio $e = 6$; these blocks are a very natural choice to be collapsed to regular convolutions (on top of SubNetwork-A) to improve hardware performance with some accuracy drop. (e) **SubNetwork-C:** We now modify the kernel sizes on some of the IBNs (since the depthwise MACs are still low) to recover the lost accuracy. (f) **Ground Truth:** Manual model designed on the SuperNet performs even better than SubNetwork-C (same accuracy and even better hardware performance), thus, highlighting the importance of the proposed search space.

Table 1. Results on ImageNet (100 epoch training) for the SuperNet and different RAN-e networks (structures shown in Fig. 4). These models are suitable for microcontroller- and mobile-scale AI accelerators. Green/Red indicate improvements or worsening of various metrics compared to the original EfficientNet-Lite-B0 (with ReLU6) [21]. Normalized throughput is Frames Per Second (FPS) normalized w.r.t. EfficientNet-Lite-B0 estimated for Arm Ethos-U55 (M-class) and Arm Ethos-U65 (A-class) microNPUs.

100 epoch training	#Parameters (in Millions)	#MACs (in Millions)	Top-1 Accuracy	Normalized Throughput (FPS)	
				Ethos-U55 M-class systems	Ethos-U65 A-class systems
EfficientNet-Lite-B0 [21] (original with ReLU6)	4.7M	**385M**	72.81%	1×	1×
EfficientNet-Lite-B0 (with H-Swish)	4.7M	**385M**	**73.82**% (**+1.01**%)	1×	1×
SuperNet (all IBNs)	4.7M	590M	74.24% (+1.43%)	0.86×	0.87×
SubNetwork-W (reduced width SuperNet, all IBNs)	**4.13M**	488M	73.75% (+0.94%)	0.91×	0.90×
RAN-e SubNetwork-A (PReLU search, trained with H-Swish)	4.6M	561M	73.26% (+0.45%)	0.88×	0.91×
RAN-e SubNetwork-B (SubNetwork-A + first 2 IBNs collapsed)	4.6M	482M	72.63% (−0.18%)	1.10×	1.34×
RAN-e SubNetwork-C (SubNetwork-B + new kernel sizes)	4.7M	488M	**73.13**% (**+0.32**%)	**1.06×**	**1.28×**
RAN-e Ground Truth (manual model within same SuperNet)	**4.5M**	**433M**	**73.04**% (**+0.23**%)	**1.16×**	**1.49×**

in terms of the location and types of AFRBs. The complete details of the SuperNet (e.g., strides, channel counts, expansion ratios, etc.) are shown in Table 5 in Appendix A. Note that, our SuperNet uses only 3×3 kernel sizes for all blocks (see green "3-R" blocks for regular convolutions or blue "3-I" blocks for IBNs in Fig. 4(b)). As we go through our semi-automatic search process, we will make a number of simple finetuning steps (including kernel sizes) to arrive at the final architecture. More details about the training setup for RAN-e are given in Appendix B.

Table 1 shows the #parameters, #MACs, accuracy, and normalized throughput (FPS) of SuperNet and various RAN-e models w.r.t. EfficientNet-Lite-B0 [21]. The normalized throughput is obtained using performance estimators for Arm Ethos-U55 [1] (with M-class system configuration) and Arm Ethos-U65 [2] (with A-class system configuration) microNPUs[3]. All models in Table 1 use Hard-

Swish (H-Swish) [29] activation function unless indicated otherwise. Finally, Table 1 results are only for 100 epoch training. We will present 350 epoch training results later.

As evident from Table 1, our SuperNet (assuming all blocks are selected as IBNs, see Fig. 4(b)) achieves 74.24% top-1 accuracy on ImageNet in 100 epochs with 4.7M parameters and 590M MACs. Our SuperNet is slower than EfficientNet-Lite-B0, e.g., 0.87× the FPS of EfficientNet-Lite-B0 on Ethos-U65. This is because the SuperNet incurs a significantly heavier MAC cost (590M vs. 385M) while still only using IBNs in its architecture which present hardware utilization issues for NPUs at some of the layers. Staying within the same IBN-based search space, we further created SubNetwork-W which is simply a reduced width version of the SuperNet (width multiplier = 0.91×, see Table 1). Clearly, this model achieves a similar accuracy as EfficientNet-Lite-B0-H-Swish (around 73.8%). Again, since this model also uses IBNs only, even with 17% lower MACs than the SuperNet (488M vs. 590M), SubNetwork-W achieves only 3% boost in normalized FPS (0.9× vs.

---

[3]The performance estimator for Ethos-U55 is available at: https://git.mlplatform.org/ml/ethos-u/ethos-u-vela.git/about/. The performance estimator for Ethos-U65 is proprietary.

$0.87\times$) on Ethos-U65. Therefore, hardware utilization issues make it very difficult to improve FPS even if MACs are greatly reduced. Hence, our objective is to make the SuperNet accelerator-hardware-aware using our proposed AFRB blocks and the semi-automatic search algorithm.

2.5.1 The Initial Search

We now perform the search, i.e., problem (3) on the PReLU SuperNet shown in Fig. 4(a). The λ hyperparameter in problem (3) controls the tradeoff between accuracy and the amount of non-linearity pruning. We performed a search on $\lambda \in (6 \times 10^{-4}, 2 \times 10^{-3})$ and found that $\lambda = 1 \times 10^{-3}$ results in minimal accuracy drop ($\sim 1\%$) over the SuperNet while collapsing 5 out of 17 IBN blocks into regular convolutions. We call this model SubNetwork-A (see Fig. 4(c)). Recall that, our search only chooses between $\alpha = 1$ (linear) and $\alpha \neq 1$ (non-linear). The linear case automatically restructures to a regular convolution (see Fig. 3), whereas we set all the high non-linearity $\alpha \neq 1$ blocks as IBNs. Since the first activation function is missing in AFRB blocks (see Fig. 3), we bring it back for all $\alpha \neq 1$ blocks and retrain the models from scratch once the search phase is complete[4]. Table 1 shows that SubNetwork-A achieves 1% lower accuracy than the all-IBN SuperNet, reduced the #MACs by 29M, and slightly improves the normalized FPS from 0.87 to 0.91 on Ethos-U65.

Clearly, the performance is not significantly better than the SuperNet. This is because we have only conducted an *accuracy-driven* search and the total number of MACs is still very high compared to the SuperNet (561M vs. 590M). Hence, we next finetune the SubNetwork-A architecture obtained with our initial PReLU search.

2.5.2 Finetuning Stage 1: Collapse High MAC Blocks

It is easy to see that our search algorithm chooses to *not* restructure some MAC-heavy blocks, e.g., the first two IBNs in Fig. 4(c). This decision by our PReLU search algorithm makes intuitive sense. Having powerful initial blocks can have a significant impact on accuracy. Since, unlike most hardware-aware NAS works [10, 51], we have *not* done any MAC- or latency-aware search, the objective in problem (3) attempts to maximize only the accuracy without trying to reduce MACs. This is why it chooses to keep the first two blocks as IBNs even if they are very expensive in MACs. Hence, as our first finetuning stage, we directly restructure the first two IBN blocks – two of the highest MAC

---

[4]SubNetwork-A (and others) use H-Swish everywhere during retraining and not PReLU. The PReLU is used only during the search phase. In practice, for the automatically restructuring ($\alpha = 1$, linear) case, the PReLU search yields α values close to 1 and not exactly 1 when we solve problem (3), e.g., α's are within a reasonable boundary like $[0.8, 1.3]$. When we train the searched SubNetwork-A from scratch, we replace the AFRBs with $\alpha \in [0.8, 1.3]$ with regular convolutions.

blocks due to large feature map sizes and high expansion ratios ($e = 6$) – into regular convolutions. This results in SubNetwork-B model (see Fig. 4(d)).

