Searching for Fast Model Families on Datacenter Accelerators

Sheng Li, Mingxing Tan, Ruoming Pang, Andrew Li, Liqun Cheng, Quoc V. Le, Norman P. Jouppi
Google
{lsheng,tanmingxing,rpang,andrewyli,liquncheng,qvl,jouppi}@google.com

Abstract

Neural Architecture Search (NAS), together with model scaling, has shown remarkable progress in designing high accuracy and fast convolutional architecture families. However, as neither NAS nor model scaling considers sufficient hardware architecture details, they do not take full advantage of the emerging datacenter (DC) accelerators. In this paper, we search for fast and accurate CNN model families for efficient inference on DC accelerators. We first analyze DC accelerators and find that existing CNNs suffer from insufficient operational intensity, parallelism, and execution efficiency and exhibit FLOPs-latency nonproportionality. These insights let us create a DC-accelerator-optimized search space, with space-to-depth, space-to-batch, hybrid fused convolution structures with vanilla and depthwise convolutions, and block-wise activation functions. We further propose a latency-aware compound scaling (LACS), the first multi-objective compound scaling method optimizing both accuracy and latency. Our LACS discovers that network depth should grow much faster than image size and network width, which is quite different from the observations from previous compound scaling. With the new search space and LACS, our search and scaling on datacenter accelerators results in a new model series named EfficientNet-X. EfficientNet-X is up to more than 2X faster than EfficientNet (a model series with state-of-the-art trade-off on FLOPs and accuracy) on TPUv3 and GPUv100, with comparable accuracy. EfficientNet-X is also up to 7X faster than recent RegNet and ResNeSt on TPUv3 and GPUv100. Source code is at https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet/tpu

1. Introduction

As Moore's Law is slowing down, more specialized datacenter (DC) accelerators such as GPUs [43, 14] and TPUs [32, 20, 15, 42] have been developed to keep up with the increasing demand of machine learning (ML) models. With the increasing complexity of ML model architectures and accelerator architectures, there is a fast-widening gap between achieved performance and available performance.

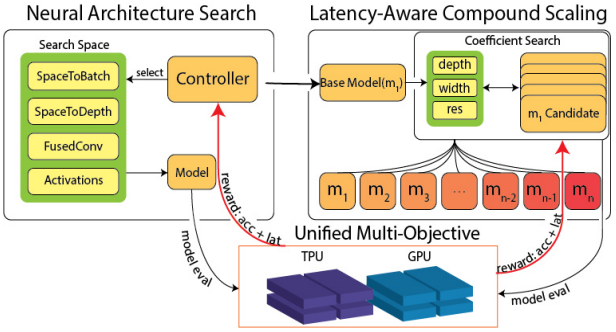

Figure 1: Unified accelerator-optimized NAS and Latency-aware Compound Scaling (LACS) to search model families optimized for TPUs and GPUs. The same multi-objective with both latency and accuracy is used for both NAS and model scaling. For a given accelerator, a base model (m1) is obtained via NAS with a new search space tailored to DC accelerators. The new latency-aware compound scaling (LACS) searches for scaling coefficients on m1 to form the model family. Both processes are executed separately on TPU and GPU, resulting in two families of final models.

Neural Architecture Search (NAS) [65, 9, 66, 11], a new paradigm of assembling models automatically, has the potential to bridge the gap. Modern NAS usually aims at designing a family of models for different accuracy-speed trade-offs for different use cases. Because of the high cost associated with searching for the entire family of models, model scaling is commonly used to achieve this goal by scaling [25, 57] up from a base model to form a model family. However, on specialized DC accelerators the fast-widening gap remains even with NAS and model scaling, because they do not have sufficient visibility into hardware architecture details and thus cannot design optimal model families for them.

In this paper, we aim at bridging this gap and designing model families with high accuracy and inference speed, by taking into consideration hardware architecture details of TPUs and GPUs for both NAS and model scaling. We first analyze DC accelerators to find performance bottlenecks. Our analysis reveals the root cause of the recent observed FLOPs-latency nonpropotionality [51]. We discover that SOTA CNNs suffer from low operational intensity

and parallelism, which causes low computation rate (*i.e.*, FLOPs/sec or Ops/sec[1]) and sub-optimal inference latency and throughput on TPU/GPU accelerators. With these insights, we augment state-of-the-art (SOTA) NAS with *DC accelerator optimized search space* to improve CNN model operational intensity and efficiency. Concretely, we create a new search space with accelerator-friendly operations including space-to-depth, space-to-batch, fused convolution structures, and block-wise searchable activation as shown in Figure 1. We propose *latency-aware compound scaling (LACS)* that uses a multi-objective of both accuracy and inference speed to search for scaling factors to generate a model family. LACS is the *first* compound scaling method with a multi-objective including both latency and accuracy.

With the improved NAS and LACS, we search for high accuracy CNNs for efficient inference on TPUv3 [20, 15, 42] and GPUv100 [14]. Our search results in a new model family named EfficientNet-X (with differences on TPU and GPU) that achieve a better accuracy and latency trade-offs than the state-of-the-art. EfficientNet-X models are up to more than 2X faster on TPUv3 and GPUv100 respectively than EfficientNet [57] with comparable accuracy. Moreover, EfficientNet-X models achieve 30% more speedup compared to EfficientNet when moving from TPUv2 to TPUv3, demonstrating the generality of our search method across different accelerator generations. EfficientNet-X is also faster than other SOTA models, with on average (geo-mean) 82% and 48% faster than RegNet and ResNeSt respectively on GPUv100 and 7X and 48% faster than RegNet and ResNeSt respectively on TPUv3.

In summary, this paper makes the following contributions:

1. We conduct quantitative analysis to reveal the root cause of FLOPs-latency nonproportionality on DC accelerators. Although recent work [51] has observed the similar behavior, our roofline model and analysis is the *first* to show the fundamental reasons for latency to be much less correlated to FLOPs on GPUs and TPUs than on CPUs. Moreover, our analysis also discovers the performance bottlenecks of CNNs and inspires enhancements for both NAS and compound model scaling.
2. We design a DC-accelerator-optimized search space, with space-to-batch, space-to-depth, fused convolution structures, and block-wise activation functions, to compose CNNs with higher operational intensity and efficiency for better accuracy and speed trade-offs.
3. We propose latency-aware compound scaling (LACS), the *first* compound scaling method with accuracy and latency as the multi-objective. After taking latency into account, our LACS discovers network depth should grow much faster than image size and network width, which is quite different from previous compound model scaling results [57].
4. Our unified NAS and LACS produce EfficientNet-X, with up to 2X speedup over the EfficientNet and up to 7X speedup over RegNet/ResNeSt on TPUs and GPUs.

