HardCoRe-NAS: Hard Constrained diffeRentiable Neural Architecture Search

Niv Nayman [*1] **Yonathan Aflalo** [*1] **Asaf Noy** [1] **Lihi Zelnik-Manor** [1]

Abstract

Realistic use of neural networks often requires adhering to multiple constraints on latency, energy and memory among others. A popular approach to find fitting networks is through constrained Neural Architecture Search (NAS), however, previous methods enforce the constraint only softly. Therefore, the resulting networks do not exactly adhere to the resource constraint and their accuracy is harmed. In this work we resolve this by introducing *Hard Constrained diffeRentiable NAS (HardCoRe-NAS)*, that is based on an accurate formulation of the expected resource requirement and a scalable search method that satisfies the hard constraint throughout the search. Our experiments show that HardCoRe-NAS generates state-of-the-art architectures, surpassing other NAS methods, while strictly satisfying the hard resource constraints without any tuning required.

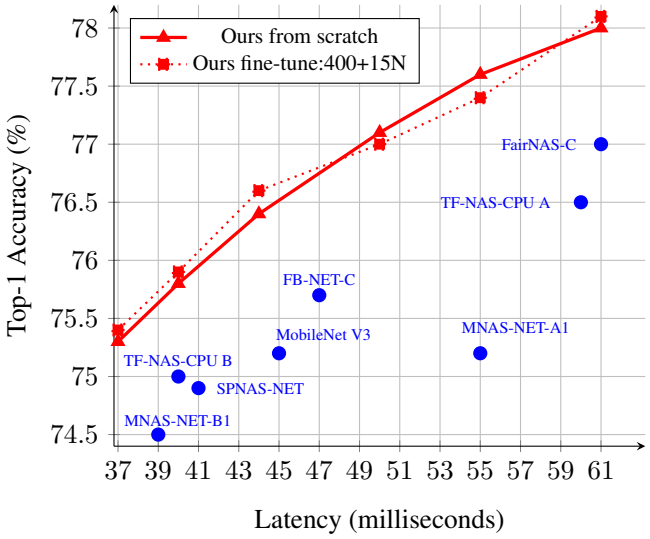

Figure 1. Top-1 accuracy on Imagenet vs Latency measured on Intel Xeon CPU for a batch size of 1. HardCoreNAS can generate a network for any given latency, with accuracy according to the red curve, which is higher than all previous methods.

1. Introduction

With the rise in popularity of Convolutional Neural Networks (CNN), the need for neural networks with fast inference speed and high accuracy, has been growing continuously. At first, manually designed architectures, such as VGG (Simonyan & Zisserman, 2015) or ResNet (He et al., 2015), targeted powerful GPUs as those were the common computing platform for deep CNNs. Many variants of those architectures were the golden standard until the need for deployment on edge devices and standard CPUs emerged. These are more limited computing platforms, requiring lighter architectures that for practical scenarios have to comply with hard constraints on the real time latency or power consumption. This has spawned a line of research aimed at finding architectures with both high performance and bounded resource demand.

The main approaches to solve this evolved from Neural Architecture Search (NAS) (Zoph & Le, 2016; Liu et al., 2018; Cai et al., 2018), while adding a constraint on the target latency over various platforms, e.g., TPU, CPU, Edge-GPU, FPGA, etc. The constrained-NAS methods can be grouped into two categories: (i) Reward based methods such as Reinforcement-Learning (RL) or Evolutionary Algorithm (EA) (Cai et al., 2019; Tan et al., 2019; Tan & Le, 2019; Howard et al., 2019), where the search is performed by sampling networks and predicting their final accuracy and resource demands by evaluation over some validation set. The predictors are expensive to acquire and oftentimes inaccurate. (ii) Resource-aware gradient based methods (Hu et al., 2020; Wu et al., 2019) formulate a differentiable loss function consisting of a trade-off between an accuracy term and a proxy soft penalty term. Therefore, the architecture can be directly optimized using stochastic gradient descent (SGD) (Bottou, 1998), however, it is hard to tune the trade-off between accuracy and resources, which deteriorates the network accuracy and fails to fully meet the resource re-

[*]Equal contribution, Author ordering determined by unbiased coin flip. [1]Alibaba Group, Tel Aviv, Israel. Correspondence to: Yonathan Aflalo <johnaflalo@gmail.com>, Niv Nayman <niv.nayman@alibaba-inc.com>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

quirements. The hard constraints over the resources are further violated due to a final discretization step projecting the architecture over the differentiable search space into the discrete space of architectures.

In this paper, we propose a search algorithm that produces architectures with high accuracy (Figure 1) that strictly satisfy any given hard latency constraint (Figure 3). The search algorithm is fast and scalable to a large number of platforms. The proposed algorithm is based on several key ideas, starting from formulating the NAS problem more accurately, accounting for hard constraints over resources, and solving every aspect of it rigorously. For clarity we focus in this paper on latency constraints, however, our approach can be generalized to other types of resources.

At the heart of our approach lies a suggested differentiable search space that induces a one-shot model (Bender et al., 2018; Chu et al., 2019; Guo et al., 2020; Cai et al., 2019) that is easy to train via a simple, yet effective, technique for sampling multiple sub-networks from the one-shot model, such that each one is properly pretrained. We further suggest an accurate formula for the expected latency of every architecture residing in that space. Then, we search the space for sub-networks by solving a hard constrained optimization problem while keeping the one-shot model pretrained weights frozen. We show that the constrained optimization can be solved via the block coordinate stochastic Frank-Wolfe (BC-SFW) algorithm (Hazan & Luo, 2016a; Lacoste-Julien et al., 2013a). Our algorithm converges faster than SGD, while tightly satisfying the hard latency constraint continuously throughout the search, including during the final discretization step.

The approach we propose has several advantages. First, the outcome networks provide high accuracy and closely comply to the latency constraint. In addition, our solution is scalable to multiple target devices and latency demands. This scalability is due to the efficient pretraining of the one-shot model as well as the fast search method that involves a relatively small number of parameters, governing only the structure of the architecture. We hope that our formulation of NAS as a constrained optimization problem, equipped with an efficient algorithm that solves it, could give rise to followup work incorporating a variety of resource and structural constraints over the search space.

2. Related Work

Efficient Neural Networks are designed to meet the rising demand of deep learning models for numerous tasks per hardware constraints. Manually-crafted architectures such as MobileNets (Howard et al., 2017; Sandler et al., 2018b) and ShuffleNet (Zhang et al., 2018) were designed for mobile devices, while TResNet (Ridnik et al., 2020) and ResNesT (Zhang et al., 2020a) are tailor-made for GPUs. Techniques for improving efficiency include pruning of redundant channels (Dong & Yang, 2019; Aflalo et al., 2020) and layers (Han et al., 2015b), model compression (Han et al., 2015a; He et al., 2018) and weight quantization methods (Hubara et al., 2016; Umuroglu et al., 2017). Dynamic neural networks adjust models based on their inputs to accelerate the inference, via gating modules (Wang et al., 2018), graph branching (Huang et al., 2017) or dynamic channel selection (Lin et al., 2017). These techniques are applied on predefined architectures, hence cannot utilize or satisfy specific hardware constraints.

