Appendix

A. More Specifications of the Search Space

Inspired by EfficientNet (Tan & Le, 2019) and TF-NAS (Hu et al., 2020), we build a layer-wise search space, as explained in Section 3.1 and depicted in Figure 2 and in Table 3. The input shapes and the channel numbers are the same as EfficientNetB0. Similarly to TF-NAS and differently from EfficientNet-B0, we use ReLU in the first three stages. As specified in Section 3.1.1, the ElasticMBInvRes block is our *elastic* version of the MBInvRes block, introduced in (Sandler et al., 2018a). Those blocks of stages 3 to 8 are to be searched for, while the rest are fixed.

Stage	Input	Operation	C_{out}	Act	b
1	$224^2 \times 3$	3×3 Conv	32	ReLU	1
2	$112^3 \times 32$	MBInvRes	16	ReLU	1
3	$112^2 \times 16$	ElasticMBInvRes	24	ReLU	$[2, 4]$
4	$56^2 \times 24$	ElasticMBInvRes	40	Swish	$[2, 4]$
5	$28^2 \times 40$	ElasticMBInvRes	80	Swish	$[2, 4]$
6	$14^2 \times 80$	ElasticMBInvRes	112	Swish	$[2, 4]$
7	$14^2 \times 112$	ElasticMBInvRes	192	Swish	$[2, 4]$
8	$7^2 \times 192$	ElasticMBInvRes	960	Swish	1
9	$7^2 \times 960$	1×1 Conv	1280	Swish	1
10	$7^2 \times 1280$	AvgPool	1280	-	1
11	1280	Fc	1000	-	1

Table 3. Macro architecture of the one-shot model. "MBInvRes" is the basic block in (Sandler et al., 2018a). "ElasticMBInvRes" denotes our *elastic* blocks (Section 3.1.1) to be searched for. "C_{out}" stands for the output channels. Act denotes the activation function used in a stage. "b" is the number of blocks in a stage, where $[\underline{b}, \bar{b}]$ is a discrete interval. If necessary, the down-sampling occurs at the first block of a stage.

The configurations of the ElasticMBInvRes blocks $c \in \mathcal{C}$ are sorted according to their expected latency as specified in Table 4.

B. Searching for the Expansion Ratio

Searching for expansion ration (er), as specified in Section 3.1.1, involves the summation of feature maps of different number of channels:

$$y^s_{b,er} = \sum_{er \in \mathcal{A}_{er}} \alpha^s_{b,er} \cdot PWC^s_{b,er}(x^s_b) \qquad (17)$$

where $PWC^s_{b,er}$ is the point-wise convolution of block b in stage s with expansion ratio er.

The summation in (17) is made possible by calculating $PWC^s_{b,\bar{er}}$ only once, where $\bar{er} = \max \mathcal{A}_{er}$, and masking its output several times as following:

$$y^s_{b,er} = \sum_{er \in \mathcal{A}_{er}} \alpha^s_{b,er} \cdot PWC^s_{b,\bar{er}}(x^s_b) \odot \mathbb{1}_{C \leq er \times C_{in}} \qquad (18)$$

where $\odot$ is a point-wise multiplication, C_{in} is the number of channels of x^s_b and the mask tensors $\mathbb{1}_{C \leq er \times C_{in}}$ are of the same dimensions as of $PWC^s_{b,\bar{er}}(x^s_b)$ with ones for all channels C satisfying $C \leq er \times C_{in}$ and zeros otherwise.

Thus, all of the tensors involved in the summation have the same number of channels, i.e. $\bar{er} \times C_{in}$, while the weights of the point-wise convolutions are shared. Thus we gain the benefits of weight sharing, as specified in Section 3.3.

c	er	k	se
1	3	3×3	off
2	3	5×5	on
3	3	3×3	off
4	3	5×5	on
5	4	3×3	off
6	4	5×5	on
7	4	3×3	off
8	4	5×5	on
9	6	3×3	off
10	6	5×5	on
11	6	3×3	off
12	6	5×5	on

Table 4. Specifications for each indexed configuration $c \in \mathcal{C}$. "er" stands for the expansion ratio of the point-wise convolutions, "k" stands for the kernel size of the depth-wise separable convolutions and "se" stands for Squeeze-and-Excitation (SE) with on and off denoting with and without SE respectively. The configurations are indexed according to their expected latency.