[1]When operations are done in different data types such as bfloat16 [15], float16 [14], and tf32 [43], the computation rate is usually denoted as OPS, *i.e.*, OPs/Second. Hereafter in this paper, we use FLOPs/sec and Ops/sec interchangeably unless noted otherwise.

2. Rethink model speed on DC accelerators: Why FLOPs and latency do not correlate

Emerging datacenter accelerators, including TPUs [32, 15] and GPUs [14], have been using new hardware architectures to keep up with the fast-increasing demand of computing power from ML models. In particular, because matrix-multiplication is the core operation in neural networks, the most special feature of these accelerators is the matrix-multiply-and-accumulate units, called tensor cores [14] in GPUs and matrix multiply units [32, 42] in TPUs. These new hardware architectures have changed the way ML models execute on the accelerators. For example, recent work [51] has observed that FLOPs and latency do not correlate on these accelerators. However, with these empirical observations, there is yet no in-depth analysis to reveal the root cause.

In this section, we find the root cause of the FLOPs-latency nonpropotionality and provide principles for designing high speed ML models on DC accelerators. To rethink the implications of the DC accelerators on model speed including the FLOPs-latency nonpropotionality, we build a generic performance model as shown in Equation 1.

$$\textbf{Latency} = \frac{W}{C} = \frac{\mathbf{W}}{C_{ideal} \times \mathbf{E}}, \qquad \mathbf{I} = \frac{\mathbf{W}}{Q}$$
$$C_{ideal} = \begin{cases} \mathbf{I} \times b & \text{if } I < \text{Ridge Point} \\ C_{max} & \text{else} \end{cases} \tag{1}$$

where $\boldsymbol{W}$ (in FLOPs) is the amount of computation required by an ML model, Q (in Bytes) is the memory traffic (bytes of memory transfers) incurred during the execution, and $\boldsymbol{I}$ is the operational intensity of the model (in FLOPs/Byte). C (in FLOPs/sec) is the computation rate determined by the ideal computation rate (C_{ideal}) and the execution efficiency $\boldsymbol{E}$, where C_{ideal} is determined by I, accelerator memory bandwidth b, and accelerator peak computation rate C_{max}. Note that b and C_{max} are accelerator hardware constants. Details of I and C are shown in Figure 2. The execution efficiency $\mathbf{E}$ is defined as the achieved C / C_{ideal}. The end-to-end inference latency of a model is a nonlinear function of W, I, and E, instead of only W— the FLOPs. This is the *root cause* of FLOPs-latency nonproportionality.

To dive deeper into the operational intensity and efficiency, we adapt the simple *roofline* analysis (as shown in Figure 2) that originated from high-performance computing (HPC)[59] and has been used in ML [60, 32, 42]. The roofline model reasonably assumes that applications are either compute-bound or memory-(bandwidth)-bound as they

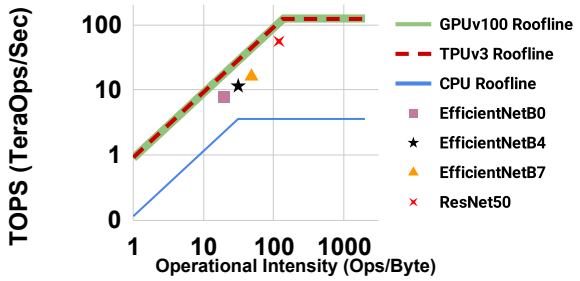

Figure 2: Rooflines of TPUv3, Volta SMX2 GPU, and Xeon Skylake CPU. TPU and GPU have overlapped rooflines because of their similar peak computation rate and memory bandwidth.

don't fit in on-chip memories. The Y-axis is computation rate C in FLOPs/sec or Ops/sec, thus the peak computation rate forms the saturation region of the roofline. The X-axis is operational intensity I in FLOPs per memory byte accessed. The memory bytes include weights, activations, and intermediate values. The slope of the linear part can be easily derived to be memory bandwidth (Bytes/Sec). An ML model can achieve peak FLOPs/sec on the accelerators only when its operational intensity is sufficient to push it into the saturation (*i.e.*, compute-bound) region in the roofline. Otherwise, the ML model is memory-bandwidth-bound. The ridge point is the transition point from the memory-bandwidth-bound performance region to the compute-bound performance region. With the roofline analysis and understanding of datacenter accelerator architectures, we can obtain a few key principles for designing high speed ML models on DC accelerators:

- Compute is significantly cheaper on DC accelerators than on previous systems because of the new matrix-multiply-and-accumulate units, which results in the ∼35X higher TeraOps/sec of GPUv100 and TPUv3 than typical of CPU as shown as the saturation regions in Figure 2.
- ML models need have high operational intensity on TPUs and GPUs to be in the compute-bound region to reach close-to-peak performance. This is because, for TPUs and GPUs, their peak computation rate (TeraOps/s) grows much faster than memory bandwidth (Bytes/s). Thus, TPUs and GPUs have ridge points farther to the right than CPUs. However, as shown in Figure 2 EfficientNets' operational intensity is an order of magnitude lower than that of the TPU/GPU ridge point (and even ResNet), which is too low to tap into the full potential of the DC accelerators despite their significantly reduced FLOPs. Specifically, EfficientNet has∼10X FLOPs reduction compared to other models such as ResNet at comparable accuracy.
- Parallelism is critical for high speed models. TPU/GPU accelerators are optimized for throughput with the new matrix/tensor units. These matrix/tensor units require large parallelism to achieve high performance. For example, a convolution operation needs to have adequately sized depth, batch, and spatial dimensions to provide enough parallelism to achieve high execution efficiency on matrix units. Additionally, because many vector/element operations such as activation functions run on vector units (*e.g.*, CUDA cores in GPUs and vector units in TPUs) instead of matrix units, sufficient parallelism between matrix and vector units is also important for ML models to achieve high performance on GPUs and TPUs.

