BINAS: Bilinear Interpretable Neural Architecture Search

Niv Nayman[*] NIVN@CAMPUS.TECHNION.AC.IL
Technion - Israel Institute of Technology, Haifa, Israel[+]

Yonathan Aflalo[*] YOAFLALO@AMAZON.COM
Amazon Alexa Shopping, Tel-Aviv, Israel[+]

Asaf Noy ASAF.NOY@ALIBABA-INC.COM
Alibaba Group, Tel Aviv, Israel

Lihi Zelnik-Manor LIHI@TECHNION.AC.IL
Technion - Israel Institute of Technology, Haifa, Israel[+]

Editors: Emtiyaz Khan and Mehmet Gönen

Abstract

Making neural networks practical often requires adhering to resource constraints such as latency, energy and memory. To solve this we introduce a *Bilinear Interpretable* approach for constrained *Neural Architecture Search (BINAS)* that is based on an accurate and simple bilinear formulation of both an accuracy estimator and the expected resource requirement, jointly with a scalable search method with theoretical guarantees. One major advantage of BINAS is providing interpretability via insights about the contribution of different design choices. For example, we find that in the examined search space, adding depth and width is more effective at deeper stages of the network and at the beginning of each resolution stage. BINAS differs from previous methods that typically use complicated accuracy predictors that make them hard to interpret, sensitive to many hyper-parameters, and thus with compromised final accuracy. Our experiments [1] show that BINAS generates comparable to or better than state of the art architectures, while reducing the marginal search cost, as well as strictly satisfying the resource constraints.

Keywords: Neural Architecture Search, Computer Vision, Deep Learning, Optimization

1. Introduction

The increasing utilization of Convolutional Neural Networks (CNN) in real systems and commercial products puts neural networks with both high accuracy and fast inference speed in high demand. Early days architectures, such as VGG Simonyan and Zisserman (2015) or ResNet He et al. (2015), were designed for powerful GPUs as those were the common computing platform for deep CNNs, however, in recent years the need for deployment on standard CPUs and edge devices emerged. These computing platforms are limited in their abilities and as a result require lighter architectures that comply with strict requirements

*. These authors contributed equally.
+. This work was partially done while all the authors were also affiliated with Alibaba Group.
1. The full code is available at `https://github.com/Alibaba-MIIL/BINAS`

on real time latency and power consumption. This has spawned a line of research aimed at finding architectures with both high performance and constrained resource demands.

The main approaches to solve this evolved from Neural Architecture Search (NAS) Zoph and Le (2016); Liu et al. (2018); Cai et al. (2018), while a constraint on the target latency is added over various platforms, e.g., CPU, TPU, FPGA, MCU etc. Those constrained-NAS methods can be grouped into two categories: (i) Reward based methods such as Reinforcement-Learning (RL) or Evolutionary Algorithms (EA) Cai et al. (2019); Tan et al. (2019); Tan and Le (2019); Howard et al. (2019), where the latency and accuracy of sampled architectures are predicted by evaluations on the target devices over some validation set to perform the search. The predictors are typically made of complicated models and hence require many samples and sophisticated fitting techniques White et al. (2021). Overall this oftentimes leads to inaccurate, expensive to acquire, and hard to optimize objective functions due to their complexity. (ii) Resource-aware gradient based methods formulate a differentiable loss function consisting of a trade-off between an accuracy term and either a proxy soft penalty term Hu et al. (2020); Wu et al. (2019) or a hard constraint Nayman et al. (2021). Therefore, the architecture can be directly optimized via bi-level optimization using stochastic gradient descent (SGD) or stochastic Frank-Wolfe (SFW) Hazan and Luo (2016), respectively. However, the bi-level nature of the problem introduces many challenges Chen et al. (2019); Noy et al. (2020); Nayman et al. (2019) and recently Wang et al. (2021) pointed out the inconsistencies associated with using gradient information as a proxy for the quality of the architectures, especially in the presence of skip connections in the search space. These inconsistencies call for making NAS more interpretable, by extending its scope from finding optimal architectures to interpretable features Ru et al. (2021) and their corresponding impact on the network performance.

In this paper, we propose an interpretable search algorithm that is fast and scalable, yet produces architectures with high accuracy that satisfy hard latency constraints. At the heart of our approach is an accuracy estimator which is interpretable, easy to optimize and does not have a strong reliance on gradient information. Our proposed predictor measures the performance contribution of individual design choices by sampling sub-networks from a one-shot model (Bender et al. (2018); Chu et al. (2019); Guo et al. (2020); Cai et al. (2019); Nayman et al. (2021)). Constructing the estimator this way allows making insights about the contribution of the design choices. It is important to note, that albeit its simplicity, our predictor's performance matches that of previously proposed predictors, that are typically expensive to compute and hard to optimize due to many hyper-parameters.

The predictor we propose has a bilinear form that allows formulating the latency constrained NAS problem as an *Integer Quadratic Constrained Quadratic Programming* (IQCQP). Thanks to this, the optimization can be efficiently solved via a simple algorithm with some off-the-shelf components. The algorithm we suggest solves it within a few minutes on a common CPU.

Overall our optimization approach has two main performance related advantages. First, the outcome networks provide high accuracy and closely comply with the latency constraint. Second, the search is highly efficient, which makes our approach scalable to multiple target devices and latency demands.

2. Related Work

Neural Architecture Search methods automate models' design per provided constraints. Early methods like NASNet Zoph and 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.

Predictor based methods recently have been proposed based on training a model to predict the accuracy of an architecture just from an encoding of the architecture. Popular choices for these models include Gaussian processes, neural networks, tree-based methods. See Lu et al. (2020) for such utilization and White et al. (2021) for comprehensive survey.

Interpretabe NAS was introduced by Ru et al. (2021) through an elaborated Bayesian optimisation with Weisfeiler-Lehman kernel to identify beneficial topological features. We propose an intuitive and simpler approach for NAS interpretibiliy for the efficient search space examined. This leads to more understanding and applicable design rules.

Hardware-aware methods such as ProxylessNAS Cai et al. (2018), Mnasnet Tan et al. (2019), SPNASNet Stamoulis et al. (2019), FBNet Wu et al. (2019), and TFNAS Hu et al. (2020) generate architectures that comply to the constraints by applying simple heuristics such as soft penalties on the loss function. OFA Cai et al. (2019) and HardCoRe-NAS Nayman et al. (2021) proposed a scalable approach across multiple devices by training an one-shot model Brock et al. (2017); Bender et al. (2018) once. This pretrained super-network is highly predictive for the accuracy ranking of extracted sub-networks, e.g. FairNAS Guo et al. (2020), SPOS Guo et al. (2020). OFA applies evolutionary search Real et al. (2019) over a complicated multilayer perceptron (MLP) Rumelhart et al. (1985) based accuracy predictor with many hyperparameters to be tuned. HardCore-NAS searches by backpropagation Kelley (1960) over a supernetwork under strict latency constraints for several GPU hours per network. Hence it requires access to a powerful GPU to perform the search and both approaches lack interpretability. This work relies on such one-shot model, for intuitively building an interpretable and simple bilinear accuracy estimator that matches in performance without any tuning and optimized under strict latency constraints by solving an IQCQP problem (for related work see the introduction section of Billionnet et al. (2016)) in several CPU minutes following by a short fine-tuning.