Neural Architecture Search methods automate models' design per provided constraints. Early methods like NASNet (Zoph & Le, 2016) and AmoebaNet (Real et al., 2019) focused solely on accuracy, producing SotA classification models (Huang et al., 2019) at the cost of GPU-years per search, with relatively large inference times. DARTS (Liu et al., 2018) introduced a differential space for efficient search and reduced the training duration to days, followed by XNAS (Nayman et al., 2019) and ASAP (Noy et al., 2020) that applied pruning-during-search techniques to further reduce it to hours. Recent methods (Lee et al., 2021) also attempt at rapidly generating architectures for unseen datasets without considering resource constraints. Hardware-aware methods such as ProxylessNAS (Cai et al., 2018), Mnasnet (Tan et al., 2019), FBNet (Wu et al., 2019), SPNASNet (Stamoulis et al., 2019) and TFNAS (Hu et al., 2020) produce architectures that satisfy the required constraints by applying simple heuristics such as soft penalties on the loss function. OFA (Cai et al., 2019) proposed a scalable approach across multiple devices by training an one-shot model (Brock et al., 2017; Bender et al., 2018) for 1200 GPU hours. This provides a strong pretrained super-network being highly predictive for the accuracy ranking of extracted sub-networks (Guo et al., 2020; Chu et al., 2019; Yu et al., 2020). OFA searches under latency constraints by conducting evolutionary search (Real et al., 2019) over an accuracy predictor learnt by sampling 16K sub-networks and measuring their accuracy on 10K validation images, hence, training the predictor costs 40 GPU hours. This work relies on such one-shot model acquired within only 400 GPU hours in a much simpler manner, without training an accuracy predictor and satisfies hard constraints tightly with less heuristics.

Frank-Wolfe (FW) algorithm (Frank et al., 1956) is commonly used by machine learning applications (Sun et al., 2019) thanks to its projection-free property (Combettes et al., 2020; Hazan & Minasyan, 2020) and ability to handle structured constraints (Jaggi, 2013). Modern adaptations aimed at deep neural networks (DNNs) optimization include more efficient variants (Zhang et al., 2020b; Combettes et al., 2020), task-specific variants (Chen et al., 2020; Tsiligkaridis & Roberts, 2020), as well as improved

convergence guarantees (Lacoste-Julien & Jaggi, 2015; d'Aspremont & Pilanci, 2020). Two prominent variants are the stochastic FW (Hazan & Luo, 2016b) and Block-coordinate FW (Lacoste-Julien et al., 2013b). While FW excels as an optimizer for DNNs (Berrada et al., 2018; Pokutta et al., 2020), this work is the first to utilize it for NAS.

3. Method

In the scope of this paper, we focus on latency-constrained NAS, searching for an architecture with the highest validation accuracy under a predefined latency constraint, denoted by T. Our architecture search space $\mathcal{S}$ is parametrized by $\zeta \in \mathcal{S}$, governing the architecture structure in a fully differentiable manner, and w, the convolution weights. A latency-constrained NAS can be expressed as the following constrained bilevel optimization problem:

$$\begin{aligned} &\min_{\zeta \in \mathcal{S}} \mathbb{E}_{\substack{x,y \sim \mathcal{D}_{val} \\ \hat{\zeta} \sim \mathcal{P}_\zeta(\mathcal{S})}} [\mathcal{L}_{CE}(x, y \mid w^*, \hat{\zeta})] \\ &\text{s.t. } \mathrm{LAT}(\zeta) \leq T \\ &w^* = \operatorname*{argmin}_w \mathbb{E}_{\substack{x,y \sim \mathcal{D}_{train} \\ \hat{\zeta} \sim \mathcal{P}_\zeta(\mathcal{S})}} [\mathcal{L}_{CE}(x, y \mid w, \hat{\zeta})] \end{aligned} \tag{1}$$

where $\mathcal{D}_{train}$ and $\mathcal{D}_{val}$ are the train and validation sets' distributions respectively, $\mathcal{P}_\zeta(\mathcal{S})$ is some probability measure over the search space parameterized by ζ, $\mathcal{L}_{CE}$ is the cross entropy loss as a differentiable proxy for the negative accuracy and $LAT(\zeta)$ is the estimated latency of the model.

To solve problem (1), we construct a fully differentiable search space parameterized by $\zeta = (\boldsymbol{\alpha}, \boldsymbol{\beta})$ (Section 3.1), that enables the formulation of a differentiable closed form formula expressing the estimated latency $LAT(\boldsymbol{\alpha}, \boldsymbol{\beta})$ (Section 3.2) and efficient acquisition of w^* (Section 3.3). Finally, we introduce rigorous constrained optimization techniques for solving the outer level problem (Section 3.4).

3.1. Search Space

Aiming at the most accurate latency, a flexible search space is composed of a micro search space that controls the internal structures of each block $b \in \{1, .., d = 4\}$, together with a macro search space that specifies the way those are connected to one another in every stage $s \in \{1, .., S = 5\}$.

3.1.1. Micro-Search

Every block is an *elastic* version of the MBInvRes block, introduced in (Sandler et al., 2018a), with expansion ratio $er \in \mathcal{A}_{er} = \{3, 4, 6\}$ of the point-wise convolution, kernel size $k \in \mathcal{A}_k = \{3 \times 3, 5 \times 5\}$ of the depth-wise separable convolution (DWS), and Squeeze-and-Excitation (SE) layer (Hu et al., 2018) $se \in \mathcal{A}_{se} = \{\text{on}, \text{off}\}$. The blocks are configurable, as illustrated at the bottom of Figure 2, using a parametrization $(\alpha^s_{b,er}, \alpha^s_{b,k}, \alpha^s_{b,se})$, defined for every block b of stage s:

$$\alpha^s_{b,er \backslash k \backslash se} \in \{0,1\} \quad \forall er \in \mathcal{A}_{er},\ k \in \mathcal{A}_k,\ se \in \mathcal{A}_{se}$$

$$\Sigma_{er \in \mathcal{A}_{er}} \alpha^s_{b,er} = 1\,;\ \Sigma_{k \in \mathcal{A}_k} \alpha^s_{b,k} = 1\,;\ \Sigma_{se \in \mathcal{A}_{se}} \alpha^s_{b,k} = 1$$

Each triplet (er, k, se) induces a block configuration c that resides within a micro-search space $\mathcal{C} = \mathcal{A}_{er} \otimes \mathcal{A}_k \otimes \mathcal{A}_{se}$, parameterized by $\boldsymbol{\alpha} \in \mathcal{A} = \bigotimes_{s=1}^{S} \bigotimes_{b=1}^{d} \bigotimes_{c \in \mathcal{C}} \alpha^s_{b,c}$, where $\otimes$ denotes the Cartesian product. Hence, for each block b of stage s we have:

$$\alpha^s_{b,c} \in \{0,1\}^{|\mathcal{C}|} \quad ; \quad \Sigma_{c \in \mathcal{C}} \alpha^s_{b,c} = 1$$

An input feature map x^s_b to block b of stage s is processed as follows:

$$x^s_{b+1} = \sum_{c \in \mathcal{C}} \alpha^s_{b,c} \cdot O^s_{b,c}(x^s_b)$$

where $O^s_{b,c}(\cdot)$ is the operation performed by the elastic MBInvRes block configured according to $c = (er, k, se)$.