Table 1 demonstrates that SubNetwork-B immediately reduces about 80M more MACs over SubNetwork-A and collapsing these two IBN blocks leads to a top-1 accuracy of 72.63% (about 0.6% lower than SubNetwork-A). However, because these newly restructured regular convolutions execute on the AI acclerators without hardware utilization issues, the normalized FPS greatly increases from 0.91 to 1.34 on Ethos-U65 (about $1.47\times$ increase compared to SubNetwork-A). This also highlights the power of our hardware-aware search space: *Even with similar MACs as SubNetwork-W which uses only the traditional IBNs, our model has $1.49\times$ better FPS than SubNetwork-W.*

2.5.3 Finetuning Stage 2: Change Kernel Sizes

Inevitably, the accuracy drops when we collapse the first two IBNs in the last finetuning stage. To make up for this lost accuracy, recall that it is very common to use kernel sizes larger than 3×3 in modern NAS search spaces (e.g., EfficientNet-Lite also uses 5×5 kernel sizes, etc.). Therefore, in this finetuning stage, we increase kernel sizes to 5×5 and 7×7 for a few IBN blocks. Since the depthwise convolutions in IBNs still incur a low MAC cost, this does not increase the overall MACs significantly. The new updated model is called SubNetwork-C and its structure is shown in Fig. 4(e). As evident from Table 1, SubNetwork-C recovers nearly all of the accuracy to the level of SubNetwork-A (73.13% vs. 73.26%) while still improving normalized FPS from 0.91 to 1.28. SubNetwork-C is the last stage of our semi-automatic search process and clearly demonstrates a systematic way to go from a completely *hardware-unaware* SuperNet to a very *hardware-friendly* SubNetwork. In summary, *compared to EfficientNet-Lite-B0 (original, with ReLU6) trained on our 100 epoch setup, we achieve slightly better accuracy and nearly $1.28\times$ higher FPS.*

2.5.4 Is this the best we can do?

So far, the process has been semi-automatic with most of the design decisions being very straightforward after the initial search. We now evaluate if we can do any better with a manual design, within the same search space and SuperNet. To this end, we created a model shown in Fig. 4(f) called the Ground Truth network. Table 1 demonstrates that *RAN-e Ground Truth achieves slightly better accuracy than EfficientNet-Lite-B0 (ReLU6) with nearly $1.5\times$ higher FPS which is even better than that for the SubNetwork-C.*

We call this manual model the Ground Truth network because ideally we expect the search algorithm to discover this or a better network on its own. However, since our semi-automatic algorithm produced a sub-optimal network (even

Table 2. ImageNet results. [†]RepVGG and ResNet accuracies are taken directly from [15] which were trained to 120 epochs. RAN-e accuracies for 100 epoch experiments (Table 1) are higher than ResNet and RepVGG below. [‡]EfficientNet-Lite-B0 as reported by [21] using a different training recipe. The remaining RAN-e and EfficientNet-Lite-B0 are trained to 350 epochs on our setup.

| | Params | MACs | Top-1 | Normalized FPS | |
				Ethos-U55 M-class	Ethos-U65 A-class
ResNet-18[†] [15,24]	11.7M	1.8B	71.2%	0.46×	0.48×
RepVGG-A0[†] [15]	8.3M	1.4B	72.4%	0.72×	0.69×
EffNet-Lite-B0-R6[‡] [21]	4.7M	**385M**	75.1%	1×	1×
EffNet-Lite-B0-R6	4.7M	**385M**	74.4%	1×	1×
EffNet-Lite-B0-HS	4.7M	**385M**	**75.6%**	1×	1×
RAN-e-C (Ours)	4.7M	488M	**74.6%**	1.06×	1.28×
RAN-e-GT (Ours)	**4.5M**	433M	**74.6%**	1.16×	1.49×

if it is $1.28\times$ better than a very strong baseline), it highlights the current limitations of the search process, i.e., lack of a full-blown NAS and no MAC-aware losses. It is possible that with a complete NAS (e.g., channel counts, expansion ratios, kernel sizes, strides, etc.) on the AFRB-based search space, along with explicit MAC-driven losses, the search algorithm may produce an even better network than the RAN-e Ground Truth model. Integrating AFRBs into a full NAS and MAC-constraints is left as a future work.

2.5.5 Comparison against reference models

The original EfficientNet-Lite-B0 (ReLU6) is used only as a reference in the previous sections to show that AFRB-based search space can come up with competitive models. For a fair comparison with our models (that use H-Swish), Table 1 also presents the accuracy for EfficientNet-Lite-B0 (H-Swish) trained on our setup. As evident, our SubNetwork-C (RAN-e-C) and Ground Truth (RAN-e-GT) networks are within 1% accuracy of EfficientNet-Lite-B0 (H-Swish) while improving FPS by up to $1.28\times$-$1.5\times$. Note that, EfficientNet-Lite-B0 was obtained using a full-blown NAS that searched over number of blocks, channel counts, expansion ratios, and kernel sizes. Again, because our search is *not* a complete NAS, better networks in AFRB search space could have been potentially obtained if we had also searched over width, depth, expansion ratios, etc. More interestingly, the FPS gain in RAN-e is despite the fact that both SubNetwork-C and Ground Truth networks require more MACs than EfficientNet-Lite-B0. This is perhaps a surprising result because most of the prior art tries to minimize the #MACs/#parameters to obtain efficient models. Therefore, a *hardware-aware search space* can significantly boost performance even with slightly higher MACs.

Next, Table 2 shows comparisons against a few existing baselines that also rely on regular convolutions, e.g., ResNets [24] and RepVGG [15]. We also train our RAN-e networks and EfficientNet-Lite-B0 (ReLU6 and H-Swish)

to 350 epochs and report their accuracy. As evident from Table 2, while models like ResNet-18 and RepVGG-A0 use only 3×3 convolutions, they result in extremely compute intensive models. Specifically, compared to RAN-e-GT (Ground Truth), ResNet-18 and RepVGG require $4.15\times$ and $3.23\times$ more MACs, respectively. The accuracy for these models reported in Table 2 is taken directly from [15] which trains them to 120 epochs only (without advanced data augmentation techniques like Mixup [62]). We also do not use Mixup in our experiments and our 100 epoch accuracies in Table 1 already exceed ResNet-18 and RepVGG accuracies. Furthermore, the normalized FPS on Ethos-U55 and Ethos-U65 clearly demonstrate the superiority of RAN-e-GT compared to ResNet-18 and RepVGG (we achieve up to $3.1\times$ and $2.1\times$ higher FPS, respectively). Note that, the improvements for M-class Ethos-U55 are significant but not as much as the A-class Ethos-U65 because M-class systems are at tiny microcontroller-scale and are limited in memory.

Finally, we have also reproduced EfficientNet-Lite-B0 (ReLU6 (R6) and H-Swish (HS)) accuracies on our setup. Again, while we are 1% away in top-1 accuracy compared to EfficientNet-Lite-B0-HS, we achieve about $1.5\times$ higher FPS. On the other hand, compared to the original EfficientNet-Lite-B0-R6 (trained on our 350 epoch setup), our proposed RAN-e networks achieve about 0.2% higher top-1 accuracy on ImageNet. Of note, with a different training recipe, [21] reports an accuracy of 75.1% for EffNet-Lite-B0-R6 (compared to 74.4% for our setup). Therefore, the accuracy for our models may improve even more if the training recipe is optimized further.

Other Remarks. We emphasize that the quality of Super-Net matters in terms of compute costs of different SubNetworks. Specifically, even though both SubNetwork-W and EfficientNet-Lite-B0 (H-Swish) are based on IBNs-only search space and achieve a similar accuracy, SubNetwork-W requires 100M more MACs than EfficientNet-Lite-B0 (see Table 1). Hence, if we had a better, more efficient SuperNet (e.g., in terms of number of blocks, channel counts, expansion ratios, stride locations, etc.), our results could be improved further. This also highlights that (1) our new hardware-aware search space will likely offer best results when used in conjunction with full NAS, and (2) perhaps newer, cheap activation functions can also be proposed in future that are specifically designed to work with AFRBs to obtain even higher accuracy (see discussion in Section 4).

Is it possible to manipulate the non-linearities in deep networks to create accelerator-hardware-aware models? We have demonstrated that an AFRB-based search space enables us to accomplish this goal. The discussion so far completes the proof-of-concept of our novel search space and shows that RAN-explicit is a new direction to achieve hardware-awareness. Next, we propose the implicit restructuring of non-linearities to reduce compute cost of models.

3. RAN-implicit and Training-Free Scaling

Given a good base model, can changing the amount of non-linearity in a network allow us to scale it up or down in a *training-free* way to obtain highly accurate deep networks that satisfy specific #MAC/#parameter constraints? To address this question, we first revisit recent literature [7] that links topological characteristics of deep networks (i.e., structural properties such as presence of skip connections, etc.) with their gradient propagation and model performance. We also briefly review the literature that studies expressivity of deep neural networks [40, 42, 46].

3.1. Preliminaries

We start by discussing the topological metric and the main theoretical result in [7].