3. Optimize search space for DC accelerators

Based on the analysis and optimization principles in the previous section, we optimize NAS to improve operational intensity and parallelism to design fast models. NAS has three pillars: the search algorithms governing the search process, the objectives determining the trade-offs of the search results, and the search space as the key link between model architectures and accelerator architectures. Thus, specializing the search space for DC accelerators is crucial to give NAS more visibility to DC accelerator details. Our optimized search space includes three key new components: accelerator-friendly space-to-depth/batch, fused convolution structures, and block-wise activation functions.

3.1. Efficient space-to-depth and space-to-batch

As pointed out in Section 2, convolutions need high parallelism in all dimensions (depth, batch, and spatial) to achieve high speed on TPUs and GPUs. However, insufficient parallelism because of the small depth and batch is the usual cause of low utilization and low performance on matrix units. We augment the search space with accelerator-friendly space-to-depth and space-to-batch ops to increase depth and batch dimensions while keeping the total tensor volume the same.

For space-to-depth ops, instead of using the memory-copy-reshape based ops provided by frameworks such as TensorFlow [7] and Pytorch [45], we customize an $n \times n$ convolution with stride-n to perform the space-to-depth operation, reshaping an $H \times W \times C$ tensor to an $H/n \times W/n \times C * n^2$ tensor. This approach has two advantages: 1) convolutions are much preferred by TPUs and GPUs because of their high operational intensity and execution efficiency; 2) in addition to reshaping the input tensor to improve operational intensity and efficiency, the $n \times n$ convolutions can also be trained to contribute to the model's capacity. For space-to-batch ops, we have to use the memory-intensive copy-reshape ops provided by common frameworks [7, 45].

3.2. Fused convolution structures

As they are the dominant operations in CNNs, it is important to ensure that convolutions in the search space are optimized for accelerator architectures. As the baseline search space already includes a rich set of convolutions with different types, sizes, and shapes, we augment the search space with fused convolution macro structures. With 4-mode input

tensor $\mathcal{I}$ and output tensor $\mathcal{O}$ of $N \times C \times H \times W$[2], the total computation load W (in FLOPs) and operational intensity I for convolution and depthwise convolution are in Equation 2. From Equation 1 and 2, it is clear that although depthwise convolutions have fewer FLOPs, they also have lower operational intensity to potentially hurt computation rate and thus hurt latency.

$$\begin{aligned} W_{\text{\_Conv2}} &= N \times H \times W \times C^2 \times K^2, \\ I_{\text{\_Conv2}} &= \frac{N \times H \times W \times C^2 \times K^2}{2 * N \times H \times W \times C + C^2 \times K^2}, \\ W_{\text{\_DWConv}} &= N \times H \times W \times C \times (C + K^2), \\ I_{\text{\_DWConv}} &= \frac{N \times H \times W \times C \times (C + K^2)}{(4 * N \times H \times W \times C + C \times K^2 + C^2)} \end{aligned} \tag{2}$$

This trade-off is more complicated in convolution macro structures such as mobile inverted bottleneck conv (MBConv) [53], an important convolution structure in the baseline search space. MBConv is a macro block that includes a expansion layer of 1x1 convolutions, a depthwise convolution, and a projection layer of 1x1 convolutions, together with activation, batch normalization, and skip-connections. A fused variant of MBConv (fused MBConv) combines the depthwise convolutions with the expansion or projection layer as a vanilla convolution. These trade-offs involving W, I, and E (as shown in Equation 1 and 2) are too complicated for manual optimization but are well-suited for NAS to explore. Concretely, fused MBConv has higher operational intensity (good for speed) but higher FLOPs (bad for speed) than MBConv. Thus, fused MBConv can possibly be either faster or slower than MBConv, depending on the shape and size of weights and activations of the macro op. Moreover, the MBConv and fused MBConv contribute differently to the final model accuracy. Thus, we added the fused MBConv into the baseline factorized search space [57]. Although recently NAS for mobile devices [22] also uses a similar op, our work is the first to provide the in-depth analysis and employ such ops into the DC accelerator search spaces.

3.3. Block-wise searchable activation functions

While activation functions have been studied thoroughly for their impact on accuracy [48, 8], their impact on speed is less well understood. With the high computing capacity on TPUs and GPUs, the FLOPs difference among different activation functions is negligible. However, because of the low operational intensity of all activation functions and the shape of rooflines (Figure 2) of TPU and GPU, all activation functions are memory-bound [5] on TPUv3 and GPUv100. These memory-bound activation functions can have large negative performance impact to the end-to-end model speed, because they can drag the overall model into the slope region of the rooflines (where ML model performance is far away from the TPU/GPU peak performance as shown in Figure 2.

The most important optimization for activation functions is fusing [1, 5] an activation function with its associated convolutions to avoid accessing memory just for computing the activation function. Because activation functions (being element-wise operations) usually run on vector units, their execution can be in parallel with the execution of convolutions when convolutions run on matrix unit. In theory, the fused activation functions can be completely hidden by the execution of convolutions. But, in practice, the software stack plays an crucial role for such optimizations, which manifests as important model accuracy and speed trade-offs.

Therefore, we enhance the baseline factorized search space [57, 56] with searchable activation functions, including ReLU and swish. To make the search space manageable, we make the activation searchable at the *block level* in the factorized search space, *i.e.*, different blocks can have different activation functions but all layers within the same block use the same activation function. More details of the TPU/GPU-optimized search space can be found in Appendix A.

4. Latency-aware compound scaling (LACS)

The optimized search space in previous section helps our goal to compose CNN model families with optimal accuracy and inference latency on different DC accelerators as shown in Figure 1. Particularly, our goal can be defined generally with Equation 3.

$$\max_{\mathcal{S}_{h_j}, m_{i,h_j}} \mathcal{O}\Big(\text{Accuracy}(m_{i,h_j}), \text{Latency}_{h_j}(m_{i,h_j})\Big) \tag{3}$$

Given a set of k DC hardware accelerators $h_1, h_2, \ldots h_k \in \mathcal{H}$ of accelerators, we aim at searching for a family of models denoted as $m_{1,h_j}, m_{2,h_j}, \ldots m_{n,h_j} \in \mathcal{M}_{h_j}$. Models in $\mathcal{M}_{h_j}$ specialize in different DC architectures in $\mathcal{H}$ and increase in accuracy at the cost of latency to serve different use cases. The search process is governed by the accuracy and latency *multi-objective* $\mathcal{O}$, evaluating all models in the family of $\mathcal{M}_{h_j}$ on accelerator h_j. The model family $\mathcal{M}_{h_j}$ is composed with a model search space of $\mathcal{S}_{h_j}$ tailored for a given accelerator h_j. In this work, the DC hardware accelerator set $\mathcal{H}$ focuses on TPUs and GPUs.

