HARDWARE-AWARE NETWORK TRANSFORMATION

Anonymous authors
Paper under double-blind review

ABSTRACT

In this paper, we tackle the problem of network acceleration by proposing hardware-aware network transformation (HANT), an approach that builds on neural architecture search techniques and teacher-student distillation. HANT consists of two phases: in the first phase, it trains many alternative operations for every layer of the teacher 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 approach. In extensive experiments, we show that HANT can successfully accelerate three different families of network architectures (EfficientNetsV1, EfficientNetsV2 and ResNests), over two different target hardware platforms with minimal loss of accuracy. For example, HANT accelerates EfficientNetsV1-B6 by 3.6× with <0.4% drop in top-1 accuracy on ImageNet. When comparing the same latency level, HANT can accelerate EfficientNetV1-B4 to the same latency as EfficientNetV1-B1 while achieving 3% higher accuracy. We also show that applying HANT to EfficientNetV1 results in the automated discovery of the same (qualitative) architecture modifications later incorporated in EfficientNetV2. Finally, HANT's efficient search allows us to examine a large pool of 197 operations per layer, resulting in new insights into the accuracy-latency tradeoffs for different operations.

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 (Rastegari et al., 2016; Zhang et al., 2017; Sze et al., 2017; He et al., 2018b; Cai et al., 2020) is to accelerate a given neural network such that it meets efficiency criteria on a device without sacrificing accuracy. 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.

The current network acceleration techniques can be grouped into: *(i) pruning* that removes inactive neurons (Blalock et al., 2020), (ii) *compile-time optimization* (Ryoo et al., 2008) *or kernel fusion* (Wang et al., 2010; Ding et al., 2021) that combines multiple operations into an equivalent operation, (iii) *quantization* that reduces the precision in which the network operates at (Wang et al., 2019; Rastegari et al., 2016), and *(iv) knowledge distillation* that distills knowledge from a larger *teacher* network into a smaller *student* network (Hinton et al., 2015; Sau & Balasubramanian, 2016; Xu et al., 2018). 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 architecture search requires domain knowledge and multiple iterations due to its manual design.

In this paper, we propose hardware-aware network transformation (HANT), a network acceleration framework that automatically replaces inefficient operations in a given network with more efficient counterparts. Given a convolutional teacher network, we formulate the problem as searching in a large pool of candidate operations to find efficient operations for different layers of the 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) (Zoph & Le, 2016; Tan et al., 2018; Vahdat et al., 2020), 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.

Method	Knowledge Distillation				Diverse Operators	Design Space Size	Pretrain Cost	Search Cost	Train Cost	Total Cost
	Layer	Block	Network	None			To Train L Architectures			
Once-For-All Cai et al. (2020)				✓		$> O\left(10^{19}\right)$	1205e	40h	75eL	1205e + 75eL
AKD Liu et al. (2020)			✓		✓	$> O\left(10^{13}\right)$	0	50000eL	400eL	50400eL
DNA Li et al. (2020)		✓				$> O\left(10^{15}\right)$	320e	14h	450eL	320e + 450eL
DONNA Moons et al. (2020)		✓			✓	$> O\left(10^{13}\right)$	1920e[1]	1500e + <1hL	50eL	3420e + 50eL
This Work	✓				✓	$> O\left(10^{100}\right)$	197e	<1hL	100eL	197e + 100eL

Table 1: Related method comparison. Time is mentioned in GPU hours by h, or ImageNet epochs by e. Our method assumes 197 candidate operations. L is the number of target architectures

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 (He et al., 2016a) to more recent transformer blocks, such as those employed by Dosovitskiy et al. (2020), 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 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 in the search space can be approximated by teacher's accuracy and a simple linear function of changes in the accuracy obtained from individual operations. Thus, we propose 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 our computational cost of search and it can be easily applied to 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 state-of-the-art 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

Methods to reduce the computational cost of neural networks can be grouped into two categories: those that modify the underlying architecture and those that focus on the efficiency of the operations within a given architecture (*e.g.* by means of inducing sparsity or reducing precision). Given that the aim of this work is on discovering new efficient architectures, in the following we focus our attention on the former. We note, however, that the two directions are highly complementary and can be applied jointly, as show in our experiments with TensorRT in Section 3.

When looking to automatically identifying architectures to maximize both accuracy and computational efficiency, there are two main lines of work. One is to design such 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 (Cai et al., 2019; Tan et al., 2019a; Wu et al., 2019; Yang et al., 2018a; Veniat & Denoyer, 2017; Yang et al., 2018b; Wu et al., 2018; Vahdat et al., 2020). 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 hardware platform one wants to target, 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. Cai et al. (2020) circumvent this issue by building a supernetwork containing every possible architecture in the search space, training it, 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

[1] potentially can be improved by parallelization

that can be considered at once. Adding new operations is also costly, since the supernetwork must be pretrained from scratch every time a new operation is added. And because of the progressive shrinking search algorithm, only a restricted set of operations are supported by this approach.

The other line of work, more in line with our objective here, focuses on modifying *existing* architectures. Approaches in this area build on teacher-student knowledge distillation, performing multiobjective NAS on the student to best approximate the teacher network. This approximation can happen at different levels and can consist of approximating the whole network at once, or approximating blocks of layers.

Liu et al. (2020) apply 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. Li et al. (2020) and Moons et al. (2020) 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 to have student blocks mimic the output of teacher blocks, but Li et al. (2020) sample random paths through a mix of operators in each block, whereas Moons et al. (2020) train several candidate blocks with a repeated single operation for each teacher block. They then both search for an optimal set of blocks, with Li et al. (2020) using a novel ranking algorithm to predict the best set of operations within each block, and then applying a traversal search, while Moons et al. (2020) train 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. Li et al. (2020) rank 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. Moons et al. (2020) sample and finetune 30 models to build a linear accuracy predictor, which incurs a significant startup cost for search.

In this work, we further increase the granularity of the knowledge distillation procedure, focusing on distilling each *layer* individually. This not only results in much greater expressivity of the search space, but it also naturally lets us search over network depth simply by including an identity operation to the search space, greatly reducing pretraining cost. On the other hand, this increase in expressivity also comes at the cost of a much larger search space, necessitating the development of a highly efficient search method based on integer linear optimization (presented in Section 2).