C. Multipath Sampling Code

We provide a simple PyTorch (Paszke et al., 2019) implementation for sampling multiple distinctive paths (subnetworks of the one-shot model) for every image in the batch, as specified in Section 3.3. The code is presented in figure 6.

```
import torch
from torch.nn.functional import
    gumbel_softmax

def multipath(a, ops, x):
    assert C = len(a) == len(ops)
    bs = x.shape[0]
    a = torch.log(a).repeat(bs)
    a = a.reshape(bs, C).transpose(0, 1)
    a_hat = gumbel_softmax(a)

    o = torch.zeros_like(x)
    for ah, op in in zip(a_hat, ops):
        o += ah.view(-1, 1, 1, 1) * op(x)

    return o
```

Figure 6. PyTorch Multipath Code

Note that the smaller batch size (16 images per GPU) during the multi-path stage (section 3.3) is due to the memory

required for any differentiable NAS applying a weighted sum of the outputs of different operations. This is due to storing gradient information for each one and is not an artifact of the multi-path sampling, i.e. the same batch size is required by the common single-path approach (Figure 4).

D. A Brief Derivation of the FW Step

Suppose $\mathcal{D}$ is a compact convex set in a vector space and $f\colon \mathcal{D} \to \mathbb{R}$ is a convex, differentiable real-valued function. The Frank-Wolfe algorithm (Frank et al., 1956) iteratively solves the optimization problem:

$$\min_{\mathbf{x}\in\mathcal{D}} f(\mathbf{x}). \tag{19}$$

To this end, at iteration $k+1$ it aims at solving:

$$\min_{\mathbf{x_k}+\mathbf{\Delta}\in\mathcal{D}} f(\mathbf{x_k}+\mathbf{\Delta}). \tag{20}$$

Using a first order taylor expansion of f, (20) is approximated in the neighborhood of $f(x_k)$, and thus the problem can be written as:

$$\min_{\mathbf{x_k}+\mathbf{\Delta}\in\mathcal{D}} \mathbf{\Delta}^T \nabla f(\mathbf{x_k}) \tag{21}$$

Replacing $\mathbf{\Delta}$ with $\gamma(\boldsymbol{s}-\mathbf{x_k})$ for $\gamma\in[0,1]$, problem (21) is equivalent to:

$$\min_{\mathbf{x_k}+\gamma(\mathbf{s}-\mathbf{x_k})\in\mathcal{D}} \gamma(s-\mathbf{x_k})^T \nabla f(\mathbf{x_k}). \tag{22}$$

Assuming that $\mathbf{x_k}\in\mathcal{D}$, since $\mathcal{D}$ is convex, $\mathbf{x_k}+\gamma(\boldsymbol{s}-\mathbf{x_k})\in\mathcal{D}$ holds for all $\gamma\in[0,1]$ iff $\boldsymbol{s}\in\mathcal{D}$. Hence, (22) can be written as following:

$$\min_{\mathbf{s}\in\mathcal{D}} s^T \nabla f(\mathbf{x_k}). \tag{23}$$

Obtaining the minimizer s_k of (23) at iteration $k+1$, the FW update step is:

$$\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \gamma(\mathbf{s}_k - \mathbf{x}_k). \tag{24}$$

E. Obtaining a Feasible Initial Point

Algorithm 1 requires a feasible initial point $(\boldsymbol{u}_0, \boldsymbol{\beta}_0)\in\mathcal{S}_{lat}$. assuming such a point exists, i.e. as t is large enough, a trivial initial point $(\boldsymbol{u}_0,\boldsymbol{\beta}_0) := (\tilde{\boldsymbol{u}},\tilde{\boldsymbol{\beta}})$ is associated with the lightest structure in the search space $\mathcal{S}\subset\mathcal{P}_\zeta(\mathcal{S})$, i.e. setting:

$$\tilde{\alpha}^s_{b,c} = \mathbb{1}\{c=1\}\,;\ \tilde{\beta}^s_b = \mathbb{1}\{b=2\} \tag{25}$$

for all $s\in\{1,..,S\}, b\in\{1,..,d\}, c\in\mathcal{C}$, where $\mathbb{1}\{\cdot\}$ is the indicator function. However, starting from this point condemns the gradients with respect to all other structures to be always zero due to the way paths are sampled from the space using the Gumbel-Softmax trick, section 3.1.3.