Even with state-of-the-art NAS and our enhanced search space as described in Section 3, it is too costly to search an entire family of models. Therefore, model scaling is commonly used together with NAS. Model scaling has changed from simple scaling [25, 63, 28] to more sophisticated compound scaling [57]. Compound scaling [57] is essentially a search algorithm as it searches for the best scaling factors for depth, width, and resolution, under a given objective and constraint. However, although the SOTA compound scaling has demonstrated better results than simple scaling, by systematically scaling depth, width, and resolution of CNNs,

[2]For simplicity, we assume that 1) input depth (C_{in}) is the same as output depth (C_{out}), 2) input height and weight (H_{in} and W_{in}) are the same as output height and width (H_{out} and W_{out}), and 3) stride-1 square kernels with size $K \times K$. N is the batch size.

there is still a major hurdle preventing it from harvesting the full potential of hardware and working optimally with NAS. Concretely, by using accuracy as the sole objective[3] during searching for scaling factors, the existing SOTA compound scaling method cannot consider the performance (e.g., inference latency) impact for the resulted model families.

As we seek to design end-to-end model family search as described in Equation 3 and Figure 1, we propose *latency-aware compound scaling (LACS)*. Unlike existing compound scaling that uses accuracy as the sole objective, LACS employs accuracy and latency as the multi-objective when searching for scaling factors of depth, width, and resolution of CNNs for better latency and accuracy trade-offs. Searching for scaling factors with LACS amounts to approximating the solution to the following optimization problem for each accelerator h_j:

$$\begin{aligned} & d_{h_j}, w_{h_j}, r_{h_j} \\ & = \arg\max_{d,w,r} \mathcal{O}\Big(\text{Accuracy}(m_{i,h_j}), \text{Latency}_{h_j}(m_{i,h_j})\Big) \\ & \quad \text{w.r.t.} \quad \text{Latency}(G(m_{i,h_j}, d, w, r)) = T_{m_{i+1,h_j}} \end{aligned} \tag{4}$$

where d, w, r are scaling coefficients for model's depth, width, and input resolution respectively while preserving basic network architecture. $T_{m_{i+1,h_j}}$ is the target latency for the $(i+1)$th model of the family on h_j. d, w, r are determined by a compound coefficient ϕ to scale the network uniformly:

$$d = \alpha^{\phi}, w = \beta^{\phi}, r = \gamma^{\phi}; \quad \text{s.t. } \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \tag{5}$$

ϕ controls how many more resources are available for model scaling. In the original compound scaling that uses accuracy as the sole objective, ϕ means the extra FLOPs for model scaling. Whereas, in our latency-aware compound scaling, ϕ means the latency budget for model scaling, with α, β and γ controlling how the latency budget is allocated to scale depth, width, and resolution, respectively. α, β and γ can be determined by a grid search. LACS is the first multi-objective compound scaling, which enables streamlined integration with multi-objective NAS with the same unified multi-objective reward including both model accuracy and latency as shown in Figure 1.

5. Searching and scaling optimized model families on DC accelerators

This section describes our process of searching and scaling to design model families on TPUs and GPUs with the unified NAS and LACS. We first use NAS with the new search space tailored for DC accelerators to search for the base model. We then use LACS to find scaling factors to compose model families on TPUs and GPUs.

[3]Although compound model scaling also uses FLOPs as the constraints of the scaling factors, the model accuracy is the only objective when searching for the compound scaling factors.

5.1. NAS for base models

We use a NAS infrastructure similar to [56, 57], where we employ the same RNN-based controller. We build an infrastructure to retrieve TPU and GPU hardware latency directly during search and run NAS on TPUv3[20, 15] and GPUv100 [14]. We used data parallelism for distributed training and searching on both TPUs and GPUs.

To establish solid baseline for comparison, we first use the original search space from EfficientNet [57] but replace total computation load (FLOPs) with inference latency of TPUv3 and GPUv100 as the performance objective. Our search finds no model better than EfficientNet-B0 with ReLU. Thus, in the original EfficientNet search space without our TPU/GPU-optimized operations such as space-to-depth/batch, fused MBConv, and searchable activation functions, the FLOPs-optimized models and latency-optimized models converge to the same model architecture as EfficientNet-B0 with ReLU[4]. This observation further necessitates the design of the new search space customized for TPUs and GPUs.

We then performance NAS on our proposed new search space as described in Section 3. We use the same multi-objective reward mechanism as in [56] to ensure fair comparison, although different objective function forms, such as additive forms [10], can potentially produce even better results. The multi-objective reward combines accuracy and latency as $ACCURACY(m) \times \left[\frac{LATENCY(m)}{Target}\right]^w$ to approximate the Pareto-optimal results on both accuracy and latency. The factor w decides the weight of latency in the reward. We re-calibrate the factor w to make the reward design suitable for TPUv3 and GPUv100. Particularly, we use a larger weight factor $w = -0.09$ because model accuracy is less sensitive to latency variations on TPUs and GPUs than on mobile platforms (original $w = -0.07$ in [56]).

Our search produces EfficientNet-X-B0, a fast network on TPUs and GPUs, as shown in Table 1. The EfficientNet-X-B0 demonstrates the effectiveness of the new accelerator-optimized search space, compared to the baseline EfficientNet-B0 [57]. *Firstly*, a space-to-depth op using convolution-2x2 with stride-2 is inserted before the second stage, which can improve the channel depth of subsequent layers to improve speed. *Secondly*, EfficientNet-X-B0 uses hybrid MBConv, with fused-MBConv in stage 4 and 5 and non-fused MBConv in the rest of the stages. *Thirdly*, EfficientNet-X-B0 employs different activation function strategy on TPUs and GPUs. On TPUs, EfficientNet-X-B0 uses swish in stages with fused-MBConv but ReLU in stages with MBConv. On GPUs, EfficientNet-X-B0 selects ReLU for all stages. *Lastly*, NAS designs EfficientNet-X-B0

[4]Note that when searching on the original EfficientNet search space, we always used ReLU because the original EfficientNet search space did not support searching for activation functions. In the original EfficientNet [57], EfficientNet-B0 was searched with ReLU and manually set to use swish for all layers after the search was done

Table 1: EfficientNet-X-B0 architecture. The architecture includes multiple stages, with each row representing a stage. Each stage includes operators, number of repeated layers denoted as #L, (input/hidden) resolution, output channel size denoted as #OC, squeeze-and-excite (SE) ratio [30], and activation functions denoted as AF. Activation functions differ on TPUs from GPUs.