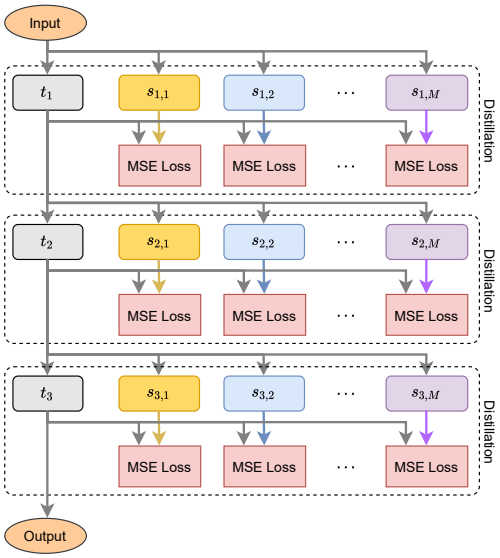
2 METHOD

Our goal in this paper is to accelerate a given pre-trained teacher network by replacing its inefficient operations with more efficient alternatives. Our method, visualized in Fig. 1, 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; (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.

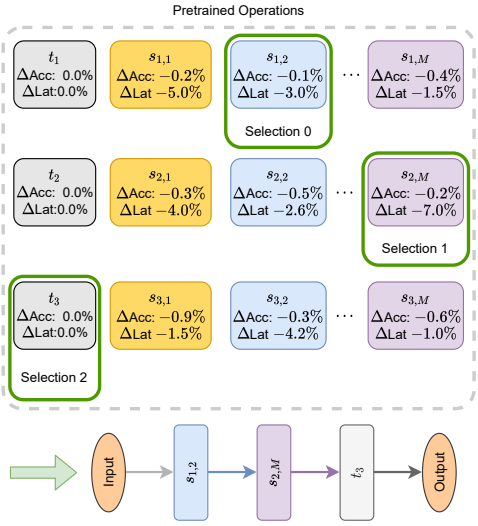
2.1 CANDIDATE PRETRAINING PHASE

Notation: We represent the teacher 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) in 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 (Liu et al., 2018; Zoph & Le, 2016). Finding the optimal architecture in hardware-aware NAS reduces to:



(a) **Candidate pretraining phase:** We minimize the MSE loss between the output of the teacher operation t_i and the output of each student operation on each layer $s_{i,j}$, where the input to each operation is the teacher output from the previous layer.



(b) **Operation Selection Phase:** We estimate and record in a lookup table the reduction of network accuracy and latency from replacing a teacher operation with one of the student operations. We then apply integer programming to minimize the accuracy reduction while attaining a target latency reduction.

Figure 1: HANT consists of two phases: a candidate pretraining phase (a) and an architecture search phase (b).

$$\min_{\mathbf{Z}} \min_{\mathbf{W}} \underbrace{\sum_{(x,y)\in X_{tr}} \mathcal{L}\big(\mathcal{S}(x;\mathbf{Z},\mathbf{W}),y\big)}_{\text{objective}}, \quad \text{s.t.} \quad \underbrace{\sum_{i=1}^{N} \mathbf{b}_i^T \mathbf{z}_i \le \mathcal{B}}_{\text{budget constraint}}; \quad \underbrace{\mathbf{1}^T \mathbf{z}_i = 1 \; \forall \, i \in [1..N]}_{\text{one op per layer}} \tag{1}$$

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 (Real et al., 2017), reinforcement learning (Zoph & Le, 2016) or differentiable search (Liu et al., 2018).

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 teacher network. That is why we assume that the functionality of each candidate operation is also similar to 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 on each layer on 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

Since our goal in search is to discover an efficient network in the proximity of the teacher network, we propose a simple linear relaxation of candidate architecture loss using $\sum_{X_{tr}} \mathcal{L}\big(\mathcal{S}(x;\mathbf{Z}),y\big) \approx \sum_{X_{tr}} \mathcal{L}\big(\mathcal{T}(x),y\big) + \sum_{i=1}^{N} \mathbf{a}_i^T \mathbf{z}_i$, 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. 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) We can easily formulate the search problem such that instead of one architecture, we obtain a set of diverse candidate architectures. (ii) 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. (iii) 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.

Formally, we denote the k^{th} solution with $\big\{\mathbf{Z}^{(k)}\big\}_{k=1}^{K}$, which is obtained by solving:

$$\min_{\mathbf{z}^{(k)}} \underbrace{\sum_{i=1}^{N} \mathbf{a}_i^T \mathbf{z}_i^{(k)}}_{\text{objective}}, \;\; \text{s.t.} \;\; \underbrace{\sum_{i=1}^{N} \mathbf{b}_i^T \mathbf{z}_i^{(k)} \leq \mathcal{B}}_{\text{budget constraint}}; \;\; \underbrace{\mathbf{1}^T \mathbf{z}_i^{(k)} = 1 \; \forall \, i}_{\text{one op per layer}}; \;\; \underbrace{\sum_{i=1}^{N} \mathbf{z}_i^{(k)^T} \mathbf{z}_i^{(b)} \leq \mathcal{O}}_{\text{overlap constraint}}, \;\; b \in [1..k-1] \tag{3}$$

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 problem. We use the off-the-shelf PuLP Python package to find feasible candidate solutions. The cost of finding the first solution is very small, typically less than 1 CPU-second. As K increases, so does the difficulty of finding a feasible solution, so we typically limit K to be about 100.

Candidate architecture evaluation. Solving Eq. 3 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.

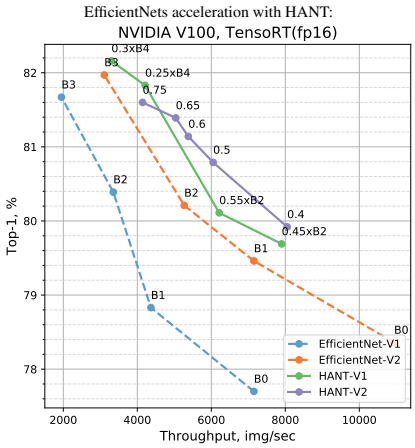
Candidate architecture fine-tuning. After selecting the best architecture among the K candidate architectures, we fine-tune it for 100 epochs using the original objective used for training the teacher. Additionally, we add the distillation loss from teacher to student during fine-tuning.

3 Experiments

In this section, we apply HANT to the family of EfficientNetV1 (Tan & Le, 2019a), EfficientNetV2 (Tan & Le, 2021) and ResNeST50 (fastest variant 1s4x24d) (Zhang et al., 2020) models. When naming our models, we use the latency reduction ratio compared to the original model according to the latency look-up table (LUT). For example, 0.25×B6 indicates 4× target speedup for the B6 model. However, we use the LUT-based latency only as the naming convention of our models but we use the actual latency on the target platform (averaged over 10 runs) when reporting the latency measurements. Note that our LUT-based latency estimation has ∼0.99 correlation with the actual on-device latency. However, they often differ by a constant value due to the fixed latency of the input and output layers that are discarded in LUT-based latency estimation. For experiments, ImageNet-1K (Russakovsky et al., 2015) is used for pretraining (1 epoch), candidate evaluation (6k training images) and finetuning (100 epochs).

