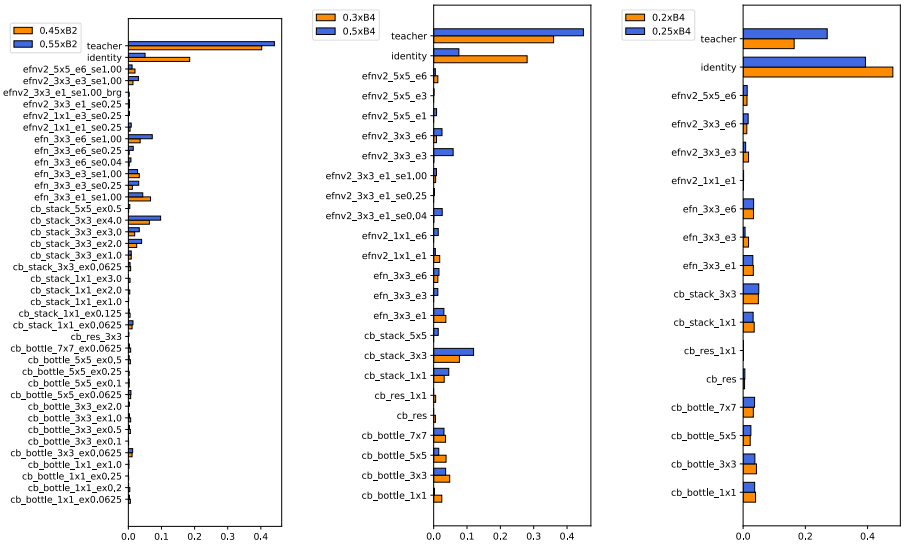
Appendix

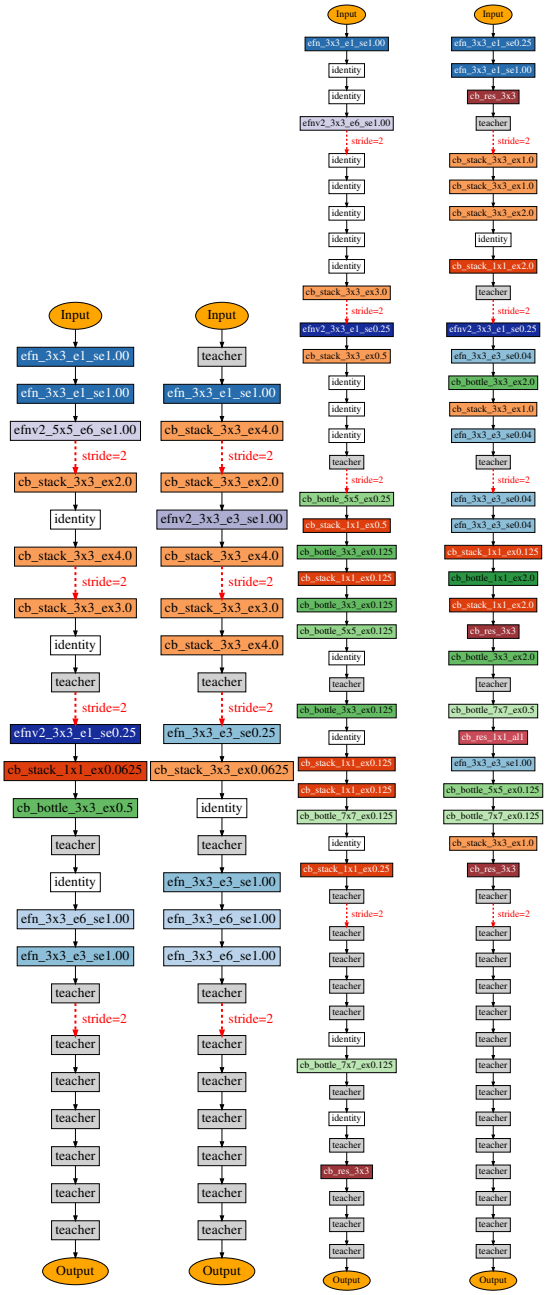
Distribution of selected operations. In Fig. 5, we provide the histogram of selected operations for three EfficientNeV1t networks and different acceleration ratios. All statistics are calculated for the first 100 architectures found via integer optimization.