Stage	Operator	Resolution	#OC	#L	SE	AF(TPU/GPU)
1	Conv3x3	224×224	32	1	N/A	swish/ReLU
2	Conv2x2 for reshaping†	112×112	128	1	N/A	ReLU/ReLU
3	MBConv1, k3x3	56×56	64	1	1	ReLU/ReLU
4	Fused MBConv6, k3x3	56×56	24	2	0.5	swish/ReLU
5	Fused MBConv6, k5x5	56×56	40	2	0.25	swish/ReLU
6	MBConv6, k3x3	28×28	80	3	0.25	ReLU/ReLU
7	MBConv6, k5x5	14×14	112	3	0.25	ReLU/ReLU
8	MBConv6, k5x5	14×14	192	4	0.25	ReLU/ReLU
9	MBConv6, k3x3	7×7	320	1	0.25	ReLU/ReLU
10	Conv1x1 & Pooling & FC	7×7	1280	1	N/A	ReLU/ReLU

with bigger squeeze-and-excite layers than EfficientNet-B0.

All the new model architectures in EfficientNet-X-B0 show the effectiveness of the DC accelerator optimized search space. We use the selection of the activation functions as an example to shed more light. The usage of swish and ReLU in EfficientNet-X-B0 is the opposite of that in mobilenetv3 [27]. Swish has ∼4X more FLOPs than ReLU, making it very expensive on mobile platforms. MobilenetV3 uses swish only in later layers, because the cost of applying nonlinearity decreases in deeper layers of the network.

However, as describe in Section 3, because of the high computing capacity of TPUs and GPUs, the FLOPs differences between swish and ReLU are negligible. Instead, activation functions are optimized with fusion and run on vector units in parallel with convolutions that usually run on matrix units. However, the software stack on GPUs only fuses ReLU (but not swish) with associated convolutions, which leads to significant slow down for GPU models with swish. As a result, EfficientNet-X-B0 on GPU chooses ReLU for all layers. In contrast, since TPU has swish fused with convolutions through XLA [1], EfficientNet-X-B0 uses swish in many layers. Our profiling results with Cloud TPU Profiler [6] reveal that depthwise convolutions on TPU run on vector units[5] instead of matrix units. Thus, severe contention on vector units happens between depthwise convolutions and swish, as swish has 4X more FLOPs than ReLU despite its benefits in improving model accuracy. Therefore, when searching on TPUs with our new search space, NAS automatically pairs ReLU with stages containing depthwise convolutions to avoid competition on vector units. Appendix B shows more ablation studies on EfficientNet-X-B0.

5.2. Scaling to form model families with LACS

With the searched base model EfficientNet-X-B0, we use LACS to search for scaling factors to build the model family. As described in Section 4, we perform Pareto frontier search to find best α, β, and γ. We start with initial grid search

[5]Coincidentally, recent experiment [2] discovers the similar behavior on GPU. Depthwise convolutions run in vector units, *i.e.*, CUDA cores, instead of the tensor cores on GPUs.

Table 2: Comparison of LACS scaling factors with existing SOTA compound scaling using accuracy as the sole objective (*i.e.*, EfficiencNet's scaling factors). α, β, and γ are the base term of the exponential scaling for depth, width, and resolution respectively, as shown in Equation 1.

Scaling Type	α (depth)	β (width)	γ (resolution)
Accuracy-only	1.2	1.1	1.15
LACS on GPU	1.29	1.16	1.07
LACS on TPU	1.29	1.14	1.08

for coefficient triplets of α, β, and γ using the same multi-objective (*i.e.*, $ACCURACY(m) \times \left[\frac{LATENCY(m)}{Target}\right]^w$) as used in NAS when searching for the base model. We search on TPUv3 and GPUv100 and find different optimal scaling coefficients as shown in Table 2.

LACS discovers network depth should grow much faster than image resolution, which is quite different from the previous SOTA compound scaling results using accuracy as the single objective. Faster increase on network depth than on image resolutions can slow down the memory inflation due to activation and intermediate tensors, which improves model speed by making a model more compute bound than memory bound. As shown in Section 2, DC accelerators prefer models to be compute-bound to achieve high performance.

We also perform direct search on TPUv3 and GPUv100 with the same latency target as EfficientNet-X-B1 and find the same model architectures as obtained by LACS, which confirms that LACS can find the same model as the direct multi-objective NAS when given the same latency target, but with much fewer accelerator resources. Appendix C shows more ablation studies on LACS.

6. Experiments

We present the accuracy and performance results on the new EfficientNet-X model family on TPUs and GPUs, to demonstrate the effectiveness of the unified NAS and LACS method. Table 3 shows the speed and accuracy on ImageNet [52] of EfficientNet-X models and comparisons with other SOTA CNN models, where a few key observations can be made. *First*, EfficientNet-X models are the fastest among each model group on TPUs and GPUs, with comparable accuracy. Specifically, EfficientNet-X models are up to more than 2X faster than EfficientNet. EfficientNet-X is on average (geomean) 82% and 48% faster than RegNet and ResNeSt respectively on GPUv100 and 7X; it is 48% faster than RegNet and ResNeSt respectively on TPUv3. *Second*, all models except for Efficient-X models in Table 3 are polarized. On one extreme, the EfficientNet family has the fewest FLOPs but the lowest operational intensity I. On the other extreme, other models such as ResNet and Inception families have the highest operational intensity but most FLOPs. While lower FLOPs improves inference speed, lower operational intensity hurts inference speed. In contrast, the