Having several configurations $O^s_{b,c}(\cdot)$ share the same values of $\alpha^s_{b,er}$ or $\alpha^s_{b,k}$ or $\alpha^s_{b,se}$ induces weight sharing between the common operations of the associated architectures. This weight sharing is beneficial for solving the inner problem (1) effectively, as will be discussed in Section 3.3.

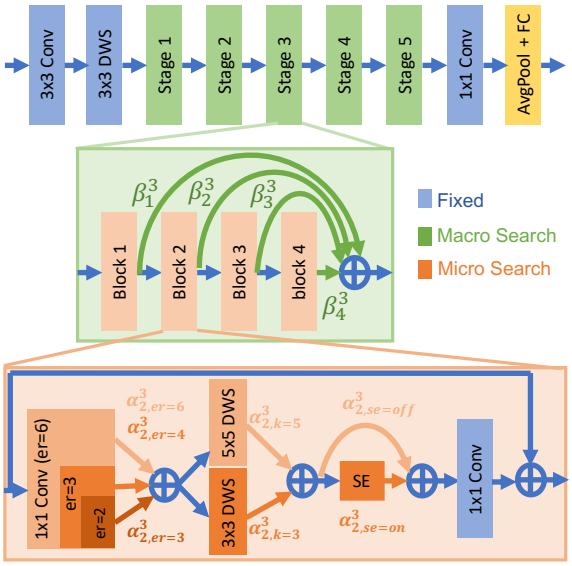

Figure 2. A search space view via the one-shot model

3.1.2. Macro-Search

Inspired by (Hu et al., 2020; Cai et al., 2019), the output of each block of every stage is also directed to the end of the stage as illustrated in the middle of Figure 2. Thus, the depth of each stage s is controlled by the parameters $\boldsymbol{\beta} \in \mathcal{B} = \bigotimes_{s=1}^{S} \bigotimes_{b=1}^{d} \beta^s_b$, such that:

$$\beta^s_b \in \{0,1\}^d \quad ; \quad \Sigma^d_{b=1} \beta^s_b = 1$$

The depth is $b^s \in \{b \mid \beta_b^s = 1, b \in \{1, .., d\}\}$, since:

$$x_1^{s+1} = \Sigma_{b=1}^{d} \beta_b^s \cdot x_{b+1}^s$$

3.1.3. The Composed Search Space

The overall search space is composed of both the micro and macro search spaces parameterized by $\boldsymbol{\alpha} \in \mathcal{A}$ and $\boldsymbol{\beta} \in \mathcal{B}$, respectively, such that:

$$\mathcal{S} = \left\{ (\boldsymbol{\alpha}, \boldsymbol{\beta}) \;\middle|\; \begin{array}{c} \boldsymbol{\alpha} \in \mathcal{A}, \boldsymbol{\beta} \in \mathcal{B} \\ \alpha_{b,c}^s \in \{0,1\}^{|\mathcal{C}|}\,;\, \Sigma_{c \in \mathcal{C}} \alpha_{b,c}^s = 1 \\ \beta_b^s \in \{0,1\}^d\,;\, \Sigma_{b=1}^d \beta_b^s = 1 \\ \forall s \in \{1, .., S\}, b \in \{1, .., d\}, c \in \mathcal{C} \end{array} \right\}$$

A continuous probability distribution is induced over the space, by relaxing $\alpha_{b,c}^s \in \{0,1\}^{|\mathcal{C}|} \rightarrow \alpha_{b,c}^s \in \mathbb{R}_+^{|\mathcal{C}|}$ and $\beta_b^s \in \{0,1\}^d \rightarrow \beta_b^s \in \mathbb{R}_+^d$ to be continuous rather than discrete. A sample sub-network is drawn $\hat{\zeta} = (\hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\beta}}) \sim \mathcal{P}_\zeta(\mathcal{S})$ using the Gumbel-Softmax Trick (Jang et al., 2016) such that $\hat{\zeta} \in \mathcal{S}$, as specified in (4) and (5). In summary, one can view the parametrization $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ as a composition of probabilities in $\mathcal{P}_\zeta(\mathcal{S})$ or as degenerated one-hot vectors in $\mathcal{S}$.

Effectively we include at least a couple of blocks in each stage by setting $\beta_1^s \equiv 0$, hence, the overall size of the search space is:

$$|\mathcal{S}| = (\sum_{b=2}^{d} |\mathcal{C}|^b)^S = (\sum_{b=2}^{d} |\mathcal{A}_{er}|^b \cdot |\mathcal{A}_k|^b \cdot |\mathcal{A}_{se}|^b)^S$$
$$= ((3 \times 2 \times 2)^2 + (3 \times 2 \times 2)^3 + (3 \times 2 \times 2)^4)^5 \approx 10^{27}$$

3.2. Formulating the Latency Constraint

Aiming at tightly satisfying latency constraints, we propose an accurate formula for the expected latency of a sub-network. The expected latency of a block b can be computed by summing over the latency $t_{b,c}^s$ of every possible configuration $c \in \mathcal{A}$:

$$\bar{\ell}_b^s = \Sigma_{c \in \mathcal{C}} \alpha_{b,c}^s \cdot t_{b,c}^s$$

Thus the expected latency of a stage of depth b' is

$$\ell_{b'}^s = \sum_{b=1}^{b'} \bar{\ell}_b^s \tag{2}$$

Taking the expectation over all possible depths yields

$$\ell^s = \sum_{b'=1}^{d} \beta_{b'}^s \cdot \ell_{b'}^s$$

and summing over all the stages results in the following formula for the overall latency:

$$LAT(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \sum_{s=1}^{S} \sum_{b'=1}^{d} \sum_{b=1}^{b'} \sum_{c \in \mathcal{C}} \alpha_{b,c}^s \cdot t_{b,c}^s \cdot \beta_{b'}^s \tag{3}$$

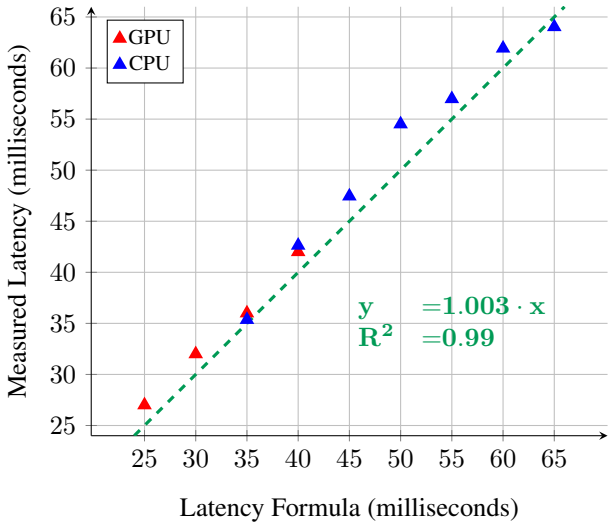

Figure 3. Experimental results showing that the latency calculated with formula (3) is very close to the latency measured in practice, on both CPU and GPU.

The summation originated in (2) differentiates our latency formulation (3) from that of (Hu et al., 2020).

Figure 3 provides empirical validation of (3), showing that in practice the actual and estimated latency are very close on both GPU and CPU. More details on the experiments are provided in Section 4.2.1.