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

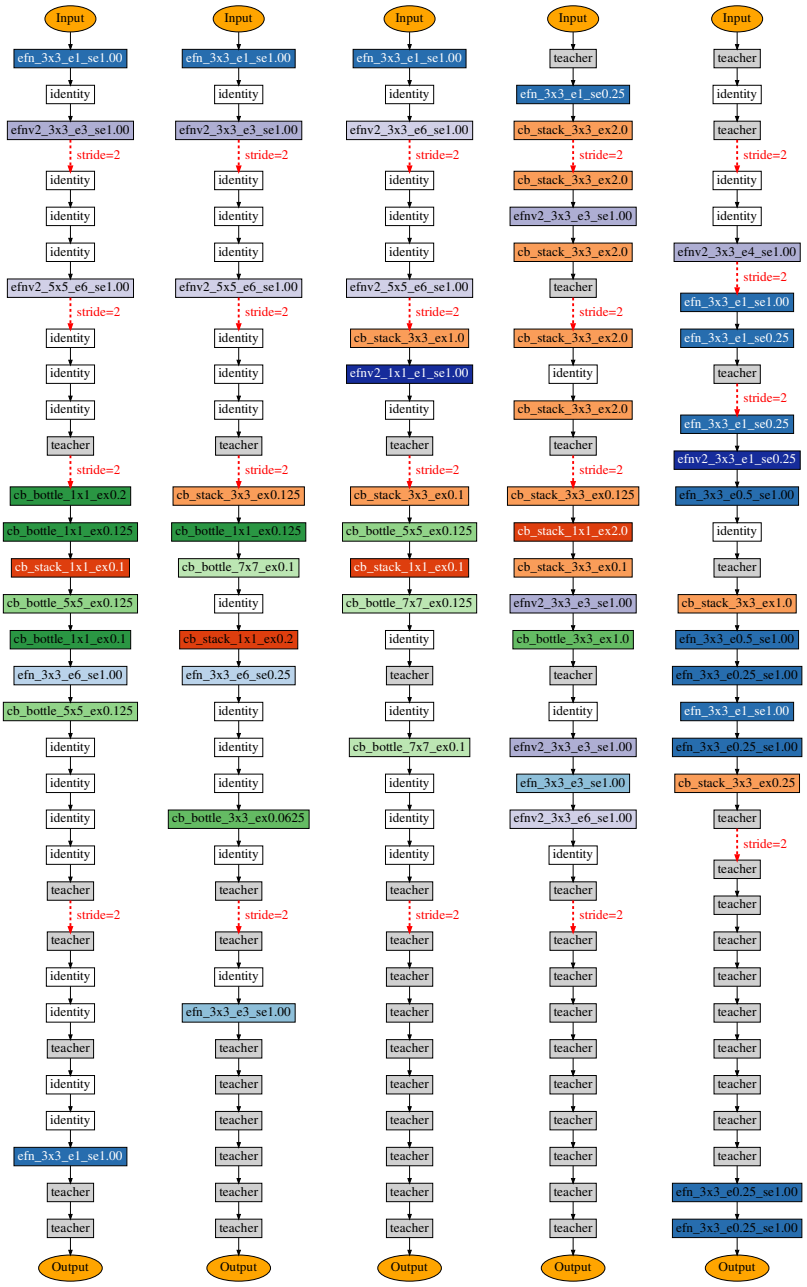
Fig. 5: The histogram of selected operations for top-100 models of EfficientNetV1 derivatives. Teacher layers are often selected especially in the deeper layers of the network as we visualize in Fig. 6 and Fig. 7. The identity layer is also selected often especially when the target latency is low. Interestingly, 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. 6 and Fig. 7 visualize the final architectures found by LANA. 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) 0.45xB2 (b) 0.55xB2 (c) 0.25xB6 (d) 0.5xB6

Fig. 6: Final architectures selected by LANA 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

Fig. 7: Final architectures selected by LANA as EfficientNetV1-B4/V2-B3 derivatives.

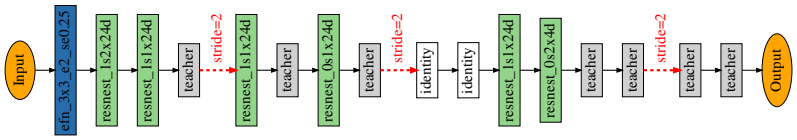


Fig. 8: Final architectures selected by LANA as 0.7xResNeST50d_1s4x24d.

4.1 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 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.

Table 8: EfficientNetV1 accelerated by LANA for CPU inference.

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	80.07	113

4.2 CPU optimized models

We extended experiments of EfficientNetV1 by optimizing LANA on CPU. In this case only LUT is different and pretrained operations are the same.

4.3 Effect of the model finetuning duration

In the paper, finetuning of the final architecture is performed for 100 epoch for most of the experiments with being 50 epochs for ablations studies. The length of finetuning affects accuracy, therefore we compare performance by changing it Table 9.

Table 9: Ablations on the length of finetuning step. On the right side, we report EfficientNet accuracy for models with the same latency as ours.

LANA	Epochs					EfficientNetV1
model	5	10	25	50	100	alternative
0.45xB2	78.69	79.08	79.19	79.58	79.71	77.70 (B0)
0.55xB2	79.05	79.47	79.84	80.00	80.11	78.83 (B1)

4.4 Candidate pretraining insights

Latency-accuracy tradeoff for different operations after pretraining is shown in the Figure 9. Observations from these plots are discussed in Section 3.2.

