LANA: Latency Aware Network Acceleration

Pavlo Molchanov[1], Jimmy Hall[2], Hongxu Yin[1], Jan Kautz[1], Nicolo Fusi[2], and Arash Vahdat[1]

[1] NVIDIA
[2] Microsoft Research

Abstract. We introduce latency-aware network acceleration (LANA)–an approach that builds on neural architecture search technique to accelerate neural networks. LANA consists of two phases: in the first phase, it trains many alternative operations for every layer of a target network using layer-wise feature map distillation. In the second phase, it solves the combinatorial selection of efficient operations using a novel constrained integer linear optimization (ILP) approach. ILP brings unique properties as it (i) performs NAS within a few seconds to minutes, (ii) easily satisfies budget constraints, (iii) works on the layer-granularity, (iv) supports a huge search space $O(10^{100})$, surpassing prior search approaches in efficacy and efficiency. In extensive experiments, we show that LANA yields efficient and accurate models constrained by a target latency budget, while being significantly faster than other techniques. We analyze three popular network architectures: EfficientNetV1, EfficientNetV2 and ResNeST, and achieve accuracy improvement (up to 3.0%) for all models when compressing larger models. LANA achieves significant speed-ups (up to 5×) with minor to no accuracy drop on GPU and CPU. Project page: https://bit.ly/3Oja2IF

1 Introduction

In many applications, we may have access to a neural network that satisfies desired performance needs in terms of accuracy but is computationally too expensive to deploy. The goal of hardware-aware network acceleration [59, 103, 67, 28, 5, 13] is to accelerate a given neural network such that it meets efficiency criteria on a device without sacrificing accuracy dramatically. Network acceleration plays a key role in reducing the operational cost, power usage, and environmental impact of deploying deep neural networks in real-world applications.

Given a trained neural network (teacher, base model), the current network acceleration techniques can be grouped into: *(i) pruning* that removes inactive neurons [51, 36, 20, 8, 49, 48, 45, 43, 98, 25, 26, 18, 29, 104, 35, 39, 94, 27, 44, 53, 22, 33, 19, 97, 24, 47, 4], (ii) *compile-time optimization* [62] *or kernel fusion* [81, 15, 14, 100] that combines multiple operations into an equivalent operation, (iii) *quantization* that reduces the precision in which the network operates at [83, 16, 65, 7, 11, 55, 86, 101, 34], and *(iv) knowledge distillation* that distills knowledge from a larger *teacher* network into a smaller *student* network [31, 64, 91, 95, 46, 52, 1]. The approaches within (i) to (iii) are restricted to the underlying network operations and they do not change the architecture. Knowledge distillation changes the network architecture from teacher to student, however, the student design requires domain knowledge and is done usually manually.

In this paper, we propose latency-aware network transformation (LANA), a network acceleration framework that replaces inefficient operations in a given trained network with more efficient counterparts (see Fig 1). Given a convolutional network as target and base model to accelerate, we formulate the problem as searching in a large pool of candidate operations to find efficient operations for different layers of the base model (teacher). The search problem is combinatorial in nature with a space that grows exponentially with the depth of the network. To solve this problem, we can turn to neural architecture search (NAS) [105, 106, 70, 6, 57, 78], which has been proven successful in discovering novel architectures. However, existing NAS solutions are computationally expensive, and usually handle only a small number of candidate operations (ranging from 5 to 15) in each layer and they often struggle with larger candidate pools.

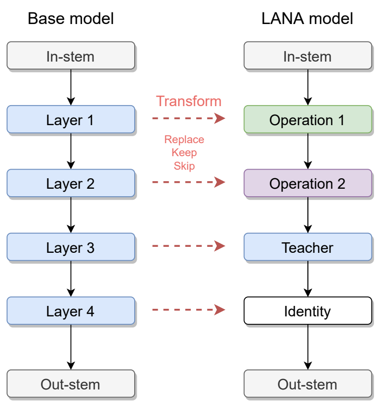


Fig. 1: LANA is a post-training model optimization method that keeps, replaces or skips layers of the trained base model (teacher).

To tackle the search problem with a large number of candidate operations in an efficient and scalable way, we propose a two-phase approach. In the first phase, we define a large candidate pool of operations ranging from classic residual blocks [21] to recent blocks [17, 15, 66, 2], with varying hyperparameters. Candidate operations are pretrained to mimic the teacher's operations via a simple layer-wise optimization. Distillation-based pretraining enables a very quick preparation of all candidate operations, offering a much more competitive starting point for subsequent searching.

In the second phase, we search among the pre-trained operations as well as the teacher's own operations to construct an efficient network. Since our operation selection problem can be considered as searching in the proximity of the teacher network in the architecture space, we assume that the accuracy of a candidate architecture can be approximated by the teacher's accuracy and a simple linear function that measures changes in the accuracy for individual operations. Our approximation allows us to relax the search problem into a constrained integer linear optimization problem that is solved in a few seconds. As we show extensively in our experiments, such relaxation can drastically cut down on the cost of our search and it can be easily applied to a huge pool of operations (197 operations per layer), while offering improvements in model acceleration by a large margin.

In summary, we make the following contributions: **(i)** We propose a simple two-phase approach for accelerating a teacher network using NAS-like search. **(ii)** We propose an effective search algorithm using constrained integer optimization that can find an architecture in seconds tailored to our setting where a fitness measure is available for each operation. **(iii)** We examine a large pool of operations including the recent vision transformers and new variants of convolutional networks. We provide insights into the operations selected by our framework and into final model architectures.

1.1 Related Work

Since our goal is to accelerate a trained model by modifying architecture, in this section we focus on related NAS-based approaches.

Hardware-aware NAS: The goal of hardware-aware NAS is to design efficient and accurate architectures from scratch while targeting a specific hardware platform. This has been the focus of an increasingly large body of work on multiobjective neural architecture search [6, 71, 87, 92, 79, 93, 88, 78]. The goal here is to solve an optimization problem maximizing accuracy while meeting performance constraints specified in terms of latency, memory consumption or number of parameters. Given that the optimization problem is set up from scratch for each target hardware platform, these approaches generally require the search to start from scratch for every new deployment target (*e.g.*, GPU/CPU family) or objective, incurring a search cost that increases linearly as the number of constraints and targets increases. [5] circumvents this issue by training a supernetwork containing every possible architecture in the search space, and then applying a progressive shrinking algorithm to produce multiple high-performing architectures. This approach incurs a high pretraining cost, but once training is complete, new architectures are relatively inexpensive to find. On the other hand, the high computational complexity of pretraining limits the number of operations that can be considered. Adding new operations is also costly, since the supernetwork must be pretrained from scratch every time for a new operation. Recent work [56] also explores prioritized paths within the supernetwork to help guide the learning of weak subnets through distillation, such that all blocks can be trained simultaneously and the strongest path constitutes a final architecture. Despite remarkable insights the searching still imposes GPU days to converge towards a final strong path overseeing a small search space.

Teacher-based NAS: Our work is more related to the line of work that focuses on modifying *existing* architectures. Approaches in this area build on teacher-student knowledge distillation, performing multiobjective NAS on the student to mimic the teacher network.