Remark: By replacing the latency measurements $t_{b,c}^s$ with other resources, e.g., memory, FLOPS, MACS, etc., one can use formula (3) to add multiple hard constraints to the outer problem of (1).

3.3. Solution to the Inner Problem w^*

Previous work proposed approximated solutions to the following unconstrained problem:

$$\min_{\zeta \in \mathcal{S}} \mathbb{E}\left[\mathcal{L}_{CE}\left(x, y \mid w_\zeta^*, \zeta\right)\right]$$
$$\text{s.t. } w_\zeta^* = \underset{w}{\operatorname{argmin}}\ \mathbb{E}_{\zeta \sim \mathcal{P}_\zeta(\mathcal{S})}\left[\mathcal{L}_{CE}\left(x, y \mid w, \zeta\right)\right]$$

typically by alternating or simultaneous updates of w and ζ (Liu et al., 2018; Xie et al., 2018; Cai et al., 2018; Wu et al., 2019; Hu et al., 2020). This approach has several limitations. First, obtaining a designated w_ζ^* with respect to every update of ζ involves a heavy training of a neural network until convergence. Instead a very rough approximation is obtained by just a few update steps for w. In turn, this approximation creates a bias towards strengthening networks with few parameters since those learn faster, hence, get sampled even more frequently, further increasing the chance to learn in a positive feedback manner. Eventually, often overly simple architectures are generated, e.g., consisting of many skip-connections (Chen et al., 2019; Liang et al., 2019). Several remedies have been proposed, e.g.,

temperature annealing, adding uniform sampling, modified gradients and updating only w for a while before the joint optimization begins (Noy et al., 2020; Wu et al., 2019; Hu et al., 2020; Nayman et al., 2019). While those mitigate the bias problem, they do not solve it.

We avoid this approximation. Instead we obtain w^* of the inner problem of (1) only once, with respect to a uniformly distributed architecture, sampling $\bar{\zeta}$ from $\mathcal{P}_{\bar{\zeta}}(\mathcal{S}) = U(\mathcal{S})$.

This is done by sampling multiple distinctive paths (sub-networks of the one-shot model) for every image in the batch in an efficient way (just a few lines of code[1]), using the Gumbel-Softmax Trick, as opposed to the common sampling of a single path per batch. For every feature map x that goes through block b of stage s, distinctive uniform random variables $\mathcal{U}^s_{b,c,x}, \mathcal{U}^s_{b,x} \sim U(0,1)$ are sampled, governing the path undertaken by this feature map:

$$\hat{\alpha}^s_{b,c,x} = \frac{e^{\log(\alpha^s_{b,c}) - \log(\log(\mathcal{U}^s_{b,c,x}))}}{\sum_{c \in \mathcal{C}} e^{\log(\alpha^s_{b,c}) - \log(\log(\mathcal{U}^s_{b,c,x}))}} \tag{4}$$

$$\hat{\beta}^s_{b,x} = \frac{e^{\log(\beta^s_{b}) - \log(\log(\mathcal{U}^s_{b,x}))}}{\sum_{c \in \mathcal{C}} e^{\log(\alpha^s_{b,c}) - \log(\log(\mathcal{U}^s_{b,x}))}} \tag{5}$$

Based on the observation that the accuracy of a sub-network with w^* should be predictive for its accuracy when optimized as a stand-alone model from scratch, we aim at an accurate prediction. Our simple approach implies that, with high probability, the number of paths sampled at each update step is as high as the number of images in the batch. This is two orders of magnitude larger than previous methods that sample a single path per update step (Guo et al., 2020; Cai et al., 2019), while avoiding the need to keep track of all the sampled paths (Chu et al., 2019). Using multiple paths reduces the variance of the gradients with respect to the paths sampled by an order of magnitude[2]. Furthermore, leveraging the weight sharing implied by the structure of the elastic MBInvRes block (Section 3.1.1), the number of gradients observed by each operation is increased by a factor of at least $\frac{|\mathcal{C}|}{\max(|\mathcal{A}_{er}|,|\mathcal{A}_k|,|\mathcal{A}_{se}|)} = \frac{3\times2\times2}{\max(3,2,2)} \approx 4$. This further reduces the variance by half.

Figure 4 shows that we obtain a one-shot model w^* with high correlation between the ranking of sub-networks directly extracted from it and the corresponding stand-alone counterpart trained from scratch. See more details in Section 4.2.2. This implies that w^* captures well the quality of sub-structures in the search space.

[1] https://github.com/Alibaba-MIIL/HardCoReNAS

[2] A typical batch consists of hundreds of i.i.d paths, thus a variance reduction of the square root of that is in place.

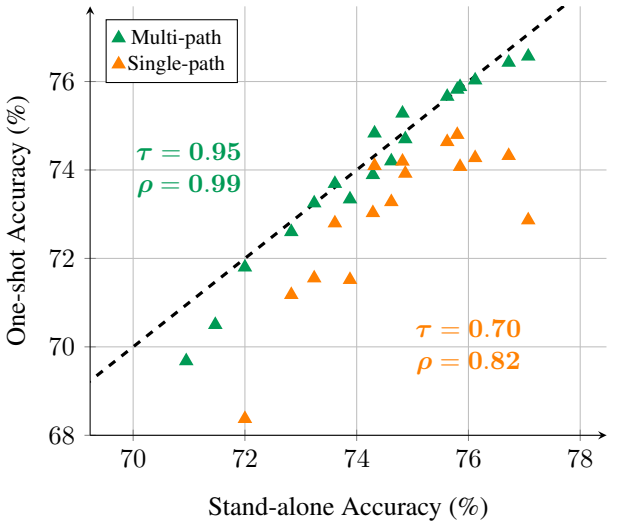

Figure 4. Top-1 accuracy on Imagenet of networks trained from scratch v.s. corresponding sub-networks extracted from our one-shot model. τ and ρ represent the Kendall-Tau and Spearman correlation coefficients, respectively.

3.4. Solving the Outer Problem

Having defined the formula for the latency $\text{LAT}(\zeta)$ and obtained a solution for w^*, we can now continue to solve the outer problem (1).

3.4.1. Searching Under Latency Constraints

Most differentiable resource-aware NAS methods account for the resources through shaping the loss function with soft penalties (Wu et al., 2019; Hu et al., 2020). This approach solely does not meet the constraints tightly. Experiments illustrating this are described in Section 4.2.3.

Our approach directly solves the constrained outer problem (1), hence, it enables the strict satisfaction of resource constraints by further restricting the search space, i.e., $\mathcal{S}_{LAT} = \{\zeta \mid \zeta \in \mathcal{P}_\zeta(\mathcal{S}), \text{LAT}(\zeta) \leq T\}$.