GPU optimized models:

Model	Res px	Accuracy (%)	Latency(ms) TensorRT	Latency(ms) PyTorch
EfficientNetV1				
EfficientNetV1-B0	224	77.70	17.9	35.6
0.45xB2	260	**79.71 (+2.01)**	**16.2**	**30.2**
EfficientNetV1-B1	240	78.83	29.3	59.0
0.55xB2	260	80.11 (+1.28)	**20.6**	**48.7**
0.2xB4	380	80.33 (+1.50)	33.0	52.2
0.25xB4	380	**81.83 (+3.00)**	30.4	64.5
EfficientNetV1-B2	260	80.39	38.2	77.1
0.3xB4	380	**82.16 (+1.77)**	**38.8**	81.8
EfficientNetV1-B3	300	81.67	67.2	125.9
0.5xB4	380	**82.66 (+0.99)**	**61.4**	148.1
EfficientNetV1-B4	380	83.02	132.0	262.4
0.25xB6	528	**83.77 (+0.75)**	**128.8**	282.1
EfficientNetV1-B5	456	83.81	265.7	525.6
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
0.45xB3	300	**80.30 (+0.84)**	**17.8**	**43.0**
EfficientNetV2-B2	260	80.21	24.3	58.9
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
0.7x	224	80.85	**22.3(1.45x)**	**52.7**



CPU optimized models:

Model	Res. (px)	Accuracy (%)	Latency Pytorch (ms)
EfficientNetB0	224	77.70	57
0.4xB2 (Xeon)	260	78.11 (+0.97)	48
0.5xB2 (Xeon)	260	78.87 (+1.17)	58
EfficientNetB1	240	78.83	86
0.7xB2 (Xeon)	260	79.89 (+1.06)	80
EfficientNetB2	260	80.39	113

Table 2: Models optimized with HANT, evaluated on ImageNet-1K. Latency is computed for a batch of 128 images over 10 runs on actual hardware. Left: 3 families of models optimized for GPU. Right top: detailed look at EfficientNets. Right bottom: EfficientNetV1 optimized for CPU inference.

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 which is used as the $\mathbf{b}_i$ vectors in Eq. 3. The candidate operation pool is prefiltered by removing operations that are slower than the teacher or have an accuracy drop of more than 5%. We explore kernel fusion and lower precision (e.g., 16-bit floating point) as for additional model optimization. Kernel fusion eliminates dead computations, folds constants and operations into single kernels (e.g., convolution, activation function, normalization). We use the state-of-the-art TensorRT (NVIDIA, 2021) framework and report corresponding numbers with a label "TensorRT" or "TRT". Additionally, we measure latency in PyTorch to remove the kernel fusion contribution. Note that the exact same setup is used for evaluating latency of **all** competing models, our models, and baselines.

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

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

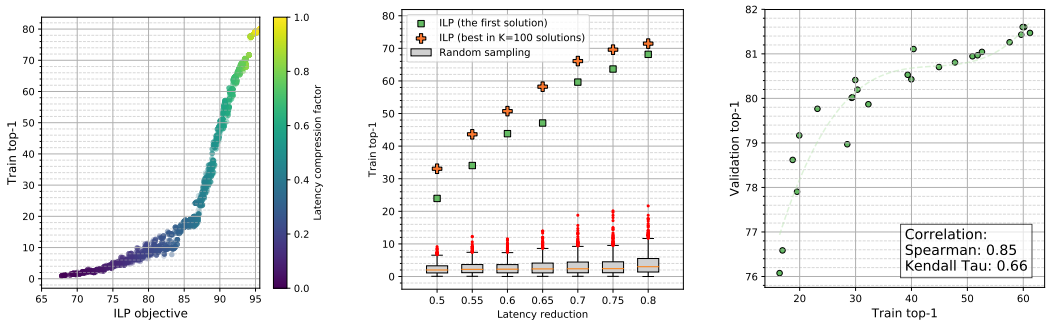
Identity is used to skip teacher's operation. It changes the depth of the network without explicitly searching for it in contrast to Cai et al. (2020) and Moons et al. (2020).

Inverted residual blocks `efn` (Sandler et al., 2018) and `efnv2` (Tan & Le, 2021) are studied with varying expansion factor $e = \{1, 3, 6\}$, squeeze and excitation factor $se = \{1.0(\text{No}), 0.04, 0.025\}$, and kernel size $k = \{1, 3, 5\}$.

Dense convolution blocks inspired by He et al. (2016b) 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 Ding et al., 2021; (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`) (Dosovitskiy et al., 2020) 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 (Srinivas et al., 2021) with 4 heads and expansion factor $e = \{1/4, 1/2, 1, 2, 3, 4\}$; (iii) lambda bottleneck layers (Bello, 2021) 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 $197^{46} \approx 10^{100}$, exponentially larger than the search space in prior works.



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

Figure 2: 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*.

3.1 EfficientNet and ResNeST Derivatives

Our experimental results on accelerating the EfficientNetV1(B2, B4, B6), EfficientNetV2(B3), and ResNeST50 family for GPUs and CPU are shown in Table 2 for ImageNet-1K. Comparison with more models from `timm` is in the Appendix (Figure 8). We make the following observations:

Throughput is significantly improved with HANT making EfficientNets up to $2\times$ faster (EfficientNetV1-B3 vs. $0.25\times$B4). EfficientNetV1s are accelerated beyond original EfficientNetV2. ResNeST50 is accelerated by $1.5\times$ with only 0.14% drop in accuracy.

Accuracy is also significantly higher compared to the models with the same latency. Our $0.25\times$B4 obtains 3.0% higher top-1 accuracy compared to the original EfficientNetV1-B1 while maintaining a similar latency. EfficientNetV2-B3 accelerated by $1.7\times$ gains 0.93% with respect to EfficientNetV2-B2. Trade-off between accuracy and latency is obtained by accelerating EfficientNet models by up to $3.6\times$ while scarifying only 0.3% accuracy as for the $0.25\times$B6 variant.

As observed in Table 2, we apply HANT to a large spectrum of model sizes on two different target hardware types. This is possible in HANT due to (i) independent and fully parallel layer-wise distillation, (ii) separating search from pretraining stage, and hence alleviating the need to retrain candidate operations before optimizing a network for a new hardware, and (iii) using a light-weight linear optimization for architecture search that allows for efficient and effective searching in a large architecture space.

3.2 Analysis

We next provide detailed ablations to analyze and validate our design choices/assumptions in HANT for both pretraining and search phases, joint with associated insights. Unless otherwise stated we used EfficientNetV2-B3 as our target for the ablation.

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 architecture. This assumption is verified by sampling 1000 architectures (different budget constraints) by minimizing the ILP objective and observing the real loss function (accuracy for simplicity). Results are shown in Fig. 2(a). 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. 2(b), where we sample 1000 random architectures per 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 due to our overlap constraint. If we increase the number of solutions found by ILP from K=1 to K=100, performance improves further.