The AKD approach [42] applies knowledge distillation at the network level, training a reinforcement learning agent to construct an efficient student network given a teacher network and a constraint and then training that student from scratch using knowledge distillation. DNA [38] and DONNA [50] take a more fine-grained approach, dividing the network into a small number of blocks, each of which contain several layers. During knowledge distillation, they both attempt

Table 1: Related method comparison. Time is mentioned in GPU hours by `h`, or ImageNet epochs by `e`. Our method assumes 197 candidate operations for the full pool, and only 2 (teacher and identity) for single shot mode. L is the number of target architectures.

Method	Knowledge Distillation	Diverse Operators	Design Space Size	Pretrain Cost	Search Cost	Train Cost	Total Cost
				To Train L Architectures			
Once-For-All [5]	None		$> O(10^{19})$	1205e	40h	75eL	1205e + 75eL
AKD [42]	Network	✓	$> O(10^{13})$	0	50000eL	400eL	50400eL
DNA [38]	Block		$> O(10^{15})$	320e	14hL	450eL	320e + 450eL
DONNA [50]	Block	✓	$> O(10^{13})$	1920e[3]	1500e + $\leqslant$ 1hL	50eL	3420e + 50eL
Cream [56]	Network	✓	$> O(10^{16})$	0	120eL	500eL	620eL
This Work	Layer	✓	$> O(10^{100})$	197e	<1hL	100eL	197e + 100eL
This Work - single shot	None		$> O(10^{2})$	0	∼0	100eL	100eL

to have student blocks mimic the output of teacher blocks, but [38] samples random paths through a mix of operators in each block, whereas [50] trains several candidate blocks with a repeated single operation for each teacher block. They then both search for an optimal set of blocks, with DNA [38] using a novel ranking algorithm to predict the best set of operations within each block, and then applying a traversal search, while [50] trains a linear model that predicts accuracy of a set of blocks and use that to guide an evolutionary search. While both methods deliver impressive results, they differ from our approach in important ways. DNA [38] ranks each path within a block, and then use this ranking to search over the blocks, relying on the low number of blocks to accelerate search. DONNA [50] samples and finetunes 30 models to build a linear accuracy predictor, which incurs a significant startup cost for search. In contrast, we formulate the search problem as an integer linear optimization problems that can be solved very quickly for large networks and large pool of operations.

Table 1 compares our work to these works in detail. We increase the granularity of network acceleration, focusing on each *layer* individually instead of blocks as done in DNA and DONNA. The main advantage of focusing on layers is that it allows us to accelerate the teacher by simply *replacing* inefficient layers whereas blockwise algorithms such as DNA and DONNA require *searching* for an efficient subnetwork that mimics the whole block. The blockwise search introduces additional constraints. For example, both DNA and DONNA enumerate over different depth values (multiplying the search space) while we reduce depth simply using an identity operation. Additionally, DONNA assumes that the same layer in each block is repeated whereas we have more expressivity by assigning different operation to different layers. The expressivity can be seen from the design space size in Table 1 in which our search space is orders of magnitude larger. On the other hand, this extremely large space necessitate the development of a highly efficient search method based on integer linear optimization (presented in Section 2). As we can see from Table 1, even with significantly larger search space, our total cost is lower than prior work. We additionally introduce one-shot formulation when LANA transforms an architecture by simply skipping blocks (cells). This setting imposes no pretraining and search cost is negligibly small.

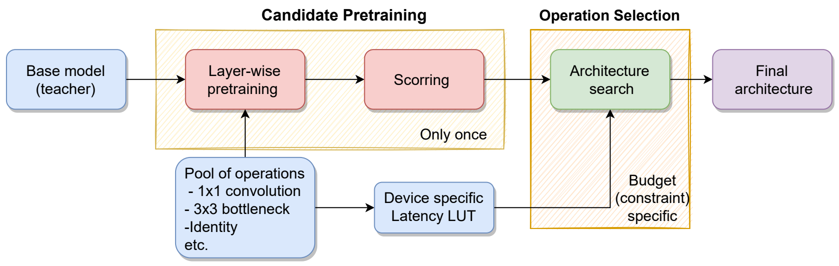


Fig. 2: **LANA framework:** A set of candidate operations is pretrained to mimic layers of the trained base (teacher) model. Then, operations are scored on their goodness metric to approximate the teacher. These 2 steps are only performed once for a given base model. Finally, an architecture search is performed to select operations per every layer to satisfy a predefined budget constraint.

2 Method

Our goal in this paper is *to accelerate* a given pre-trained teacher/base network by replacing its inefficient operations with more efficient alternatives. Our method, visualized in Fig. 2, is composed of two phases: (i) *Candidate pretraining phase* (Sec. 2.1), in which we use distillation to train a large set of operations to approximate different layers in the original teacher architecture; and (ii) *Operation selection phase* (Sec. 2.2), in which we search for an architecture composed of a combination of the original teacher layers and pretrained efficient operations via linear optimization.

2.1 Candidate Pretraining Phase

We represent the teacher (base) network as the composition of N teacher operations by $\mathcal{T}(x) = t_N \circ t_{N-1} \circ \ldots \circ t_1(x)$, where x is the input tensor, t_i is the i^{th} operation (i.e., layer) in the network. We then define the set of *candidate student operations* $\bigcup_{i=1}^{N} \{s_{ij}\}_{j=1}^{M}$, which will be used to approximate the teacher operations. Here, M denotes the number of candidate operations per layer. The student operations can draw from a wide variety of operations – the only requirement is that all candidate operations for a given layer must have the same input and output tensor dimensions as the teacher operation t_i. We denote all the parameters (e.g., trainable convolutional filters) of the operations as $\mathbf{W} = \{w_{ij}\}_{i,j}^{N,M}$, where w_{ij} denotes the parameters of the student operation s_{ij}. We use a set of binary vectors $\mathbf{Z} = \{\mathbf{z}_i\}_{i=1}^{N}$, where $\mathbf{z}_i = \{0,1\}^M$ is a one-hot vector, to represent operation selection parameters. We denote the candidate network architecture specified by $\mathbf{Z}$ using $\mathcal{S}(x; \mathbf{Z}, \mathbf{W})$.

The problem of optimal selection of operations is often tackled in NAS. This problem is usually formulated as a bi-level optimization that selects operations and optimizes their weights jointly [41, 105].

Finding the optimal architecture in hardware-aware NAS reduces to:

$$\min_{\mathbf{Z}} \min_{\mathbf{W}} \underbrace{\sum_{(x,y)\in X_{tr}} \mathcal{L}\big(\mathcal{S}(x;\mathbf{Z},\mathbf{W}),y\big)}_{\text{objective}}, \tag{1}$$

$$\text{s.t.} \quad \underbrace{\sum_{i=1}^{N} \mathbf{b}_i^T \mathbf{z}_i \leqslant \mathcal{B}}_{\text{budget constraint}} \;\; ; \quad \underbrace{\mathbf{1}^T \mathbf{z}_i = 1 \;\forall\; i \in [1..N]}_{\text{one op per layer}}$$

where $\mathbf{b}_i \in \mathbb{R}_+^M$ is a vector of corresponding cost of each student operation (latency, number of parameters, FLOPs, etc.) in layer i. The total budget constraint is defined via scalar $\mathcal{B}$. The objective is to minimize the loss function $\mathcal{L}$ that estimates the error with respect to the correct output y while meeting a budget constraint. In general, the optimization problem in Eq. 1 is an NP-hard combinatorial problem with an exponentially large state space (i.e., M^N). The existing NAS approaches often solve this optimization using evolutionary search [60], reinforcement learning [105] or differentiable search [41].