Definition 1 (NN-Mass [7]). *NN-Mass is defined as the sum (over all N_b blocks) of product of total #input channels (i_b) at all layers in a block and Cell-Density (ρ_b).*

$$m = \sum_{b=1}^{N_b} i_b \times \rho_b, \quad \rho_b = \frac{\text{\#Actual Skip Connections}}{\text{Total Possible Skip Connections}} \tag{4}$$

For DenseNet-type models[5], NN-Mass was shown to be related to the average degree, i.e., average number of connections for each channel in the network – both short-range, i.e., layer-by-layer connections, and long-range, i.e., skip connections going across multiple layers. Specifically, for a network with width w at all blocks, [7] proved that the average degree $\hat{k} = w + m/2$. Intuitively, [7] argues that if we have two networks with similar average connectivity (e.g., same width and NN-Mass), then the amount of information flowing through them is constrained similarly. Therefore, such topological constraints can also have a profound impact on how learning happens in different networks.

Proposition 1 (NN-Mass vs. Dynamical Isometry [7]). *Given a deep linear DenseNet-type MLP network, the Layerwise Dynamical Isometry (LDI) is defined as the mean singular value of layerwise Jacobians at initialization. Then, the LDI is bounded as follows:*

$$\sqrt{q\hat{k}} - \sqrt{qw} \leq \mathbb{E}[\sigma] \leq \sqrt{q\hat{k}} + \sqrt{qw}, \tag{5}$$

where, $\hat{k} = w + m/2$, q is the initialization variance. If $q = 1/\hat{k}$, then the mean singular value of layerwise Jacobians (LDI) is bounded near 1, i.e., the gradients flow through the network without amplification or attenuation [31].

Proposition 1 shows that models with similar width and NN-Mass have similar gradient flow properties and,

thus, training convergence (irrespective of their depths, #parameters, and #MACs) since their mean singular value is bounded in a similar way. Therefore, models with similar width and NN-Mass can achieve a similar accuracy even if they have highly different #parameters/#MACs/#layers. Extensive empirical results were presented in [7] to demonstrate the effectiveness of NN-Mass.

The existing gradient flow-based theory in [7] shows the relationship between model topology and training convergence but does *not* say anything about expressivity of deep networks. Understanding the expressivity of deep networks is just as important as understanding their gradient properties [40, 42, 46]. One way to quantify expressivity of deep networks is to count the number of linear regions that a function represents. These definitions are given below:

Definition 2 (Linear Regions [40]). *Given a function f, a linear region is a maximal connected set of inputs x where f is linear.*

The number of linear regions can be found by counting the number of unique activation patterns, e.g., how different ReLU units are activating for different inputs [46]. That is, counting the number of unique activation patterns quantifies how many different linear regions are contained in the function represented by the given deep network. Therefore, this can be directly used as a measure of expressivity of the model. Montúfar *et al.* [40] provide an aysmptotic lower bound on maximal number of linear regions as follows:
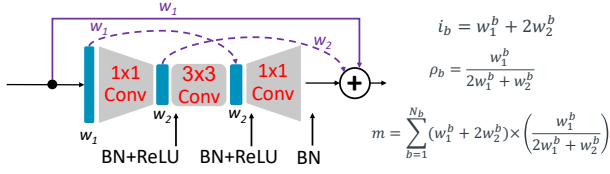
Proposition 2 (Linear Regions for ReLU Networks [40]). *Given an input $x \in \mathbb{R}^{n_0}$ and a rectifier (ReLU) deep network function $f : \mathbb{R}^{n_0} \to \mathbb{R}^n$ with n_0 input neurons, L layers with n neurons each ($n \geq n_0$), f can compute functions with $\Omega((n/n_0)^{(L-1)n_0} \times n^{n_0})$ linear regions.*

In the next section, we demonstrate for ResNet- or ConvNext-type networks that NN-Mass is related to the expressivity of deep networks. This is particularly important because while [7] theoretically analyzes DenseNet-type networks, it neither discusses *why* NN-Mass works for ResNets and other models with residual additions, nor provides any connection with the expressivity of deep networks.

3.2. Expressivity vs. NN-Mass

Bhardwaj *et al.* [7] empirically derive the NN-Mass expression for ResNet-type networks as shown in Fig. 5(a). We also adapt the NN-Mass for ConvNexts (see Fig. 5(b)). Note that, for models with a very uniformly repeating structure (e.g., residual blocks or ConvNext blocks), the intermediate width (w_2) is generally related to the input channels for the block ($w1$) as $w_2 = e \cdot w_1$, where e is the expansion or shrinking factor and is generally the same for all blocks in the network. Therefore, for this special class of networks, we have the following result:

---

[5]In [7], DenseNet-type networks contain concatenation-type skip connections and the density of skip connections can be varied.

a. Bottleneck Residual Block

$i_b = w_1^b + 2w_2^b$

$\rho_b = \dfrac{w_1^b}{2w_1^b + w_2^b}$

$m = \displaystyle\sum_{b=1}^{N_b} (w_1^b + 2w_2^b) \times \left(\dfrac{w_1^b}{2w_1^b + w_2^b} \right)$

b. ConvNext Block

$i_b = 2w_1^b + w_2^b$

$\rho_b = \dfrac{w_1^b}{3w_1^b} = \dfrac{1}{3}$

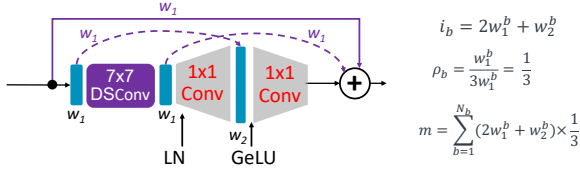$m = \displaystyle\sum_{b=1}^{N_b} (2w_1^b + w_2^b) \times \dfrac{1}{3}$

Figure 5. (a) NN-Mass calculation for ResNet bottleneck block. (b) NN-Mass calculation for ConvNext block. In both cases, the solid skip connection is the one actually present in the blocks. The dotted skip connections are shown as *possible* skip connections in the blocks as required by the cell-density (ρ_b) definition. Since the skip connections for both blocks involve channel-wise additions, the solid skip connection supplies w_1 links, i.e., the skip connection carries information from all w_1 input channels (at first layer); similar ideas apply to possible (dotted) skip connections.

Proposition 3 (NN-Mass vs. Linear Regions). *For models with residual additions and a uniform structure (e.g., ResNets/ConvNexts), NN-Mass is proportional to the total number of non-linear units in the network. That is, if the total number of non-linear units in the network = $\mathcal{X}$, then $\mathcal{X} \propto m$ or $\mathcal{X} = k \times m$, where k is a constant. Then, upper bound on maximal number of linear regions = $2^{\mathcal{X}} = 2^{km}$.*

Proof. Assuming $w_2 = e \cdot w_1$ for all blocks, NN-Mass for ResNets (m_R) is given by (see Fig. 5(a)):

$$m_R = \sum_{b=1}^{N_b} (w_1^b(1+2e)) \left(\frac{w_1^b}{(2+e)w_1^b} \right) = \left(\frac{1+2e}{2+e} \right) \sum_{b=1}^{N_b} w_1^b \tag{6}$$

Total number of non-linear units for ResNets ($\mathcal{X}_R$) is:

$$\mathcal{X}_R = \sum_{b=1}^{N_b} 2w_2^b = \sum_{b=1}^{N_b} 2e \times w_1^b = 2e \sum_{b=1}^{N_b} w_1^b \tag{7}$$

Therefore, $\mathcal{X}_R \propto m_R$ or $\mathcal{X}_R = k_R \times m_R$, where $k_R = (2e(2+e))/(1+2e)$. A similar calculation for ConvNext class of networks reveals that the total number of non-linear units for ConvNexts ($\mathcal{X}_C$) is also proportional to its NN-Mass (m_C), i.e., $\mathcal{X}_C = k_C \times m_C$, where $k_C = (3e/(2+e))$.

As a result, for models with residual additions and uniform structure, NN-Mass is proportional to the total number of non-linear units in the network. Using [40], the maximal number of linear regions is bounded by $2^{\mathcal{X}} = 2^{km}$. Hence,

NN-Mass directly impacts the number of linear regions and, thus, the expressivity of this class of deep networks. □

Next, we prove one more result on the relationship between NN-Mass and expressivity. Specifically, NN-Mass has known limitations, e.g., depending on the difficulty of the given task, the models need to be deep enough and wide enough for NN-Mass to produce best results in practice[6]. Proofs in [7] explicitly assumed conditions on depth while deriving the results, so it is understood that the models need to be deep. However, it is unclear why the width also impacts the effectiveness of NN-Mass. We analyze this relationship between width and NN-Mass below.