As commonly done for gradient based approaches, e.g., (Liu et al., 2018), we relax the discrete search space $\mathcal{S}$ to be continuous by searching for $\zeta \in \mathcal{S}_{LAT}$. As long as $\mathcal{S}_{LAT}$ is convex, it could be leveraged for applying the stochastic Frank-Wolfe (SFW) algorithm (Hazan & Luo, 2016a) to directly solve the constrained outer problem:

$$\min_{\zeta \in \mathcal{S}_{LAT}} \mathbb{E}_{x,y \sim \mathcal{D}_{val}} \left[\mathcal{L}_{CE}\left(x, y \mid w^*, \zeta\right)\right] \tag{6}$$

following the update step:

$$\zeta_{t+1} = (1 - \gamma_t) \cdot \zeta_t + \gamma_t \cdot \xi_t \tag{7}$$

$$\xi_t = \underset{\zeta \in \mathcal{S}_{LAT}}{\text{argmin}}\ \zeta^T \cdot \nabla_{\zeta_t} \mathcal{L}_{CE}\left(x_t, y_t \mid w^*, \zeta_t\right) \tag{8}$$

where (x_t, y_t) and γ_t are the sampled data and the learning rate at step t, respectively. For $\mathcal{S}_{LAT}$ of linear constraints, the linear program (8) can be solved efficiently, using the Simplex algorithm (Nash, 2000).

A convex $\mathcal{S}_{LAT}$ together with $\gamma_t \in [0,1]$ satisfy $\zeta_t \in \mathcal{S}_{LAT}$ anytime, as long as $\zeta_0 \in \mathcal{S}_{LAT}$. We provide a method for satisfying the latter in the supplementary materials.

The benefits of such optimization are demonstrated in Figure 5 through a toy problem, described in Section 4.2.3. While SGD is sensitive to the trade-off involved with a soft penalty, SFW converges faster to the optimum with zero penalty.

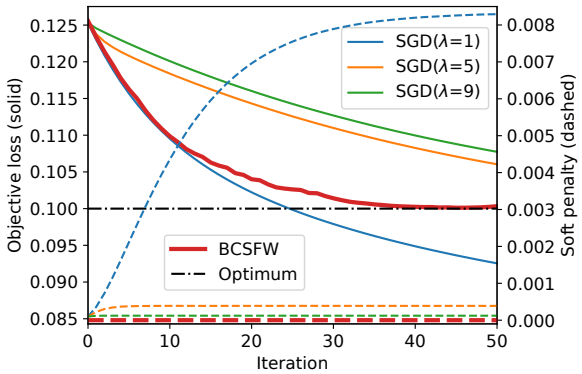

Figure 5. Objective loss and soft penalty for BCSFW and SGD for different values of the penalty coefficient λ. Solid and dashed curves represent the objective loss (left y-axis) and soft penalty (right y-axis), respectively. BCSFW converges to the optimum much faster than SGD.

All this requires $\mathcal{S}_{LAT}$ to be convex. While $\mathcal{P}_\zeta(\mathcal{S})$ is obviously convex, formed by linear constraints, the latency constraint $\text{LAT}(\zeta) \leq T$ is not necessarily so. The latency formula (3) can be expressed as a quadratic constraint by constructing a matrix $\Theta \in \mathbb{R}_+^{S\cdot d\cdot|\mathcal{A}|\times S\cdot d}$ from $t^s_{b,c}$, such that,

$$LAT(\zeta) = LAT(\boldsymbol{\alpha},\boldsymbol{\beta}) = \boldsymbol{\alpha}^T\Theta\boldsymbol{\beta} \quad ; \quad \zeta \in \mathcal{P}_\zeta(\mathcal{S}) \quad (9)$$

Since Θ is constructed from measured latency, it is not guaranteed to be positive semi-definite, hence, the induced quadratic constraint could make $\mathcal{S}_{LAT}$ non-convex.

To overcome this, we introduce the Block Coordinate Stochastic Frank-Wolfe (BCSFW) Algorithm 1, that combines Stochastic Frank-Wolfe with Block Coordinate Frank-Wolfe (Lacoste-Julien et al., 2013a). This is done by forming separated convex feasible sets at each step, induced by linear constraints only:

$$\boldsymbol{\alpha}_t \in \mathcal{S}_t^{\boldsymbol{\alpha}} = \{\boldsymbol{\alpha} \mid \boldsymbol{\alpha} \in \mathcal{A}, \quad \boldsymbol{\beta}_t^T\Theta^T\cdot\boldsymbol{\alpha} \leq T\} \quad (10)$$

$$\boldsymbol{\beta}_t \in \mathcal{S}_t^{\boldsymbol{\beta}} = \{\boldsymbol{\beta} \mid \boldsymbol{\beta} \in \mathcal{B}, \quad \boldsymbol{\alpha}_t^T\Theta\cdot\boldsymbol{\beta} \leq T\} \quad (11)$$

This implies that $\zeta_t = (\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t) \in \mathcal{S}_{LAT}$ for all t. Moving inside the feasible domain at anytime avoids irrelevant infeasible structures from being promoted and hiding feasible structures.

3.4.2. Projection Back to the Discrete Space

As differentiable NAS methods are inherently associated with a continuous search space, a final discretizaiton step $\mathbb{P}$:

Algorithm 1 Block Coordinate SFW (BCSFW)

input $(\boldsymbol{\alpha}_0, \boldsymbol{\beta}_0) \in \mathcal{S}_{LAT}$
for $t = 0, \ldots, K$ **do**
  Pick $\delta := \boldsymbol{\alpha}$ or $\delta := \boldsymbol{\beta}$ at random
  Sample an i.i.d validation batch $(x_t, y_t) \sim \mathcal{D}_{val}$
  $\xi_t = \text{argmin}_{\,\xi\in\mathcal{S}_t^\delta}\, \xi^T\cdot\nabla_{\delta_t}\mathcal{L}_{CE}\left(x_t, y_t \mid w^*, \delta_t\right)$
  Update $\delta_{t+1} = (1-\gamma_t)\cdot\delta_t + \gamma_t\cdot\xi_t$ with $\gamma_t = \frac{4}{t+4}$
end for

$\mathcal{P}_\zeta(\mathcal{S}) \to \mathcal{S}$ is required for extracting a single architecture. Most methods use the *argmax* operator:

$$\bar{\alpha}^s_{b,c} := \mathbb{1}\{c = \underset{c\in\mathcal{C}}{\text{argmax}}\ \alpha^s_{b,c}\} \quad (12)$$

$$\bar{\beta}^s_b := \mathbb{1}\{b = \underset{b=2,..,d}{\text{argmax}}\ \beta^s_b\}$$

for all $s \in \{1,..,S\}, b \in \{1,..,d\}, c \in \mathcal{C}$, where $(\bar{\boldsymbol{\alpha}}, \bar{\boldsymbol{\beta}})$ is the solution to the outer problem of (1).

For resource-aware NAS methods, applying such projection results in possible violation of the resource constraints, due to the shift from the converged solution in the continuous space. Experiments showing that latency constraints are violated due to (12) are provided in Section 4.2.4.

While several methods mitigate this violation by promoting sparse probabilities during the search, e.g., (Noy et al., 2020; Nayman et al., 2019), our approach completely eliminates it by introducing an alternative projection step, described next.

Viewing the solution of the outer problem $(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$ as the credit assigned to each configuration, we introduce a projection step that maximizes the overall credit while strictly satisfying the latency constraints. It is based on solving the following two linear programs:

$$\max_{\boldsymbol{\alpha}\in\mathcal{S}^{\boldsymbol{\alpha}^*}} \boldsymbol{\alpha}^T\cdot\boldsymbol{\alpha}^* \quad ; \quad \max_{\boldsymbol{\beta}\in\mathcal{S}^{\boldsymbol{\beta}^*}} \boldsymbol{\beta}^T\cdot\boldsymbol{\beta}^* \quad (13)$$

Note, that when there is no latency constraint, e.g., $T \to \infty$, (13) coincides with (12).