However, the goal of NAS is to find an architecture in the whole search space from scratch, whereas our goal is to improve efficiency of a given teacher network by replacing operations. Thus, our search can be considered as searching in the architecture space in the proximity of the already trained model. That is why we assume that the functionality of each candidate operation is also similar to the teacher's operation, and we train each candidate operation to mimic the teacher operation

using layer-wise feature map distillation with the mean squared error (MSE) loss:

$$\min_{\mathbf{W}} \sum_{x\in X_{\text{tr}}} \sum_{i,j}^{N,M} \left\| t_i(x_{i-1}) - s_{ij}(x_{i-1}; w_{ij}) \right\|_2^2, \tag{2}$$

where X_{tr} is a set of training samples, and $x_{i-1} = t_{i-1} \circ t_{i-2} \circ \ldots \circ t_1\,(x)$ is the output of the previous layer of the teacher, fed to both the teacher and student operations.

Our layer-wise pretraining has several advantages. First, the minimization in Eq. 2 can be decomposed into $N \times M$ independent minimization problems as $w_{i,j}$ is specific to one minimization problem per operation and layer. This allows us to train all candidate operations simultaneously in parallel. Second, since each candidate operation is tasked with an easy problem of approximating *one* layer in the teacher network, we can train the student operation quickly in one epoch. In this paper, instead of solving all $N \times M$ problems in separate processes, we train a single operation for each layer in the same forward pass of the teacher to maximize reusing the output features produced in all the teacher layers. This way the pretraining phase roughly takes $O(M)$ epochs of training a full network.

2.2 Operation Selection Phase

Fig. 3 shows steps involved to find an accelerated architecture. Since our goal in search is to discover an efficient network in the proximity of the teacher network,

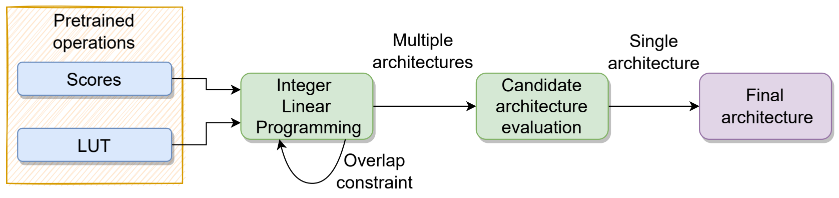


Fig. 3: **Architecture search with LANA:** Given scores of pretrained operations and their associated cost, LANA formulates the architecture search as an integer linear programming problem. Multiple architectures are found to satisfy the budget constraint via penalizing overlaps. Then, a single architecture is picked during candidate architecture evaluation phase. The cost of the overall search is minor comparing to existing NAS approaches as no training is involved.

we propose a simple linear relaxation of candidate architecture loss using

$$\sum_{X_{tr}} \mathcal{L}\big(\mathcal{S}(x;\mathbf{Z}), y\big) \approx \sum_{X_{tr}} \mathcal{L}\big(\mathcal{T}(x), y\big) + \sum_{i=1}^{N} \mathbf{a}_i^T \mathbf{z}_i, \tag{3}$$

where the first term denotes the training loss of teacher which is constant and $\mathbf{a}_i$ is a vector of change values in the training loss per operation for layer i. We refer to $\mathbf{a}_i$ as a score vector. Our approximation bears similarity to the first-degree Taylor expansion of the student loss with the teacher as the reference point (since the teacher architecture is a member of the search space). To compute $\{\mathbf{a}_i\}_i^N$, after pretraining operations in the first stage, we plug each candidate operation one-by-one in the teacher network and we measure the change on training loss on a small labeled set. Our approximation relaxes the non-linear loss to a linear function. Although this is a weak approximation that ignores how different layers influence the final loss together, we empirically observe that it performs well in practice as a proxy for searching the student.

Approximating the architecture loss with a linear function allows us to formulate the search problem as solving an integer linear program (ILP). This has several main advantages: (i) Although solving integer linear programs is generally NP-hard, there exist many off-the-shelf libraries that can obtain a high-quality solutions in a few seconds. (ii) Since integer linear optimization libraries easily scale up to millions of variables, our search also scales up easily to very large number of candidate operations per layer. (iii) We can easily formulate the search problem such that instead of one architecture, we obtain a set of diverse candidate architectures. Formally, we denote the k^{th} solution with $\big\{\mathbf{Z}^{(k)}\big\}_{k=1}^{K}$, which is obtained by solving:

$$\underbrace{\min_{\mathbf{Z}^{(k)}} \sum_{i=1}^{N} \mathbf{a}_i^T \mathbf{z}_i^{(k)}}_{\text{objective}}, \quad \text{s.t.} \underbrace{\sum_{i=1}^{N} \mathbf{b}_i^T \mathbf{z}_i^{(k)} \leqslant \mathcal{B}}_{\text{budget constraint}}; \quad \underbrace{\mathbf{1}^T \mathbf{z}_i^{(k)} = 1 \ \forall\ i}_{\text{one op per layer}};$$

$$\underbrace{\sum_{i=1}^{N} {\mathbf{z}_i^{(k)}}^T \mathbf{z}_i^{(k')}}_{\text{overlap constraint}} \leqslant \mathcal{O}, \forall k' < k \tag{4}$$

where we minimize the change in the loss while satisfying the budget and overlap constraint. The scalar $\mathcal{O}$ sets the maximum overlap with any previous solution which is set to $0.7N$ in our case. We obtain K diverse solutions by solving the minimization above K times.

Solving the integer linear program (ILP). We use the off-the-shelf PuLP Python package to find feasible candidate solutions. The cost of finding the first solution is very small, often less than 1 CPU-second. As K increases, so does the difficulty of finding a feasible solution. We limit K to $\sim$100.

Candidate architecture evaluation. Solving Eq. 4 provides us with K architectures. The linear proxy used for candidates loss is calculated in an isolated setting for each operation. To reduce the approximation error, we evaluate all K architectures with pretrained weights from phase one on a small part of the training set (6k images on ImageNet) and select the architecture with the lowest loss. This step assumes that the accuracy of the model before finetuning is positively correlated with the accuracy after finetuning. Batch normalization layers have to use current batch statistics (instead of precompted) to adopted for distribution change with new operations.

Candidate architecture fine-tuning. After selecting the best architecture among the K candidate architectures, we fine-tune it for 100 epochs using the original objective used for training the teacher. Additionally, we add the distillation loss from teacher to student during fine-tuning.

3 Experiments

We apply LANA to the family of EfficientNetV1 [73], EfficientNetV2 [75] and ResNeST50 [102]. When naming our models, we use the latency reduction ratio compared to the original model according to latency look-up table (LUT). For example, $0.25\times$B6 indicates $4\times$ target speedup for the B6 model. For experiments, ImageNet-1K [61] is used for pretraining (1 epoch), candidate evaluation (6k training images) and finetuning (100 epochs).

We use the NVIDIA V100 GPU and Intel Xeon Silver 4114 CPU as our target hardware. A hardware specific look-up table is precomputed for each candidate operation (vectors $\mathbf{b}_i$ in Eq. 4). We measure latency in 2 settings: (i) in Pytorch framework, and (ii) TensorRT [54]. The latter performs kernel fusion for additional model optimization making it even harder to accelerate models. The exact same setup is used for evaluating latency of **all** competing models, our models, and baselines. Actual latency on target platforms is reported in tables.