Hence for a balanced propagation of gradients, the closest to uniformly distributed structure in $\mathcal{S}_{LAT}$ is encouraged. For this purpose we solve the following quadratic programs (QP) alternately,

$$\begin{aligned}
&\min_{\boldsymbol{u}\in\mathcal{S}^{\tilde{\boldsymbol{u}}}} \sum_{s=1}^{s}\sum_{b=1}^{d-1}\sum_{c=1}^{|\mathcal{C}|-1}(\alpha^s_{b,c+1}-\alpha^s_{b,c})^2 \\
&\min_{\boldsymbol{\beta}\in\mathcal{S}^{\tilde{\boldsymbol{\beta}}}} \sum_{s=1}^{s}\sum_{b=1}^{d-1}(\beta^s_{b+1}-\beta^s_b)^2
\end{aligned} \tag{26}$$

using a QP solver at each step.

Sorting the indices of configurations according to their expected latency (see Table 4), the objectives in (26) promote probabilities of consecutive indices to be close to each other, forming chains of non-zero probability with a balanced distribution up to an infeasible configuration, there a chain of zero probability if formed. Illustrations of the formation of such chains are shown in Figure 7 for several latency constraints. Preferring as many blocks participating as possible over different configurations, the alternating optimization in (26) starts with β. This yields balanced β probabilities as long as the constraint allows it.

The benefits from starting with such initial point are quantified by averaging the relative improvements in top-1 accuracy for several latency constraints $\mathcal{T}=\{35,40,45,50,55\}$ milliseconds as following:

$$\frac{100}{|\mathcal{T}|}\sum_{T\in\mathcal{T}}\frac{Acc^T_{\text{balance init}} - Acc^T_{\text{lighest init}}}{Acc^T_{\text{lighest init}}} \tag{27}$$

where $Acc^T_{\text{balance init}}$ and $Acc^T_{\text{lighest init}}$ are the top-1 accuracy measured for fine-tuned models generated by searching the space initialized with (26) and (25) respectively, under latency constraint T. The calculation in (27) yields 8.3% of relative improvement in favour of (26) on average.

F. Proof of Theorem 3.1

In order to proof 3.1, we start with proving auxiliary lemmas. To this end we define the *relaxed* Multiple Choice Knapsack Problem (MCKP):

Definition F.1. *Given $n\in\mathbb{N}$, and a collection of k distinct covering subsets of $\{1,2,\cdots,n\}$ denoted as $N_i, i\in\{1,2,\cdots,k\}$, such that $\cup_{i=1}^k N_i = \{1,2,\cdots,n\}$ and $\cap_{i=1}^k N_i = \varnothing$ with associated values and costs $p_{ij}, t_{ij}\ \forall i\in\{1,\dots,k\}, j\in N_i$ respectively, the relaxed Multiple Choice Knapsack Problem (MCKP) is formulated as following:*

$$\begin{aligned}
&\max_{vu}\sum_{i=1}^{k}\sum_{j\in N_i} p_{ij}\boldsymbol{u}_{ij} \\
&\textit{subject to}\sum_{i=1}^{k}\sum_{j\in N_i} t_{ij}\boldsymbol{u}_{ij}\le T \\
&\sum_{j\in N_i}\boldsymbol{u}_{ij}=1 \qquad \forall i\in\{1,\dots,k\} \\
&\boldsymbol{u}_{ij}\ge 0 \qquad \forall i\in\{1,\dots,k\}, j\in N_i
\end{aligned} \tag{28}$$