Corollary 1 (NN-Mass matters less when width is low)**.** *Suppose we are given a deep network with n_0 input neurons, L layers with n neurons each and repeating residual addition skip connections. For such a network, NN-Mass $m = nL$. Then, the function represented by this model $f : \mathbb{R}^{n_0} \to \mathbb{R}^n$ can compute $\Omega((n/n_0)^{(m-n)\frac{n_0}{n}} \times n^{n_0})$ linear regions. When the width is low (e.g., when $n \to n_0$), model's expressivity is mostly determined by its width.*

Proof. Since all layers have the same width n, and because the network consists of repeating residual blocks (similar to Fig. 5(a)), the expansion/shrinking factor $e = 1$. Substituting this in Eq. (6), we obtain NN-Mass $m = \sum_{b=1}^{N_b} w_1^b = nL$. For the remaining proof, we simply substitute $L = m/n$ in Proposition 2. Then, it is easy to see that if $n \to n_0$, $n/n_0 \to 1$ and, thus, the term with NN-Mass does not contribute to the maximal number of linear regions. Instead, the bound is mostly dictated by the second term that depends only on n. This explains why NN-Mass depends on model width: When the width is low, NN-Mass matters less and the expressivity depends more on width. □

Therefore, model topology not only impacts the gradient properties as proved in [7], it also directly affects the expressive power of deep neural networks. Since computing metrics like NN-Mass does not require any training or even a single forward or backward pass, they can serve as excellent training-free methods to search for high quality models.

3.3. Training-Free Scaling with NN-Mass

We now exploit NN-Mass for training-free model scaling. Specifically, given a base model $\mathcal{M}$ and a few different hardware constraints $\mathcal{H} = \{H_1, H_2, \ldots, H_p\}$ (in terms of #MACs/#parameters), the problem is to scale the model up or down in a training-free fashion to find high quality models for all constraints. Once an appropriate model is found using this training-free *search*, it is trained to obtain the final model. Note that, all other conditions from the last section

---

[6]This was evident in some of the results presented in [7] (e.g., R^2 for relationship between accuracy and NN-Mass increases with increasing width, see Fig. 15 in Appendix H.4 of [7]). We saw similar patterns in our own experiments as well.
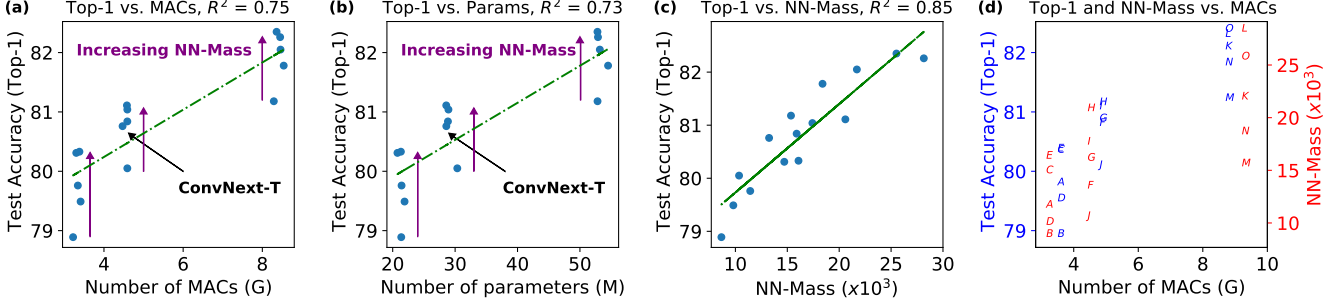
Figure 6. ImageNet accuracy for models scaled up or down from the base ConvNext-Tiny network. All networks are trained for 100 epochs. (a) Top-1 accuracy vs. #MACs for three MAC budgets: 3.3G, 4.5G, and 8.5G MACs ($R^2 = 0.75$). (b) Top-1 accuracy vs. #parameters for three parameter count budgets: 21M, 28M, and 50M ($R^2 = 0.73$). (c) Accuracy increases with NN-Mass ($R^2 = 0.85$). (d) Each letter denotes a model: blue shows its test accuracy and red shows its NN-Mass. Clearly, higher NN-Mass results in higher test accuracy for all models except O and L. Accuracy saturates as NN-Mass becomes high for a given constraint (O and L have less than 0.1% accuracy gap).

must hold true for model $\mathcal{M}$: It should have a large depth and width, and must consist of a uniform block structure that repeats throughout the network.

Since computing NN-Mass does not require training, we scale the base model $\mathcal{M}$ as follows: We scale the depth and width of $\mathcal{M}$ using a set of width and depth multipliers, e.g., $W \in (w_{min}, w_{max})$ and $D \in (d_{min}, d_{max})$. For each $w_m \in W$ and $d_m \in D$, we compute the {#MACs, #parameters, NN-Mass}. Then, for the models that satisfy the given hardware budget constraint $H_i \in \mathcal{H}$, we pick the model with highest NN-Mass and train it. Based on the theory discussed in Section 3.2, we expect this model to have superior gradient flow properties and expressive power than other models with similar #MACs/#parameters. Therfore, model M_i for hardware budget H_i is simply the network that maximizes NN-Mass for that hardware budget. Since #MACs/#parameters/NN-Mass do not require training, this process is completely training free.

Note that, scaling the base model by w_m and d_m *implicitly* restructures the non-linear activation functions because it changes NN-Mass which in turn changes the total number of non-linear units in the network (see Proposition 3). Therefore, we call our networks RAN-implicit (RAN-i). We next present detailed experimental results on ImageNet to show the effectiveness of our training-free scaling.

3.4. RAN-i: ImageNet Evaluation

We start with the base model ConvNext-Tiny [39] and scale it to three hardware budgets: (1) H_1: 3.3B MACs and 21M parameters, (2) H_2: 4.5B MACs and 28M parameters, i.e., the same hardware budget as ConvNext-Tiny, and (3) H_3: 8.5B MACs and 50M parameters. To sample models that satisfy the above hardware budgets, we use width multipliers $W \in (0.25, 1.6)$ and depth multipliers $D \in (0.6, 2.56)$. In total, we sampled 800 different models using the above width and depth multipliers. For each network, we then compute {#MACs, #parameters,

NN-Mass}. We found total 15 networks that satisfied $\mathcal{H} = \{H_1, H_2, H_3\}$ budgets defined above (i.e., 5 models for each of H_1, H_2, H_3). As explained in Section 3.3, for each budget, the higher NN-Mass models are expected to achieve the higher accuracy due to better gradient properties and expressivity. To verify this, we trained all 15 networks on ImageNet for 100 epochs to evaluate if this is indeed the case. Detailed training setup is given in Appendix C.

We show the top-1 accuracy and its relationship with #MACs/#parameters/NN-Mass for all 15 networks in Fig. 6. Note that, all models belong to the same ConvNext family of networks and the only difference is that their widths and depths are scaled up or down from ConvNext-Tiny (ConvNext-T) network. Yet, Fig. 6(a,b) show that for exactly the same hardware budget, there can be a significant difference in accuracy. We found that for all three hardware budgets, models with increasing NN-Mass result in higher accuracy (see Fig. 6(c)). Specifically, $R^2 = 0.85$ for NN-Mass is higher than that for #MACs ($R^2 = 0.75$) and #parameters ($R^2 = 0.73$).

Fig. 6(d) also shows the exact ranking of top-1 accuracy and NN-Mass vs. #MACs. Here, each letter denotes a model, blue color shows its accuracy, and red color shows its NN-Mass. Going from top to bottom for each hardware budget, we can see that the ranking for accuracy and NN-Mass is the same for almost all cases[7]. Note that, for each hardware budget, accuracy eventually saturates and, hence, increasing NN-Mass stops improving the models. This is visible from models O and L in Fig. 6(d). This also indicates that NN-Mass should have some lower and upper bounds for it to work optimally. While we have derived some of these conditions in Corollary 1, more theoretical analysis can certainly improve our understanding of NN-Mass. Nevertheless, our results clearly demonstrate that NN-Mass sig-

---

[7]For instance, going from top to bottom for 3.3B MACs in Fig. 6(d), we see the ranking of networks as E-C-A-D-B for both NN-Mass and accuracy, thus showing that higher NN-Mass results in higher accuracy.
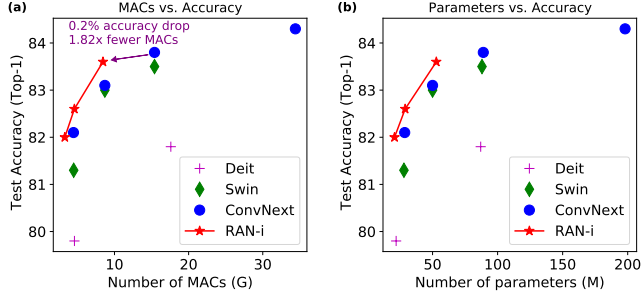
Figure 7. ImageNet results: RAN-i networks (i.e., highest NN-Mass models for various hardware budgets) achieve state-of-the-art accuracy and establish a new pareto frontier over existing networks. Models are trained for 300 epochs. (a) MACs vs. Top-1 Accuracy: RAN-i can achieve 83.6% top-1 accuracy which is only 0.2% lower than ConvNext-B [39] while requiring $1.82\times$ fewer MACs. (b) Parameter count vs. Top-1 Accuracy: RAN-i leads to significant savings in number of parameters as well.

nificantly cuts down the search space of possible depths and widths by providing us the most promising candidates in a completely training-free manner.