Table 3: EfficientNet-X inference speed and accuracy results on ImageNet on TPUv3 and GPUv100. ConvNets with similar top-1 accuracy are grouped together. *Original reported model accuracies in papers are used in the comparisons. †Following common practices, #FLOPs refer to #multiply-and-add operations. ‡E is the execution efficiency measured on TPUv3, w.r.t to roofline instead of peak hardware FLOPs/sec as shown in Equation 1. Only in the compute-bound region as shown in Figure 2, the roofline and hardware peak hardware FLOPs/sec are the same. §The inference latency are measured for inferencing 128 images on TPUv3 and GPUv100, with mini batch size of 128. All the measured speed is verified to be the same or faster than the reported results in original papers with the same batch size to ensure fair and correct measurements. Note that the results are to demonstrate the effectiveness of our unified search and scaling method on different DC accelerators. And direct comparing TPU and GPU results is not meaningful and beyond the scope of this paper, because we focus on evaluating the model architecture themselves on different DC accelerators and run models directly on both GPUs and TPUs without extra offline model optimizations (*e.g.*, TensorRT [3] and model tuning [50]).

Models	**Acc.***	**#Params (Million)**	**#FLOPs**† **(Billion)**	**I (Ops/Byte)**	**E**‡	**Inference Latency**§**(ms) (TPUv3 / GPUv100)**
EfficientNet-X-B0	77.3%	7.6	0.91	63.8	57.3%	**8.71 / 22.5**
EfficientNet-B0 [57]	77.3%	5.3	0.39	19.7	52.4%	13.4 / 38.1
ResNet-50 [25]	76.0%	26	4.1	122.5	57.2%	35.1 / 35.6
RegNetY-800MF [47]	76.3%	6.3	0.8	12.7	30%	45.1 / 33.9
EfficientNet-X-B1	79.4%	10.4	1.58	65.5	59.2%	**13.6 / 34.4**
EfficientNet-B1	79.2%	7.8	0.70	21.4	51.3%	22.3 / 60.5
Inception-v3 [55]	78.8%	24	5.7	94.6	34.5%	104.8 /55.6
RegNetY-4.0GF [47]	79.4%	26	4.0	19.4	29.2%	109.5 / 75.1
EfficientNet-X-B2	80.0%	11.5	1.89	73.0	54.8%	**15.7 / 45.5**
EfficientNet-B2	80.3%	9.2	1.0	24.1	48.8%	29.8 / 77.2
Inception-v4 [54]	80.0%	48	13	148.5	35.3%	75.1 / 119.9
RegNetY-8.0GF [47]	79.9%	39.2	8.0	27.9	32.4%	190.5 / 122.1
EfficientNet-X-B3	81.4%	16	4.3	84.0	51.2%	**31.9 / 66.6**
EfficientNet-B3	81.7%	12	1.8	26.1	51.3%	48.1 / 128.8
EfficientNet-X-B4	83.0%	34	10.4	101.5	47.7%	**64.9 / 149.2**
EfficientNet-B4	83.0%	19	4.2	31.29	47.8%	102.6 / 310.7
NASNet-A [66]	82.7%	89	24	55.2	43.8%	269.5 / 481.2
ResNeSt-101 [63]	83.0%	48	13	71.7	28.1%	92.3 / 149.4
EfficientNet-X-B5	83.7%	60	22.2	126.1	47.8%	**125.9 / 290.2**
EfficientNet-B5	83.7%	30	9.9	39.7	46.8%	192.5 / 640.1
ResNeSt-200 [63]	83.9%	70	36.3	68.7	69.9%	244.3 / 415.6
EfficientNet-X-B6	84.4%	137	52	167.5	36.2%	**258.1 / 467.2**
EfficientNet-B6	84.4%	43	19	43.9	45.0%	334.2 / 1040.6
EfficientNet-X-B7	84.7%	199	93	194.3	39.4%	**396.1 / 847.7**
EfficientNet-B7	84.7%	66	37	48.3	43.4%	621.4 / 1471.3
ResNeSt-269 [63]	84.5%	111	77	72.9	70.2%	501.9 / 864.9

EfficientNet-X models strike a balance between computation load and computation rate, having both FLOPs and operational intensity in the middle between the two extremes, which makes EfficientNet-X to be the fastest in each group.

Figure 3 shows the speedup details due to our new search and scaling method. Overall, EfficientNet-X achieves up to 2X+ speedup on TPUv3 and GPUv100 over EfficientNet, with geometric mean speedup as 56% and 91% on TPUs and GPUs respectively. Figure 3 also shows the ablation study on the speedup breakdown due to NAS with the new search space and LACS. EfficientNet-X-single-objective-scaling composes the model family using EfficientNet-X-B0 as the base model but the EfficientNet's orginal scaling factors that are obtained by single-objective compound scaling with accuracy as the sole objective. Thus, the speedup on EfficientNet-X-B0 over EfficientNet-B0 shows the benefits of the NAS with new search space, and the relative speedup of EfficientNet-X over EfficientNet-X-single-objective-scaling in Figure 3 indicates the benefits of LACS over previous SOTA compound scaling with accuracy as the only objective. Concretely, NAS with new search space generates ∼50% speedup on TPUv3 and GPUv100, respectively. LACS further increases performance by 14% and 25% average (geometric mean) on TPUs and GPUs respectively, atop

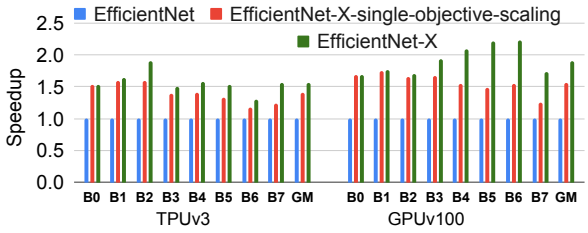

Figure 3: Speedup of EfficientNet-X and EfficientNet-X-single-objective-scaling over the baseline EfficientNet. EfficientNet-X-single-objective-scaling forms the model family use EfficientNet-X-B0 as the base model but uses original EfficientNet's scaling factors that are obtained by compound scaling with accuracy as the sole objective. GM is geometric mean.