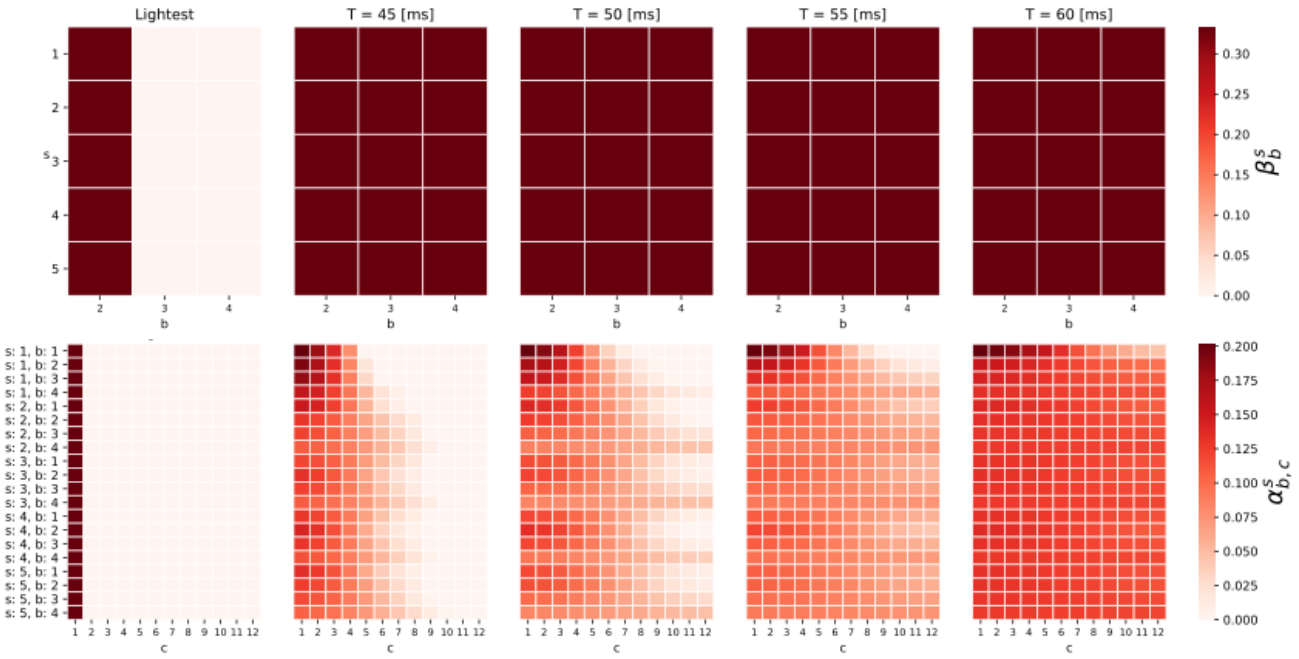

Figure 7. Heat maps representing the initial probability of picking each stage depth (top) and block configuration (bottom), sorted from the lightest to the heaviest (Table 4). Each couple of frames shows the initial point for a different latency constraint. Rows in each frames stand for different stages (top) and blocks (bottom). The lightest feasible initial point (25) involves only a single configuration of the first block in each stage, avoiding gradients from propagating to others (left). The balanced initial points (26) form chains of similar non-zero probability followed by chains of zero probabilities, such that gradients are propagated through feasible paths with a balanced distribution.

where the binary constraints $\boldsymbol{u}_{ij} \in \{0, 1\}$ *of the original MCKP formulation (Kellerer et al., 2004) are replaced with* $\boldsymbol{u}_{ij} \geq 0$.

Definition F.2. *An one-hot vector* $\boldsymbol{u}_i$ *satisfies:*

$$||\boldsymbol{u}_i^*||^0 = \sum_{j \in N_i} |\boldsymbol{u}_{ij}^*|^0 = \sum_{j \in N_i} \mathbb{1}_{\boldsymbol{u}_{ij}^* > 0} = 1$$

where $\mathbb{1}_A$ *is the indicator function that yields* 1 *if* A *holds and* 0 *otherwise.*

Lemma F.1. *The solution* $\boldsymbol{u}^*$ *of the relaxed MCKP (28) is composed of vectors* $\boldsymbol{u}_i^*$ *that are all one-hot but a single one.*

Proof. Suppose that $\boldsymbol{u}^*$ is an optimal solution of (28), and two indices i_1, i_2 exist such that $\boldsymbol{u}_{i_1}^*, \boldsymbol{u}_{i_2}^*$ are not one-hot vectors. As a consequence, we show that four indices, j_1, j_2, j_3, j_4 exist, such that $\boldsymbol{u}_{i_1 j_1}^*, \boldsymbol{u}_{i_1 j_2}^*, \boldsymbol{u}_{i_2 j_3}^*, \boldsymbol{u}_{i_2 j_4}^* \notin \{0, 1\}$.