We next provide a theorem that guarantees that the projection (13) yields a sparse solution, representing a valid sub-network of the one-shot model. Specifically, a single probability vector from those composing $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ contains up to two non-zero entries each, as all the rest are one-hot vectors.

Theorem 3.1. *The solution* $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ *of* (13) *admits:*

$$\sum_{c\in\mathcal{C}} |\alpha^s_{b,c}|^0 = 1 \;\forall (s,b) \in \{1,..,S\}\otimes\{1,..,d\} \setminus \{(s_{\boldsymbol{\alpha}}, b_{\boldsymbol{\alpha}})\}$$

$$\sum_{b=1}^{d} |\beta^s_b|^0 = 1 \quad \forall s \in \{1,..,S\}\setminus\{s_{\boldsymbol{\beta}}\}$$

where $|\cdot|^0 = \mathbb{1}\{\cdot > 0\}$ *and* $(s_{\boldsymbol{\alpha}}, b_{\boldsymbol{\alpha}})$, $s_{\boldsymbol{\beta}}$ *are single block*

and stage respectively, satisfying:

$$\sum_{c\in\mathcal{C}} |\alpha^{s_{\boldsymbol{\alpha}}}_{b_{\boldsymbol{\alpha}},c}|^0 \leq 2 \quad ; \quad \sum_{b=1}^{d} |\beta^{s_{\boldsymbol{\beta}}}_{b}|^0 \leq 2 \tag{14}$$

Refer to the supplementary materials for the proof.

Remark: A negligible deviation is associated with taking the argmax (12) over the only two couples referred to in (14). Experiments supporting this are described in Section 4.2.4.

4. Experimental Results

4.1. Search for State-of-the-Art Architectures

4.1.1. Dataset and Setting

For all of our experiments, we train our networks using SGD with a learning rate of 0.1 with cosine annealing, Nesterov momentum of 0.9, weight decay of 10^{-4}, applying label smoothing (Szegedy et al., 2016) of 0.1, cutout, Autoaugment (Cubuk et al., 2018), mixed precision and EMA-smoothing.

We obtain the solution of the inner problem w^* as specified in sections 3.3 and 4.2.2 over 80% of a random 80-20 split of the ImageNet train set. We utilize the remaining 20% as a validation set and search for architectures with latencies of $40, 45, 50, 55, 60$ and $25, 30, 40$ milliseconds running with a batch size of 1 and 64 on an Intel Xeon CPU and and NVIDIA P100 GPU, respectively. The search is performed according to section 3.4 for only 2 epochs of the validation set, lasting for 8 GPU hours[5].

4.1.2. Comparisons with other methods

We compare our generated architectures to other state-of-the-art NAS methods in Table 1 and Figure 1. For the purpose of comparing the generated architectures alone, excluding the contribution of evolved pretraining techniques, for each model in Table 1, the official PyTorch implementation (Paszke et al., 2019) is trained from a random initialization (but OFA [6]) using the exact same techniques and code, as specified in section 4.1.1. We report the maximum accuracy between the original paper and our training. We emphasize that all latency values presented are measured without any formula but through actual time measurements of the models, running on a single thread with the exact same settings and on the same hardware. We excluded further optimizations, such as Intel MKL-DNN (Intel, R), therefore, the latency we report may differ from the one originally reported. It can be seen that networks generated by our method meet the latency target closely, while at the same time surpassing all the others methods on the top-1 accuracy over ImageNet with a reduced scalable search time.

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

	Model	Latency (ms)	Top-1 (%)	Total Cost (GPU hours)
NVIDIA P100 GPU (batch:64)	MobileNetV3	28	75.2	180N
	TFNAS-D	30	74.2	236N
	Ours 25 ms	**27**	**75.7**	400 + **15N**
	MnasNetA1	37	75.2	40,000N
	MnasNetB1	34	74.5	40,000N
	FBNet	41	75.7	576N
	SPNASNet	36	74.9	288 + 408N
	TFNAS-B	44	76.3	263N
	TFNAS-C	37	75.2	263N
	Ours 30 ms	**32**	**77.3**	400 + **15N**
	TFNAS-A	54	76.9	263N
	EfficientNetB0	48	77.3	
	MobileNetV2	50	76.5	150N
	Ours 40 ms	**41**	**77.9**	400 + **15N**
Intel Xeon CPU (batch:1)	MnasNetB1	39	74.5	40,000N
	TFNAS-B	40	75.0	263N
	SPNASNet	41	74.9	288 + 408N
	OFA CPU[3]	42	75.7	1200 + 25N
	Ours 40 ms	**40**	**75.8**	400 + **15N**
	MobileNetV3	45	75.2	180N
	FBNet	47	75.7	576N
	MnasNetA1	55	75.2	40,000N
	Ours 45 ms	**44**	**76.4**	400 + **15N**
	MobileNetV2	70	76.5	150N
	TFNAS-A	60	76.5	263N
	Ours 50 ms	**50**	**77.1**	400 + **15N**
	EfficientNetB0	85	77.3	
	Ours 55 ms	**55**	**77.6**	400 + **15N**
	FairNAS-C	60	77.0	240N
	Ours 60 ms	**61**	**78.0**	400 + **15N**

Table 1. ImageNet top-1 accuracy, latency and cost comparison with other methods. The total cost stands for the search and training cost of N networks.

4.2. Empirical Analysis of Key Components

4.2.1. Validation of the Latency Formula

One of our goals is to provide a practical method to accurately meet the given resource requirements. Hence, we validate empirically the accuracy of the latency formula (3), by comparing its estimation with the measured latency. Experiments were performed on two platforms: Intel Xeon CPU and NVIDIA P100 GPU, and applied to multiple networks. Results are shown in Figure 3, which confirms a linear relation between estimated and measured latency, with a ratio of 1.003 and a coefficient of determination of $R^2 = 0.99$. This supports the accuracy of the proposed formula.

4.2.2. Evaluating the Solution of the Inner Problem w^*

The ultimate quality measure for a generated architecture is arguably its accuracy over a test set when trained as a stand-alone model from randomly initialized weights. To evaluate the quality of our one-shot model w^* we compare

the accuracy of networks extracted from it with the accuracy of the corresponding architectures when trained from scratch. Naturally, when training from scratch the accuracy could increase. However, a desired behavior is that the ranking of the accuracy of the networks will remain the same with and without training from scratch. The correlation can be calculated via the Kendall-Tau (Maurice, 1938) and Spearman's (Spearman, 1961) rank correlation coefficients, denoted as τ and ρ, respectively.

To this end, we first train for 250 epochs a one-shot model $\bar{w}^*$ using the heaviest possible configuration, i.e., a depth of 4 for all stages, with $er = 6, k = 5 \times 5, se = on$ for all the blocks. Next, to obtain w^*, we apply the multi-path training of Section 3.3 for additional 100 epochs of fine-tuning $\bar{w}^*$ over 80% of a 80-20 random split of the ImageNet train set (Deng et al., 2009). The training settings are specified in Section 4.1.1. The first 250 epochs took 280 GPU hours[4] and the additional 100 fine-tuning epochs took 120 GPU hours[5], summing to a total of 400 hours on NVIDIA V100 GPU to obtain w^*. To further demonstrate the effectiveness of our multi-path technique, we repeat this procedure also without it, sampling a single path for each batch.

