Block-Level Surrogate Models for Inference Time Estimation in Hardware Aware Neural Architecture Search

Block-Level Surrogate Models for Inference Time Estimation in Hardware-Aware Neural Architecture Search

Kurt Stolle[1,2], Sebastian Vogel[2], Fons van der Sommen[1], and Willem Sanberg[2(✉)]

[1] Eindhoven University of Technology, Eindhoven, The Netherlands
{k.h.w.stolle,fvdsommen}@tue.nl
[2] NXP Semiconductors, Eindhoven, The Netherlands
{sebastian.vogel,willem.sanberg}@nxp.com

Abstract. Hardware-Aware Neural Architecture Search (HA-NAS) is an attractive approach for discovering network architectures that balance task accuracy and deployment efficiency. In an iterative search algorithm, inference time is typically determined at every step by directly profiling architectures on hardware. This imposes limitations on the scalability of search processes because access to specialized devices for profiling is required. As such, the ability to assess inference time without hardware access is an important aspect to enable deep learning on resource-constrained embedded devices. Previous work estimates inference time by summing individual contributions of the architecture's parts. In this work, we propose using block-level inference time estimators to find the network-level inference time. Individual estimators are trained on collected datasets of independently sampled and profiled architecture block instances. Our experiments on isolated blocks commonly found in classification architectures show that gradient boosted decision trees serve as an accurate surrogate for inference time. More specifically, their Spearman correlation coefficient exceeds 0.98 on all tested platforms. When such blocks are connected in sequence, the sum of all block estimations correlates with the measured network inference time, having Spearman correlation coefficients above 0.71 on evaluated CPUs and an accelerator platform. Furthermore, we demonstrate the applicability of our Surrogate Model (SM) methodology in its intended HA-NAS context. To this end, we evaluate and compare two HA-NAS processes: one that relies on profiling via hardware-in-the-loop and one that leverages block-level surrogate models. We find that both processes yield similar Pareto-optimal architectures. This shows that our method facilitates a similar task-performance outcome without relying on hardware access for profiling during architecture search.

Keywords: AutoML · Inference time estimation · Neural network design

M.-R. Amini et al. (Eds.): ECML PKDD 2022, LNAI 13717, pp. 463–479, 2023.
https://doi.org/10.1007/978-3-031-26419-1_28

1 Introduction

Neural networks consistently achieve competitive results in a wide variety of machine learning contexts. It is thus not surprising that both academia and industry address challenging tasks in multiple domains with neural networks. Furthermore, with cloud services offering specialized neural network training infrastructure, a network can be trained and deployed with minimal operational investment. Since both environment perception and the interpretation of sensor data on-device benefit from the use of deep learning, neural networks are deployed outside data centers with increasing frequency. This shift to edge devices brings a new challenge: hardware cost of deployed neural networks, such as inference latency (i.e. execution time), memory usage, bandwidth utilization, etc. These hardware metrics must be reduced such that neural networks can effectively be executed on more computationally and power constrained devices [23].

Designing machine learning models targeting multiple objectives is a tedious task, which traditionally entails many hours of manual tuning by human experts. As a consequence, the adoption of hardware-optimal neural network design for practical innovations is often deferred due to domain knowledge scarcity. As a response to an increasing demand for task-specific solutions, automated machine learning (AutoML) has emerged to successfully address this limitation [30]. In the case of neural networks, the field of Neural Architecture Search (NAS) studies AutoML that yields optimal architectures. Latest state-of-the-art performance on a number of challenging computer vision benchmarks has been achieved with neural networks found via architecture search [31].

One of the toughest challenges for realizing NAS in practice is the amount of computational resources and power required per search. An important reason for this, is that many NAS methods estimate the accuracy of candidate networks by training them for several epochs [12,15]. Recent research has revealed that this can be alleviated by estimating the task performance directly from the architecture using a surrogate model. As a consequence, the amount of training required is drastically reduced [2,4].

As indicated previously, there is increasing demand for models that meet challenging hardware deployment cost requirements, such as inference time, memory requirements, or power consumption. This is addressed by hardware-aware neural architecture search (HA-NAS) which aims to optimize for both task-performance and hardware costs [3]. In this work, we consider the co-optimization of both classification accuracy and execution time on hardware. The hardware-aware search strategy determines architectures that perform optimally on both metrics. In the remainder of this paper, we use 'inference time', 'latency', and 'execution time' interchangeably.