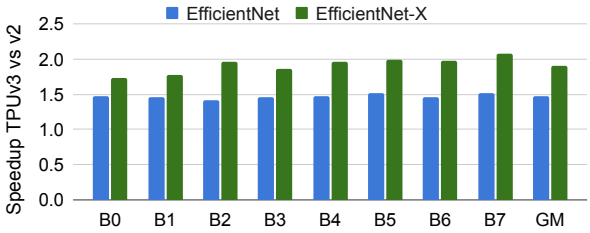

Figure 4: Speedup of EfficientNet-X and EfficientNet when migrating from TPUv2 to TPUv3 with 3X hardware peak performance. GM is geometric mean.

the speedup due to the new search space. The more detailed ablation studies on search space and LACS can be found in Appendix B and C respectively.

Moreover, the DC accelerator-friendliness of EfficientNet-X generalizes well across accelerator generations. TPUv3 has 3X of the TPUv2's peak performance. When migrating from TPUv2 to TPUv3 as shown in Figure 4, EfficientNet-X models achieve ∼1.9X average (geometric mean) speedup while EfficientNet models only achieve ∼1.5X speedup. In other words, EfficientNet-X materializes ∼30% better speedup than EfficientNet when migrating from TPUv2 to TPUv3, demonstrating good generality.

All these results demonstrate the effectiveness of our method. Specifically, our method, including NAS with the search space optimized for DC-accelerators and LACS, emphasizes on simultaneously optimizing total computation W, operational intensity I, and execution efficiency E.

We also perform search and model scaling on Xeon Platinum 8180 CPUs that are representative server-class CPUs in datacenters. The results on CPUs are similar to that on DC accelerators when the vector units/instructions [4] are enabled on CPUs. However, when the vector units/instructions are disabled, the results on CPUs are very different. The detailed results on CPUs can be found in Appendix D.

7. Related work

Neural Architecture Search (NAS) attempts to automate the design process of machine learning models with reinforcement learning [65, 66], evolutionary search [49], differentiable search [37, 17], and other methods [40, 33]. Recent work in NAS has also reduced search costs [46, 36, 64] and improved inference efficiency [56, 61, 57, 41, 35]. When designing fast models for inference with NAS, previous work employed multi-objective search [56, 18, 13, 29, 64, 27, 12, 21, 39, 16] to consider accuracy together with performance/efficiency. However, their methods only passively use high level signals such as model size and latency.

Targeted ML optimizations are also used extensively to improve model accuracy and efficiency trade-offs. These targeted optimizations include automated approaches such as model pruning and quantization [23, 26, 58, 44, 38, 34, 19, 31, 65] as well as manual optimizations on specific platforms especially mobile devices [28, 53].

Initial model scaling involves taking a fixed architecture and individually increasing depth [25] and width [62] in separation or together [28, 63]. Further work in compound scaling yielded model families varying in depth, width, and resolution simultaneously and systematically [57]. Scaling is also more recently used in constructing larger models in conjunction with NAS [66, 57].

Specialized datacenter accelerators have been playing a critical role in powering machine learning. These accelerators, including TPUs [15, 32] and GPUs [14, 43], provide the computing power for both training and inference at scale.

8. Conclusions

This work presents a new method to search for CNN model families targeting datacenter accelerators for high accuracy and efficient inference. We first provide analysis to show the root cause of FLOPs-latency nonproportionality and ways to improve CNN performance on DC accelerators. Guided by the insights gained from our analysis, the new search method incorporates a NAS search space tailored for DC accelerators and a new scaling approach called latency-aware compound scaling. Our new method provides the search and scaling for model families with critical visibility into accelerator details, and compose model families with optimized FLOPs, operational intensity, and efficiency to achieve better accuracy and speed. The resulted EfficientNet-X model family achieves up to 2X+ faster speed and comparable accuracy to SOTA model families on TPUv3 and GPUv100. EfficientNet-X also achieves 30% better speedup when migrating from TPUv2 to TPUv3, demonstrating the generality of our method across different accelerator generations. These results highlight the impressive possibilities available through careful accelerator-aware optimizations on NAS and compound scaling for increasingly demanding computer vision models on emerging DC accelerators.

References

[1] Accelerated linear algebra (xla): Optimizing compiler for machine learning. *https://www.tensorflow.org/xla/*. 4, 6

[2] Depth-wise separable convolutions: Performance investigation. *https://tlkh.dev/depsep-convs-perf-investigations/*. 6

[3] Developer guide: Nvidia deep learning tensorrt. *https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html*. 7

[4] Intel® Advanced Vector Extensions 512 (Intel® AVX-512). *https://www.intel.com/content/www/us/en/architecture-and-technology/avx-512-overview.html*. 8, 14

[5] Nvidia deep learning performance: Activation. `https://docs.nvidia.com/deeplearning/performance/dl-performance-memory-limited/index.html`. 4

[6] Using cloud tpu tools. *https://cloud.google.com/tpu/docs/cloud-tpu-tools*. 6

[7] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 3

[8] Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding deep neural networks with rectified linear units. In *ICLR*, 2018. 4

[9] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *International Conference on Learning Representations*, 2017. 1

[10] Gabriel Bender, Hanxiao Liu, Bo Chen, Grace Chu, Shuyang Cheng, Pieter-Jan Kindermans, and Quoc Le. Can weight sharing outperform random architecture search? an investigation with TuNAS. 2020. 5

[11] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Reinforcement learning for architecture search by network transformation. *AAAI*, 2018. 1

[12] Han Cai, Chuang Gan, and Song Han. Once for all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019. 8

[13] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *ICLR*, 2019. 8

[14] Jack Choquette, Olivier Giroux, and Denis Foley. Volta: Performance and programmability. *IEEE Micro*, 2018. 1, 2, 5, 8

[15] Jeffrey Dean. The deep learning revolution and its implications for computer architecture and chip design, 2019. 1, 2, 5, 8

[16] Jin-Dong Dong, An-Chieh Cheng, Da-Cheng Juan, Wei Wei, and Min Sun. Ppp-net: Platform-aware progressive search for pareto-optimal neural architectures. 2018. 8

[17] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE Conference on computer vision and pattern recognition*, pages 1761–1770, 2019. 8

[18] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. *arXiv preprint arXiv:1804.09081*, 2018. 8

[19] Amir Gholami, Kiseok Kwon, Bichen Wu, Zizheng Tai, Xiangyu Yue, Peter Jin, Sicheng Zhao, and Kurt Keutzer. Squeezenext: Hardware-aware neural network design. *ECV Workshop at CVPR'18*, 2018. 8