Finally, we pick the highest NN-Mass networks for each hardware budget and train them to 300 epochs on ImageNet. We call these highest NN-Mass networks as RAN-i; more architecture details are given in Appendix D. The results are shown in Fig. 7. Clearly, RAN-i establishes a new state-of-the-art on ImageNet as it beats the Pareto frontier of ConvNexts. Specifically, our 8.45B MACs RAN-i network achieves only 0.2% lower top-1 accuracy than ConvNext-B that requires 15.4B MACs. This results in up to $1.82\times$ fewer MACs with nearly the same accuracy. Significant improvements are also obtained in #parameters. Moreover, for a concrete FPS evaluation, we deploy our 4.59B MACs RAN-i network and 8.7B MACs ConvNext-S on a single core Arm Neoverse-based datacenter CPU. We found that RAN-i achieves nearly 40% ($1.38\times$) higher FPS than ConvNext-S with about 0.5% lower accuracy. Therefore, NN-Mass is a very inexpensive method to push the Pareto frontier and to scale models in a training-free way to various hardware constraints once a good base model is known.

3.5. Object Detection with NN-Mass Scaling

To further illustrate the utility of network scaling with NN-Mass, we compare the performance of RAN-i models against that of ConvNext models when deployed as object detection backbones. We design a simple two-stage object detector and show that it performs better when it uses our scaled models than when it uses ConvNexts.

Two-stage object detectors, such as the popular Faster-RCNN and Mask-RCNN, are common because of their ease of use and competitive detection accuracies [23,44]. A two-stage detector funnels the input image through a *backbone*

Table 3. Object Detection Results. On the COCO dataset, object detectors backboned by NN-Mass scaled RAN-i models (see Appendix D) achieve competitive accuracies with significantly less computation requirement than when backboned by ConvNexts.

Backbone	Params	MACs	mAP	FPS Improvement
ConvNext-S	94.9M	150.6B	34.4%	$1.33\times$
RAN-i-S (Ours)	**63.6M**	**82.5B**	**34.7%**	
ConvNext-B	149.9M	219.4B	**35.0%**	$1.21\times$
RAN-i-B (Ours)	**93.2M**	**123.4B**	34.9%	

network (RAN-i or ConvNext, in this case) and then sends the extracted features to a *Region Proposal Network (RPN)*, which proposes *Regions of Interest (RoIs)* around objects to be classified by the *head*.

Our detector architecture is largely the same as that of Faster-RCNN, but we make the following changes to slim it down: (1) In the RPN, we only use PyTorch's design of a single 3×3 convolutional layer, followed by two 1×1 convolutions [30]. We restrict the model to produce 512 RoIs instead of 800 RoIs. (2) In the head, we only use two fully connected layers with 512 neurons each before the output.

For an apples-to-apples comparison, we compare the performance of our object detector in two cases: (1) the backbone is RAN-i, and (2) the backbone is ConvNext. As shown in Table 3, when RAN-i backbones are used, object detectors run up to $1.33\times$ faster (measured on an Arm Neoverse-based datacenter CPU) than when ConvNext equivalents are used, while MACs are reduced by $1.83\times$ and parameters by $1.49\times$. On the COCO dataset [36], accuracy of our RAN-i backbone model either exceeds or is similar to that of the ConvNext backbone model. It is also competitive with ResNet50-FPN-Faster-RCNN's 36% mAP [13], despite requiring $1.6\times$ fewer MACs. ResNet50-FPN-Faster-RCNN is a Faster-RCNN model with a ResNet-50 backbone and a Feature Pyramid Network (FPN), intended to handle objects at different sizes [35]. We do not use a FPN in our model. Further, our architecture facilitates easy training, achieving 34.7% mAP in just 26 epochs (15 hours wall-clock time, see Appendix E for training details). Therefore, RAN-i can be used to significantly reduce object detection compute costs without affecting accuracy.

The benefits of our object detection experiments are further emphasized by the architecture's ease of use and versatility. By incorporating NN-Mass scalable backbones, an existing detector can be easily modified to achieve substantial computation improvements. The rapid trainability of the design (no layer-freezing required) also facilitates training with fewer resources. The backbone can be replaced by any other feature extractor, and the RPN and head are further modularized. Users of this design can thus strike a comfortable balance between accuracy and computational requirements in a relatively inexpensive manner.

a. ConvNext

c. Activation function on new block

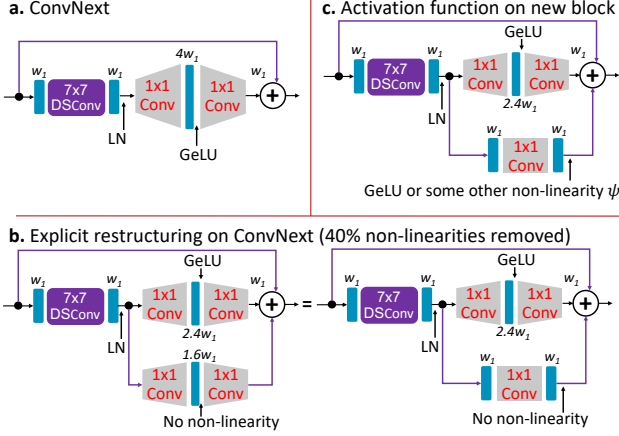**b. Explicit restructuring on ConvNext (40% non-linearities removed)**

Figure 8. Block and Activation Function Co-Design: (a) Typical ConvNext block consists of an inverted bottleneck with $4\times$ expansion ratio (see $4w_1$ GeLUs between the 1×1 layers). (b) If 40% non-linear units are removed, the inverted bottleneck can be analytically separated into two branches. Since the lower branch does not have any non-linearity, it restructures into a single 1×1 convolution with w_1 input and output channels, thereby saving #MACs/#parameters over the base model. (c) Can we regain some accuracy by re-introducing cheap non-linearity on second branch?

4. Block and Activation Function Co-Design

So far, we have implicitly restructured the amount on non-linearity in ConvNext. Is it possible to directly manipulate the non-linear units in ConvNext to explicitly restructure it into a different network architecture? If yes, can we co-design novel activation functions that make up for lost expressivity due to restructuring? We note that the ConvNext block also contains an inverted bottleneck kind of structure with a 1×1 convolution expanding the w_1 input channels by a factor of 4, followed by $4w_1$ non-linear GeLU units, and then a 1×1 convolution projecting back to w_1 channels (see Fig. 8(a)). Total #MACs can be significantly reduced if all or at least some of those GeLUs can be linearized. This can be similar to RAN-e that fully removes non-linear activations from expanded channels. However, we found that the accuracy drops significantly in ConvNexts if we linearize all GeLU units in a block.

To this end, we focus on the following task: We first remove 40% GeLUs from each block of ConvNext-T network. As shown in Fig. 8(b), the 40% linearized channels can then be *analytically* separated out as a secondary branch. That is, the primary (upper) branch has 60% channels ($0.6 \times 4w_1 = 2.4w_1$) with GeLUs, and the secondary (lower) branch has 40% channels ($0.4 \times 4w_1 = 1.6w_1$) with no non-linear activation function. Then, similar to RAN-e, the lower branch explicitly restructures into a single 1×1 convolution. This process reduces the number of MACs from $(H \times W \times w_1 \times 4w_1) \times 2 = 8HWw_1^2$ in ConvNext-T

to $(H \times W \times w_1 \times 2.4w_1) \times 2 + (H \times W \times w_1^2) = 5.8HWw_1^2$, i.e., 27.5% fewer MACs compared to the initial block.