Candidate architecture evaluation plays an important role in HANT. This step finds the best architecture quickly out of multiple candidates, generated by the ILP solver, by evaluating them on 6k images from the train data. The procedure is built on the assumption that accuracy of the model before finetuning (just by plugging all candidate operations) is a reasonable indicator of the relative model performance after finetuning. As shown in Fig. 2(b), when plugging pretrained operations 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.

Verifying pre/post finetuning model ranking. We finetuned 31 models sampled from the pertained operations and measured their top1 accuracy. The linear objective used in our ILP ranks those model architectures with the correlation score of 0.66 for Kendall Tau index and 0.85 for Spearman. When we measure the training accuracy of the candidate architecture before finetuning as the ranking indicator, the correlation is significantly improved to 0.835 for Kendall Tau index and 0.945 for Spearman. Those observations are shown in Fig. 2(c). In Table 3, we use different metrics to rank candidate architectures, and observe that most correlates well with the final model accuracy after the finetuning.

Correlation	Accuracy	Cross-entropy	KL divergence	MSE
Kendal Tau	0.835	0.824	0.832	0.811
Spearman	0.945	0.947	0.954	0.944

Table 3: Candidate evaluation metrics for model ranking before and after finetuning.

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 (Xie et al., 2019; Wu et al., 2019; Vahdat et al., 2020). We follow SNAS (Xie et al., 2019) in this experiment; (iii) *REINFORCE* is a stochastic optimization framework borrowed from reinforcement learning and adopted for architecture search (Pham et al., 2018; Zoph et al., 2018; Tan et al., 2019b). We follow an E-NAS-like (Pham et al., 2018) architecture search for (iii) and use weight sharing for (ii) and (iii). Experiments are conducted on EfficientNetV1-B2 accelerated to $0.45\times$ original latency. The final validation top-1 accuracy after finetuning are presented in Table 4. Our proposed ILP achieves higher accuracy ($+0.43\%$) compared to the second best method E-NAS while being $821\times$ faster in search.

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 (Xie et al., 2019)	74.20	16.3 GPU/h
E-NAS (Pham et al., 2018)	78.85	61.6 GPU/h

Table 4: Comparing methods for candidate selection (NAS). Our proposed ILP is better ($+0.43\%$) and $821\times$ faster.

Zero-shot HANT. 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 score vector for the *Identity* operator will be required alongside the LUT for teacher and Identity operations. We report zero-shot results in Table 5. We observe that HANT efficiently finds residual blocks that can be skipped. This unique property of HANT is enabled by search with ILP, and, to the best of our knowledge, no other NAS-based accelerating techniques are capable of zero-shot constrained NAS.

Setup	Zero-shot	Full
0.55xEfficientNetV1-B2	79.40	80.11
0.45xEfficientNetV1-B2	78.68	79.71
1.00xEfficientNetV1-B2	80.39	

Table 5: Zero-shot HANT with only skip connections.

Pretraining insights. To gain more insights into the tradeoff between the accuracy and speed of each operation, we analyze the the pretrained candidate operation pool tailored for EffientNetV1B2. A detailed Figure 7 is shown in the Appendix A.2, which highlights trade-offs for three layers spaced equally throughout the network. Here, we provide general observations as follows.

We observe that no operation outperforms the teacher in terms of accuracy; changing pretraining loss from MSE to 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. The identity operation unsurprisingly achieves very high speedups at the expense of increasingly poor accuracy compared to the teacher as a function of depth. Visual transformer blocks (`ViT`) and bottleneck transformer blocks (`bot_trans`) show neither a speedup advantage nor are able to recover the accuracy of the teacher.

3.2.1 ARCHITECTURE INSIGHTS

Figures 4-6 in the appendices visualize the final architectures discovered by HANT. Next, we share the insights observed on these architectures.

EfficientNetV1. Observing final architectures obtained by HANT on EfficientNetV1 family, particularly 0.55×B2 version optimized for GPU, we discover that most of the modifications are done to the first half of the model: (i) squeeze-and-excitation are removed, while later layers are still equipped with them; (ii) dense convolutions (like inverted stacked or bottleneck residual blocks) replace depth-wise separable counterparts; (iii) expansion factor is reduced from 6 to 3.5 on average. *Surprisingly, HANT automatically discovers the same design choices that are introduced in the EfficeintNetV2 family when optimized for datacenter inference.*

EfficientNetV2. HANT accelerates EfficientNetV2-B3 by 2×, leading to 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, the accelerated model still outperforms EfficientNetV2-B2 and B1.

ResNeST50. HANT applied on ResNeST50d_1s4x24d 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.

3.2.2 ABLATIONS ON FINETUNING

In Table 6, we look deeper into the finetuning step. For this experiment, we select our 0.45×EfficientNetV1-B2 with the final accuracy of 79.71%. Reinitializing all weights in the model, in contrary to loading them from the pretraining stage, results in 79.42%, while loading only teacher cells results in 79.69%. The results indicate the importance of pretraining stage (i) to find a good architecture and (ii) to boost finetuning.

Setting	Preloaded weights	Final accuracy (%)
0.45xEfficeintNetV1-B2	all	79.71
No Knowledge distil.	all	79.06
0.45xB2	none	79.42
0.45xB2	teacher	79.69
B0 (default)	all	77.70
B0 with KD loss	all	78.72
B0 from scratch	none	78.01

Table 6: Ablations on the finetuning

Knowledge distillation plays a key role in student-teacher setups. When it is disabled, we observe an accuracy degradation of 0.65%. This emphasises the benefit of training a larger model and then distilling towards a smaller one for inference.

We further verify whether we can achieve a similar high accuracy through knowledge distillation from EfficeintNetV1-B2 to EfficeintNetV1-B0 in the same setting. The top-1 accuracy of 78.72% is still 1% less than HANT's accuracy. Compared with training 0.45xB2 from scratch (79.42%), B0 achieves 78.01% – a 1.4% drop in accuracy. We conclude that transforming the teacher into more efficient architecture is significantly more beneficial than fine-tuning smaller models.

4 CONCLUSION

In this paper, we proposed HANT, a hardware-aware network transformation framework for accelerating pretrained neural networks. HANT 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 HANT accelerates these architectures by several folds with only a small drop in the accuracy. We analyzed the selected operations and we provided new insights for future neural architecture designs.

REPRODUCIBILITY STATEMENT

We provide details on how to reimplement our work in the Appendix. We share hyper parameters for pretraining, search and finetuning. We are in the midst of releasing the code with exact commands to reproduce our results. Latency measurements were conducted 10 times and the mean value is reported.

REFERENCES

Irwan Bello. Lambdanetworks: Modeling long-range interactions without attention. *arXiv preprint arXiv:2102.08602*, 2021.