Figure 1 shows a diagram of an iterative HA-NAS process. The latency of an architecture must be assessed at every step in the search process. Three categories of assessment methods are commonly found in the literature. Firstly, hardware-in-the-loop (HIL) methods profile a given architecture ad-hoc on the targeted hardware whenever a new architecture is emitted [7,24]. Secondly, lookup table (LUT) methods query latencies of parts of a neural networks based

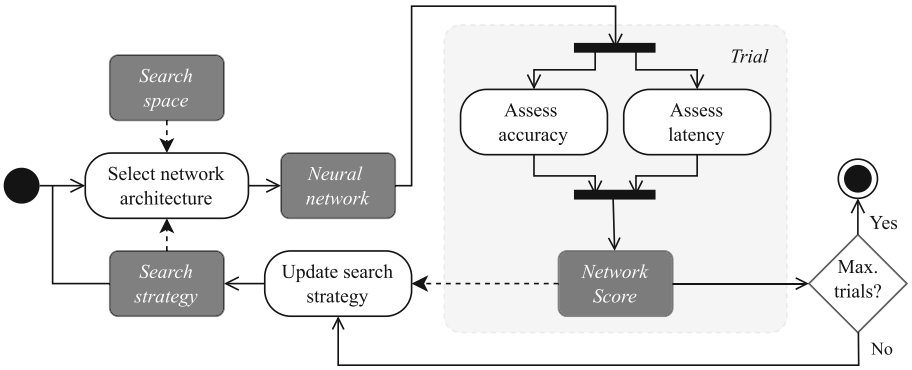


Fig. 1. UML activity diagram of hardware-aware neural architecture search. Neural networks are selected from the search space using a search strategy. The combined task and hardware performance scores are used to determine the next trial network. This work proposes a surrogate-model-based latency assessment.

on their building blocks [1,6,28,29,32,34]. The LUT contains an entry for each architecture block configuration in the search space. These part latencies are then combined, usually via summation, to yield the overall architecture latency. Thirdly, surrogate model (SM) methods define a prediction model to infer the approximate latency of a network [2,4,14,33].

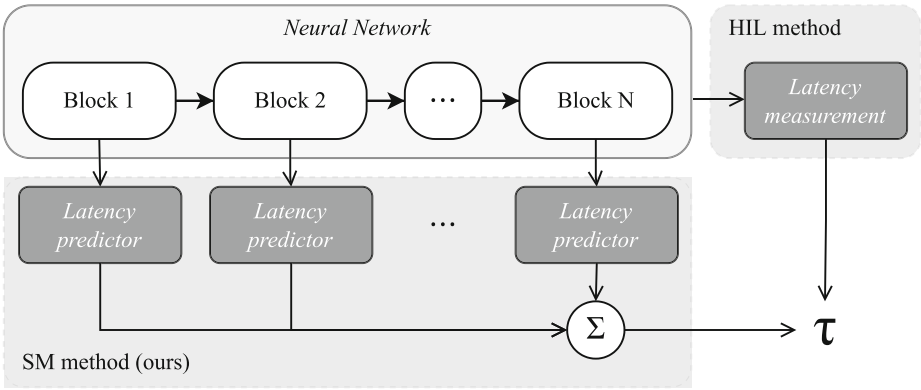


Fig. 2. Schematic showing the assessment methodolgy of full-network latency τ using HIL profiling and our proposed block-level based estimate.

We aim to combine various machine learning methodologies to engineer a latency estimation framework for block-based convolutional neural networks (CNN) in order to overcome the limitations of HIL and lookup table based latency assessment systems. As such, our approach can be categorized as an SM

method. Figure 2 shows an overview of how our proposed method estimates the latency τ of a neural network architecture using a set of dedicated block-level predictors. The true execution time can be determined by HIL-measurements. In broad terms, we train and evaluate various regression models on a generated dataset of individually profiled architecture block instances. The architecture block configurations together with their measured inference time represent the respective input and ground truth in the training data. A successful prediction model will learn the influence of the block configuration parameters on the block latency. This way the model may generalize from trained block-configurations to configurations that were not part of the training set. This overcomes the limitations of lookup tables where every possible configuration of an architecture building block would need to be profiled and stored before being used in HA-NAS. As such, our proposed block-level surrogate model (BLSM) greatly increases the cardinality of a search space in a HA-NAS algorithm without requiring hardware access at search-time.

Our key contributions can be summarized as:

- We introduce a block-level surrogate-model methodology abbreviated BLSM that uses trained inference time estimators to overcome the need for hardware access during HA-NAS while preserving the flexibility benefits of HIL-methodologies.
- We propose a novel definition of blocks using a bijective relation between instantiation parameters and a computational sub-graph to make the BLSM pipeline generalize to any block type that may be uniquely described by a structured collection of constructor variables.
- We assess the feasibility of block-level latency estimates for guiding HA-NAS in comparison to HIL-guided search.

2 Related Work

The use of machine learning models that predict the task performance of neural network architectures has recently become a relevant aspect for improving the search time of NAS methods. Baker *et al.* were among the first to explore the use of an SM for accuracy prediction based on support vector regression in combination with an early stopping scheme [2]. Similarly, Moons *et al.* integrate an accuracy predictor in their pipeline for rapidly searching neural networks on various hardware targets [24]. They employ HIL to test whether an architecture is feasible on a target device. Our SM method aims to be a drop-in replacement for such a HIL approach. This would allow a fully prediction-based optimization scheme when optimizing for both accuracy and latency.