Candidate operations. We construct a large of pool of diverse candidate operation including $M = 197$ operations for each layer of teacher. Our operations include:

Teacher operation is used as is in the pretrained (base) model with teacher model accuracy.

Identity is used to skip teacher's operation. It changes the depth of the network.

Table 2: Comparison to prior art on ImageNet1K. Latency is measured on NVIDIA V100 with various batch size and inference precision.

Method	Accuracy	Latency (ms), bs128/fp16	bs32/fp32
EfficientNetV1-B0	77.7	35.6	
Cream-S [56]	77.6	36.7	
DNA-D [38]	77.1	33.8	
DONNA [50]	78.9		20.0
OFA_flops@482M [5]	79.6	39.3	
LANA(0.45xEFNv1-B2)	79.7	30.2	18.9
LANA(0.4xEFNv2-B3)	**79.9**	39.0	
EfficientNetV1-B1	78.8	59.0	
DNA-D [38]	78.4	61.3	
DONNA [50]	79.5		25.0
OFA_flops@595M [5]	80.0	50.0	
Cream-L [56]	80.0	84.0	
LANA(0.55xEFNv1-B2)	80.1	48.7	24.1
LANA(0.5xEFNv2-B3)	**80.8**	48.1	

Table 3: Models optimized with LANA for latency-accuracy trade-off on ImageNet1K.

Method	Variant	Res px	Accuracy (%)	Latency(ms) TensorRT	PyTorch
EfficientNetV1					
EfficientNetV1-B1		240	78.83	29.3	59.0
LANA	0.25xB4	380	**81.83 (+3.00)**	30.4	64.5
EfficientNetV1-B2		260	80.07	38.2	77.1
LANA	0.3xB4	380	**82.16 (+2.09)**	38.8	81.8
EfficientNetV1-B3		300	81.67	67.2	125.9
LANA	0.5xB4	380	**82.66 (+0.99)**	**61.4**	148.1
EfficientNetV1-B4		380	83.02	132.0	262.4
LANA	0.25xB6	528	**83.77 (+0.75)**	**128.8**	282.1
EfficientNetV1-B5		456	83.81	265.7	525.6
LANA	0.5xB6	528	**83.99 (+0.18)**	266.5	561.2
EfficientNetV1-B6		528	84.11	466.7	895.2
EfficientNetV2					
EfficientNetV2-B1		240	79.46	17.9	44.7
LANA	0.45xB3	300	**80.30 (+0.84)**	**17.8**	**43.0**
EfficientNetV2-B2		260	80.21	24.3	58.9
LANA	0.6xB3	300	**81.14 (+0.93)**	**23.8**	**56.1**
EfficientNetV2-B3		300	81.97	41.2	91.6
ResNeST50d_1s4x24d					
ResNeST50		224	80.99	32.3	74.0
LANA	0.7x	224	80.85	**22.3(1.45x)**	**52.7**

Inverted residual blocks `efn` [63] and `efnv2` [75] with varying expansion factor e=$\{1, 3, 6\}$, squeeze and excitation ratio se=$\{\text{No}, 0.04, 0.025\}$, and kernel size k=$\{1, 3, 5\}$.
Dense convolution blocks inspired by [22] with (i) two stacked convolution (`cb_stack`) with CBRCB structure, C-conv, B-batchnorm, R-Relu; (ii) bottleneck architecture (`cb_bottle`) with CBR-CBR-CB; (ii) CB pair (`cb_res`); (iii) RepVGG block [15]; (iv) CBR pairs with perturbations as `conv_cs`. For all models we vary kernel size $k = \{1, 3, 5, 7\}$ and width $w = \{1/16, 1/10, 1/8, 1/5, 1/4, 1/2, 1, 2, 3, 4\}$.
Transformer variations (i) visual transformer block (`vit`) [17] with depth $d = \{1, 2\}$, dimension $w = \{2^5, 2^6, 2^7, 2^8, 2^9, 2^{10}\}$ and heads $h = \{4, 8, 16\}$; (ii) bottleneck transformers [66] with 4 heads and expansion factor $e = \{1/4, 1/2, 1, 2, 3, 4\}$; (iii) lambda bottleneck layers [2] with expansion $e = \{1/4, 1/2, 1, 2, 3, 4\}$.

With the pool of 197 operations, distilling from an EfficientNet-B6 model with 46 layers yields a design space of the size $197^{46} \approx 10^{100}$.

3.1 EfficientNet and ResNeST Derivatives

Our experimental results on accelerating EfficientNetV1(B2, B4, B6), EfficientNetV2(B3), and ResNeST50 family for GPUs are shown in Tables 2 and 3. Comparison with more models from `timm` is in the Appendix.

At first we compare to prior NAS-like models tailored to EfficientNetV1-B0 and B1 in Table 2. LANA has clear advantages in terms of accuracy and latency.

Next, we demonstrate a capability of LANA for a variety of larger architectures in Table 3. Results show that:

– LANA achieves an accuracy boost of 0.18-3.0% for all models when compressing larger models to the latency level of smaller models (see EfficientNet models and the corresponding LANA models in the same latency group).

Table 4: Local pretraining of the teacher model initialized from scratch on ImageNet1K at different granularity.

Model	Original	Distillation	
		Block-wise [50], [38]	Layer-wise (LANA)
EfficientNetV1-B2	82.01	68.21	76.52
EfficientNetV2-B3	81.20	73.53	76.52
ResNeST-50	86.35	77.54	80.61

Table 5: Comparing methods for candidate selection (NAS). Our proposed ILP is better (+0.43%) and 821× faster.

Method	Accuracy	Search cost
ILP, K=100 (ours)	79.28	4.5 CPU/m
Random, found 80 arch	76.44	1.4 CPU/m
SNAS [90]	74.20	16.3 GPU/h
E-NAS [57]	78.85	61.6 GPU/h

Table 6: Impact of the search space on 0.55×EfficientNetV1-B2 compression. Two operations correspond to Single-shot LANA.

Operations	2	5	10	All
Space size	$O(10^7)$	$O(10^{16})$	$O(10^{23})$	$O(10^{46})$
Accuracy	79.40	79.52	79.66	80.00
STD		±0.208	±0.133	

Table 7: Single-shot LANA with only skip connections.

Setup	Top-1 Acc.		Latency(ms)
	Single-shot	All	TensorRT
0.45xEfficientNetV1-B2	78.68	79.71	16.2
0.55xEfficientNetV1-B2	79.40	80.11	20.6

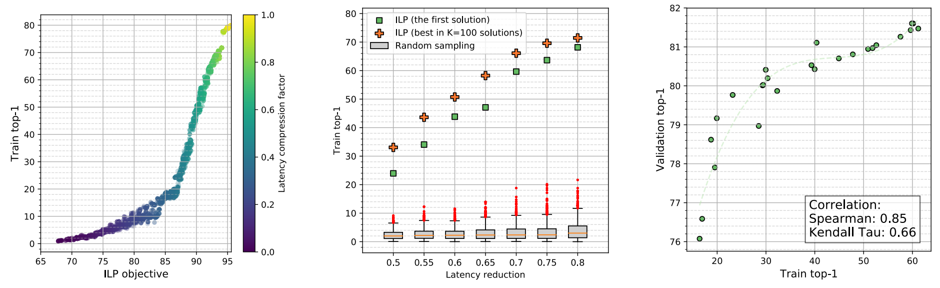
– LANA achieves significant real speed-ups with little to no accuracy drop: (i) EfficieintNetV1-B6 accelerated by 3.6x and 1.8x times by trading-off 0.34% and 0.11% accuracy, respectively; (ii) B2 variant is accelerated 2.4x and 1.9x times with 0.36% and no accuracy loss, respectively. (iii) ResNeST50 is accelerated 1.5x with 0.14% accuracy drop.