Define

$$q = \frac{t_{i_2 j_2} - t_{i_1 j_1}}{t_{i_2 j_3} - t_{i_2 j_4}} \tag{29}$$

and

$$f = (t_{i_1 j_1} - t_{i_1 j_2}) \left(\frac{p_{i_1 j_1} - p_{i_1 j_2}}{t_{i_1 j_1} - t_{i_1 j_2}} - \frac{p_{i_2 j_3} - p_{i_2 j_4}}{t_{i_2 j_3} - t_{i_2 j_4}} \right)$$

and assume, without loss of generality, that $f > 0$, otherwise one could swap the indices j_1 and j_2 so that this assumption holds.

Set

$$\Delta = \min \left((1 - \boldsymbol{u}_{i_1 j_1}^*), \frac{1 - \boldsymbol{u}_{i_2 j_3}^*}{|q|}, \boldsymbol{u}_{i_1 j_2}^*, \frac{\boldsymbol{u}_{i_2 j_4}^*}{|q|} \right) \tag{30}$$

such that $\Delta > 0$ and construct another feasible solution of (28) $\boldsymbol{u}_{ij} \leftarrow \boldsymbol{u}_{ij}^*$ for all i, j but for the following indices:

$$\begin{aligned}
\boldsymbol{u}_{i_1 j_1} &\leftarrow \boldsymbol{u}_{i_1 j_1}^* + \Delta \\
\boldsymbol{u}_{i_1 j_2} &\leftarrow \boldsymbol{u}_{i_2 j_2}^* - \Delta \\
\boldsymbol{u}_{i_2 j_3} &\leftarrow \boldsymbol{u}_{i_2 j_3}^* + q\Delta \\
\boldsymbol{u}_{i_2 j_4} &\leftarrow \boldsymbol{u}_{i_2 j_4}^* - q\Delta
\end{aligned}$$

The feasibility of $\boldsymbol{u}$ is easily verified by the definitions in (29) and (30), while the objective varies by:

$$\begin{aligned}
&\sum_{i=1}^{k} \sum_{j \in N_i} p_{ij} (\boldsymbol{u}_{ij} - \boldsymbol{u}_{ij}^*) \\
&\quad = \Delta (p_{i_1 j_1} - p_{i_1 j_2}) + q\Delta (p_{i_2 j_3} - p_{i_2 j_4}) \\
&\quad = \Delta (t_{i_1 j_1} - t_{i_1 j_2}) \left(\frac{p_{i_1 j_1} - p_{i_1 j_2}}{t_{i_1 j_1} - t_{i_1 j_2}} - \frac{p_{i_2 j_3} - p_{i_2 j_4}}{t_{i_2 j_3} - t_{i_2 j_4}} \right) \\
&\quad = \Delta f > 0
\end{aligned} \tag{31}$$

where the last inequality holds due to (30) together with the assumption $f > 0$. Equation (31) holds in contradiction to

$\boldsymbol{u}^*$ being the optimal solution of (28). Hence all the vectors of $\boldsymbol{u}^*$ but one are one-hot vectors. □

Lemma F.2. *The single non one-hot vector of the solution $\boldsymbol{u}^*$ of the relaxed MCKP (28) has at most two nonzero elements.*

Proof. Suppose that $\boldsymbol{u}^*$ is an optimal solution of (28) and an index $\hat{i}$ and three indices j_1, j_2, j_3 exist such that $\boldsymbol{u}^*_{\hat{i}j_1}, \boldsymbol{u}^*_{\hat{i}j_2}, \boldsymbol{u}^*_{\hat{i}j_3} \notin \{0, 1\}$.

Consider the variables $\boldsymbol{\Delta} = (\Delta_1, \Delta_2, \Delta_3)^T \in \mathbb{R}^3$ and the following system of equations:

$$t_{\hat{i}j_1} \cdot \Delta_1 + t_{\hat{i}j_2} \cdot \Delta_2 + t_{\hat{i}j_3} \cdot \Delta_3 = 0 \qquad (32)$$
$$\Delta_1 + \Delta_2 + \Delta_3 = 0$$

At least one non-trivial solution $\boldsymbol{\Delta}^*$ to (32) exists, since the system consists of two equations and three variables.

Assume, without loss of generality, that

$$p_{\hat{i}j_1} \cdot \Delta_1 + p_{\hat{i}j_2} \cdot \Delta_2 + p_{\hat{i}j_3} \cdot \Delta_3 > 0 \qquad (33)$$

Otherwise replace $\boldsymbol{\Delta}^*$ with $-\boldsymbol{\Delta}^*$.