Irwan Bello, William Fedus, Xianzhi Du, Ekin D. Cubuk, Aravind Srinivas, Tsung-Yi Lin, Jonathon Shlens, and Barret Zoph. Revisiting resnets: Improved training and scaling strategies, 2021.

Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *arXiv preprint arXiv:2003.03033*, 2020.

Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HylVB3AqYm.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/pdf/1908.09791.pdf.

Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation, 2018.

Yunpeng Chen, Jianan Li, Huaxin Xiao, Xiaojie Jin, Shuicheng Yan, and Jiashi Feng. Dual path networks. *arXiv preprint arXiv:1707.01629*, 2017.

François Chollet. Xception: Deep learning with depthwise separable convolutions, 2017.

Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. Repvgg: Making vgg-style convnets great again. *arXiv preprint arXiv:2101.03697*, 2021.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016a.

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

Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks, 2018a.

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*, pp. 784–800, 2018b.

Byeongho Heo, Sangdoo Yun, Dongyoon Han, Sanghyuk Chun, Junsuk Choe, and Seong Joon Oh. Rethinking spatial dimensions of vision transformers, 2021.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. Squeeze-and-excitation networks, 2019.

Changlin Li, Jiefeng Peng, Liuchun Yuan, Guangrun Wang, Xiaodan Liang, Liang Lin, and Xiaojun Chang. 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.

Xiang Li, Wenhai Wang, Xiaolin Hu, and Jian Yang. Selective kernel networks, 2019.

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

Yu Liu, Xuhui Jia, Mingxing Tan, Raviteja Vemulapalli, Yukun Zhu, Bradley Green, and Xiaogang Wang. 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.

Bert Moons, Parham Noorzad, Andrii Skliar, Giovanni Mariani, Dushyant Mehta, Chris Lott, and Tijmen Blankevoort. Distilling optimal neural networks: Rapid search in diverse spaces. *arXiv preprint arXiv:2012.08859*, 2020.

NVIDIA. TensorRT Library. `https://developer.nvidia.com/tensorrt`, 2021. [Online; accessed 10-May-2021].

Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces, 2020.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pp. 525–542. Springer, 2016.

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, pp. 2902–2911. PMLR, 2017.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115 (3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Shane Ryoo, C. Rodrigues, Sara S. Baghsorkhi, S. S. Stone, D. Kirk, and W. Hwu. 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.

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

Bharat Bhusan Sau and Vineeth N Balasubramanian. Deep model compression: Distilling knowledge from noisy teachers. *arXiv preprint arXiv:1610.09650*, 2016.

Aravind Srinivas, Tsung-Yi Lin, Niki Parmar, Jonathon Shlens, Pieter Abbeel, and Ashish Vaswani. Bottleneck transformers for visual recognition. *arXiv preprint arXiv:2101.11605*, 2021.

Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017.

Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pp. 6105–6114. PMLR, 2019a.

Mingxing Tan and Quoc V Le. Mixconv: Mixed depthwise convolutional kernels. *arXiv preprint arXiv:1907.09595*, 2019b.

Mingxing Tan and Quoc V Le. Efficientnetv2: Smaller models and faster training. *arXiv preprint arXiv:2104.00298*, 2021.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. *arXiv preprint arXiv:1807.11626*, 2018.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019a.

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

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. *arXiv preprint arXiv:2012.12877*, 2020.

Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. *arXiv preprint arXiv:2103.17239*, 2021.

Arash Vahdat, Arun Mallya, Ming-Yu Liu, and Jan Kautz. Unas: Differentiable architecture search meets reinforcement learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11266–11275, 2020.

Tom Veniat and Ludovic Denoyer. Learning time/memory-efficient deep architectures with budgeted super networks. *arXiv preprint arXiv:1706.00046*, 2017.

Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. 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, 2020a.

Guibin Wang, Yisong Lin, and Wei Yi. 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.

Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, et al. Deep high-resolution representation learning for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 2020b.

Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. HAQ: Hardware-aware automated quantization with mixed precision. In *CVPR*, 2019.

Qilong Wang, Banggu Wu, Pengfei Zhu, Peihua Li, Wangmeng Zuo, and Qinghua Hu. Eca-net: Efficient channel attention for deep convolutional neural networks. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020c.

Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019.

Bichen Wu, Yanghan Wang, Peizhao Zhang, Yuandong Tian, Peter Vajda, and Kurt Keutzer. Mixed precision quantization of convnets via differentiable neural architecture search. *arXiv preprint arXiv:1812.00090*, 2018.

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/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431, 2016. URL http://arxiv.org/abs/1611.05431.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: stochastic neural architecture search. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rylqooRqK7.

Zheng Xu, Yen-Chang Hsu, and Jiawei Huang. Training shallow and thin networks for acceleration via knowledge distillation with conditional adversarial networks. In *ICLR Workshop*, 2018.

Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. *Energy*, 41:46, 2018a.

Yifan Yang, Qijing Huang, Bichen Wu, Tianjun Zhang, Liang Ma, Giulio Gambardella, Michaela Blott, Luciano Lavagno, Kees Vissers, John Wawrzynek, et al. Synetgy: Algorithm-hardware co-design for convnet accelerators on embedded fpgas. *arXiv preprint arXiv:1811.08634*, 2018b.

Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor Darrell. Deep layer aggregation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2403–2412, 2018.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016. URL http://arxiv.org/abs/1605.07146.

Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Haibin Lin, Zhi Zhang, Yue Sun, Tong He, Jonas Mueller, R Manmatha, et al. Resnest: Split-attention networks. *arXiv preprint arXiv:2004.08955*, 2020.

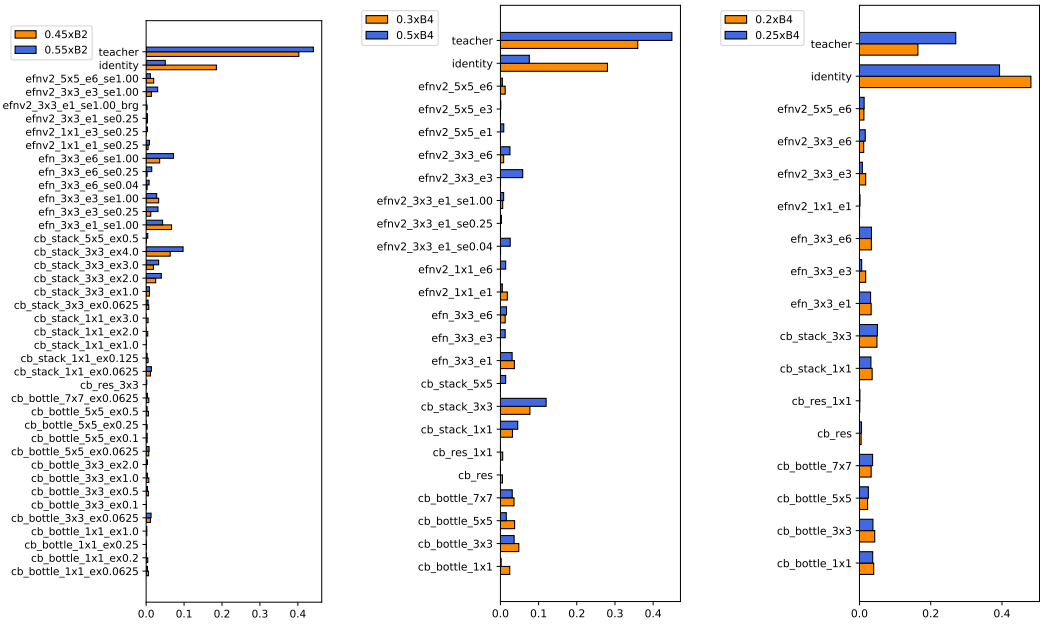
X Zhang, X Zhou, M Lin, and J Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*, 2017.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, pp. 8697–8710, 2018.