Different strategies have been employed to estimate hardware metrics. For example, Dong *et al.* define an analytical model to estimate DNN latency on an FPGA platform [10]. This analytical strategy requires detailed understanding of low-level execution of DNN blocks on the target hardware, which is not feasible for a wide variety of deployment scenarios, e.g., if specialized black-box inference engines are involved. Alternatively, Gupta *et al.* implement an

SM that uses hardware virtualization. Such cycle-accurate simulations can accurately predict the performance of a hardware target. However, cycle-accurate simulations are often prohibitive in HA-NAS for assessing execution time due to their significant computational effort. Additionally, profiling a neural network via simulation requires cycle-accurate emulation of the platform. Such software may not be available for the target hardware. Our method overcomes the limitations of simulation and analytical models by predicting the latency through a machine-learning-based surrogate model trained on a-priori profiled data of the target hardware platform.

Bouzidi *et al.* compare a variety of modeling algorithms to predict the latency of CNN-architectures on edge GPU platforms [4]. Whereas the method proposed in their work aims to make a single prediction on the network level, our approach uses block-level latency predictions to infer the network latency.

HELP and BRP-NAS are similar approaches for introducing predictor-based latency estimators in NAS [11,21]. Especially the focus of HELP lies on generalizing the latency estimations towards multiple HW-targets. However, the individual neural networks for latency estimation stem from one search space only. Our work focuses on block-level latency estimation and therefore enables latency estimation in arbitrary search spaces derived form a composition of blocks.

We evaluate our method as a drop-in module for the *AutoKeras* framework proposed in [19], which renders the NAS process into a hyperparameter optimization context. In their system, the authors programatically define a parameterized template network called the 'hypermodel', which we can represent as a mapping from a set of network instantiation parameters to a network architecture. By means of hyperparameter optimization, trials can be performed until an optimal configuration is found [19]. By contrast, the canonical term 'search space' is used for the remainder of this paper to refer to the set of all possible network instances that are in the range of a specific hypermodel. To clarify, the range of a hypermodel entails all architectures that can be defined on the cross product of the domain of each instantiation parameter.

HW-NAS-Bench is a recent dataset that includes latency measurements of networks for benchmarking *hardware-aware* NAS approaches [22]. Our work is based on block-level latency estimates. Unfortunately, HW-NAS-Bench only provides the latency of entire networks and not of their composing blocks. Therefore, our approach cannot be assessed on the HW-NAS-Bench dataset.

3 Method

3.1 Block Instances and Parameters

Before proceeding to formalize our models, let us first introduce the concept of block-based neural networks. This work has a focus on image classification networks that consist of sequentially connected block instances.

Definition 1. *A block instance is a sub-graph of a neural network that is the specific realization from the space of all block variants.*

Table 1. Evaluated block types

Block name	Reference implementation	Parameters cardinality[(i)]
`MBConv`	MobileNet convolutional block [18]	$3 + 5$
`Res V1`	ResNet original implementation [16]	$3 + 4$
`Res V2`	ResNet with identity mappings [17]	$3 + 4$

[i] Cardinality includes three input dimensions plus the number of searchable block configuration hyperparameters.

The networks investigated in this paper can be represented as a list of *parameters* that unambiguously define a series of block instances. Inversely, these parameters can be uniquely inferred from the network graph by inspecting the block instances it contains.

Definition 2. *Block parameters are a collection of mixed-domain values that uniquely map to a block instance.*

The bijective properties of the mapping from parameters to instances enables constructing a dataset of block parameters together with the measured inference latency of the respective instance for training a block-level latency predictor. Additionally, the block parameters of each block in a candidate neural network can be determined during search-time by parsing the computational graph of the full network. By design, this effectively isolates the training of latency predictors from the search process. The blocks defined in this work are parameterized versions of blocks commonly found in literature. Figure 1 summarizes the block types evaluated in this paper.

As is common practice in HA-NAS methods [3,6,26,27] and also experimentally confirmed [27], we assume that network latency can be sufficiently approximated by the sum of stacked block-instance latencies. As such, blocks can be profiled individually and mixed-and-matched to yield new networks that retain the summed-latency property. Thus, summing latency predictions for each block of a neural network yields an estimate of the entire architecture inference time. We hypothesize that there exist regression models that can predict the block-level latency from the block serialization. This requires a predictor model for each pair of block families and hardware targets, which can be trained on profiling data of single blocks.

3.2 Predictor Model

Having defined the relation between blocks and the full-network latency, the estimator model can now be formalized. Let $\tau_{\mathbf{c}} \in \mathbb{R}_+$ be the measured latency of a block B with parameters $\mathbf{c} \in \mathcal{C}_\mathrm{B}$, where $\mathcal{C}_\mathrm{B}$ represents the set of all possible block configurations. The objective of our block-level estimators is to learn the transformation

$$\hat{\tau} : \mathcal{C}_\mathrm{B} \rightarrow \mathbb{R}_+, \tag{1}$$

such that $\hat{\tau}(\mathbf{c})$ predicts the measured latency. This is identified as a regression problem with error function

$$\zeta = \hat{\tau}(\mathbf{c}) - \tau_{\mathbf{c}}. \tag{2}$$

We propose to use a separate estimator $\hat{\tau}(\cdot)$ for each block type and hardware platform. This enables to two degrees of freedom for designing inference time surrogate models. First, each block predictor can leverage the machine learning algorithm best suited for the domain of instantiation parameters that it defines. Second, it is not required to train predictors that generalize across all hardware-platforms. Instead, predictors may be fine-tuned to fit specific hardware characteristics.