3. Method

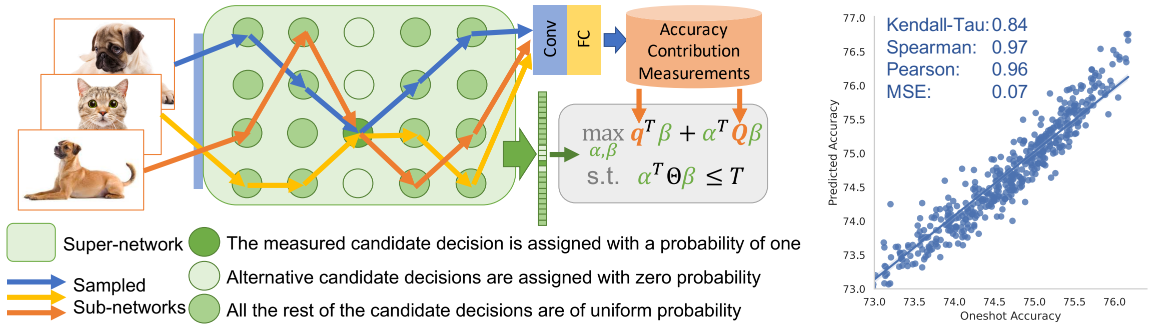
Conv
FC
Accuracy Contribution Measurements
$\max_{\alpha,\beta} q^T\beta + \alpha^T Q \beta$
s.t. $\alpha^T \Theta \beta \le T$
Super-network
The measured candidate decision is assigned with a probability of one
Sampled Sub-networks
Alternative candidate decisions are assigned with zero probability
All the rest of the candidate decisions are of uniform probability
Kendall-Tau: 0.84
Spearman: 0.97
Pearson: 0.96
MSE: 0.07
Predicted Accuracy
Oneshot Accuracy

Figure 1: (Left) The BINAS scheme constructs a bilinaer accuracy estimator by measuring the accuracy contribution of individual design choices and then maximizing this objective under bilinear latency constraints. (Right) Accuracy predictions vs measured accuracy of 500 subnetworks sampled uniformly at random. High ranking correlations are achieved.

In this section we propose our method for latency-constrained NAS. We search for an architecture with the highest validation accuracy under a predefined latency constraint, denoted by T. We start by following Bender et al. (2018), training a supernetwork that accommodates the search space $\mathcal{S}$, as described in section 3.1. The supernetwork training ensures that the accuracy of subnetworks extracted from it, together with their corresponding weights, are properly ranked Guo et al. (2020); Chu et al. (2019); Cai et al. (2019); Nayman et al. (2021) as if those were trained from scratch. Given such a supernetwork, we sample subnetworks from it for estimating the individual accuracy contribution of each design choice (section 3.2), and construct a bilinear accuracy estimator for the expected accuracy of every possible subnetwork in the search space (section 3.3). Finally, the latency of each possible block configuration is measured on the target device and aggregated to form a bilinear latency constraint. Putting it all together (section 3.4) we formulate an IQCQP:

$$\begin{aligned} &\max_{\zeta} ACC(\zeta) = q^T\zeta + \zeta^T Q\zeta \\ &\text{s.t. } LAT(\zeta) = \zeta^T\Theta\zeta \le T, \quad A_{\mathcal{S}}\cdot\zeta \le b_{\mathcal{S}}, \quad \zeta\in\mathbb{Z}^N \end{aligned} \tag{1}$$

where $\zeta=(\boldsymbol{\alpha},\boldsymbol{\beta})\in\mathcal{S}\subset\mathbb{Z}^N$, is the parametrization of the design choices in the search space that govern the architecture structure, $q\in\mathbb{R}^N$, $Q\in\mathbb{R}^{N\times N}$, $\Theta\in\mathbb{R}^{N\times N}$, $A_{\mathcal{S}}\in\mathbb{R}^{C\times N}$, $b_{\mathcal{S}}\in\mathbb{R}^C$ and $\zeta\in\mathcal{S}$ can be expressed as a set of C linear equations. Finally, in section 3.5 we propose an optimization method to efficiently solve Problem 1. Figure 1 (Left) shows a high level illustration of the scheme.

3.1. The Search Space

We consider a general search space that integrates a macro search space and a micro search space. The macro search space is composed of S stages $s\in\{1,..,S\}$ of different input resolutions, each composed of blocks $b\in\{1,..,D\}$ with the same input resolution, and defines how the blocks are connected, see Figure 2. The micro search space controls the internal structures of each block. Specifically, this search space includes latency efficient search spaces introduced in Wu et al. (2019); Howard et al. (2019); Tan et al. (2019); Hu et al. (2020); Cai et al. (2019); Nayman et al. (2021).

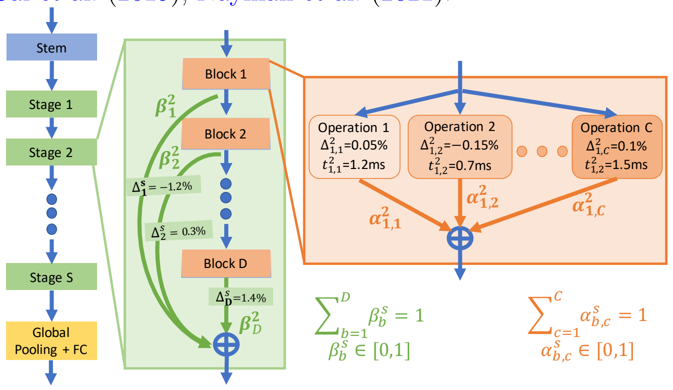
Stem
Stage 1
Stage 2
Stage S
Global Pooling + FC
Block 1
Block 2
Block D
β_1^2
β_2^2
β_D^2
$\Delta_1^s = -1.2\%$
$\Delta_2^s = 0.3\%$
$\Delta_D^s = 1.4\%$
Operation 1 $\Delta_{1,1}^2 = 0.05\%$ $t_{1,1}^2 = 1.2$ms
Operation 2 $\Delta_{1,2}^2 = -0.15\%$ $t_{1,2}^2 = 0.7$ms
Operation C $\Delta_{1,C}^2 = 0.1\%$ $t_{1,2}^2 = 1.5$ms
$\alpha_{1,1}^2$
$\alpha_{1,2}^2$
$\alpha_{1,C}^2$
$\sum_{b=1}^D \beta_b^s = 1$
$\beta_b^s \in [0,1]$
$\sum_{c=1}^C \alpha_{b,c}^s = 1$
$\alpha_{b,c}^s \in [0,1]$

Figure 2: BINAS search space. The individual accuracy contribution and latency of each configured operation are measured.

A block configuration $c\in\mathcal{C}$ (specified in Appendix B) corresponds to parameters $\boldsymbol{\alpha}$. For each block b of stage s we have $\alpha_{b,c}^s\in\{0,1\}^{|\mathcal{C}|}$ and $\Sigma_{c\in\mathcal{C}}\alpha_{b,c}^s=1$. An input feature map x_b^s to block b of stage s is processed as follows: $x_{b+1}^s=\sum_{c\in\mathcal{C}}\alpha_{b,c}^s\cdot O_{b,c}^s(x_b^s)$, where $O_{b,c}^s(\cdot)$ is the operation configured by $c\in\mathcal{C}$. The depth of each stage s is controlled by the parameters $\boldsymbol{\beta}$: $x_1^{s+1}=\Sigma_{b=1}^D\beta_b^s\cdot x_{b+1}^s$, such that $\beta_b^s\in\{0,1\}^D$ and $\Sigma_{b=1}^D\beta_b^s=1$.