A APPENDIX

Distribution of selected operations. In Fig. 3, we provide the histogram of selected operations for three EfficientNeV1t networks and different acceleration ratios. All statistics are calculated for the first 100 architectures found via integer optimization.



(a) B2 selected ops (b) B4 selected ops (c) B4 selected ops

Figure 3: The histogram of selected operations for top-100 models of EfficientNetV1 derivatives. Teacher layers are often selected especially in the deeper layers of the network as we visualize in Fig. 4 and Fig. 5. The identity layer is also selected often especially when the target latency is low. Interestingly, simple layers with two stacked convolution (`cb_stack`) in the CBRCB structure (C-convolution, B-batchnorm, R-ReLU) are selected most frequently after the teacher and identity operations. Additionally we see a higher chance of selecting inverted residual blocks (`efn` and `efn2`) with no squeeze-and-excitation operations `se1.00`.

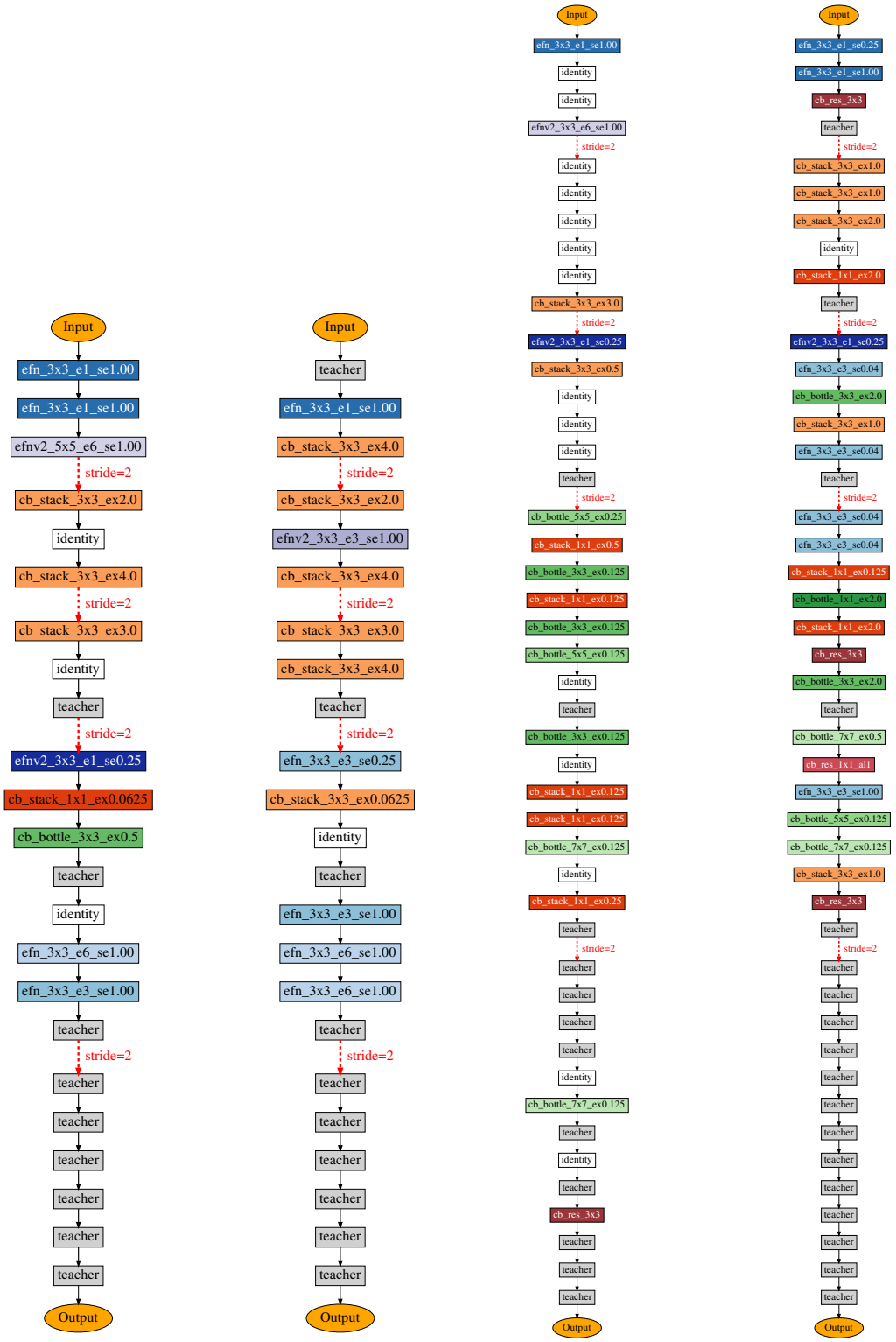
Final architectures. Fig. 4 and Fig. 5 visualize the final architectures found by HANT. We observe that teacher ops usually appear towards the end of the networks. Identity connections appear in the first few resolution blocks where the latency is the highest to speed up inference, for example 0.2×B4 has 2, 1, 1 in the first 3 resolution blocks from original 3, 4, 4.

A.1 ADDITIONAL IMPLEMENTATION DETAILS

We next provide details on chosen batch size defined as `bs` and learning rate `lr`, joint with other details required to replicate results in the paper.