This paper evaluates four estimator models, namely Linear, Random Forest, Boosted Trees, and Dense NAS. These different estimator models should address the mixed-domain characteristics of block parameters and are further explained in the following.

Linear Regression. This model is best suited for configuration spaces that consist of continuous independent variables. We make use of an exponentially activated linear regression model

$$\hat{\tau}(\mathbf{c}) = \exp(\mathbf{W}\mathbf{c} + b) \tag{3}$$

with weighting matrix $\mathbf{W}$ and bias b. We iteratively approximate the weighting matrix using gradient descent. Hyperparameters are the learning rate (0.01) and batch size (64), which were determined using 20 trials of grid search.

Decision Forest Regression. These models are expected to achieve high performance when the block configuration consists of mostly categorical parameters or when the input values are from mixed domains. In a decision forest model, the (regression) output is given by taking the mean output of a decision tree ensemble. We explore two flavors of decision-forest construction algorithms. First, the Random Forest predictor uses the method described in [5], which adds uncorrelated trees to the forest that minimize the prediction error. Second, the Boosted Treespredictor uses the gradient boosting algorithm [13], which iteratively reduces the error of the forest by adding trees that minimize the error of prediction as a product of the learning rate. For both models, hyperparameters were configured as suggested in the reference implementation [8].

Deep Neural Network Regression. The fourth estimator method under consideration is a deep neural network. We exploit a NAS algorithm to find an optimal fully connected deep neural network, tailored towards each block and hardware platform. Models of this class are best suited to inputs where each element is from the same domain. Additionally, inputs that have a high degree of interdependent relations are likely to benefit from deep neural networks. The network is trained using gradient descent, with the learning rate tuned by the NAS algorithm.

3.3 Experimental Set-Up

The methodology proposed in this paper is evaluated on four platforms. These platforms cover a wide range of hardware architectures to illustrate the extent to which our methodology is applicable. More specifically, we employ a high-performance CPU for high-end compute (CPU Cloud), a low-power CPU that is typically used in energy-constrained edge platforms (CPU Edge), a general-purpose GPU designed for parallel processing (GPU), and a specialized parallel compute platform (ASIC)[1]. For each platform, latency is measured in a profiling experiment consisting of 10 000 invocations on isolated block instances with random data and batch size 1. Note that in edge devices, achieving low latency is often more important than realizing high throughput. We therefore choose a batch size of 1. A number of external factors may affect the measured latency, such as processes executed in parallel on the targeted hardware and general profiling inaccuracies. To mitigate these effects in our approach, we rely only on the minimal measured latency of the block profiling dataset. This represents a feasible best-case execution time and is more robust than, e.g., a mean value, since we observed that inference measurements typically follow an exponential distribution.

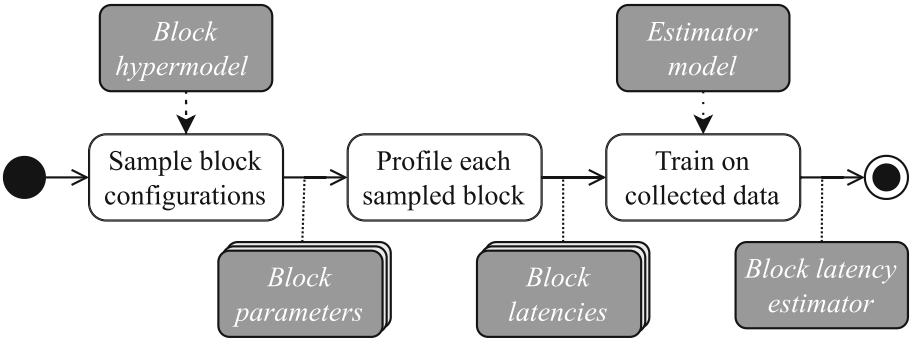


Fig. 3. UML activity diagram showing the end-to-end process of sampling configurations of a block, profiling these samples, and fitting a predictor model on this data. This yields a latency predictor that is specifically designed for a single block type and the hardware platform it was profiled on.

Figure 3 shows the process of sampling configurations of a particular block, profiling these block instances on hardware and fitting a regression model. This yields a block latency predictor as can be used in Fig. 2.

We evaluate out proposed block-level surrogate models (BLSM) for latency in a HA-NAS on a small-scale image classification task. For this, we use the

[1] Note that the experiments are meant to demonstrate our methodology. The experiments are not intended to benchmark specific hardware and deployment toolchains, hence those details are left out.

CIFAR-10 dataset [20]. This dataset covers 60 000 colored images, each having a width and height of 32 pixels. The objective is to classify each image into one of ten classes. The classification accuracy is defined as the percentage of correctly predicted classes over a test set of 10 000 samples.