To summarize, the search space is composed of both the micro and macro search spaces parameterized by $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, respectively:

$$\mathcal{S} = \left\{ (\boldsymbol{\alpha}, \boldsymbol{\beta}) \middle| \begin{array}{c} \alpha^s_{b,c} \in \{0,1\}^{|\mathcal{C}|} \,;\, \Sigma_{c\in\mathcal{C}}\alpha^s_{b,c} = 1 \\ \beta^s_b \in \{0,1\}^D \,;\, \Sigma^D_{b=1}\beta^s_b = 1 \end{array} \;;\; \begin{array}{l} \forall s \in \{1,..,S\} \\ \forall b \in \{1,..,D\}, c \in \mathcal{C} \end{array} \right\} \tag{2}$$

such that a continuous probability distribution is induced over the space, by relaxing $\alpha^s_{b,c} \in \{0,1\}^{|\mathcal{C}|}$ to $\alpha^s_{b,c} \in \mathbb{R}^{|\mathcal{C}|}_+$ and $\beta^s_b \in \{0,1\}^D$ to $\beta^s_b \in \mathbb{R}^D_+$ to be continuous rather than discrete. Therefore, this probability distribution can be expressed by a set of linear equations and one can view the parametrization $\zeta = (\boldsymbol{\alpha}, \boldsymbol{\beta})$ as a composition of probabilities in $\mathcal{P}_\zeta(\mathcal{S}) = \{\zeta \mid A_\mathcal{S}\zeta \le b_\mathcal{S}\} = \{(\boldsymbol{\alpha}, \boldsymbol{\beta}) \mid A^\alpha_\mathcal{S} \cdot \boldsymbol{\alpha} \le b^\alpha_\mathcal{S}, A^\beta_\mathcal{S} \cdot \boldsymbol{\beta} \le b^\beta_\mathcal{S}\}$ or as degenerate one-hot vectors in $\mathcal{S}$.

3.2. Estimating the Accuracy Contribution of Design Choices

Next we introduce a simple way to estimate the accuracy contribution of each design choice, given a trained one-shot model. With this at hand, we will be able to select those design choices that contribute the most to the accuracy under some latency budget. Suppose a supernetwork is constructed, such that every sub-network of it resides in the search space of Section 3.1. The supernetwork is trained to rank well different subnetworks Guo et al. (2020); Chu et al. (2019); Cai et al. (2019); Nayman et al. (2021). The expected accuracy of such a supernetwork $\mathbb{E}[Acc]$ is estimated by uniformly sampling a different subnetwork for each input image from 20% of the Imagenet train set (considered as a validation set), as illustrated in Figure 3. This estimate serves as the base accuracy of the

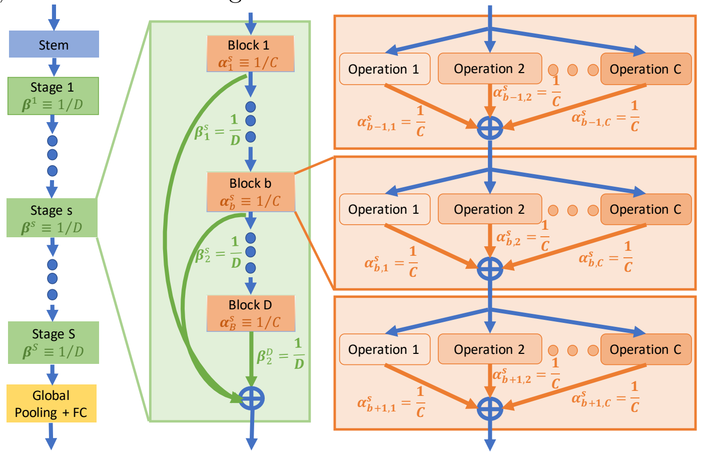
Stem
Stage 1
$\beta^1 \equiv 1/D$
Stage s
$\beta^s \equiv 1/D$
Stage S
$\beta^S \equiv 1/D$
Global Pooling + FC
Block 1
$\alpha^s_1 \equiv 1/C$
$\beta^s_1 = \frac{1}{D}$
Block b
$\alpha^s_b \equiv 1/C$
$\beta^s_2 = \frac{1}{D}$
Block D
$\alpha^s_B \equiv 1/C$
$\beta^D_2 = \frac{1}{D}$
Operation 1
Operation 2
Operation C
$\alpha^s_{b-1,1} = \frac{1}{C}$
$\alpha^s_{b-1,2} = \frac{1}{C}$
$\alpha^s_{b-1,C} = \frac{1}{C}$
$\alpha^s_{b,1} = \frac{1}{C}$
$\alpha^s_{b,2} = \frac{1}{C}$
$\alpha^s_{b,C} = \frac{1}{C}$
$\alpha^s_{b+1,1} = \frac{1}{C}$
$\alpha^s_{b+1,2} = \frac{1}{C}$
$\alpha^s_{b+1,C} = \frac{1}{C}$

Figure 3: Estimating the base accuracy of the supernetwork is done by random uniform sampling of subnetworks for each input image.

supernetwork and thus every design choice should be evaluated by its contribution on top of it. Hence, the individual accuracy contribution of setting the depth of stage s to b is the gap: $\Delta^s_b = \mathbb{E}[Acc|d^s = b] - \mathbb{E}[Acc]$, where the first expectation is estimated by setting $\beta^s_b = 1$ and uniform distribution for the rest of the design choices. This is done for every possible depth of every stage (Figure 4).

Similarly, the individual accuracy contribution of choosing configuration c in block b of stage s is given by the gap: $\Delta^s_{b,c} = \mathbb{E}[Acc|O^s_{b,c} = O_c, d^s = b] - \mathbb{E}[Acc]$, where the first expectation is estimated by setting $\alpha^s_{b,c} = 1$ and $\beta^s_b = 1$, while keeping a uniform distribution for all the rest of the design choices in the supernetwork, as illustrated in Figure 5.

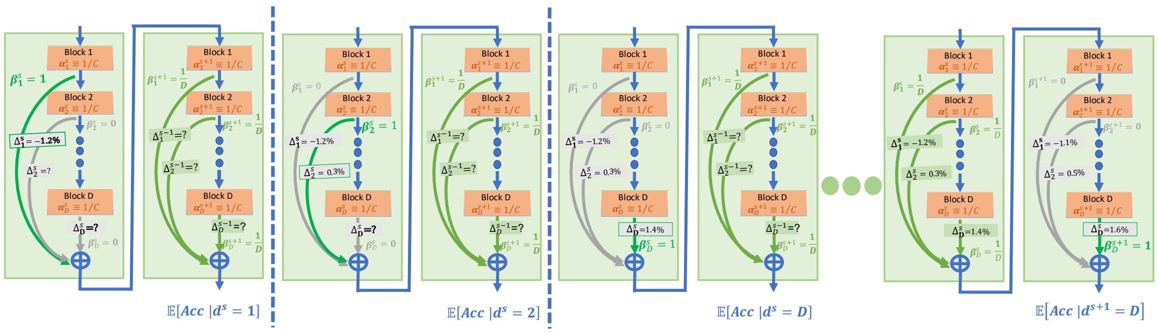

Figure 4: Estimating the expected accuracy gap caused by macroscopic design choices of the depth of the stages.