Inevitably, removing non-linear units would result in some loss of accuracy. It is natural to ask if there is anything we can do on the lower branch to recover the lost accuracy. For instance, can we use a GeLU or some other activation function ψ on the lower branch? Note that, the inverted bottleneck structure seems to be a common theme in most of the state-of-the-art models and results in a significant improvement in accuracy. The main defining characteristic of the inverted bottleneck is that the non-linearity is applied in *higher dimensions*, e.g., after expanding the initial number of channels using a 1×1 convolution. We hypothesize that this "applying non-linearity in higher dimensions" is responsible for high accuracy achieved by most of the networks, e.g., EfficientNets [51], ConvNext [39], Swin-Transformers [38], etc. Therefore, we ask the following question w.r.t. the lower branch in our restructured block: *Is there an inexpensive way we can operate higher dimensions without increasing computational costs?*

The above question has been very well-studied in the machine learning community. Specifically, the *kernel trick* [9] used in Support Vector Machines [47] can project low-dimensional inputs into high-dimensional spaces without ever leaving the original low-dimensional space. Towards this, we consider the non-linear activation function ψ for the lower branch as the exponential function:

$$\psi(\boldsymbol{x}, \boldsymbol{\beta}) = e^{\langle \boldsymbol{x}, \boldsymbol{\beta} \rangle} = \sum_{n=0}^{\infty} \frac{\langle \boldsymbol{x}, \boldsymbol{\beta} \rangle^n}{n!}, \qquad (8)$$

where, $\boldsymbol{x}$ is the input data patch and $\boldsymbol{\beta}$ is the learnable weight for the 1×1 convolution on the lower branch. Clearly, similar to all kernel tricks, the exponential kernel implicitly operates in an infinite-dimensional space without ever explicitly computing the sum in Eq. (8). Note that, by designing an activation function to make up for the lost expressivity during our explicit restructuring, we are attempting to *co-design a restructurable block with a novel activation function*. We next evaluate whether this co-design can help us achieve a higher accuracy.

Table 4 demonstrates the results for explicit restructuring of ConvNext-T network. Here, all models are trained on ImageNet for 100 epochs. As evident, if we remove the lower branch completely, this results in a 40% channel pruned version of ConvNext-T. This network loses about 1.3% accuracy over the baseline. Next, we evaluate two networks: (1) A 27.5% channel pruned version of ConvNext-T which is about 0.9% below the baseline, and (2) A 40% non-linearity restructured network as shown in Fig. 8(b), right (we call this as Model A). Both of these networks achieve a similar accuracy and incur exactly the same #MACs and #parameters. Next, we train Model B which appends a GeLU activation function at the end of lower branch (see

Table 4. Block and Activation Function Co-Design for ConvNext-Tiny. Models are trained for 100 epochs on ImageNet.

100 epoch training	Params	MACs	Top-1
ConvNext-T	28.6M	4.47B	80.2%
ConvNext-T (40% pruned)	18.2M	2.8B	78.9%
ConvNext-T (27.5% pruned)	21.5M	3.32B	79.3%
Model A (40% restructured, Fig. 8(b) right)	21.5M	3.32B	79.3%
Model B (Model A + [ψ = GeLU], Fig. 8(c))	21.5M	3.32B	79.4%
Model C (Model A + [ψ = EXP], Fig. 8(c))	21.5M	3.32B	**79.7**%

Fig. 8(c)). Model B achieves only 0.1% higher accuracy than Model A. Finally, we train Model C using the exponential activation function. Even though Model C has exactly the same #MACs and #parameters as Models A, B, and the 27.5% channel-pruned ConvNext-T, it achieves nearly 0.3%-0.4% higher accuracy. This supports our hypothesis that operating in higher dimensions can still be beneficial in deep networks even if it is done using kernel tricks.

While we achieve accuracy improvement with our proposed activation function, there are clear limitations: introducing an exponential in the network makes it highly unstable. Specifically, we observed that during training, it can often lead to NaN loss. However, when the model does train, we get better convergence and accuracy than no-activation and GeLU cases. This is a key limitation of our kernel trick. Therefore, more stable activation functions that implicitly operate in high dimensions should be designed in future.

5. Outstanding Problems

So far, we have demonstrated that explicit and implicit restructuring of non-linear activation functions is valuable for deep learning. Based on our insights, we now discuss the following open problems in this new domain:

- **Manipulating Non-Linearity as a Key NAS Goal:** In the AI accelerator age, high utilization yet high accuracy building blocks are a prerequisite for hardware-aware networks. However, existing NAS does not use a search space containing restructurable blocks like AFRBs. Therefore, future NAS research should exploit non-linearity manipulation as a key objective.

- **Non-Linearity and Model Architecture Co-Design:** More generally (i.e., beyond NAS), a new research direction is to co-design novel restructurable blocks along with ways to induce the non-linearity elsewhere in the network. We have attempted to do this in our Section 4, e.g., we restructured a known block with a more powerful, theoretically-grounded activation function. However, since our exponential function has clear stability issues, more research is needed in this area to potentially discover completely new building blocks that are friendly to AI accelerators.

- **Better Theory and Generic Topological Metrics:** Our NN-Mass [7] based method allows us to scale base models to various hardware constraints in a training-free manner and achieves state-of-the-art accuracy on ImageNet. There are still limitations which can be targeted in future work: (*i*) Currently NN-Mass works for large networks. There must be specific bounds on depth and width to obtain optimal results with NN-Mass (see Section 3.4). While Corollary 1 is a step in this direction, more theory is needed to better understand NN-Mass. (*ii*) Also, NN-Mass works only for uniform structures and does not work for irregular strides and full NAS search spaces. Thus, better topological metrics are required for generic NAS as well.

- **More Theory for Explicit Restructuring:** We need better theoretical grounding for explicit restructuring of activation functions. For instance, how does the dual objective in problem (3) change the deep learning optimization landscape? Can we build better optimizers that specifically work well for AFRBs? Improvements in this space can directly improve the overall accuracy without changing the computational costs.

The research directions above can significantly impact efficient deep learning, particularly for AI accelerators.

6. Related Work

Linear overparameterization in deep networks. The benefit of linear overparameterization in accelerating the training of deep neural networks and improving accuracy has received considerable attention in recent works [3, 11, 14, 15, 22, 57]. Several of these previous works propose overparameterizing a convolutional layer during training by using a series of linear convolutional layers. More recently, RepVGG [15] demonstrates the importance of linear residual connections in parallel branches of a neural network during training, which can be folded during inference to boost the accuracy of single-branch networks. In contrast to these prior works on model overparameterization, RAN-e seeks to identify where in the network the non-linear activation functions can be removed. This results in a sequence of linear convolution layers that can be collapsed into a single, small, regular convolution layer. Overall, our approach produces networks that use a mix of IBNs and regular convolutions, and achieve significantly higher accuracy at lower computational cost than RepVGG [15].

Non-linearity manipulation. Concurrent with our work, Fu *et al.* [20] proposed DepthShrinker, which combines irregular blocks into dense operations to create hardware-efficient, compact neural networks. DepthShrinker also proposes to replace low-utilization blocks with regular convolutions by pruning non-linear units. Despite the synergies

between our work and DepthShrinker, there are significant differences and advantages of our work:

1. **More generality:** RANs are much more general than just non-linearity pruning: We propose a hardware-aware search space for future NAS methods. We further propose training-free model scaling with theoretically grounded non-linearity manipulation, as well as a co-design between blocks and activation functions which could also be useful for networks where complete removal of non-linear units may not be possible.

2. **Fully differentiable restructuring:** Our RAN-e is a fully differentiable restructuring algorithm. In contrast, DepthShrinker [20] relies on approximate techniques like Straight-Through Estimators (STE) [6] which can be unstable under certain conditions [61].

3. **No self-distillation:** DepthShrinker exploits methods like self-distillation to regain the accuracy drop. Distillation-based techniques are known to result in significant accuracy improvements [41]. We do not use any distillation-based methods to improve accuracy.

4. **Better accuracy:** RAN-e achieves significantly higher accuracy than DepthShrinker. In particular, on ImageNet, RAN-e-C achieves a 2.1% higher accuracy under comparable MACs compared to DepthShrinker's MBV2-1.4-DS-D model (e.g., 488M MACs RAN-e-C achieves 74.6% accuracy vs. 72.5% accuracy for 484M MACs DepthShrinker; no self-distillation used in either network). Additionally, for similar MACs, RAN-e-GT trained *without* self-distillation significantly outperforms DepthShrinker trained *with* self-distillation by **4.47**% in top-1 accuracy (e.g., 74.6% for 433M MACs RAN-e-GT vs. 70.13% for DepthShrinker's 415M MACs MBV2-1.4-DS-F).