3.4 Applicability of Block-Level Surrogate Models in HA-NAS

To assess the applicability of a block-level surrogate model for inference time, we compare the distribution of found neural networks of a HIL-based search with the distribution of architectures generated by a search that relies on our BLSM. If the found Pareto-optimal neural architectures are sampled in similar accuracy-latency regimes in both setups, we consider the use of a BLSM in HA-NAS is *feasible* for a block type and hardware platform. As a result, we can compare the optimal networks from a session guided by a predictor with the optimal networks found when HIL-measurements were used.

For the purpose of evaluating our proposed BLSM, we sample 200 architectures with random block configurations and assess the predicted latency with *true* latency measured in a HIL-setup. Note, random sampling of architectures reflects an unguided search, i.e. each trial is chosen independently of any previously evaluated accuracy or latency in a search process.

In contrast, a HA-NAS with a Bayesian Optimization search strategy samples architectures for training and profiling based on previous trials' accuracy and latency. In order to render the Bayesian Optimization as a hardware-aware multi-objective optimization, we use a scalarization paradigm [25]. To this end, we propose using the balanced sum of the task-related objective (accuracy) and the hardware performance metric (inference time). Specifically, we investigate the multi-objective-optimization of classification accuracy α on CIFAR-10 and inference latency τ on target hardware platforms. Thus, we define the following multi-objective scalarization to derive a *balanced accuracy-latency score*

$$\mathrm{BALS} = \frac{\alpha}{\alpha_{\mathrm{ref}}} + \frac{\log(\tau - \tau_{\mathrm{ref}})}{\log \tau_{\mathrm{ref}}} - 1, \tag{4}$$

where constants α_{ref} and τ_{ref} can be tuned according to the importance of each objective. For experiments, we employ an adapted version of the Bayesian Optimization search algorithm from the AutoKeras framework [19].

Tabel 2 summarizes the hyperparameters of the HA-NAS benchmark. We use 20 epochs of training as a proxy for the final validation accuracy in accordance with NAS-bench-201 [9].

Table 2. HA-NAS benchmark hyperparameters

Hyperparameter	Value
Primary objective	CIFAR-10 classification accuracy
Input image dimensions	$32 \times 32 \times 3$
Training epochs to estimate accuracy score	20
Batch size	64
Optimizer	Adam
Learning rate	0.01
Reference accuracy α_{ref}	1.0
Reference latency τ_{ref}	100 ms

4 Results and Discussion

Our latency predictors are trained on a generated dataset of block parameters and their corresponding measured latency on a specific target platform. To assess performance and compare results, each block-hardware-dataset is split into a training, validation, and test set. The training set is used to fit the model, while the hyperparameters of this model are tuned to maximize accuracy on the validation split. Finally, the test split is used to quantitatively assess the quality of predictors with respect to measured values and in relation to other predictors.

4.1 Block-Level Surrogate Model Performance

Figure 4 shows the predicted latency versus the measured latency. The amount of deviation from the $x = y$ line is a proxy for the quality of the predictor. From this visualization, it is evident that the Boosted Trees estimator appears to yield the least amount of predictive error compared to the other estimator models. Overall, the GPU hardware is most difficult to predict. This could be explained by inaccuracies in profiling at the small-scale range of GPU inference times. In order for a NAS algorithm to be guided towards optimal solutions using a predictor, the prediction and measured values ideally have a monotonic relation. To assess the strength of all trained predictors in this regard, Spearman's ρ is calculated, and results are summarized in Table 3. All predictor methods produce highly correlated results.

4.2 Comparing Recall of Optimal Networks Under Prediction

Previously, we computed the Spearman ρ to quantitatively assess the monotonic relation between block-level latency estimates and the measured latency of an isolated block. In this section, we analyze both the monotonic relation strength between network-level latency estimates using summed block-level SM estimators and mean measured latency. For our analysis, we use a set of 200 networks