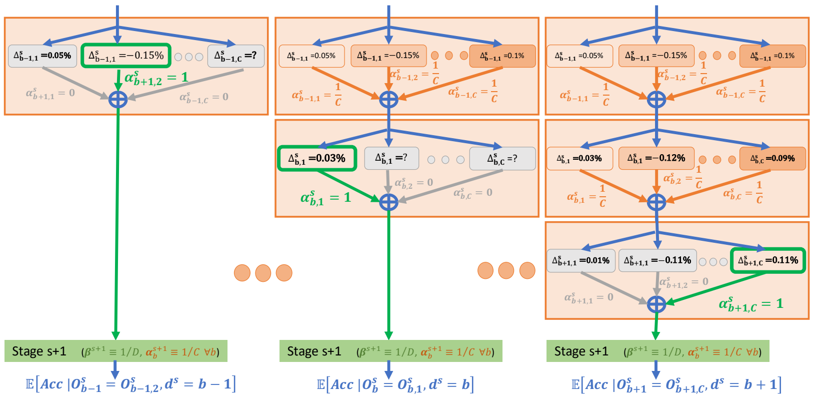

Figure 5: Estimating the expected accuracy gap caused by microscopic design choices on the operation applied at every block one at a time.

3.3. Constructing a Bilinear Accuracy Estimator

We next propose an intuitive and effective way to utilize the estimated individual accuracy contribution of each design choice (section 3.2) to estimate the expected overall accuracy of a certain architecture in the search space.

The expected accuracy contribution of a block b can be computed by summing over the accuracy contributions $\Delta^s_{b,c}$ of every possible configuration $c \in \mathcal{C}$: $\bar{\delta}^s_b = \Sigma_{c\in\mathcal{C}} \alpha^s_{b,c} \cdot \Delta^s_{b,c}$ Thus the expected accuracy contribution of the stage s of depth b' is $\delta^s_{b'} = \Delta^s_{b'} + \Sigma^{b'}_{b=1} \bar{\delta}^s_b$, where the first term is the accuracy contribution associated solely with the choice of depth b' and the second term is the aggregation of the expected accuracy contributions of the first b' blocks in the stage. Taking the expectation over all possible depths for stage s yields $\delta^s = \sum_{b'=1}^{D} \beta^s_{b'} \cdot \delta^s_{b'}$ and summing over all the stages results in the aggregated accuracy contribution of all design choices. Hence the accuracy of a subnetwork can be calculated by this estimated contribution on top of the estimated base accuracy $\mathbb{E}[Acc]$ (Figure 3):

$$ACC(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \mathbb{E}[Acc] + \sum_{s=1}^{S}\sum_{b=1}^{D} \beta^s_b \cdot \Delta^s_b + \sum_{s=1}^{S}\sum_{b=1}^{D}\sum_{b'=b}^{D}\sum_{c\in\mathcal{C}} \alpha^s_{b,c} \cdot \Delta^s_{b,c} \cdot \beta^s_{b'} \qquad (3)$$

And its vectorized form can be expressed as the following bilinear formula in $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$: $ACC(\boldsymbol{\alpha}, \boldsymbol{\beta}) = r + q_\beta^T \boldsymbol{\beta} + \boldsymbol{\alpha}^T Q_{\alpha\beta} \boldsymbol{\beta}$, where $r = E[Acc]$, $q_\beta \in \mathbb{R}^{D\cdot S}$ is a vector composed of Δ^s_b and $Q_{\alpha\beta} \in \mathbb{R}^{\mathcal{C}\cdot D\cdot S\times D\cdot S}$ is a matrix composed of $\Delta^s_{b,c}$.

We next present a theorem (with proof in Appendix E) that states that the estimator in equation 3 approximates well the expected accuracy of an architecture.

Theorem 3.1 *Assume $\{O_b^s, d_s\}$ for $s = 1, \dots, S$ and $b = 1, \dots, D$ are conditionally independent with the accuracy Acc. Suppose that there exists a positive real number $0 < \epsilon \ll 1$ such that for any $X \in \{O_b^s, d_s\}$ the following holds $|\mathbb{P}[Acc|X] - \mathbb{P}[Acc]| < \epsilon \mathbb{P}[Acc]$. Then:*

$$\mathbb{E}\left[Acc \middle| \cap_{s=1}^S \cap_{b=1}^D O_b^s, \cap_{s=1}^S d^s\right] = \mathbb{E}[Acc] + (1 + \mathcal{O}(N\epsilon)) \cdot \sum_{s=1}^S \sum_{b=1}^D \beta_b^s \cdot \Delta_b^s \\ + (1 + \mathcal{O}(N\epsilon)) \cdot \sum_{s=1}^S \sum_{b=1}^D \sum_{b'=b}^D \sum_{c \in \mathcal{C}} \alpha_{b,c}^s \cdot \Delta_{b,c}^s \cdot \beta_{b'}^s \tag{4}$$

Some of the assumptions we adopted were to enable and facilitate the analysis. Our empirical evaluation shows that in certain search spaces and tasks this assumption holds nicely, while for others it might hold to some extent.

Theorem 3.1 and Figure 1 (right) demonstrate the effectiveness of relying on $\Delta_{b,c}^s$, Δ_b^s to express the expected accuracy of networks. Since those terms measure the accuracy contributions of individual design decisions, many insights and design rules can be extracted from those, as discussed in section 5.3, making the proposed estimator intuitively interpretable. Furthermore, the transitivity of ranking correlations is used in appendix I for guaranteeing good prediction performance with respect to architectures trained from scratch.

3.4. The Integer Quadratic Constraints Quadratic Program

In this section we formulate latency-constrained NAS as an IQCQP of bilinear objective function and bilinear constraints. For the purpose of maximizing the validation accuracy of the selected subnetwork under latency constraint, we utilize the bilinear accuracy estimator derived in section 3.3 as the objective function and define the bilinear latency constraint similarly to Hu et al. (2020); Nayman et al. (2021):

$$LAT(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \sum_{s=1}^S \sum_{b=1}^D \sum_{b'=b}^D \sum_{c \in \mathcal{C}} \alpha_{b,c}^s \cdot t_{b,c}^s \cdot \beta_{b'}^s = \boldsymbol{\alpha}^T \Theta \boldsymbol{\beta} \tag{5}$$

where $\Theta \in \mathbb{R}^{\mathcal{C} \cdot D \cdot S \times D \cdot S}$ is a matrix composed of the latency measurements $t_{b,c}^s$ on the target device of each configuration $c \in \mathcal{C}$ of every block b in every stage s (Figure 2). The bilinear version of problem 1 turns to be:

$$\begin{aligned} \max_{\alpha_{b,c}^s, \beta_b^s} \quad & \sum_{s=1}^S \sum_{b=1}^D \beta_b^s \cdot \Delta_b^s + \sum_{s=1}^S \sum_{b=1}^D \sum_{b'=b}^D \sum_{c \in \mathcal{C}} \alpha_{b,c}^s \cdot \Delta_{b,c}^s \cdot \beta_{b'}^s \\ \text{s.t.} \quad & \sum_{s=1}^S \sum_{b=1}^D \sum_{b'=b}^D \sum_{c \in \mathcal{C}} \alpha_{b,c}^s \cdot t_{b,c}^s \cdot \beta_{b'}^s \leq T \\ & \Sigma_{c \in \mathcal{C}} \alpha_{b,c}^s = 1\,;\ \alpha_{b,c}^s \in \{0,1\}^{|\mathcal{C}|}\ \forall s \in \{1,..,S\}, b \in \{1,..,D\}, c \in \mathcal{C} \\ & \Sigma_{b=1}^D \beta_b^s = 1\,;\ \beta_b^s \in \{0,1\}^D\ \forall s \in \{1,..,S\}, b \in \{1,..,D\} \end{aligned} \tag{6}$$