Scale $\boldsymbol{\Delta}^*$ such that

$$0 < \boldsymbol{u}^*_{\hat{i}j_1} + \Delta^*_k < 1 \quad \forall k \in \{1, 2, 3\} \qquad (34)$$

and construct another feasible solution of (28) $\boldsymbol{u}_{ij} \leftarrow \boldsymbol{u}^*_{ij}$ for all i, j but for the following indices:

$$\boldsymbol{u}_{\hat{i}j_1} \leftarrow \boldsymbol{u}^*_{\hat{i}j_1} + \Delta_1$$
$$\boldsymbol{u}_{\hat{i}j_2} \leftarrow \boldsymbol{u}^*_{\hat{i}j_2} + \Delta_2$$
$$\boldsymbol{u}_{\hat{i}j_3} \leftarrow \boldsymbol{u}^*_{\hat{i}j_3} + \Delta_3$$

Since $\boldsymbol{\Delta}^*$ satisfies (32) and (34), the feasibility of $\boldsymbol{u}$ is easily verified while the objective varies by:

$$\sum_{i=1}^{k} \sum_{j \in N_i} p_{ij}(\boldsymbol{u}_{ij} - \boldsymbol{u}^*_{ij})$$
$$= p_{\hat{i}j_1} \cdot \Delta_1 + p_{\hat{i}j_2} \cdot \Delta_2 + p_{\hat{i}j_3} \cdot \Delta_3 > 0 \qquad (35)$$

where the last inequality is due to (33). Equation (35) holds in contradiction to $\boldsymbol{u}^*$ being the optimal solution of (28). Hence the single non one-hot vector of $\boldsymbol{u}^*$ has at most two nonzero entries. □

In order to prove Theorem 3.1, we use Lemmas F.1 and F.1 for $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ separately, based on the observation that each problem in (13) forms a *relaxed* MCKP (28). Thus replacing $\boldsymbol{u}$ in (28) with $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, p is replaced with $\boldsymbol{\alpha}^*$ and $\boldsymbol{\beta}^*$ and the elements of t are replaced with the elements of $\boldsymbol{\beta}^{*T}\Theta^T$ and $\boldsymbol{\alpha}^{*T}\Theta$ respectively.

Remark One can further avoid the two nonzero elements by applying an iterative greedy solver as introduced in (Kellerer et al., 2004), instead of solving a linear program, with the risk of obtaining a sub-optimal solution.

G. 2 for 1: w^* Bootstrap - Accuracy vs Cost

In this section we compare the accuracy and total cost for generating trained models in three ways:

1. Training from scratch
2. Fine-tuning w^* for 10 epochs with knowledge distillation from the heaviest model loaded with $\bar{w}^*$.
3. Fine-tuning w^* for 50 epochs with knowledge distillation from the heaviest model loaded with $\bar{w}^*$.

The last two procedures are specified in Section 4.2.2.

The results, presented in Figure 8, show that with a very short fine-tuning procedure of less than 7 GPU hours (10 epochs) as specified in Section 4.1.1, in most cases, the resulted accuracy surpasses the accuracy obtained by training from scratch. Networks of higher latency benefit less from the knowledge distillation, hence a longer training is required. A training of 35 GPU hours (50 epochs) results in a significant improvement of the accuracy for most of the models.

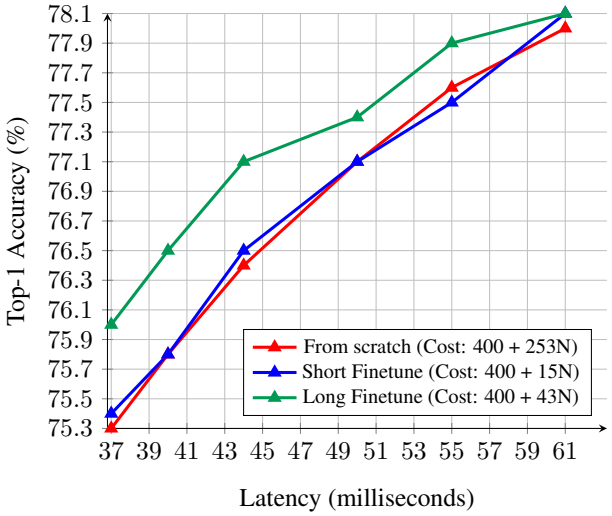

Figure 8. Top-1 accuracy on Imagenet vs Latency measured on Intel Xeon CPU for a batch size of 1, for architectures found with our method trained from scratch and fine-tuned from the pretrained super-network

H. Solving the Mathematical Programs Requires a Negligible Computation Time

In this section we measure the computation time for solving the mathematical programs associated with the initialization point, the LP associated with the FW step and the LP associated with our projection. We show that the measured times are negligible compared to the computation time attributed to backpropagation.