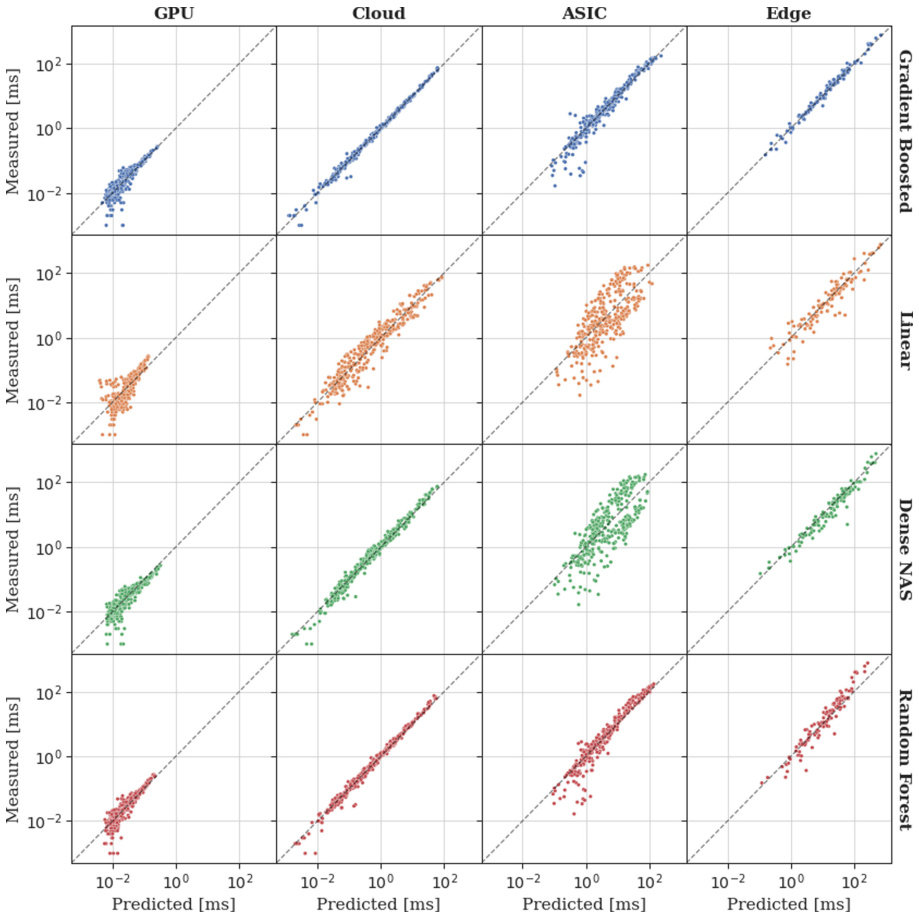


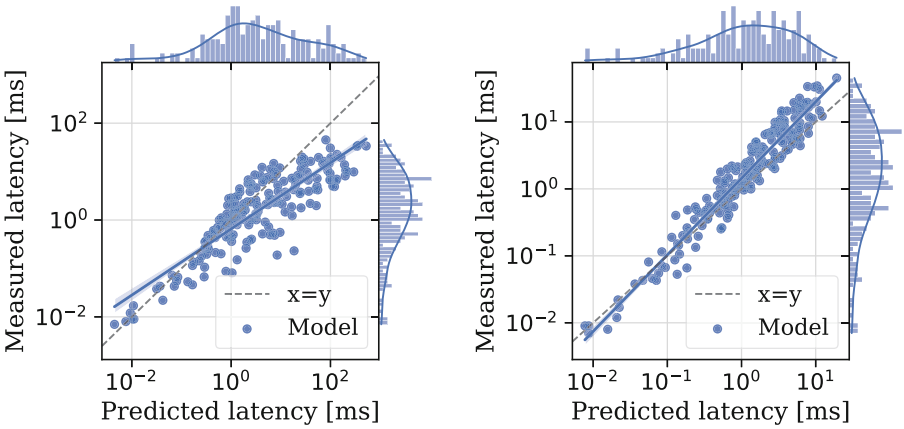
Fig. 4. Predicted latency versus measured latency for all MobileNet convolutional block estimators in isolation on different hardware platforms. Every point corresponds to a configuration of the ResBlock1 in the set of configuration samples not used in the training process.

randomly sampled from the search space, which consists of sequentially placed instances of each block type. While a block-level latency estimate may produce network-level latencies with a large degree of error, we expect a guided NAS algorithm to still find the set of Pareto-optimal networks when errors are proportionally incorrect. Figure 5 visually illustrates the relation between measured versus BLSM-predicted inference time on the network-level. Table 4 summarizes the resulting metrics for each predictor on the set of randomly sampled networks using the ASIC platform. On average, the Boosted Trees predictor produces the largest correlation coefficients, closely followed by the Random Forest predictor. This is expected, as the block-level Spearman's ρ correlation is similarly large when evaluated on isolated blocks (Table 3).

Table 3. Spearman's correlation of block-level latency estimates to measured latencies

Block	Platform	Dense NAS	Gradient boosted	Linear	Random forest
MBConvBlock	ASIC	0.79	0.98	0.74	0.97
	Cloud	0.99	1.00	0.96	0.99
	Edge	0.97	0.99	0.94	0.97
	GPU	0.88	0.89	0.78	0.90
ResBlock1	ASIC	n/a	n/a	n/a	n/a
	Cloud	0.99	1.00	0.97	0.99
	Edge	0.99	1.00	0.96	0.99
	GPU	0.91	0.98	0.93	0.97
ResBlock2	ASIC	n/a	n/a	n/a	n/a
	Cloud	1.00	1.00	0.97	0.99
	Edge	n/a	n/a	n/a	n/a
	GPU	0.95	0.99	0.94	0.98

$p < 0.001$ for all r and ρ correlation values.



(a) Linear (b) Boosted Trees

Fig. 5. Normal probability (center axis) and KDE (marginal axes) plots of measured versus predicted latency on full neural network architecture scale. Networks (blue dots) were discovered via random search consisting of stacked MBConvBlock(*Colorfigureonline*) profiled on the CPU Cloud platform. While distributions estimated on both axes appear similar, deviation from the dotted line shows the error of prediction accumulates exponentially.