And in its vectorized form:

$$\max_{\boldsymbol{\alpha}, \boldsymbol{\beta} \in \{0,1\}} q_\beta^T \boldsymbol{\beta} + \boldsymbol{\alpha}^T Q_{\alpha\beta} \boldsymbol{\beta} \text{ s.t. } \boldsymbol{\alpha}^T \Theta \boldsymbol{\beta} \leq T\,;\ A_{\mathcal{S}}^\alpha \cdot \boldsymbol{\alpha} \leq b_{\mathcal{S}}^\alpha\,;\ A_{\mathcal{S}}^\beta \cdot \boldsymbol{\beta} \leq b_{\mathcal{S}}^\beta \tag{7}$$

3.5. Solving the Integer Quadratic Constraints Quadratic Program

By formulating the latency-constrained NAS as a binary problem in section 3.4, we can now use out-of-the-box *Mixed Integer Quadratic Constraints Programming* (MIQCP) solvers to optimize problem 6. We use IBM CPLEX IBM ILOG CPLEX that supports non-convex binary QCQP and utilizes the Branch-and-Cut algorithm Padberg and Rinaldi (1991) for this purpose. A heuristic alternative for optimizing an objective function under integer constraints is evolutionary search Real et al. (2019), as next we propose a more theoretically sound alternative.

3.5.1. Utilizing the Block Coordinate Frank-Wolfe Algorithm

As pointed out by Nayman et al. (2021), since Θ is constructed from measured latency in equation 5, it is not guaranteed to be positive semi-definite, hence, the induced quadratic constraint makes the feasible domain in problem 1 non-convex in general. To overcome this we adapt the *Block-Coordinate Frank-Wolfe* (BCFW) Lacoste-Julien et al. (2013) for solving a continuous relaxation of problem 1, such that $\zeta \in \mathbb{R}_+^N$. Essentially BCFW adopts the Frank-Wolfe Frank et al. (1956) update rule for each block of coordinates in $\delta \in \{\boldsymbol{\alpha}, \boldsymbol{\beta}\}$ picked up at random at each iteration k, such that $\delta_{k+1} = (1-\gamma_k)\cdot\delta_k + \gamma_k \cdot \hat{\delta}$ with $0 \leq \gamma_k \leq 1$, for any partially differentiable objective function $ACC(\boldsymbol{\alpha}, \boldsymbol{\beta})$:

$$\hat{\boldsymbol{\alpha}} = \underset{\boldsymbol{\alpha}}{\operatorname{argmax}} \ \nabla_{\boldsymbol{\alpha}} ACC(\boldsymbol{\alpha}, \boldsymbol{\beta}_k)^T \cdot \boldsymbol{\alpha} \quad \text{s.t. } \boldsymbol{\beta}_k^T \Theta^T \cdot \boldsymbol{\alpha} \leq T \quad ; \quad A_{\mathcal{S}}^{\alpha} \cdot \boldsymbol{\alpha} \leq b_{\mathcal{S}}^{\alpha} \tag{8}$$

$$\hat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{\operatorname{argmax}} \ \nabla_{\boldsymbol{\beta}} ACC(\boldsymbol{\alpha}_k, \boldsymbol{\beta})^T \cdot \boldsymbol{\beta} \quad \text{s.t. } \boldsymbol{\alpha}_k^T \Theta \cdot \boldsymbol{\beta} \leq T \quad ; \quad A_{\mathcal{S}}^{\beta} \cdot \boldsymbol{\beta} \leq b_{\mathcal{S}}^{\beta} \tag{9}$$

where ∇_δ stands for the partial derivatives with respect to δ. Convergence guarantees are provided in Lacoste-Julien et al. (2013). Then, once converged to the solution of the continuous relaxation of the problem, we need to project the solution back to the discrete space of architectures, specified in equation 2, as done in Nayman et al. (2021). This step could deviate from the solution and cause degradation in performance.

Due to the formulation of the NAS problem 7 as a *Bilinear Programming* (BLP) Gallo and Ülkücü (1977) with bilinear constraints (BLCP) we design Algorithm 1 that applies the BCFW with line-search for this special case. Thus more specific convergence guarantees can be provided together with the sparsity of the solution, hence no additional discretization step is required. The following theorem states that after $\mathcal{O}(1/\epsilon)$ iterations, Algorithm 1 obtains an ϵ-approximate solution to problem 7.

Algorithm 1 BCFW with Line Search for BLCP

```
input $(\boldsymbol{\alpha}_0, \boldsymbol{\beta}_0) \in \left\{ (\boldsymbol{\alpha}, \boldsymbol{\beta}) \,\middle|\, \boldsymbol{\alpha}^T \Theta \boldsymbol{\beta} \leq T, A_{\mathcal{S}}^{\alpha}\boldsymbol{\alpha} \leq b_{\mathcal{S}}^{\alpha}, A_{\mathcal{S}}^{\beta}\boldsymbol{\beta} \leq b_{\mathcal{S}}^{\beta} \right\}$
1: for $k = 0, \dots, K-1$ do
2:   if $Bernoulli(p) == 1$ then
3:     $\boldsymbol{\alpha}_{k+1} = \operatorname{argmax}_{\boldsymbol{\alpha}} (q_\alpha^T + \beta_k^T Q_{\alpha\beta}^T) \cdot \boldsymbol{\alpha}$ s.t. $\boldsymbol{\beta}_k^T \Theta^T \cdot \boldsymbol{\alpha} \leq T$ ; $A_{\mathcal{S}}^{\alpha} \cdot \boldsymbol{\alpha} \leq b_{\mathcal{S}}^{\alpha}$ and $\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k$
4:   else
5:     $\boldsymbol{\beta}_{k+1} = \operatorname{argmax}_{\boldsymbol{\beta}} (q_\beta^T + \alpha_k^T Q_{\alpha\beta}) \cdot \boldsymbol{\beta}$ s.t. $\boldsymbol{\alpha}_k^T \Theta \cdot \boldsymbol{\beta} \leq T$ ; $A_{\mathcal{S}}^{\beta} \cdot \boldsymbol{\beta} \leq b_{\mathcal{S}}^{\beta}$ and $\boldsymbol{\alpha}_{k+1} = \boldsymbol{\alpha}_k$
6:   end if
7: end for
output $\zeta^* = (\boldsymbol{\alpha}_K, \boldsymbol{\beta}_K)$
```

Theorem 3.2 *For each $k > 0$ the iterate $\zeta_k = (\boldsymbol{\alpha}_k, \boldsymbol{\beta}_k)$ of Algorithm 1 satisfies:*

$$E[ACC(\zeta_k)] - ACC(\zeta^*) \leq \frac{4}{k+4}\left(ACC(\zeta_0) - ACC(\zeta^*)\right)$$

where $\zeta^* = (\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$ is the solution of a continuous relaxation of problem 7 and the expectation is over the random choice of the block $\boldsymbol{\alpha}$ or $\boldsymbol{\beta}$.

The proof is in Appendix G. We next provide a guarantee that Algorithm 1 directly yields a *sparse solution*, representing a valid sub-network without the additional discretization step required by other continuous methods Liu et al. (2018); Wu et al. (2019); Hu et al. (2020); Nayman et al. (2021).