5. **SotA ImageNet results on multiple scales:** Finally, our techniques result in state-of-the-art results on ImageNet at multiple scales, ranging from micro-NPUs to datacenter CPUs. In contrast, DepthShrinker does not cover such a broad range of ImageNet networks.

DNN compression techniques. Numerous research efforts have been devoted in recent years to compressing neural networks for increasing hardware efficiency in accelerators via filter pruning [12, 19, 25], layer pruning [16, 17, 59], quantization [5, 18, 48, 49], and low-rank matrix factorization [50, 56, 60]. Nonetheless, because these model compression techniques are orthogonal to our hardware-aware block search paradigm, they can be combined with our search space to improve hardware efficiency even further.

NAS for improving hardware efficiency. Recent research on automated efficient DNN design has been able to take advantage of significant advances in neural architecture search (NAS) to select from a variety of hardware-efficient convolutional blocks, layer widths, depths, connectivities, and per-layer quantization bitwidths during training while building a network architecture [4, 10, 32, 34, 52–55, 58]. For example, MobileDet's NAS search space included IBN as well as other hardware-aware convolutions like fused and tucker convolutions [58]. While it is possible to naively construct a NAS search space from parallel branches of IBN and hardware-friendly regular convolutions, our work proposes using non-linearity manipulation to choose between IBN and hardware-friendly convolutional blocks from the same underlying weight-shared block. This, unlike previous works, will enable the search process between different convolutional blocks to take advantage of weight-sharing NAS. Manipulating non-linearity can essentially be added as another search dimension during NAS. We leave this exploration for future work.

7. Conclusion

In this paper, we have proposed the new RAN paradigm that manipulates the amount of non-linearity in networks to improve their hardware-efficiency. Specifically, we have proposed RAN-explicit (RAN-e) and RAN-implicit (RAN-i) techniques for hardware-aware search spaces and training-free model scaling, respectively. For certain classes of networks, we have also theoretically proved the link between model expressivity as defined by the amount of non-linearity and its topological properties. With extensive experiments, we have demonstrated that our networks achieve state-of-the-art results on ImageNet at different scales and for various types of hardware ranging from micro-NPUs to datacenter CPUs. Our proposed RAN-e achieves a similar accuracy as EfficientNet-Lite-B0 while improving FPS by up to $1.5\times$ on Arm micro-NPUs. Moreover, with a similar or better accuracy, our RAN-i networks demonstrate nearly $2\times$ reduction in #MACs and about 40% improvement in FPS on Arm Neoverse-based datacenter CPUs compared to ConvNexts. When used as backbones in object detection, RAN-i achieve a similar or higher mAP over ConvNexts with 33% higher FPS on datacenter CPUs. Finally, we have also discussed a new research direction of model architecture-activation function co-design.

Overall, we have demonstrated several useful scenarios where manipulating non-linear activation functions in deep networks directly results in significant hardware-awareness and efficiency. For future work, we have outlined several outstanding research problems in this new area of restructurable deep networks.

References

[1] Arm. Ethos-U55 micro-Neural Processing Unit (micro-NPU), 2020. Link: https://www.arm.com/products/silicon-ip-cpu/ethos/ethos-u55. Accessed: December 8, 2021. 1, 6

[2] Arm. Ethos-U65 micro-Neural Processing Unit (micro-NPU), 2020. Link: https://www.arm.com/products/silicon-ip-cpu/ethos/ethos-u65. Accessed: December 8, 2021. 1, 6

[3] Sanjeev Arora, Nadav Cohen, and Elad Hazan. On the optimization of deep networks: Implicit acceleration by over-parameterization. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 244–253, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. 14

[4] Colby Banbury, Chuteng Zhou, Igor Fedorov, Ramon Matas, Urmish Thakker, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, and Paul Whatmough. Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers. In *Proceedings of Machine Learning and Systems*, 2021. 1, 15

[5] Ron Banner, Yury Nahshan, and Daniel Soudry. Post training 4-bit quantization of convolutional networks for rapid-deployment. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, 2019. 15

[6] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 15

[7] Kartikeya Bhardwaj, Guihong Li, and Radu Marculescu. How does topology influence gradient propagation and model performance of deep networks with densenet-type skip connections? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13498–13507, 2021. 2, 9, 10, 14

[8] Kartikeya Bhardwaj, Milos Milosavljevic, Liam O'Neil, Dibakar Gope, Ramon Matas, Alex Chalfin, Naveen Suda, Lingchuan Meng, and Danny Loh. Collapsible linear blocks for super-efficient super resolution. *Proceedings of Machine Learning and Systems*, 4:529–547, 2022. 4

[9] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. 13

[10] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019. 1, 7, 15

[11] Jinming Cao, Yangyan Li, Mingchao Sun, Ying Chen, Dani Lischinski, Daniel Cohen-Or, Baoquan Chen, and Changhe Tu. Do-conv: Depthwise over-parameterized convolutional layer. *IEEE Transactions on Image Processing*, 31:3726–3736, 2022. 14

[12] Ting-Wu Chin, Ruizhou Ding, Cha Zhang, and Diana Marculescu. Towards efficient model compression via learned global ranking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020. 15

[13] Torch Contributors. Faster-rcnn-resnet50-fpn documentation. 12, 19

[14] Xiaohan Ding, Yuchen Guo, Guiguang Ding, and Jungong Han. Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019. 14

[15] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. Repvgg: Making vgg-style convnets great again. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13733–13742, 2021. 8, 14

[16] Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. 15

[17] Sara Elkerdawy, Mostafa Elhoushi, Abhineet Singh, Hong Zhang, and Nilanjan Ray. To filter prune, or to layer prune, that is the question. In *Proceedings of the Asian Conference on Computer Vision (ACCV)*, November 2020. 15

[18] Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. Learned step size quantization. In *International Conference on Learning Representations*, 2020. 15

[19] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. 15

[20] Yonggan Fu, Haichuan Yang, Jiayi Yuan, Meng Li, Cheng Wan, Raghuraman Krishnamoorthi, Vikas Chandra, and Yingyan Lin. Depthshrinker: A new compression paradigm towards boosting real-hardware efficiency of compact neural networks. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, 2022. 14, 15

[21] Google. EfficientNet-Lite models, 2020. Link: https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet/lite. Accessed: December 8, 2021. 6, 8

[22] Shuxuan Guo, Jose M. Alvarez, and Mathieu Salzmann. Expandnets: Linear over-parameterization to train compact convolutional networks. In *Advances in Neural Information Processing Systems*, volume 33, pages 1298–1310, 2020. 4, 14

[23] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 12

[24] 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. 8

[25] Yang He, Yuhang Ding, Ping Liu, Linchao Zhu, Hanwang Zhang, and Yi Yang. Learning filter pruning criteria for deep

convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 15

[26] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European conference on computer vision (ECCV)*, pages 784–800, 2018. 1

[27] Dan Hendrycks and Kevin Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. 2016. 2

[28] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. 1

[29] 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 *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324, 2019. 2, 6

[30] N Inkawhich. Finetuning torchvision models-pytorch tutorials 1.2.0 documentation, 2021. 12

[31] Namhoon Lee, Thalaiyasingam Ajanthan, Stephen Gould, and Philip H. S. Torr. A signal propagation perspective for pruning neural networks at initialization. In *International Conference on Learning Representations*, 2020. 9

[32] Sheng Li, Mingxing Tan, Ruoming Pang, Andrew Li, Liqun Cheng, Quoc V. Le, and Norman P. Jouppi. Searching for fast model families on datacenter accelerators. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8085–8095, June 2021. 1, 15

[33] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *International conference on machine learning*, pages 2849–2858. PMLR, 2016. 1

[34] Ji Lin, Wei-Ming Chen, Yujun Lin, Chuang Gan, and Song Han. Mcunet: Tiny deep learning on iot devices. *Advances in Neural Information Processing Systems*, 33, 2020. 1, 15

[35] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017. 12

[36] 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 *European conference on computer vision*, pages 740–755. Springer, 2014. 12, 19

[37] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 4

[38] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022, 2021. 13

[39] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11976–11986, 2022. 1, 2, 11, 12, 13, 18

[40] Guido F Montúfar, 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. 3, 9, 10

[41] Bert Moons, Parham Noorzad, Andrii Skliar, Giovanni Mariani, Dushyant Mehta, Chris Lott, and Tijmen Blankevoort. Distilling optimal neural networks: Rapid search in diverse spaces. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12229–12238, 2021. 1, 15