4.3 Effects of Prediction on Guided Search Results

A quantitative comparison between search results becomes more involved when a strategy selects subsequent trials based on the previous trial's results, as is the case in guided search. Figure 6 shows networks from a search guided respectively

Table 4. Evaluation metrics of BLSM predictors for MBConvBlock based-networks.

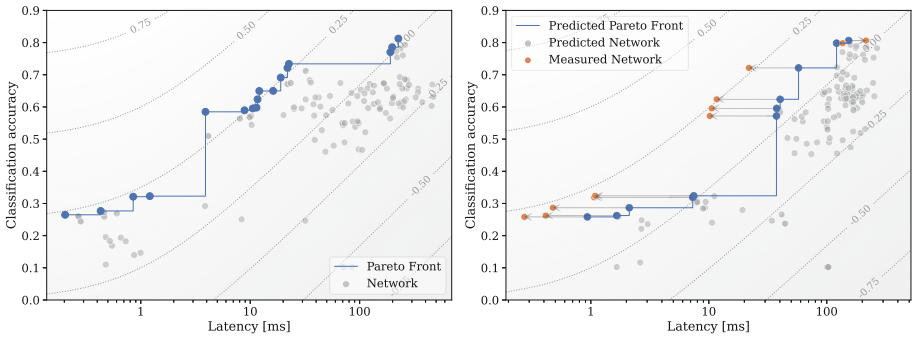
Platform	Predictor	r	ρ
Cloud CPU	Linear	0.86	0.83
	Random forest	0.96	0.95
	Gradient boosted trees	**0.97**	**0.96**
	Dense NAS	0.96	0.95
Edge CPU	Linear	0.83	0.79
	Random forest	0.87	0.85
	Gradient boosted trees	**0.91**	**0.89**
	Dense NAS	0.87	0.8
GPU	Linear	0.57	0.54
	Random forest	0.45	0.41
	Gradient boosted trees	**0.58**	**0.57**
	Dense NAS	0.53	0.48
ASIC	Linear	0.80	0.60
	Random forest	**0.96**	**0.71**
	Gradient boosted trees	0.94	0.70
	Dense NAS	0.81	0.70

$p < 0.001$ for all r and ρ correlation values.

Table 5. Distance between SM and HIL guided sets of MBConvBlock networks.

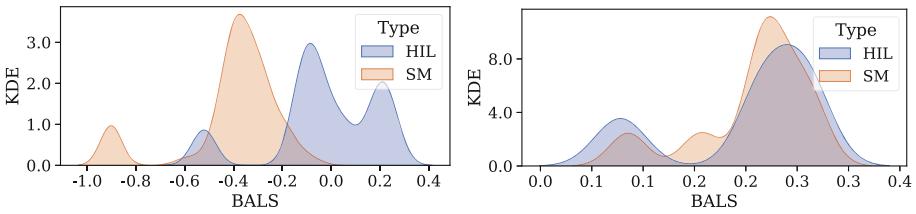
Platform	Predictor	W_{overall}	W_{optimal}	HNQ
Cloud CPU	Linear	0.1052	0.3118	2.9
Cloud CPU	Boosted Trees	0.1278	0.0198	**0.2**
ASIC	Linear	0.2032	0.1574	0.8
ASIC	Boosted Trees	0.0923	0.0142	**0.2**

by HIL and SM assessments with the Bayesian Optimization strategy in the accuracy versus latency plane. To gain insight into the score of networks selected by each search, an estimate of the HA-score distribution of networks is shown in Fig. 7. We compute the Wasserstein distance W of HA-score distributions as a measure of similarity between the networks sampled in the HIL-setup and in the SM-setup. We report both the Wasserstein distance W_{overall} between all sampled networks and W_{optimal} between the found Pareto-optimal networks. A small value of W_{pareto} indicates that the BLSM-based HA-NAS process is *feasible*, because the SM set of optimal networks is similar to the HIL set of optimal networks. The range of the Wasserstein distance metric does not give an inherent qualitative insight, but requires comparison against a baseline distance to yield an interpretable result. We use the overall distance between SM and HIL results as a baseline for each qualitative assessment. The search process can



(a) HIL (b) SM (Boosted Trees)

Fig. 6. Guided HA-NAS results under different latency assessment sources (HIL or SM) visualized in an accuracy versus latency scatter plot.



(a) All sampled networks (b) Pareto-optimal networks only

Fig. 7. KDE plot of HA-scores found via HIL and SM (Boosted Trees) Bayesian Optimization search. A stack of MBConvBlock models on the ASIC platform was used. Visually, the distances between both sets is much smaller, indicating the search strategy yields similar networks.

thus be asserted as *feasible* when the distance between optimal results is less than the overall distance of networks trialed during search. Therefore, we define the *HA-NAS quality ratio*

$$\text{HNQ} = \frac{W_{\text{optimal}}}{W_{\text{overall}}} \in \mathbb{R}_+, \tag{5}$$

and the *necessary condition for feasibility* of the search process,

$$\text{HNQ} < 1. \tag{6}$$

Table 5 summarizes the resulting metrics for each search experiment. Results indicate that the Boosted Trees BLSM is the most feasible on the CPU Cloud and ASIC platform with HNQ = 0.2. Additionally, the linear regression SM does not meet the necessary condition for feasibility Eq. (6) on the Cloud CPU platform and has a relatively large NHQ ratio on the ASIC platform.