The average time, measured during the search, for solving the linear programs specified in Algorithm 1 and in

Section 3.4.2 and the quadratic program specified in Appendix E is 1.15×10^{-5} CPU hours.

The average time, measured during the search, for a single backpropagation of gradients through the one-shot model is 2.15×10^{-3} GPU Hours.

The overall cost of solving the mathematical programs for generating N networks is about $0.02N$ CPU hours, which is negligible compared to the overall $400 + 15N$ GPU hours.

I. Comparing FLOPS

Figure 9 shows a comparison of Imagenet top-1 accuracy and FLOPS between generated models by our method and other leading NAS methods, some of which optimized for FLOPS. Our models outperform the rest in terms of the tradeoff between accuracy and FLOPS.

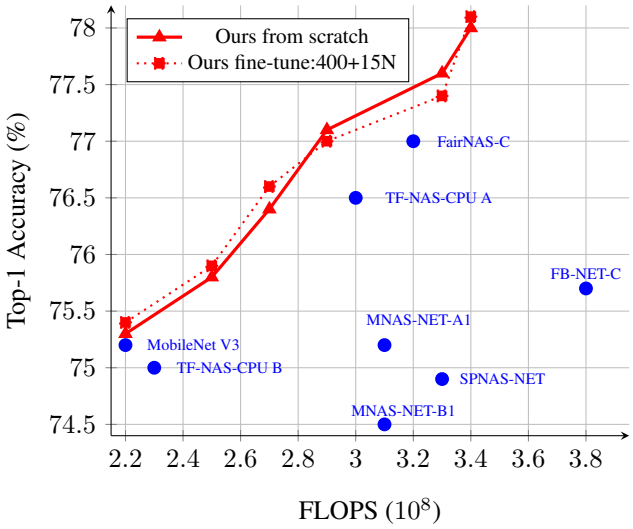

Figure 9. Top-1 accuracy on Imagenet vs FLOPS for architectures found with our method trained from scratch and fine-tuned from the pretrained super-network

J. Comparing other Resources

Talbe 5 presents comparisons in terms of model size, e.g. number of parameters, FLOPS together with latency.

All of the models in Table 5 (but OFA) are trained from scratch using the exact same techniques and code, as specified in section 4.1.1. We report the maximum accuracy and minimal size and FLOPS between the original paper and our training and measurements. We emphasize that all latency values presented are measured without any formula but through actual time measurements of the models, following the exact same settings, on the same hardware.

Model	Size (10^6)	FLOPS (10^8)	Latency (ms)	Top-1 (%)
MnasNetB1	4.4	3.1	39	74.5
TFNAS-B	4.9	2.3	40	75.0
SPNASNet	4.4	3.3	41	74.9
OFA CPU[6]	4.9	3.6	42	75.7
Ours 40 ms	5.3	2.5	40	75.8
MobileNetV3	5.4	2.2	45	75.2
FBNet	5.5	3.8	47	75.7
MnasNetA1	3.9	3.1	55	75.2
Ours 45 ms	5.2	2.7	44	76.4
MobileNetV2	6.1	5.8	70	76.5
TFNAS-A	7.1	3.0	60	76.5
Ours 50 ms	7.5	2.9	50	77.1
EfficientNetB0	5.3	3.9	85	77.3
Ours 55 ms	8.1	3.3	55	77.6
FairNAS-C	4.4	3.2	60	77.0
Ours 60 ms	8.2	3.4	61	78.0

Table 5. ImageNet top-1 accuracy, model size (number of parameters), FLOPS and latency comparison with other methods. The latency is reported for Intel Xeon CPU with a batch size of 1.

[6]Finetuning a model obtained by 1200 GPU hours.