[42] Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. On the expressive power of deep neural networks. In *international conference on machine learning*, pages 2847–2854. PMLR, 2017. 3, 9

[43] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017. 2

[44] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. 12

[45] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks, 2019. 1, 3

[46] Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. Bounding and counting linear regions of deep neural networks. In *International Conference on Machine Learning*, pages 4558–4566. PMLR, 2018. 9

[47] Alex J Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14(3):199–222, 2004. 13

[48] Pierre Stock, Angela Fan, Benjamin Graham, Edouard Grave, Rémi Gribonval, Herve Jegou, and Armand Joulin. Training with quantization noise for extreme model compression. In *International Conference on Learning Representations*, 2021. 15

[49] Pierre Stock, Armand Joulin, Rémi Gribonval, Benjamin Graham, and Hervé Jégou. And the bit goes down: Revisiting the quantization of neural networks. In *International Conference on Learning Representations*, 2020. 15

[50] Cheng Tai, Tong Xiao, Xiaogang Wang, and Weinan E. Convolutional neural networks with low-rank regularization. *CoRR*, abs/1511.06067, 2015. 15

[51] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019. 1, 3, 7, 13

[52] Mingxing Tan and Quoc Le. Efficientnetv2: Smaller models and faster training. In *Proceedings of the 38th International Conference on Machine Learning*, 2021. 1, 15

[53] Stefan Uhlich, Lukas Mauch, Fabien Cardinaux, Kazuki Yoshiyama, Javier Alonso Garcia, Stephen Tiedemann,

Thomas Kemp, and Akira Nakamura. Mixed precision dnns: All you need is a good parametrization. In *International Conference on Learning Representations*, 2020. 1, 15

[54] Arash Vahdat, Arun Mallya, Ming-Yu Liu, and Jan Kautz. Unas: Differentiable architecture search meets reinforcement learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 1, 15

[55] Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, Peter Vajda, and Joseph E. Gonzalez. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 1, 15

[56] Wei Wen, Cong Xu, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Coordinating filters for faster deep neural networks. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 658–666, 2017. 15

[57] Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying Graph Convolutional Networks. In *Proceedings of the 36th International Conference on Machine Learning*, pages 6861–6871. PMLR, 2019. 14

[58] Yunyang Xiong, Hanxiao Liu, Suyog Gupta, Berkin Akin, Gabriel Bender, Yongzhe Wang, Pieter-Jan Kindermans, Mingxing Tan, Vikas Singh, and Bo Chen. Mobiledets: Searching for object detection architectures for mobile accelerators. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3825–3834, 2021. 1, 3, 15

[59] Pengtao Xu, Jian Cao, Fanhua Shang, Wenyu Sun, and Pu Li. Layer pruning via fusible residual convolutional block for deep neural networks. *CoRR*, abs/2011.14356, 2020. 15

[60] Miao Yin, Yang Sui, Siyu Liao, and Bo Yuan. Towards efficient tensor decomposition-based dnn model compression with optimization framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10674–10683, June 2021. 15

[61] Penghang Yin, Jiancheng Lyu, Shuai Zhang, Stanley Osher, Yingyong Qi, and Jack Xin. Understanding straight-through estimator in training activation quantized neural nets. *arXiv preprint arXiv:1903.05662*, 2019. 15

[62] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017. 8, 18

[63] Yanqi Zhou, Xuanyi Dong, Tianjian Meng, Mingxing Tan, Berkin Akin, Daiyi Peng, Amir Yazdanbakhsh, Da Huang, Ravi Narayanaswami, and James Laudon. Towards the co-design of neural networks and accelerators. In *Proceedings of Machine Learning and Systems*, volume 4, pages 141–152, 2022. 1, 3

A. RAN-e: SuperNet Details

The SuperNet architecture details are given in Table 5. The top architecture for our SuperNet is different from the typical top convolutions used in networks like EfficientNet. The detailed structure of our top is given in Table 6.

Table 5. Detailed architecture for the RAN-e SuperNet. H_i and W_i are height and width of input feature maps, respectively. $\{e, s, C_o\}$ refer to expansion ratio, stride, and output channels at the given stage, respectively. All kernel sizes in SuperNet are 3×3.

Stage	Operator	$H_i \times W_i$	e	s	C_o
1	Conv 3×3	224×224	–	2	16
2	AFRB-1	112×112	6	1	32
3	AFRB-1	112×112	6	2	48
4	AFRB-1	56×56	6	2	64
5	AFRB-1	28×28	6	1	80
6	AFRB-1	28×28	6	2	80
7	AFRB-2	14×14	6	1	80
8	AFRB-1	14×14	4	1	96
9	AFRB-2	14×14	4	1	96
10	AFRB-1	14×14	6	1	128
11	AFRB-3	14×14	6	1	128
12	AFRB-1	14×14	6	2	160
13	AFRB-1	7×7	4	1	176
14	AFRB-2	7×7	4	1	176
15	AFRB-2	7×7	4	1	176
16	AFRB-1	7×7	6	1	224
17	AFRB-2	7×7	6	1	224
–	Top	7×7	–	–	1000

Table 6. Top convolution architecture for RAN-e networks.

Stage	Operator	$H_i \times W_i$	C_i	C_o
1	Conv 1×1	7×7	224	1344
2	DSConv 7×7	7×7	1344	1344
3	Average Pool	1×1	1344	1344
4	Conv 1×1	1×1	1344	1000

B. RAN-e: Training Details

We train all RAN-e networks and SuperNet on ImageNet using Autoaugment data augmentation and label smoothing with value 0.1. We also use RMSprop optimizer with an initial learning rate 0.005 which follows a cosine annealing decay schedule after an initial warmup of 5 epochs, batch size 768, decay 0.9, momentum 0.9, epsilon 0.001, and weight decay 5e-6. We do not use Mixup data augmentation [62] in our experiments. We implement our PReLU search as well as finetuning experiments in Tensorflow. All models are trained on 8 NVIDIA V100 GPUs.

C. RAN-i: Training Details

Our training setup for RAN-i networks is nearly identical to that used in the ConvNext paper and its public code [39]. The only difference is that we reduced the batch size to 80

for our networks in order to fit within the GPU memory. The batch size was lowered to 80 for both the initial 15 networks sampled using NN-Mass in Fig. 6, and the final models trained in Fig. 7. For the initial 15 networks in Fig. 6, we used drop path = 0.1 for all networks. In the next section, we provide more details for the final RAN-i networks that were trained to 300 epochs.

D. Final RAN-i Architecture Details

Table 7 more details on the final RAN-i networks. The base width and depth configurations are the same as that for ConvNext-Tiny network. For example, the first group in ConvNext-Tiny has 3 blocks with 96 input and output channels at each block, followed by 3 blocks with 192 channels each, and so on. To obtain RAN-i networks, these base widths and depths are multiplied by width multiplier w_m and depth multiplier d_m, respectively. As evident, the resulting network configurations satisfy the hardware constraints like {3.3B, 4.5B, 8.5B} MACs. The RAN-i networks in Table 7 are the highest NN-Mass models for the aforementioned hardware constraints.

Table 7. Detailed configurations for RAN-i networks. These are the final networks that were trained to 300 epochs in Fig. 7.

	RAN-i-T (Tiny)	RAN-i-S (Small)	RAN-i-B (Base)
Base Width Config	[96,192,384,768]		
Base Depth Config	[3,3,9,3]		
w_m	0.666	0.789	0.9105
d_m	1.65	1.65	2.30
New Width Config	[64,128,256,511]	[76,151,303,606]	[87,175,350,699]
New Depth Config	[5,5,15,5]	[5,5,15,5]	[7,7,21,7]
#Parameters	20.76M	28.93M	52.89M
#MACs	3.3B	4.59B	8.45B
Drop Path	0.1	0.2	0.4
Top-1	82.03%	82.63%	83.61%

E. Object detection: Training Details

Our training setup is nearly identical to that used by Facebook to train their ResNet50-FPN-Faster-RCNN model [13]. We do not freeze any layers in our detectors, and start training the RPN and head against ImageNet-pretrained backbones. As with Facebook's procedure, minimal data augmentation is performed (resize image to 800x800, add 50% probability of horizontal image flips). We use Facebook's modified COCO loss [36] with stochastic gradient descent (learning rate controlled by a LR Scheduler, with momentum of 0.9 and weight decay of 0.0001). Training on the Microsoft COCO 2017 dataset lasts for 26 epochs, using roughly 15 hours of wall-clock time on 8 NVIDIA V100 GPUs, each with a batch size of 2.