A detailed look on EfficientNets results demonstrate that EfficientNetV1 models are accelerated beyond EfficientNetV2. LANA generates models that have better accuracy-throughput trade-off when comparing models under the same accuracy or the same latency. LANA also allows us to optimize models for different hardware at a little cost. Only a new LUT is required to get optimal model for a new hardware without pretraining the candidate operations again. We present models optimized for CPU in supplementary materials Table 8 which are obtained using a different LUT only.

3.2 Analysis

Here, we provide detailed ablations to analyze our design choices in LANA for both pretraining and search phases, along with observed insights. We use EfficientNetV1-B2 and EfficientNetV2-B3 as our base models for the ablation. **Operator pretraining**. Previous work ([50] and [38]) applies per block distillation for pretraining. Main reason for that is costly search if they operate in per layer setting. With per layer pretraining the search cost increases exponentially. Main advantage of our work is ILP that it is faster by several orders of magnitude. Therefore, we can perform per layer distillation.

To study the benefit of using per layer distillation we perform a teacher mimicking test where all parameters of the teacher are re-initialized. Then, we



(a) Optimization objective (b) ILP vs random (c) Candidate evaluation

Fig. 4: Analyzing ILP performance on EfficientNetV2-B3. ILP results in significantly higher model accuracy before finetuning than 1k randomly sampled architectures in (b). Accuracy monotonically increases with ILP objective (a). Model accuracy before finetuning correctly ranks models after finetuning (c). Train top-1 is measured *before* finetuning, while Validation top-1 is *after*.

perform isolated pretraining (with MSE loss) of teacher blocks/layers and report the final top1 accuracy on the training set on the ImageNet. Results in Table 4 clearly demonstrate significant advantage of per layer pretraining. By carefully studying we found that per block supervision provides very little guidance to first layers of the block and therefore lacks performance. We conclude that per layer distillation is more efficient and provides extra advantages.

Linear relaxation in architecture search assumes that a candidate architecture can be scored by a fitness metric measured independently for all operations. Although this relaxation is not accurate, we observe a strong correlation between our linear objective and the training loss of the full architecture. This assumption is verified by sampling 1000 architectures (different budget constraints), optimizing the ILP objective, and measuring the real loss function. Results are shown in Fig. 4a using the train accuracy as the loss. We observe that ILP objective ranks models with respect to the measured accuracy correctly under different latency budgets. The Kendall Tau correlation is 0.966.

To evaluate the quality of the solution provided with ILP, we compare it with random sampling. The comparison is shown in Fig. 4b, where we sample 1000 random architectures for 7 latency budgets. The box plots indicate the poor performance of the randomly sampled architectures. The first ILP solution has significantly higher accuracy than random architecture. Furthermore, finding multiple diverse solutions is possible with ILP using the overlap constraint. If we increase the number of solutions found by ILP from $K{=}1$ to $K{=}100$, performance improves further. When plugging pretrained operations (Fig. 4b) into the teacher, the accuracy is high (it is above 30%, even at an acceleration factor of $2\times$). For EfficientNetV1, this is above 50% for the same compression factor.

Candidate architecture evaluation plays an important role in LANA. This step finds the best architecture quickly out of the diverse candidates generated by the ILP solver, by evaluating them on 6k images from the train data. The procedure is built on the assumption that the accuracy of the model on the training data before finetuning (just by plugging all candidate operations) is a reasonable indicator of the relative model performance after finetuning. We verify this hypothesis in the Fig. 4c and see positive correlation. Same observation was is present in other works like [37].

Comparing with other NAS approaches. We compare our search algorithm with other popular approaches to solve Eq. (1), including: (i) *Random* architecture sampling within a latency constrain; (ii) *Differentiable search with Gumbel Softmax* – a popular approach in NAS to relax binary optimization as a continuous variable optimization via learning the sampling distribution [90, 87, 78]. We follow SNAS [90] in this experiment; (iii) *REINFORCE* is a stochastic optimization framework borrowed from reinforcement learning and adopted for architecture search [57, 106, 72]. We follow an E-NAS-like [57] architecture search for (iii) and use weight sharing for (ii) and (iii).

Experiments are conducted on EfficientNetV1-B2 accelerated to 0.45× original latency. The final validation top-1 accuracy after finetuning are presented in Table 5. Our proposed ILP achieves higher accuracy (+0.43%) compared to the second best method E-NAS while being 821× faster in search.

Single-shot LANA. Our method can be applied without pretraining procedure if only teacher cells and *Identity* (skip) operation are used ($M = 2$ operations per layer). Only the vector for the change in loss for the *Identity* operator will be required alongside the LUT for the teacher operations. This allows us to do single-shot network acceleration without any pretraining as reported in Table 7. We observe that LANA efficiently finds residual blocks that can be skipped. This unique property of LANA is enabled because of layer-wise granularity.

Pretraining insights. To gain more insights into the tradeoff between the accuracy and speed of each operation, we analyze the pretrained candidate operation pool for EffientNetV1B2. A detailed figure is shown in the appendix. Here, we provide general observations.

We observe that no operation outperforms the teacher in terms of accuracy; changing pretraining loss from per-layer MSE to full-student cross-entropy may change this but that comes with an increased costs of pretraining. We also see that it is increasingly difficult to recover the teacher's accuracy as the depth in the network increases. The speedups achievable are roughly comparable across different depths, however, we note that achieving such speedups earlier in the network is particularly effective towards reducing total latency due to the first third of the layers accounting for 54% of the total inference time.

Looking at individual operations, we observe that inverted residual blocks (`efn`, `efnv2`) are the most accurate throughout the network, at the expense of increased computational cost (*i.e.,* lower speedups). Dense convolutions (`cb_stack`, `cb_bottle`, `conv_cs`, `cb_res`) exhibit a good compromise between accuracy and speed, with stacked convolutions being particularly effective earlier in the network.

Visual transformer blocks (`ViT`) and bottleneck transformer blocks (`bot_trans`) show neither a speedup advantage nor ability to recover teacher accuracy.

Common Operations. In Appendix 5, we analyze the distribution of the selected operations by solving for 100 architecture candidates with Eq. 4 for 0.5×EfficientNetV1B2. We observe: (i) teacher operations are selected most frequently, it is expected as we only transform the model and no operations can beat teacher in terms of accuracy; (ii) identity operation is selected more often for large architectures with higher compression rates; (iii) dense convolution blocks tend to appear more for larger models like EfficientNet-B6 to speed them up.

Architecture insights In the appendices, we visualize the final architectures discovered by LANA. Next, we share the insights observed on these architectures.

Common across all models: (i) Teacher's operations are usually selected in the tail of the network as they are relatively fast and approximating them results in the highest error. (ii) Teacher operation is preferred for downsampling layers (e.g. stride is 2). Those cells are hard to approximate as well, we hypothesize this is due to high nonlinearity of these blocks. (iii) Transformer blocks are never selected, probably because they require significantly longer pretraining on larger datasets. (v) LANA automatically adjusts depth of every resolutional block by replacing teacher cells with Identity operations.