Theorem 3.3 *The output solution $(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \zeta^*$ of Algorithm 1 contains only one-hot vectors for $\alpha^s_{b,c}$ and β^s_b, except from a single one for each of those blocks, which contains a couple of non-zero entries.*

The proof is in Appendix H. In practice, a negligible latency deviation is associated with taking the argmax over the only two couples. Differently from the sparsity guarantee for the discretization step in Nayman et al. (2021), Theorem 3.3 guarantees similar desirable properties but for the fundamentally different Algorithm 1 that does not involve an additional designated projection step.

4. Experimental Results

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

4.1.1. Search Space Specifications.

Aiming at latency efficient architectures, we adopt the search space introduced in Nayman et al. (2021), which is closely related to those used by Wu et al. (2019); Howard et al. (2019); Tan et al. (2019); Hu et al. (2020); Cai et al. (2019). The macro search space is composed of $S = 5$ stages, each composed of at most $D = 4$ blocks. The micro search space is based on *Mobilenet Inverted Residual* (MBInvRes) blocks Sandler et al. (2018) and controls the internal structures of each block. Every MBInvRes block is configured by an expansion ratio $er \in \{3, 4, 6\}$ of the point-wise convolution, kernel size $k \in \{3 \times 3, 5 \times 5\}$ of the Depth-Wise Separable convolution (DWS), and Squeeze-and-Excitation (SE) layer Hu et al. (2018) $se \in \{\text{on}, \text{off}\}$ (details in Appendix B).

4.1.2. Comparisons with Other Methods.

We compare our generated architectures to other state-of-the-art NAS methods in Table 1 and Figures 6 and 7 (Right). Aiming for surpassing the previous state-of-the-art Nayman et al. (2021) search methods, we use its search space (Section 4.1.1) and official supernetwork training. This way we can show that the improved search method (Section 5.4 and Figure 7 (Middle)) leads to superior results for the same marginal search cost of 15 GPU hours per additional generated model, as shown in Figure 7 (Right).

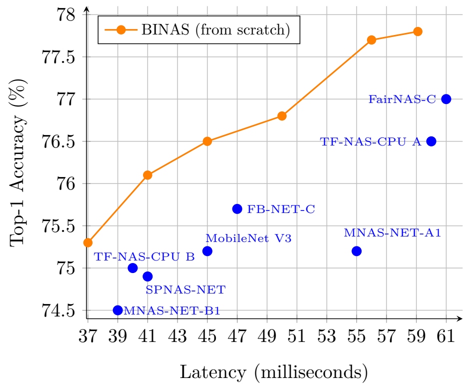

Figure 6: Imagenet Top-1 accuracy vs latency. All models are trained from scratch.

For the purpose of comparing the generated architectures of other methods alone, excluding the contribution of evolved pretraining techniques, for each model in Table 1 and Figure 6, the official PyTorch implementation is trained from a scratch using the exact same code and hyperparameters, as specified in appendix C. The maximum accuracy between our training and the original paper is reported. The latency values presented are actual time measurements of the models, running on a single thread with the exact same settings and on the same hardware. We disabled optimizations, e.g., Intel MKL-DNN Intel (R), hence 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 are comparable to or surpassing all the other methods on the top-1 Imagenet accuracy with a reduced scalable search cost. The same results hold when comparing model size and FLOPS, as shown in appendix D. The total search time consists of 435 GPU hours computed only once as preprocessing and additional 8 GPU hours for fine-tuning each generated network, while the search itself requires negligible several CPU minutes, see appendix A for more details. Due to the negligible search cost, one can choose to train the models longer (e.g. 15 GPU hours) to achieve better results with no larger marginal cost than other methods.

5. Empirical Analysis of Key Components

In this section we analyze and discuss different aspects of the proposed method.

5.1. The Contribution of Different Terms of the Accuracy Estimator

The accuracy estimator in equation 3 aggregates the contributions of multiple architectural decisions. In section 3.3, those decisions are grouped into two groups: (1) macroscopic decisions about the depth of each stage are expressed by q_β and (2) microscopic decisions about the configuration of each block are expressed by $Q_{\alpha\beta}$.

Table 2 quantifies the contribution of each of those terms to the ranking correlations by setting the corresponding terms to zero. We conclude that the depth of the network is very significant for estimating the accuracy of architectures, as setting q_β to zero specifically decreases the Kendall-Tau and Spearman's correlation coefficients from 0.84 and 0.97 to 0.29 and 0.42 respectively. The significance of microscopic decisions about the configuration of blocks is also viable but not as much, as setting $Q_{\alpha\beta}$ to zero decreases the Kendall-Tau and Spearman's correlation to 0.66 and 0.85 respectively.

Variant	Kendall-Tau	Spearman
$q_\beta \equiv 0$	0.29	0.42
$Q_{\alpha\beta} \equiv 0$	0.66	0.85
$ACC(\boldsymbol{\alpha}, \boldsymbol{\beta})$	0.84	0.97

Table 2: Contribution of terms.

5.2. Comparison to Learning the Accuracy Predictors

While the purpose of this work is not to compare many accuracy predictors, as this has been already done comprehensively by White et al. (2021), comparisons to certain learnt predictors support the validity and benefits of the proposed accuracy estimator (section 3.3) despite its simple functional form. Hence each of the following comparisons is chosen for a reason: (1) Showing that it is more sample efficient than learning the parameters of a bilinear predictor of the same functional form. (2) Comparing to a quadratic predictor shows that reducing the functional form to bilinear by omitting the interactions between microscopic decisions ($\boldsymbol{\alpha}$ parameters) to each other and of macroscopic decisions ($\boldsymbol{\beta}$ parameters) to each other does not cause much degradation. (3) Comparing to a parameter heavy MLP predictor shows that the simple bilinear parametric form does not lack the expressive power required for properly ranking architectures.

Model	Latency (ms)	Top-1 (%)	Total Cost (GPU hours)
MnasNetB1	**39**	74.5	40,000N
TFNAS-B	40	75.0	263N
SPNASNet	41	74.9	288 + 408N
OFA CPU	42	75.7	1200 + 25N
HardCoRe A	40	75.8	400 + 15N
BINAS(40)	40	**76.1**	435 + **8N**
BINAS*(40)	40	**76.5**	435 + 15N
MobileNetV3	**45**	75.2	180N
FBNet	47	75.7	576N
MnasNetA1	55	75.2	40,000N
HardCoRe B	44	76.4	400 + 15N
BINAS(45)	**45**	**76.5**	435 + **8N**
BINAS*(45)	**45**	**77.0**	435 + 15
MobileNetV2	70	76.5	150N
TFNAS-A	60	76.5	263N
HardCoRe C	**50**	77.1	400 + 15N
BINAS(50)	**50**	76.8	435 + **8N**
BINAS*(50)	**50**	**77.6**	435 + 15N
EfficientNetB0	85	77.3	
HardCoRe D	**55**	77.6	400 + 15N
BINAS(55)	**55**	77.7	435 + **8N**
BINAS*(55)	**55**	**77.8**	435 + 15N
FairNAS-C	60	77.0	240N
HardCoRe E	61	**78.0**	400 + 15N
BINAS(60)	**59**	77.8	435 + **8N**
BINAS*(60)	**59**	**78.0**	435 + 15N

Model	Latency (ms)	Top-1 (%)
MobileNetV3	28	75.2
TFNAS-D	30	74.2
HardCoRe A	27	75.7
BINAS(25)	**26**	**76.1**
BINAS*(25)	**26**	**76.6**
MnasNetA1	37	75.2
MnasNetB1	34	74.5
FBNet	41	75.7
SPNASNet	36	74.9
TFNAS-B	44	76.3
TFNAS-C	37	75.2
HardCoRe B	32	**77.3**
BINAS(30)	**31**	76.8
BINAS*(30)	**31**	77.2
TFNAS-A	54	76.9
EfficientNetB0	48	77.3
MobileNetV2	50	76.5
HardCoRe C	41	**77.9**
BINAS(40)	**40**	77.6
BINAS*(40)	**40**	77.7

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. Latency is reported for (Left) Intel Xeon CPU and (Right) NVIDIA P100 GPU with a batch size of 1 and 64 respectively. In Appendix D we compare also size and FLOPS.