For the evaluation of the ranking correlations, we extract 18 sub-networks of common configurations for all stages of depths in $\{2, 3, 4\}$ and blocks with an expansion ratio in $\mathcal{A}_{er} = \{3, 4, 6\}$, a kernel size in $\mathcal{A}_k = \{3 \times 3, 5 \times 5\}$ and Squeeze and Excitation being applied. We train each of those as stand-alone from random initialized weights for 200 epochs over the full ImageNet train set, and extract their final top-1 accuracy over the validation set of ImageNet.

Figure 4 shows for each extracted sub-network its accuracy without and with stand-alone training. It further shows results for both multi-path and single-path sampling. It can be seen that the multi-path technique improves τ and ρ by 0.35 and 0.17 respectively, leading to a highly correlated rankings of $\tau = 0.95$ and $\rho = 0.99$.

2 for 1 - w^* Bootstrap:
A nice benefit of the training scheme described in this section is that it further shortens the generation of trained models. We explain this next.

The common approach of most NAS methods is to re-train the extracted sub-networks from scratch. Instead, we leverage having two sets of weights: $\bar{w}^*$ and w^*. Instead of retraining the generated sub-networks from a random initialization we opt for fine-tuning w^* guided by knowledge distillation (Hinton et al., 2015) from the heaviest model $\bar{w}^*$. Empirically, we observe that this surpasses the accuracy obtained when training from scratch at a fraction of the time. More specifically, we are able to generate a trained model within a small marginal cost of 15 GPU hours. The total cost for generating N trained models is $400 + 15N$, much lower than the $1200 + 25N$ reported by OFA (Cai et al., 2019). See Table 1. This makes our method scalable for many devices and latency requirements. Note, that allowing for longer training further improves the accuracy significantly .

[4]Running with a batch size of 200 on 8×NVIDIA V100
[5]Running with a batch size of 16 on 8×NVIDIA V100

4.2.3. Outer Problem: Hard vs Soft

Next, we evaluate our method's ability to satisfy accurately a given latency constraint. We compare our hard-constrained formulation (1) with the common approach of adding soft penalties to the loss function (Hu et al., 2020; Wu et al., 2019). The experiments were performed over a simple and intuitive stochastic toy problem:

$$\min_x \Sigma_{i=1}^d (1+\epsilon_i) \cdot x_i^2 \quad \text{s.t.} \quad \Sigma_{i=1}^d x_i = 1 \quad ; \quad x, \epsilon \in \mathbb{R}^d \tag{15}$$

where $\epsilon \sim \mathcal{N}(0, 0.01I)$. This problem can be solved using the Frank-Wolfe (FW) (Frank et al., 1956) update rule:

$$x_{t+1} = (1-\gamma_t) \cdot x_t + \gamma_t \cdot \xi_t \tag{16}$$
$$\xi_t = \underset{\xi}{\operatorname{argmin}} \ \xi^T \cdot x \cdot diag(1+\epsilon) \text{ s.t. } \Sigma_{i=1}^d \xi_i = 1$$

starting from an arbitrary random feasible point, e.g. sample a random vector and normalize it. The soft-constraint approach minimizes $\Sigma_{i=1}^d (1+\epsilon_i) \cdot x_i^2 + \lambda \left(\sum_i x_i - 1\right)^2$ using stochastic gradient descent (SGD), where λ is a coefficient that controls the trade-off between the objective function and the soft penalty representing the constraint. Instead of using FW, we implement a version of Block Coordinate Stochastic Frank-Wolfe (BCSFW) by randomly sampling one of two predefined blocks of coordinates at every iteration and applying (16) for the sampled block only.

Figure 5, shows the objective value for $d = 10$ and the corresponding soft penalty value along the optimization for both BCSFW and SGD with several values of λ. It can be seen that SGD is very sensitive to the trade-off tuned by λ, often violating the constraint or converging to a sub-optimal objective value. On the contrary, BCSFW converges faster to the optimal solution while strictly satisfying the constraint throughout the optimization.

4.2.4. Evaluating the Discretizing Projection

Table 2 evaluates the projection of architectures to the discrete space, as proposed in Section 3.4.2. While the commonly used argmax projection violates the constraints by up to 10%, those are strictly satisfied by our proposed projection.

Constraint	35	40	45	50	55	60
argmax	36	42	50	54	58	66
Our Projection	35	40	45	49	54	60

Table 2. The effect of discretization methods

5. Conclusion

The problem of resource-aware differentiable NAS is formulated as a bilevel optimization problem with hard constraints. Each level of the problem is addressed rigorously for efficiently generating well performing architectures that strictly satisfy the hard resource constraints. HardCoRe-NAS turns to be a fast search method, scalable to many devices and requirements, while the resulted architectures perform better than architectures generated by other state-of-the-art NAS methods. We hope that the proposed methodologies will give rise to more research and applications utilizing constrained search for inducing unique structures over a variety of search spaces and resource specifications.

References

Aflalo, Y., Noy, A., Lin, M., Friedman, I., and Zelnik, L. Knapsack pruning with inner distillation. *arXiv preprint arXiv:2002.08258*, 2020.

Bender, G., Kindermans, P.-J., Zoph, B., Vasudevan, V., and Le, Q. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning*, pp. 550–559. PMLR, 2018.

Berrada, L., Zisserman, A., and Kumar, M. P. Deep frank-wolfe for neural network optimization. *arXiv preprint arXiv:1811.07591*, 2018.

Bottou, L. Online algorithms and stochastic approxima-p tions. *Online learning and neural networks*, 1998.

Brock, A., Lim, T., Ritchie, J. M., and Weston, N. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.

Cai, H., Zhu, L., and Han, S. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

Cai, H., Gan, C., Wang, T., Zhang, Z., and Han, S. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019.

Chen, J., Zhou, D., Yi, J., and Gu, Q. A frank-wolfe framework for efficient and effective adversarial attacks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 3486–3494, 2020.

Chen, X., Xie, L., Wu, J., and Tian, Q. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1294–1303, 2019.

Chu, X., Zhang, B., Xu, R., and Li, J. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. *arXiv preprint arXiv:1907.01845*, 2019.

Combettes, C. W., Spiegel, C., and Pokutta, S. Projection-free adaptive gradients for large-scale optimization. *arXiv preprint arXiv:2009.14114*, 2020.

Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., and Le, Q. V. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.

d'Aspremont, A. and Pilanci, M. Global convergence of frank wolfe on one hidden layer networks. *arXiv preprint arXiv:2002.02208*, 2020.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

Dong, X. and Yang, Y. Network pruning via transformable architecture search. In *Advances in Neural Information Processing Systems*, pp. 760–771, 2019.

Frank, M., Wolfe, P., et al. An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2): 95–110, 1956.

Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., and Sun, J. Single path one-shot neural architecture search with uniform sampling. In *European Conference on Computer Vision*, pp. 544–560. Springer, 2020.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015a.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28: 1135–1143, 2015b.