EfficientNetV1. Observing final architectures obtained by LANA on the EfficientNetV1 family, particularly the 0.55×B2 version optimized for GPUs, we discover that most of the modifications are done to the first half of the model: (i) squeeze-and-excitation is removed in the early layers; (ii) dense convolutions (like inverted stacked or bottleneck residual blocks) replace depth-wise separable counterparts; (iii) the expansion factor is reduced from 6 to 3.5 on average. *Surprisingly, LANA automatically discovers the same design choices that are introduced in the EfficientNetV2 family when optimized for datacenter inference.*

EfficientNetV2. LANA accelerates EfficientNetV2-B3 by 2×, with the following conclusions: (i) the second conv-bn-act layer is not needed and can be removed; (ii) the second third of the model benefits from reducing the expansion factor from 4 to 1 without squeeze-and-excitation operations. With these simplifications, accelerated model still outperforms EfficientNetV2-B2 and B1.

ResNeST50. LANA discovers that cardinality can be reduced from 4 to 1 or 2 for most blocks without any loss in accuracy, yielding a 1.45× speedup.

Ablations on finetuning

We select 0.45×EfficientNetV1-B2 with the final accuracy of 79.71%.

Pretrained weights. We look deeper into the finetuning step. Reinitializing all weights in the model, as opposed to loading them from the pretraining stage, results in 79.42%. The result indicates the importance of pretraining stage (i) to find a strong architecture and (ii) to boost finetuning.

Knowledge distillation plays a key role in student-teacher setups. When it is disabled, we observe an accuracy degradation of 0.65%. This emphasizes the benefit of training a larger model and then self-distilling to a smaller one. We further verify whether we can achieve a similar high accuracy using knowledge distillation from EfficientNetV1-B2 to vanilla EfficientNetV1-B0 in the same setting. The top-1 accuracy of 78.72% is still 1% less than LANA's accuracy. When both models are trained from scratch with the distillation loss, LANA 0.45xB2 achieves 79.42% while EfficientNetV1-B0 achieves 78.01%.

Length of finetuning. Pretrained operations have already been trained to mimic the teacher layer. Therefore, even before finetuning the student model can be already adept at the task. Next, we evaluate how does the length of finetuning affects the final accuracy in Table 9 (Supplementary). Even with only 5 epochs LANA outperforms the vanilla EfficientNet counterparts.

Search space size. ILP enables us to perform NAS in a very large space ($O(10^{100})$). To verify the benefit of large search space, we experiment with a restricted search space. For this, we randomly pick 2, 5 and 10 operations per layer to participate in search and finetuning for 50 epochs. We observe clear improvements from increasing the search space, shown Table 6 (results are averaged over 5 runs). When more than 2 operations are present we select the teacher cell, identity operation and the rest to be random operations with the constrain to have score difference of no more than 5.0% and being faster.

4 Conclusion

In this paper, we proposed LANA, a hardware-aware network transformation framework for accelerating pretrained neural networks. LANA uses a NAS-like search to replace inefficient operations with more efficient alternatives. It tackles this problem in two phases including a candidate pretraining phase and a search phase. The availability of the teacher network allows us to estimate the change in accuracy for each operation at each layer. Using this, we formulate the search problem as solving a linear integer optimization problem, which outperforms the commonly used NAS algorithms while being orders of magnitude faster. We applied our framework to accelerate EfficientNets (V1 and V2) and ResNets with a pool of 197 operations per layer and we observed that LANA accelerates these architectures by several folds with only a small drop in the accuracy.

Limitations. The student performance in LANA is bounded by the base model, and it rarely passes in terms of accuracy. Additionally, the output dimension of layers in the student can not be changed, and must remain the same as in the original base model.

Future work. We envision that a layer-wise network acceleration framework like LANA can host a wide range of automatically and manually designed network operations, developed in the community. Our design principals in LANA consisting of extremely large operation pool, efficient layer-wise pretraining, and lightening fast search help us realize this vision. Components of LANA can be further improved in the follow-up research: i) pretraining stage to consider error propagation; ii) scoring metric; iii) ILP with finetuning; iv) candidate architecture evaluation.

References

1. Belagiannis, V., Farshad, A., Galasso, F.: Adversarial network compression. In: Proceedings of the European Conference on Computer Vision (ECCV) Workshops. pp. 0–0 (2018)
2. Bello, I.: Lambdanetworks: Modeling long-range interactions without attention. arXiv preprint arXiv:2102.08602 (2021)
3. Bello, I., Fedus, W., Du, X., Cubuk, E.D., Srinivas, A., Lin, T.Y., Shlens, J., Zoph, B.: Revisiting resnets: Improved training and scaling strategies (2021)
4. Blalock, D., Ortiz, J.J.G., Frankle, J., Guttag, J.: What is the state of neural network pruning? arXiv preprint arXiv:2003.03033 (2020)
5. Cai, H., Gan, C., Wang, T., Zhang, Z., Han, S.: Once for all: Train one network and specialize it for efficient deployment. In: International Conference on Learning Representations (2020), https://arxiv.org/pdf/1908.09791.pdf
6. Cai, H., Zhu, L., Han, S.: ProxylessNAS: Direct neural architecture search on target task and hardware. In: International Conference on Learning Representations (2019), https://openreview.net/forum?id=HylVB3AqYm
7. Cai, Y., Yao, Z., Dong, Z., Gholami, A., Mahoney, M.W., Keutzer, K.: ZeroQ: A novel zero shot quantization framework. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 13169–13178 (2020)
8. Chauvin, Y.: A back-propagation algorithm with optimal use of hidden units. In: NIPS (1989)
9. Chen, L.C., Zhu, Y., Papandreou, G., Schroff, F., Adam, H.: Encoder-decoder with atrous separable convolution for semantic image segmentation (2018)
10. Chen, Y., Li, J., Xiao, H., Jin, X., Yan, S., Feng, J.: Dual path networks. arXiv preprint arXiv:1707.01629 (2017)
11. Choi, J., Wang, Z., Venkataramani, S., Chuang, P., Srinivasan, V., Gopalakrishnan, K.: Pact: Parameterized clipping activation for quantized neural networks. ArXiv **abs/1805.06085** (2018)
12. Chollet, F.: Xception: Deep learning with depthwise separable convolutions (2017)
13. Dai, X., Zhang, P., Wu, B., Yin, H., Sun, F., Wang, Y., Dukhan, M., Hu, Y., Wu, Y., Jia, Y., Vajda, P., Uyttendaele, M., Jha, N.K.: ChamNet: Towards efficient network design through platform-aware model adaptation. In: CVPR (2019)
14. Ding, X., Guo, Y., Ding, G., Han, J.: Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 1911–1920 (2019)
15. Ding, X., Zhang, X., Ma, N., Han, J., Ding, G., Sun, J.: Repvgg: Making vgg-style convnets great again. arXiv preprint arXiv:2101.03697 (2021)
16. Dong, Z., Yao, Z., Gholami, A., Mahoney, M.W., Keutzer, K.: Hawq: Hessian aware quantization of neural networks with mixed-precision. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 293–302 (2019)
17. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al.: An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929 (2020)
18. Frankle, J., Carbin, M.: The lottery ticket hypothesis: Finding sparse, trainable neural networks. In: International Conference on Learning Representations (2018)
19. Gordon, A., Eban, E., Nachum, O., Chen, B., Wu, H., Yang, T.J., Choi, E.: Morphnet: Fast & simple resource-constrained structure learning of deep networks. In: CVPR (2018)