5.2.1. Learning Quadratic Accuracy Predictors

One can wonder whether setting the coefficients $Q_{\alpha\beta}$, q_β and r of the bilinear form in section 3.3 according to the estimates in section 3.2 yields the best predictions of the accuracy of architectures. An alternative approach is to learn those coefficients by solving a linear regression:

$$\min_{\tilde{r},\tilde{q}_\alpha,\tilde{q}_\beta,\tilde{Q}_{\alpha\beta}} \sum_{i=1}^{n} ||\mathcal{B}_{\tilde{r},\tilde{q}_\alpha,\tilde{q}_\beta,\tilde{Q}_{\alpha\beta}}(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i) - Acc(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i)||_2^2 \tag{10}$$

$$\mathcal{B}_{\tilde{r},\tilde{q}_\alpha,\tilde{q}_\beta,\tilde{Q}_{\alpha\beta}}(\boldsymbol{\alpha}_i,\beta_i) = \tilde{r} + \boldsymbol{\alpha}_i^T\tilde{q}_\alpha + \boldsymbol{\beta}_i^T\tilde{q}_\beta + \boldsymbol{\alpha}_i^T\tilde{Q}_{\alpha\beta}\boldsymbol{\beta}_i \tag{11}$$

where $\{\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i\}_{i=1}^n$ and $Acc(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i)$ represent n uniformly sampled subnetworks and their measured accuracy, respectively.

One can further unlock the full capacity of a quadratic predictor by coupling of all components and solving the following linear regression problem:

$$\min_{\tilde{r},\tilde{q}_\alpha,\tilde{q}_\beta,\tilde{Q}_{\alpha\beta},\tilde{Q}_\alpha,\tilde{Q}_\beta} \sum_{i=1}^{n} ||\mathcal{Q}_{\tilde{r},\tilde{q}_\alpha,\tilde{q}_\beta,\tilde{Q}_{\alpha\beta},\tilde{Q}_\alpha,\tilde{Q}_\beta}(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i) - Acc(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i)||_2^2$$

$$\mathcal{Q}_{\tilde{r},\tilde{q}_\alpha,\tilde{q}_\beta,\tilde{Q}_{\alpha\beta},\tilde{Q}_\alpha,\tilde{Q}_\beta}(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i) = \mathcal{B}_{\tilde{r},\tilde{q}_\alpha,\tilde{q}_\beta,\tilde{Q}_{\alpha\beta}}(\boldsymbol{\alpha}_i,\beta_i) + \boldsymbol{\alpha}_i^T\tilde{Q}_\alpha\boldsymbol{\alpha}_i + \boldsymbol{\beta}_i^T\tilde{Q}_\beta\boldsymbol{\beta}_i \tag{12}$$

A closed form solution to these problems is derived in appendix F. While effective, this solution requires avoiding memory issues associated with inverting $N^2 \times N^2$ matrix and also reducing overfitting by tuning regularization effects over train-val splits of the data points. The data points for training all the accuracy predictors is composed of subnetworks uniformly sampled from the supernetwork and their corresponding validation accuracy is measured over the same 20% of the Imagenet train set split used in section 3.2.

Figure 7 (Left) presents the Kendall-Tau ranking correlation coefficients and mean square error (MSE), measured over 500 test data points generated uniformly at random in the same way, of different accuracy predictors versus the number of data points corresponding to the number of epochs of the validation set required for obtaining their parameters. It is noticable that the simple bilinear accuracy estimator (section 3.3) is more sample efficient, as its parameters are efficiently estimated according to section 3.2 rather than learned.

5.2.2. Beyond Quadratic Accuracy Predictors

The reader might question the expressiveness of a simple bilinear parametric form and its ability to capture the complexity of architectures. To alleviate such concerns we show in Figure 7 (Left) that the proposed bilinear estimator of section 3.3 matches the performance of the commonly used parameters heavy Multi-Layer-Perceptron (MLP) accuracy predictor Cai et al. (2019); Lu et al. (2020). Moreover, the MLP predictor is more complex and requires extensive hyperparameter tuning, e.g., of the depth, width, learning rate and its scheduling, weight decay, optimizer etc. It is also less efficient, lacks interpretability, and of limited utility as an objective function for NAS (Section 3.5).

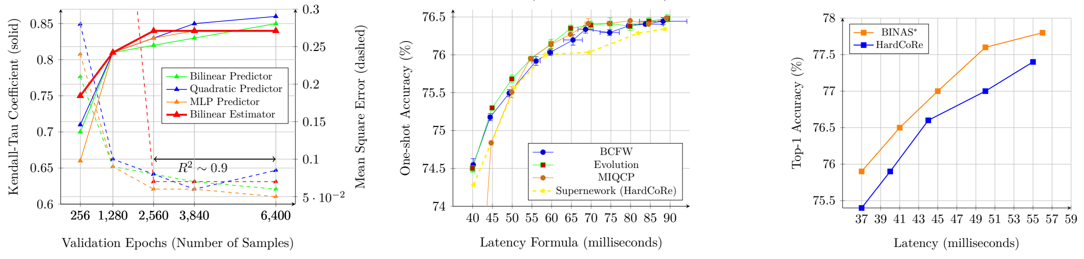

Figure 7: (Left) Performance of predictors vs samples. Ours is comparable to complex alternatives and sample efficient. (Middle) Comparing optimizers for solving the IQCQP over 5 seeds. All surpass optimizing the supernetwork (HardCoreNAS) directly. (Right) Surpassing State-of-the-Art. BINAS generates superior models than HardCoReNAS for the same search space, supernetwork weights and marginal cost of 15 GPU hours.

5.3. Interpretability of the Accuracy Estimator

Given that the accuracy estimator in section 3.3 ranks architectures well, as demonstrated in Figure 1 (Right), this accountability together with the way it is constructed bring insights about the contribution of different design choices to the accuracy, as shown in Figure 8.

The exact way that those insights are deduced is detailed in appendix J.
Deepen later stages: In the left figure $\Delta_b^s - \Delta_{b-1}^s$ are presented for $b = 3, 4$ and $s = 1, \ldots, 5$. This graph shows that increasing the depth of deeper stages is more beneficial than doing so for shallower stages. Showing also the latency cost for adding a block to each stage, we see that there is a strong motivation to make later stages deeper.
Add width and S&E to later stages and shallower blocks: In the middle and right figures, $\Delta_{b,c}^s$ are averaged over different configurations and blocks or stages respectively for showing the contribution of microscopic design choices. Those show that increasing the expansion ratio and adding S&E are more significant in deeper stages and at sooner blocks within each stage.
Prefer width and S&E over bigger kernels: Increasing the kernel size is relatively less significant and is more effective at intermediate stages.