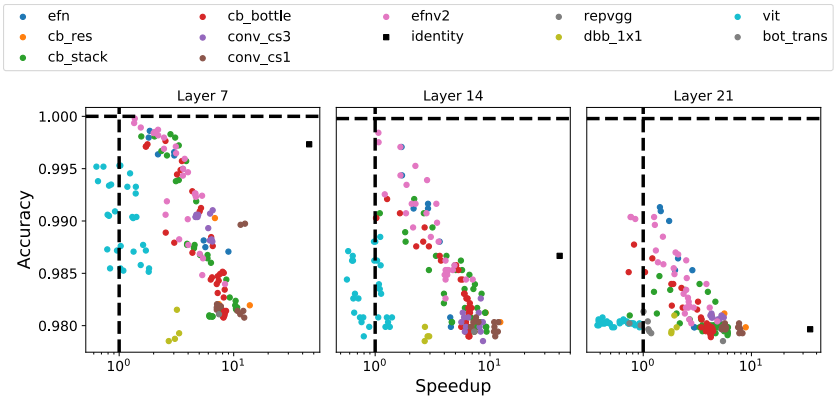


Fig. 9: Result of the pretraining stage for EfficientNetB2, showing three layers equally spaced throughout the network: 7, 14 and 21. Speedup is measured as the ratio between the latency of the teacher and the latency of the student operation (higher is better). We measure latency using Pytorch FP16. Accuracy is the ratio of the operation's accuracy and the teacher's (higher is better). The dashed black lines correspond to the teacher.

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.

4.5 Detailed comparison to prior work

For comparison to prior work we look into latest models from the out-of-the-box *timm package* [85] with Apache-2.0 License. We include detailed individual method names and references as follows:

`efficientnet`: Efficientnet [73].
`cait`: Class-attention in image transformers [77].
`cspnet`: Cross-stage partial network [80].
`deit`: (Data-efficient) vision transformer [76].
`dla`: Deep layer aggregation [96].
`dpn`: Dual-path network [10].
`ecanet`: Efficient channel attention network [84].
`hrnet`: High-resolution network [82].
`inception`: Inception V3 [69] and V4 [68].
`mixnet`: MixConv-backed network [74].
`ofa`: Once-for-all network [5].
`pit`: Pooling Vision Transforms [30].
`regnetX`: Regnet network [58], accuracy is taken from the original paper.
`regnetY`: Regnet network [58] with squeeze-and-excitation operations, accuracy is taken from the original paper.
`repvgg`: RepVGG [15].
`resnest101_e`: Resnest101 (with bag of tricks) [23].
`resnest50_d`: Resnest50 (with bag of tricks) [23].
`resnet50_d`: Resnet50 (with bag of tricks) [23].
`resnetrs10_1`: Resnet rescaled [3].
`resnetrs15_1`: Resnet rescaled [3].
`resnetrs5_0`: Resnet rescaled [3].
`resnext50d_32x4d`: Resnext network (with average pooling downsampling) [89].
`seresnet5_0`: Squeeze Excitement Resnet50 [32].
`skresnext50_32x4d`: Selective kernel Resnext50 [40].
`vit-base`: Visual Transformer, base architecture.
`vit-large_384`: Visual Transformer, large architecture, 384 resolution [17].
`wide_resnet50_2`: Resnet50 with 2× channel width [99].
`xception6_5`: Xception network (original) [12].
`xception7_1`: Xception network aligned [9].

A more detailed comparison with other models is shown in the Figure 10. We observe that models resulted from LANA acceleration are performing better than the most of other approaches. All of the models for LANA 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.

PyTorch FP16: TensorRT:

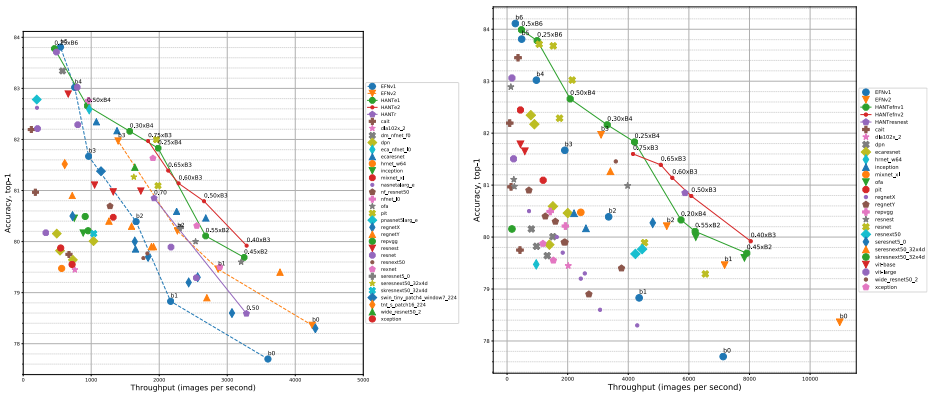


Fig. 10: Comparison with other models from `TIMM` package.