20. Hanson, S.J., Pratt, L.Y.: Comparing biases for minimal network construction with back-propagation. In: NIPS (1989)
21. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016)
22. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. In: ECCV (2016)
23. He, T., Zhang, Z., Zhang, H., Zhang, Z., Xie, J., Li, M.: Bag of tricks for image classification with convolutional neural networks (2018)
24. He, Y., Dong, X., Kang, G., Fu, Y., Yang, Y.: Progressive deep neural networks acceleration via soft filter pruning. arXiv preprint arXiv:1808.07471 (2018)
25. He, Y., Kang, G., Dong, X., Fu, Y., Yang, Y.: Soft filter pruning for accelerating deep convolutional neural networks. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence. pp. 2234–2240 (2018)
26. He, Y., Liu, P., Wang, Z., Hu, Z., Yang, Y.: Filter pruning via geometric median for deep convolutional neural networks acceleration. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 4340–4349 (2019)
27. He, Y., Han, S.: Adc: Automated deep compression and acceleration with reinforcement learning. arXiv preprint arXiv:1802.03494 (2018)
28. He, Y., Lin, J., Liu, Z., Wang, H., Li, L.J., Han, S.: Amc: Automl for model compression and acceleration on mobile devices. ECCV pp. 784–800 (2018)
29. He, Y., Zhang, X., Sun, J.: Channel pruning for accelerating very deep neural networks. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 1389–1397 (2017)
30. Heo, B., Yun, S., Han, D., Chun, S., Choe, J., Oh, S.J.: Rethinking spatial dimensions of vision transformers (2021)
31. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)
32. Hu, J., Shen, L., Albanie, S., Sun, G., Wu, E.: Squeeze-and-excitation networks (2019)
33. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: CVPR (2017)
34. Krishnamoorthi, R.: Quantizing deep convolutional networks for efficient inference: A whitepaper. ArXiv **abs/1806.08342** (2018)
35. Lebedev, V., Lempitsky, V.: Fast convnets using group-wise brain damage. In: CVPR. pp. 2554–2564 (2016)
36. LeCun, Y., Denker, J.S., Solla, S., Howard, R.E., Jackel, L.D.: Optimal brain damage. In: NIPS (1990)
37. Li, B., Wu, B., Su, J., Wang, G.: Eagleeye: Fast sub-net evaluation for efficient neural network pruning. In: European Conference on Computer Vision. pp. 639–654. Springer (2020)
38. Li, C., Peng, J., Yuan, L., Wang, G., Liang, X., Lin, L., Chang, X.: Block-wisely supervised neural architecture search with knowledge distillation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2020)
39. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient ConvNets. In: ICLR (2017)
40. Li, X., Wang, W., Hu, X., Yang, J.: Selective kernel networks (2019)
41. Liu, H., Simonyan, K., Yang, Y.: Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055 (2018)

42. Liu, Y., Jia, X., Tan, M., Vemulapalli, R., Zhu, Y., Green, B., Wang, X.: Search to distill: Pearls are everywhere but not the eyes. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2020)
43. Liu, Z., Sun, M., Zhou, T., Huang, G., Darrell, T.: Rethinking the value of network pruning. arXiv preprint arXiv:1810.05270 (2018)
44. Louizos, C., Welling, M., Kingma, D.P.: Learning sparse neural networks through l_0 regularization. arXiv preprint arXiv:1712.01312 (2017)
45. Luo, J.H., Wu, J., Lin, W.: ThiNet: A filter level pruning method for deep neural network compression. In: ICCV (2017)
46. Mishra, A., Marr, D.: Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. In: ICLR (2018)
47. Molchanov, D., Ashukha, A., Vetrov, D.: Variational dropout sparsifies deep neural networks. In: International Conference on Machine Learning. pp. 2498–2507. PMLR (2017)
48. Molchanov, P., Mallya, A., Tyree, S., Frosio, I., Kautz, J.: Importance estimation for neural network pruning. In: CVPR (2019)
49. Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J.: Pruning convolutional neural networks for resource efficient transfer learning. In: ICLR (2017)
50. Moons, B., Noorzad, P., Skliar, A., Mariani, G., Mehta, D., Lott, C., Blankevoort, T.: Distilling optimal neural networks: Rapid search in diverse spaces. arXiv preprint arXiv:2012.08859 (2020)
51. Mozer, M.C., Smolensky, P.: Skeletonization: A technique for trimming the fat from a network via relevance assessment. In: NIPS (1989)
52. Nayak, G.K., Mopuri, K.R., Shaj, V., Babu, R.V., Chakraborty, A.: Zero-shot knowledge distillation in deep networks. In: CVPR (2019)
53. Neklyudov, K., Molchanov, D., Ashukha, A., Vetrov, D.P.: Structured bayesian pruning via log-normal multiplicative noise. In: Advances in Neural Information Processing Systems. pp. 6775–6784 (2017)
54. NVIDIA: TensorRT Library. https://developer.nvidia.com/tensorrt (2021), [Online; accessed 10-May-2021]
55. Park, E., Yoo, S., Vajda, P.: Value-aware quantization for training and inference of neural networks. In: ECCV (2018)
56. Peng, H., Du, H., Yu, H., Li, Q., Liao, J., Fu, J.: Cream of the crop: Distilling prioritized paths for one-shot neural architecture search. Advances in Neural Information Processing Systems **33**, 17955–17964 (2020)
57. Pham, H., Guan, M.Y., Zoph, B., Le, Q.V., Dean, J.: Efficient neural architecture search via parameter sharing. arXiv preprint arXiv:1802.03268 (2018)
58. Radosavovic, I., Kosaraju, R.P., Girshick, R., He, K., Dollár, P.: Designing network design spaces (2020)
59. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: Xnor-net: Imagenet classification using binary convolutional neural networks. In: European conference on computer vision. pp. 525–542. Springer (2016)
60. Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y.L., Tan, J., Le, Q.V., Kurakin, A.: Large-scale evolution of image classifiers. In: International Conference on Machine Learning. pp. 2902–2911. PMLR (2017)
61. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV) **115**(3), 211–252 (2015). https://doi.org/10.1007/s11263-015-0816-y