[20] Google. Cloud TPU. *https://cloud.google.com/tpu*. 1, 2, 5

[21] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *European Conference on Computer Vision*, pages 544–560. Springer, 2020. 8

[22] Suyog Gupta and Mingxing Tan. Efficientnet-edgetpu: Creating accelerator-optimized neural networks with automl. 2019. 4

[23] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both Weights and Connections for Efficient Neural Networks. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2015. 8

[24] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, J. Law, K. Lee, J. Lu, P. Noordhuis, M. Smelyanskiy, L. Xiong, and X. Wang. Applied machine learning at facebook: A datacenter infrastructure perspective. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 620–629, 2018. 14

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CVPR*, pages 770–778, 2016. 1, 4, 7, 8

[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. *ECCV*, 2018. 8

[27] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. *ICCV*, 2019. 6, 8

[28] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 4, 8

[29] Chi-Hung Hsu, Shu-Huan Chang, Da-Cheng Juan, Jia-Yu Pan, Yu-Ting Chen, Wei Wei, and Shih-Chieh Chang. MONAS: Multi-objective neural architecture search using reinforcement learning. *arXiv preprint arXiv:1806.10332*, 2018. 8

[30] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *CVPR*, 2018. 6

[31] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016. 8

[32] Norman P. Jouppi, Cliff Young, Nishant Patil, David A. Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc

Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, Richard C. Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. In *ISCA*, 2017. 1, 2, 8

[33] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric Xing. Neural architecture search with bayesian optimisation and optimal transport. *arXiv preprint arXiv:1802.07191*, 2018. 8

[34] Sheng Li, Jongsoo Park, and Ping Tak Peter Tang. Enabling sparse winograd convolution by native pruning. *CoRR*, abs/1702.08597, 2017. 8

[35] Xin Li, Yiming Zhou, Zheng Pan, and Jiashi Feng. Partial order pruning: for best speed/accuracy trade-off in neural architecture search. In *Proceedings of the IEEE Conference on computer vision and pattern recognition*, pages 9145–9153, 2019. 8

[36] Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. *ECCV*, 2018. 8

[37] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 8

[38] Xingyu Liu, Jeff Pool, Song Han, and William J. Dally. Efficient sparse-winograd convolutional neural networks. *ICLR*, 2018. 8

[39] Zhichao Lu, Ian Whalen, Vishnu Boddeti, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, and Wolfgang Banzhaf. Nsga-net: neural architecture search using multi-objective genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 419–427, 2019. 8

[40] Renqian Luo, Fei Tian, Tao Qin, and Tie-Yan Liu. Neural architecture optimization. *arXiv preprint arXiv:1808.07233*, 2018. 8

[41] Li Lyna Zhang, Yuqing Yang, Yuhang Jiang, Wenwu Zhu, and Yunxin Liu. Fast hardware-aware neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 692–693, 2020. 8

[42] Norman P. Jouppi, Doe Hyun Yoon, George Kurian, Sheng Li, Nishant Patil, James Laudon, Cliff Young, and David Patterson. A domain-specific supercomputer for training deep neural networks. In *Communications of the ACM*, volume 67, pages 67–78, 2020. 1, 2

[43] NVIDIA. Nvidia a100 tensor core gpu architecture. *White Paper*, 2020. 1, 2, 8

[44] Jongsoo Park, Sheng Li, Wei Wen, Hai Li, Yiran Chen, and Pradeep Dubey. Holistic sparsecnn: Forging the trident of accuracy, speed, and size. *ICLR*, 2017. 8

[45] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 3

[46] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *ICML*, 2018. 8

[47] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. *CVPR*, 2020. 7

[48] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2018. 4

[49] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *AAAI*, 2019. 8

[50] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, Ramesh Chukka, Cody Coleman, Sam Davis, Pan Deng, Greg Diamos, Jared Duke, Dave Fick, J. Scott Gardner, Itay Hubara, Sachin Idgunji, Thomas B. Jablin, Jeff Jiao, Tom St. John, Pankaj Kanwar, David Lee, Jeffery Liao, Anton Lokhmotov, Francisco Massa, Peng Meng, Paulius Micikevicius, Colin Osborne, Gennady Pekhimenko, Arun Tejusve Raghunath Rajan, Dilip Sequeira, Ashish Sirasao, Fei Sun, Hanlin Tang, Michael Thomson, Frank Wei, Ephrem Wu, Lingjie Xu, Koichi Yamada, Bing Yu, George Yuan, Aaron Zhong, Peizhao Zhang, and Yuchen Zhou. Mlperf inference benchmark, 2020. 7

[51] Tal Ridnik, Hussam Lawen, Asaf Noy, Emanuel Ben Baruch, Gilad Sharir, and Itamar Friedman. Tresnet: High performance gpu-dedicated architecture, 2020. 1, 2

[52] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 6

[53] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *CVPR*, 2018. 4, 8

[54] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. *AAAI*, 4:12, 2017. 7

[55] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CVPR*, pages 2818–2826, 2016. 7

[56] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan,

Mark Sandler, Andrew Howard, and Quoc V. Le. MnasNet: Platform-aware neural architecture search for mobile. *CVPR*, 2019. 4, 5, 8

[57] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *ICML*, 2019. 1, 2, 4, 5, 7, 8, 12, 13

[58] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning Structured Sparsity in Deep Neural Networks. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2016. 8

[59] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Communications of the ACM*, 52(4):65–76, Apr. 2009. 2

[60] Samuel Williams, Charlene Yang, and Yunsong Wang. Roofline performance model for hpc and deep-learning applications. In *GPU Technology Conference (GTC)*, 2020. 2

[61] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019. 8

[62] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *BMVC*, 2016. 8

[63] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Zhi Zhang, Haibin Lin, Yue Sun, Tong He, R. Manmatha Jonas Mueller, Mu Li, and Alexander Smola. Resnest: Split-attention networks. *https://arxiv.org/abs/2004.08955*, 2020. 4, 7, 8

[64] Yanqi Zhou, Siavash Ebrahimi, Sercan Ö Arık, Haonan Yu, Hairong Liu, and Greg Diamos. Resource-efficient neural architect. *arXiv preprint arXiv:1806.07912*, 2018. 8

[65] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *ICLR*, 2017. 1, 8

[66] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. *CVPR*, 2018. 1, 7, 8