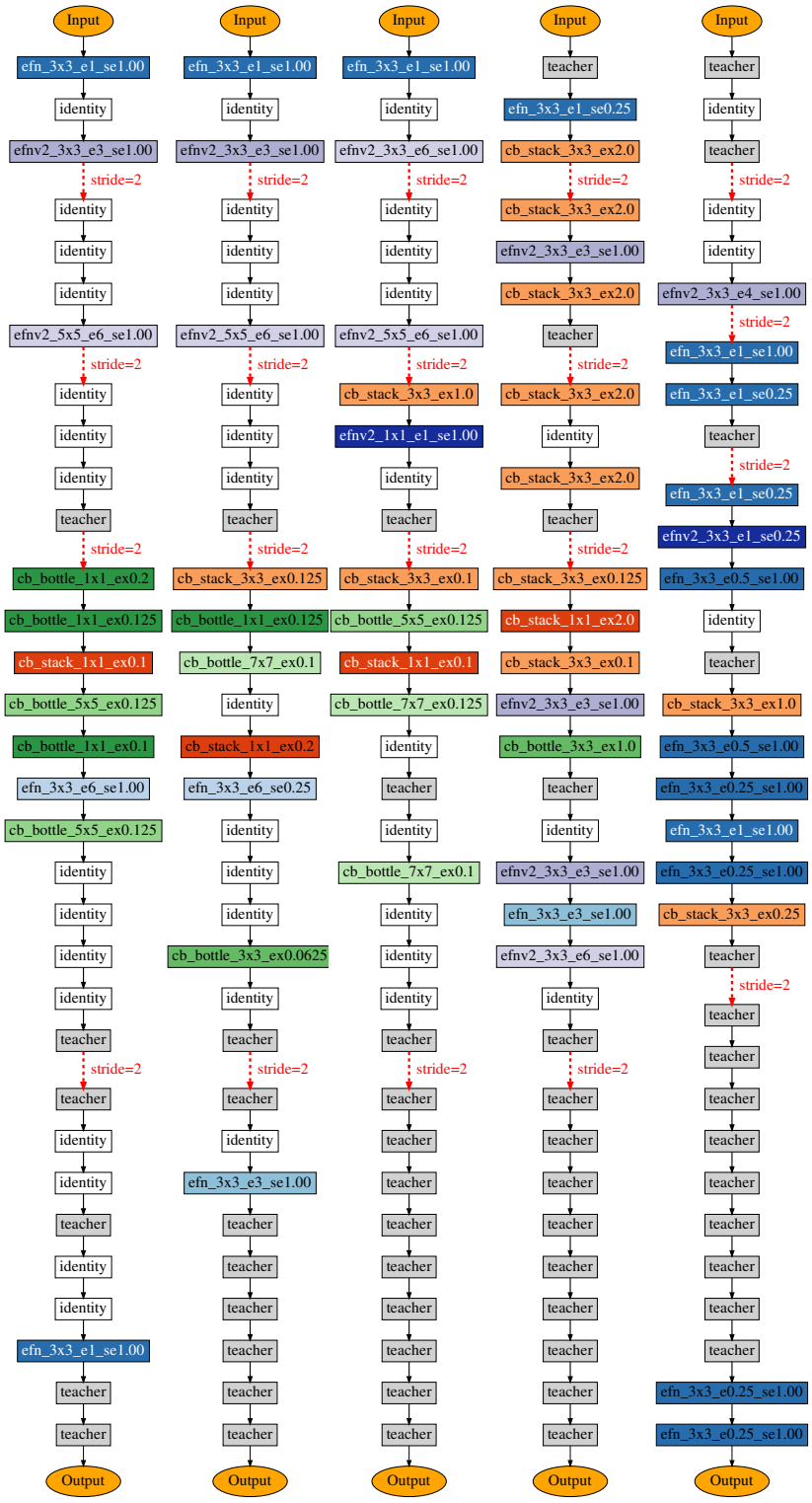
Pretraining implementation. Pretraining stage was implemented to distill a single operator over all layers in parallel on 4xV100 NVIDIA GPU with 32GB. For EfficientNet-B2 we set `lr`=0.008 with `bs`=128, for EfficientNet-B4 `lr`=0.0005 with `bs`=40, and EfficientNet-B6 `lr`=0.0012 with `bs`=12. We set $\gamma_{\text{MSE}} = 0.001$. We run optimization with an SGD optimizer with no weight decay for 1 epoch only.

Finetuning implementation. Final model finetuning runs for 100 epochs. We set `bs`=128 and `lr`=0.02 for EfficientNet-B2 trained on 2x8 V100 NVIDIA GPU; for EfficientNet-B4 derivatives



(a) 0.45xB2 (b) 0.55xB2 (c) 0.25xB6 (d) 0.5xB6

Figure 4: Final architectures selected by HANT as EfficientNet-B2/B6 derivatives.



(a) 0.2xV1-B4 (b) 0.25xV1-B4 (c) 0.3xV1-B4 (d) 0.5xV1-B4 (e) 0.5xV2-B3

Figure 5: Final architectures selected by HANT as EfficientNetV1-B4/V2-B3 derivatives.

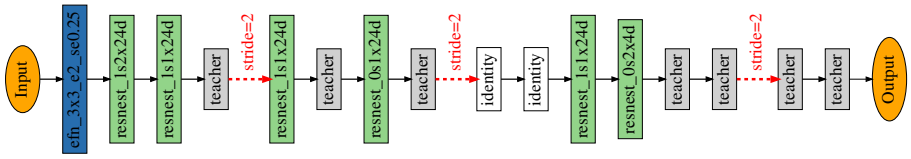


Figure 6: Final architectures selected by HANT as 0.7xResNeST50d_1s4x24d

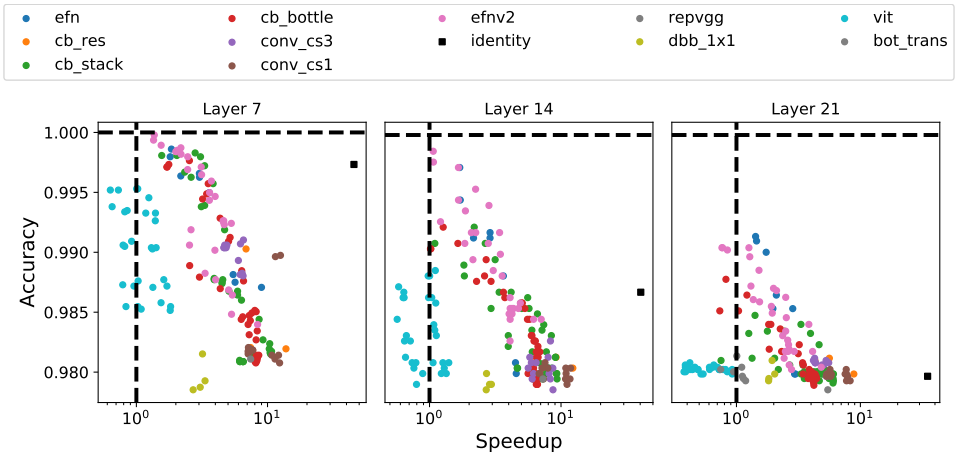


Figure 7: Result of the pretraining stage for EfficientNetB2, showing three layers equally spaced throughout the network: 7, 14 and 21. Speedup is measured as the ratio between the latency of the teacher and the latency of the student operation (higher is better). We measure latency using Pytorch FP16. Accuracy is the ratio of the operation's accuracy and the teacher's (higher is better). The dashed black lines correspond to the teacher.

we set `bs`=128 and `lr`=0.04, for EfficientNet-B6 `bs`=48 and `lr`=0.08 on 4x8 V100 NVIDIA GPU. Learning rate was set to be 0.02. We set γ_{CE} and γ_{KL} to 1.