Hazan, E. and Luo, H. Variance-reduced and projection-free stochastic optimization. In *International Conference on Machine Learning*, pp. 1263–1271. PMLR, 2016a.

Hazan, E. and Luo, H. Variance-reduced and projection-free stochastic optimization. In *International Conference on Machine Learning*, pp. 1263–1271. PMLR, 2016b.

Hazan, E. and Minasyan, E. Faster projection-free online learning. In *Conference on Learning Theory*, pp. 1877–1893. PMLR, 2020.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2015.

He, Y., Lin, J., Liu, Z., Wang, H., Li, L.-J., and Han, S. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 784–800, 2018.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015. URL http://arxiv.org/abs/1503.02531.

Howard, A., Pang, R., Adam, H., Le, Q. V., Sandler, M., Chen, B., Wang, W., Chen, L., Tan, M., Chu, G., Vasudevan, V., and Zhu, Y. Searching for mobilenetv3. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pp. 1314–1324. IEEE, 2019. doi: 10.1109/ICCV.2019.00140. URL https://doi.org/10.1109/ICCV.2019.00140.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Hu, J., Shen, L., and Sun, G. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7132–7141, 2018.

Hu, Y., Wu, X., and He, R. Tf-nas: Rethinking three search freedoms of latency-constrained differentiable neural architecture search. *arXiv preprint arXiv:2008.05314*, 2020.

Huang, G., Chen, D., Li, T., Wu, F., van der Maaten, L., and Weinberger, K. Q. Multi-scale dense networks for resource efficient image classification. *arXiv preprint arXiv:1703.09844*, 2017.

Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, D., Chen, M., Lee, H., Ngiam, J., Le, Q. V., Wu, Y., et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *Advances in neural information processing systems*, pp. 103–112, 2019.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. Binarized neural networks. *Advances in neural information processing systems*, 29:4107–4115, 2016.

Intel(R). Intel(r) math kernel library for deep neural networks (intel(r) mkl-dnn), 2019. URL https://github.com/rsdubtso/mkl-dnn.

Jaggi, M. Revisiting frank-wolfe: Projection-free sparse convex optimization. In *International Conference on Machine Learning*, pp. 427–435. PMLR, 2013.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Kellerer, H., Pferschy, U., and Pisinger, D. *The Multiple-Choice Knapsack Problem*, pp. 317–347. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. ISBN 978-3-540-24777-7. doi: 10.1007/978-3-540-24777-7_11. URL https://doi.org/10.1007/978-3-540-24777-7_11.

Lacoste-Julien, S. and Jaggi, M. On the global linear convergence of frank-wolfe optimization variants. *arXiv preprint arXiv:1511.05932*, 2015.

Lacoste-Julien, S., Jaggi, M., Schmidt, M., and Pletscher, P. Block-coordinate frank-wolfe optimization for structural svms. In *International Conference on Machine Learning*, pp. 53–61. PMLR, 2013a.

Lacoste-Julien, S., Jaggi, M., Schmidt, M., and Pletscher, P. Block-coordinate frank-wolfe optimization for structural svms. In *International Conference on Machine Learning*, pp. 53–61. PMLR, 2013b.

Lee, H., Hyung, E., and Hwang, S. J. Rapid neural architecture search by learning to generate graphs from datasets. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=rkQuFUmUOg3.

Liang, H., Zhang, S., Sun, J., He, X., Huang, W., Zhuang, K., and Li, Z. Darts+: Improved differentiable architecture search with early stopping. *arXiv preprint arXiv:1909.06035*, 2019.

Lin, J., Rao, Y., Lu, J., and Zhou, J. Runtime neural pruning. In *Advances in neural information processing systems*, pp. 2181–2191, 2017.

Liu, H., Simonyan, K., and Yang, Y. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

Maurice, K. A new measure of rank correlation. *Biometrika*, 30(1-2):81–89, 1938.

Nash, J. C. The (dantzig) simplex method for linear programming. *Computing in Science and Engg.*, 2(1):29–31, January 2000.

Nayman, N., Noy, A., Ridnik, T., Friedman, I., Jin, R., and Zelnik, L. Xnas: Neural architecture search with expert advice. In *Advances in Neural Information Processing Systems*, pp. 1977–1987, 2019.

Noy, A., Nayman, N., Ridnik, T., Zamir, N., Doveh, S., Friedman, I., Giryes, R., and Zelnik, L. Asap: Architecture search, anneal and prune. In *International Conference on Artificial Intelligence and Statistics*, pp. 493–503. PMLR, 2020.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Pokutta, S., Spiegel, C., and Zimmer, M. Deep neural network training with frank-wolfe. *arXiv preprint arXiv:2010.07243*, 2020.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pp. 4780–4789, 2019.

Ridnik, T., Lawen, H., Noy, A., Ben Baruch, E., Sharir, G., and Friedman, I. Tresnet: High performance gpu-dedicated architecture. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 1400–1409, 2020.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018a.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018b.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

Spearman, C. "general intelligence" objectively determined and measured. 1961.

Stamoulis, D., Ding, R., Wang, D., Lymberopoulos, D., Priyantha, B., Liu, J., and Marculescu, D. Single-path nas: Designing hardware-efficient convnets in less than 4 hours. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 481–497. Springer, 2019.

Sun, S., Cao, Z., Zhu, H., and Zhao, J. A survey of optimization methods from a machine learning perspective. *IEEE transactions on cybernetics*, 50(8):3668–3681, 2019.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and 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.

Tan, M. and Le, Q. V. Efficientnet: Rethinking model scaling for convolutional neural networks. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6105–6114. PMLR, 2019. URL http://proceedings.mlr.press/v97/tan19a.html.

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2820–2828, 2019.

Tsiligkaridis, T. and Roberts, J. On frank-wolfe optimization for adversarial robustness and interpretability. *arXiv preprint arXiv:2012.12368*, 2020.

Umuroglu, Y., Fraser, N. J., Gambardella, G., Blott, M., Leong, P., Jahre, M., and Vissers, K. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 65–74, 2017.

Wang, X., Yu, F., Dou, Z.-Y., Darrell, T., and Gonzalez, J. E. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 409–424, 2018.

Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., and Keutzer, K. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pp. 10734–10742. Computer Vision Foundation / IEEE, 2019. doi: 10.1109/CVPR.2019.01099.

Xie, S., Zheng, H., Liu, C., and Lin, L. Snas: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926*, 2018.

Yu, K., Ranftl, R., and Salzmann, M. How to train your super-net: An analysis of training heuristics in weight-sharing nas. *arXiv preprint arXiv:2003.04276*, 2020.

Zhang, H., Wu, C., Zhang, Z., Zhu, Y., Zhang, Z., Lin, H., Sun, Y., He, T., Mueller, J., Manmatha, R., et al. Resnest: Split-attention networks. *arXiv preprint arXiv:2004.08955*, 2020a.

Zhang, M., Chen, L., Mokhtari, A., Hassani, H., and Karbasi, A. Quantized frank-wolfe: Faster optimization, lower communication, and projection free. In *International Conference on Artificial Intelligence and Statistics*, pp. 3696–3706. PMLR, 2020b.

Zhang, X., Zhou, X., Lin, M., and Sun, J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6848–6856, 2018.

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