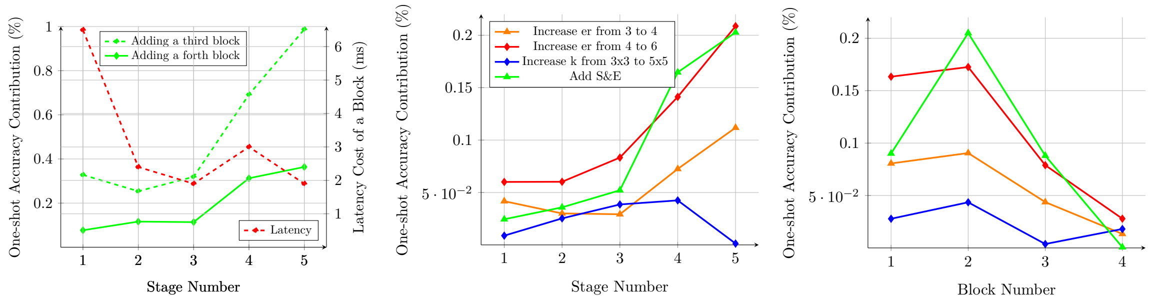

Figure 8: Design choices insights deduced from the accuracy estimator: The contribution of (Left) depth for different stages, (Middle) expansion ration, kernel size and S&E for different stages and (Right) for different blocks within a stage.

5.4. Comparison of Optimization Algorithms

Formulating the NAS problem as IQCQP affords the utilization of a variety of optimization algorithms. Figure 7 (Middle) compares the one-shot accuracy and latency of networks generated by utilizing the algorithms suggested in section 3.5 for solving problem 1 with the bilinear estimator introduced in section 3.3 serving as the objective function. Error bars for both accuracy and latency are presented for 5 different seeds. All algorithms satisfy the latency constraints up to a reasonable error of less than 10%. While all of them surpass the performance of BCSFW Nayman et al. (2021), given as reference, BCFW is superior at low latency, evolutionary search does well over all and MIQCP is superior at high latency. Hence, for practical purposes we apply the three of them for search and take the best one, with negligible computational cost of less than three CPU minutes overall.

6. Conclusion

The problem of resource-aware NAS is formulated as an IQCQP optimization problem. Bilinear constraints express resource requirements and a bilinear accuracy estimator serves as the objective function. This estimator is constructed by measuring the individual contribution of design choices, which makes it intuitive and interpretable. Indeed, its interpretability brings several insights and design rules. Its performance is comparable to complex predictors that are more expensive to acquire and harder to optimize. Efficient optimization algorithms are proposed for solving the resulted IQCQP problem. BINAS is a faster search method, scalable to many devices and requirements, while generating comparable or better architectures than those of other state-of-the-art NAS methods.

References

Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning*, pages 550–559. PMLR, 2018.

Alain Billionnet, Sourour Elloumi, and Amélie Lambert. Exact quadratic convex reformulations of mixed-integer quadratically constrained problems. *Mathematical Programming*, 158(1):235–266, 2016.

Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.

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

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019.

Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1294–1303, 2019.

Xiangxiang Chu, Bo Zhang, Ruijun Xu, and Jixiang Li. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. *arXiv preprint arXiv:1907.01845*, 2019.

Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.

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

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

Giorgio Gallo and Aydin Ülkücü. Bilinear programming: an exact algorithm. *Mathematical Programming*, 12(1):173–194, 1977.

Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *European Conference on Computer Vision*, pages 544–560. Springer, 2020.

Elad Hazan and Haipeng Luo. Variance-reduced and projection-free stochastic optimization. In *International Conference on Machine Learning*, pages 1263–1271. PMLR, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015.

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

Andrew Howard, Ruoming Pang, Hartwig Adam, Quoc V. Le, Mark Sandler, Bo Chen, Weijun Wang, Liang-Chieh Chen, Mingxing Tan, Grace Chu, Vijay Vasudevan, and Yukun Zhu. Searching for mobilenetv3. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 1314–1324. IEEE, 2019. doi: 10.1109/ICCV.2019.00140. URL https://doi.org/10.1109/ICCV.2019.00140.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

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

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *Advances in neural information processing systems*, pages 103–112, 2019.

IBM ILOG CPLEX. Ibm ilog cplex miqcp optimizer. https://www.ibm.com/docs/en/icos/12.7.1.0?topic=smippqt-miqcp-mixed-integer-programs-quadratic-terms-in-constraints.

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

Hans Kellerer, Ulrich Pferschy, and David Pisinger. *The Multiple-Choice Knapsack Problem*, pages 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.

Henry J Kelley. Gradient theory of optimal flight paths. *Ars Journal*, 30(10):947–954, 1960.

Simon Lacoste-Julien, Martin Jaggi, Mark Schmidt, and Patrick Pletscher. Block-coordinate frank-wolfe optimization for structural svms. In *International Conference on Machine Learning*, pages 53–61. PMLR, 2013.

Eric Langford, Neil Schwertman, and Margaret Owens. Is the property of being positively correlated transitive? *The American Statistician*, 55(4):322–325, 2001.

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

Zhichao Lu, Kalyanmoy Deb, Erik Goodman, Wolfgang Banzhaf, and Vishnu Naresh Boddeti. NSGANetV2: Evolutionary multi-objective surrogate-assisted neural architecture search. In *European Conference on Computer Vision (ECCV)*, 2020.

Niv Nayman, Asaf Noy, Tal Ridnik, Itamar Friedman, Rong Jin, and Lihi Zelnik. Xnas: Neural architecture search with expert advice. In *Advances in Neural Information Processing Systems*, pages 1977–1987, 2019.

Niv Nayman, Yonathan Aflalo, Asaf Noy, and Lihi Zelnik. Hardcore-nas: hard constrained differentiable neural architecture search. In *International Conference on Machine Learning*, pages 7979–7990. PMLR, 2021.

Asaf Noy, Niv Nayman, Tal Ridnik, Nadav Zamir, Sivan Doveh, Itamar Friedman, Raja Giryes, and Lihi Zelnik. Asap: Architecture search, anneal and prune. In *International Conference on Artificial Intelligence and Statistics*, pages 493–503. PMLR, 2020.

Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

Binxin Ru, Xingchen Wan, Xiaowen Dong, and Michael Osborne. Interpretable neural architecture search via bayesian optimisation with weisfeiler-lehman kernels. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=j9Rv7qdXjd.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

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

Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-path nas: Designing hardware-efficient convnets in less than 4 hours. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 481–497. Springer, 2019.

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*, pages 2818–2826, 2016.

Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *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*, pages 6105–6114. PMLR, 2019. URL http://proceedings.mlr.press/v97/tan19a.html.

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 Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019.

Ruochen Wang, Minhao Cheng, Xiangning Chen, Xiaocheng Tang, and Cho-Jui Hsieh. Rethinking architecture selection in differentiable nas. *arXiv preprint arXiv:2108.04392*, 2021.

Colin White, Arber Zela, Binxin Ru, Yang Liu, and Frank Hutter. How powerful are performance predictors in neural architecture search? *arXiv preprint arXiv:2104.01177*, 2021.

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 *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 10734–10742. Computer Vision Foundation / IEEE, 2019. doi: 10.1109/CVPR.2019.01099.

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