Latency look up table creations. We measure the latency on V100 NVIDIA GPU with TensorRT in FP16 mode for batch size of 128 images. For Xeon CPU latency we use a batch size of 1. Input and output stems are not included in latency LUT. This results in a small discrepancy between theoretical and real speed. As a result, we use latency LUT for operator evaluation, and report the final real latency for the unveiled final models.

A.2 Candidate pretraining insights

Latency-accuracy tradeoff for different operations after pretraining is shown in the Figure 7. Observations from these plots are discussed in Seection 3.2.

The choice of pretraining loss. To motivate our choice of MSE for pretraining, we investigate the distribution of activations at the output of residual blocks. We observe that for all blocks, activations follow a Gaussian-like distribution. Shapiro-Wilk test for normality averaged over all layers is 0.99 for EfficientNetV1-B2, and 0.988 for EfficientNetV2-B3. Given this observation, MSE error seems a reasonable loss function to minimize.

A.3 Detailed comparison to prior work

For comparison to prior work we look into latest models from the out-of-the-box *timm package* Wightman (2019) with Apache-2.0 License. We include detailed individual method names and references as follows:

- `efficientnet`: Efficientnet Tan & Le (2019a).
- `cait`: Class-attention in image transformers Touvron et al. (2021).
- `cspnet`: Cross-stage partial network Wang et al. (2020a).
- `deit`: (Data-efficient) vision transformer Touvron et al. (2020).
- `dla`: Deep layer aggregation Yu et al. (2018).
- `dpn`: Dual-path network Chen et al. (2017).
- `ecanet`: Efficient channel attention network Wang et al. (2020c).
- `hrnet`: High-resolution network Wang et al. (2020b).
- `inception`: Inception V3 Szegedy et al. (2016) and V4 Szegedy et al. (2017).
- `mixnet`: MixConv-backed network Tan & Le (2019b).
- `ofa`: Once-for-all network Cai et al. (2020).
- `pit`: Pooling Vision Transforms Heo et al. (2021).
- `regnetX`: Regnet network Radosavovic et al. (2020), accuracy is taken from the original paper.
- `regnetY`: Regnet network Radosavovic et al. (2020) with squeeze-and-excitation operations, accuracy is taken from the original paper.
- `repvgg`: RepVGG Ding et al. (2021).
- `resnest101_e`: Resnest101 (with bag of tricks) He et al. (2018a).
- `resnest50_d`: Resnest50 (with bag of tricks) He et al. (2018a).
- `resnet50_d`: Resnet50 (with bag of tricks) He et al. (2018a).
- `resnetrs10_1`: Resnet rescaled Bello et al. (2021).
- `resnetrs15_1`: Resnet rescaled Bello et al. (2021).
- `resnetrs5_0`: Resnet rescaled Bello et al. (2021).
- `resnext50d_32x4d`: Resnext network (with average pooling downsampling) Xie et al. (2016).
- `seresnet5_0`: Squeeze Excitement Resnet50 Hu et al. (2019).
- `skresnext50_32x4d`: Selective kernel Resnext50 Li et al. (2019).
- `vit-base`: Visual Transformer, base architecture.
- `vit-large_384`: Visual Transformer, large architecture, 384 resolution Dosovitskiy et al. (2020).
- `wide_resnet50_2`: Resnet50 with $2\times$ channel width Zagoruyko & Komodakis (2016).
- `xception6_5`: Xception network (original) Chollet (2017).
- `xception7_1`: Xception network aligned Chen et al. (2018).

A more detailed comparison with other models is shown in the Figure 8. We observe that models resulted from HANT acceleration are performing better than the most of other approaches. All of the models for HANT used LUTs computed with TensorRT and clearly the speed up in the TensorRT figure is larger when compared with other methods. On the same time if model latency is estimated with Pytorch, we still get top models that outperform many other models.

Additional comparison to prior work. For additional insights, we also include detailed experimental comparisons to DONNA proposed by Moons et al. (2020). In Table 7, we demonstrate consistent performance improvements by HANT compared to DONNA across varying latency budgets. We matched the latency of final models with settings (batch size, hardware etc) to those of DONNA.

PyTorch FP16: TensorRT:

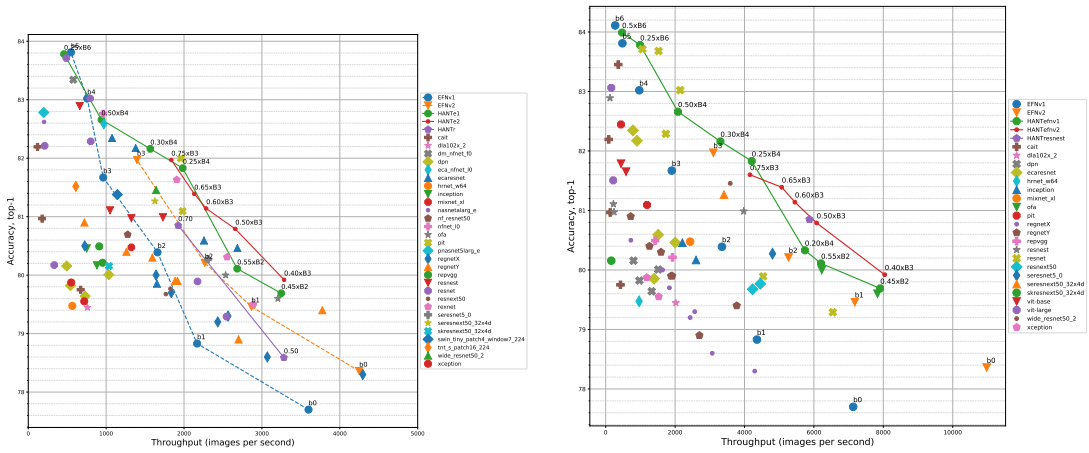


Figure 8: Comparison with other models from `TIMM` package.

Models	Latency (ms) ↓	Top-1 Acc. (%) ↑
DONNA Moons et al. (2020)	20.0	78.9
0.45×EfficientNetV1-B2 **(HANT)**	**18.9**	**79.7 (+0.8)**
DONNA Moons et al. (2020)	25.0	79.5
0.55×EfficientNetV1-B2 **(HANT)**	**24.1**	**80.1 (+0.6)**

Table 7: Additional comparison to prior work. Latency values measured at batch size 32 with PyTorch FP32. In brackets are our improvements.