62. Ryoo, S., Rodrigues, C., Baghsorkhi, S.S., Stone, S.S., Kirk, D., Hwu, W.: Optimization principles and application performance evaluation of a multithreaded gpu using cuda. Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming (2008)
63. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: CVPR (2018)
64. Sau, B.B., Balasubramanian, V.N.: Deep model compression: Distilling knowledge from noisy teachers. arXiv preprint arXiv:1610.09650 (2016)
65. Shen, S., Dong, Z., Ye, J., Ma, L., Yao, Z., Gholami, A., Mahoney, M.W., Keutzer, K.: Q-BERT: Hessian based ultra low precision quantization of BERT. In: Proceedings of the AAAI Conference on Artificial Intelligence. pp. 8815–8821 (2020)
66. Srinivas, A., Lin, T.Y., Parmar, N., Shlens, J., Abbeel, P., Vaswani, A.: Bottleneck transformers for visual recognition. arXiv preprint arXiv:2101.11605 (2021)
67. Sze, V., Chen, Y.H., Yang, T.J., Emer, J.S.: Efficient processing of deep neural networks: A tutorial and survey. Proceedings of the IEEE **105**(12), 2295–2329 (2017)
68. Szegedy, C., Ioffe, S., Vanhoucke, V., Alemi, A.: Inception-v4, inception-resnet and the impact of residual connections on learning. In: Proceedings of the AAAI Conference on Artificial Intelligence (2017)
69. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2818–2826 (2016)
70. Tan, M., Chen, B., Pang, R., Vasudevan, V., Le, Q.V.: Mnasnet: Platform-aware neural architecture search for mobile. arXiv preprint arXiv:1807.11626 (2018)
71. Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., Le, Q.V.: Mnasnet: Platform-aware neural architecture search for mobile. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2019)
72. Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., Le, Q.V.: Mnasnet: Platform-aware neural architecture search for mobile. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 2820–2828 (2019)
73. Tan, M., Le, Q.: Efficientnet: Rethinking model scaling for convolutional neural networks. In: International Conference on Machine Learning. pp. 6105–6114. PMLR (2019)
74. Tan, M., Le, Q.V.: Mixconv: Mixed depthwise convolutional kernels. arXiv preprint arXiv:1907.09595 (2019)
75. Tan, M., Le, Q.V.: Efficientnetv2: Smaller models and faster training. arXiv preprint arXiv:2104.00298 (2021)
76. Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., Jégou, H.: Training data-efficient image transformers & distillation through attention. arXiv preprint arXiv:2012.12877 (2020)
77. Touvron, H., Cord, M., Sablayrolles, A., Synnaeve, G., Jégou, H.: Going deeper with image transformers. arXiv preprint arXiv:2103.17239 (2021)
78. Vahdat, A., Mallya, A., Liu, M.Y., Kautz, J.: Unas: Differentiable architecture search meets reinforcement learning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 11266–11275 (2020)
79. Veniat, T., Denoyer, L.: Learning time/memory-efficient deep architectures with budgeted super networks. arXiv preprint arXiv:1706.00046 (2017)

80. Wang, C.Y., Liao, H.Y.M., Wu, Y.H., Chen, P.Y., Hsieh, J.W., Yeh, I.H.: Cspnet: A new backbone that can enhance learning capability of cnn. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops. pp. 390–391 (2020)
81. Wang, G., Lin, Y., Yi, W.: Kernel fusion: An effective method for better power efficiency on multithreaded gpu. 2010 IEEE/ACM Int'l Conference on Green Computing and Communications and Int'l Conference on Cyber, Physical and Social Computing pp. 344–350 (2010)
82. Wang, J., Sun, K., Cheng, T., Jiang, B., Deng, C., Zhao, Y., Liu, D., Mu, Y., Tan, M., Wang, X., et al.: Deep high-resolution representation learning for visual recognition. IEEE transactions on pattern analysis and machine intelligence (2020)
83. Wang, K., Liu, Z., Lin, Y., Lin, J., Han, S.: HAQ: Hardware-aware automated quantization with mixed precision. In: CVPR (2019)
84. Wang, Q., Wu, B., Zhu, P., Li, P., Zuo, W., Hu, Q.: Eca-net: Efficient channel attention for deep convolutional neural networks. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2020)
85. Wightman, R.: Pytorch image models. https://github.com/rwightman/pytorch-image-models (2019). https://doi.org/10.5281/zenodo.4414861
86. Wu, B., Wang, Y., Zhang, P., Tian, Y., Vajda, P., Keutzer, K.: Mixed precision quantization of convnets via differentiable neural architecture search. ArXiv **abs/1812.00090** (2018)
87. Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., Keutzer, K.: Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2019)
88. Wu, B., Wang, Y., Zhang, P., Tian, Y., Vajda, P., Keutzer, K.: Mixed precision quantization of convnets via differentiable neural architecture search. arXiv preprint arXiv:1812.00090 (2018)
89. Xie, S., Girshick, R.B., Dollár, P., Tu, Z., He, K.: Aggregated residual transformations for deep neural networks. CoRR **abs/1611.05431** (2016), http://arxiv.org/abs/1611.05431
90. Xie, S., Zheng, H., Liu, C., Lin, L.: SNAS: stochastic neural architecture search. In: International Conference on Learning Representations (2019), https://openreview.net/forum?id=rylqooRqK7
91. Xu, Z., Hsu, Y.C., Huang, J.: Training shallow and thin networks for acceleration via knowledge distillation with conditional adversarial networks. In: ICLR Workshop (2018)
92. Yang, T.J., Howard, A., Chen, B., Zhang, X., Go, A., Sandler, M., Sze, V., Adam, H.: Netadapt: Platform-aware neural network adaptation for mobile applications. Energy **41**, 46 (2018)
93. Yang, Y., Huang, Q., Wu, B., Zhang, T., Ma, L., Gambardella, G., Blott, M., Lavagno, L., Vissers, K., Wawrzynek, J., et al.: Synetgy: Algorithm-hardware co-design for convnet accelerators on embedded fpgas. arXiv preprint arXiv:1811.08634 (2018)
94. Ye, J., Lu, X., Lin, Z., Wang, J.Z.: Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. ICLR (2018)
95. Yin, H., Molchanov, P., Alvarez, J.M., Li, Z., Mallya, A., Hoiem, D., Jha, N.K., Kautz, J.: Dreaming to distill: Data-free knowledge transfer via deepinversion. In: CVPR (2020)

96. Yu, F., Wang, D., Shelhamer, E., Darrell, T.: Deep layer aggregation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2403–2412 (2018)
97. Yu, R., Li, A., Chen, C.F., Lai, J.H., Morariu, V.I., Han, X., Gao, M., Lin, C.Y., Davis, L.S.: NISP: Pruning networks using neuron importance score propagation. CVPR (2017)
98. Yu, R., Li, A., Chen, C.F., Lai, J.H., Morariu, V.I., Han, X., Gao, M., Lin, C.Y., Davis, L.S.: NISP: Pruning networks using neuron importance score propagation. In: CVPR (2018)
99. Zagoruyko, S., Komodakis, N.: Wide residual networks. CoRR **abs/1605.07146** (2016), http://arxiv.org/abs/1605.07146
100. Zagoruyko, S., Komodakis, N.: Diracnets: Training very deep neural networks without skip-connections. arXiv preprint arXiv:1706.00388 (2017)
101. Zhang, D., Yang, J., Ye, D., Hua, G.: Lq-nets: Learned quantization for highly accurate and compact deep neural networks. ArXiv **abs/1807.10029** (2018)
102. Zhang, H., Wu, C., Zhang, Z., Zhu, Y., Lin, H., Zhang, Z., Sun, Y., He, T., Mueller, J., Manmatha, R., et al.: Resnest: Split-attention networks. arXiv preprint arXiv:2004.08955 (2020)
103. Zhang, X., Zhou, X., Lin, M., Sun, J.: Shufflenet: An extremely efficient convolutional neural network for mobile devices. arXiv preprint arXiv:1707.01083 (2017)
104. Zhu, C., Han, S., Mao, H., Dally, W.J.: Trained ternary quantization. In: ICLR (2017)
105. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578 (2016)
106. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: CVPR. pp. 8697–8710 (2018)