While the measured latencies of networks found via SM-based guided search are close to that of networks found via HIL-based search, there is an offset

between predicted and measured latencies in BLSM-based search that is likely due to the accumulation of predictive errors in the search strategy algorithm. An explanation for the ability of the SM-based search to find optimal networks despite a large degree of error between the predicted and measured latency follows from the random search experiment. Namely, the monotonic relation between predictions and measured latencies on network-level causes the search strategy to propose networks in the correct relative order, albeit with a large absolute error. These results demonstrate the utility of block-level latency predictors in guided search.

5 Conclusion

This paper presents a block-level surrogate model for inference latency prediction and shows its applicability by integrating it in a HA-NAS method. Our BLSM overcomes the need for hardware access and latency lookup tables during a neural architecture search process, thereby greatly improving both flexibility and scalability of HA-NAS applications. As a key design choice, our BLSM operates on block-parameters, which by definition are available for any block, current or new. As a result, our method generalizes over many architectures and hardware platforms using a simple training procedure.

Predictors are robust to varying block domains by choosing an appropriate regression model from a diverse set of model types. Experiments on a representative set of target platforms and block types have validated our methodology, with three key findings. First, results demonstrate that predictors based on decision trees, Random Forest and Boosted Trees, perform optimally for the evaluated block types. Second, when blocks are placed in sequence, the sum of block predictions correlates with the measured latency of the full network, achieving Spearman coefficients of 0.96, 0.89, 0.57 and 0.71 on the Cloud CPU, Edge CPU, GPU, and ASIC platforms respectively. Third, NAS algorithms guided by our proposed predictor were able to find Pareto-optimal neural networks with similar HA-score to those found via a HIL-based process.

Combined, the experiments confirm that our BLSM enables HA-NAS to scale out by removing the burden of hardware access during search. More specifically, it paves the way to deploy HA-NAS in a distributed fashion, by allowing the latencies of multiple block-based architectures to be assessed anywhere and in parallel. Ultimately, this facilitates large-scale automation of designing efficient and effective neural networks for a wide variety of applications in resource constrained edge devices.

References

1. Abdelfattah, M.S., Dudziak, Ł., Chau, T., Lee, R., Kim, H., Lane, N.D.: Best of both worlds: Automl codesign of a cnn and its hardware accelerator. In: ACM/IEEE DAC. IEEE (2020)

2. Baker, B., Gupta, O., Raskar, R., Naik, N.: Accelerating neural architecture search using performance prediction. ICLR Workshop (2018)
3. Benmeziane, H., Maghraoui, K.E., Ouarnoughi, H., Niar, S., Wistuba, M., Wang, N.: A comprehensive survey on hardware-aware neural architecture search. arXiv preprint arXiv:2101.09336 (2021)
4. Bouzidi, H., Ouarnoughi, H., Niar, S., Cadi, A.A.E.: Performance prediction for convolutional neural networks on edge gpus. In: ACM ICCF, p. 54–62 (2021)
5. Breiman, L.: Random forests. In: Machine Learning. vol. 45, pp. 5–32. Springer Science and Business Media LLC (2001). https://doi.org/10.1023/a:1010933404324
6. Cai, H., Gan, C., Wang, T., Zhang, Z., Han, S.: Once for all: Train one network and specialize it for efficient deployment. In: ICLR (2020)
7. Cai, H., Zhu, L., Han, S.: ProxylessNAS: Direct neural architecture search on target task and hardware. In: ICLR (2019)
8. Dillon, J.V., et al.: Tensorflow distributions. arXiv preprint arXiv:1711.10604 (2017)
9. Dong, X., Yang, Y.: NAS-Bench-201: Extending the scope of reproducible neural architecture search. In: ICLR (2019)
10. Dong, Z., Gao, Y., Huang, Q., Wawrzynek, J., So, H.K., Keutzer, K.: HAO: Hardware-aware neural architecture optimization for efficient inference. In: IEEE FCCM, pp. 50–59. IEEE (2021)
11. Dudziak, L., Chau, T., Abdelfattah, M., Lee, R., Kim, H., Lane, N.: BRP-NAS: prediction-based NAS using GCNs. NeurIPS **33**, 10480–10490 (2020)
12. Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: a survey. J. Mach. Learn. Res. **20**(1), 1997–2017 (2019)
13. Friedman, J.H.: Greedy function approximation: a gradient boosting machine. Ann. Statist. **29**(5), 1189–1232 (2001)
14. Gupta, S., Akin, B.: Accelerator-aware neural network design using automl. arXiv preprint arXiv:2003.02838 (2020)
15. Guyon, I., et al.: Analysis of the automl challenge series, pp. 191–236 2015–2018
16. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: IEEE/CVF CVPR, pp. 770–778 (2016)
17. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9908, pp. 630–645. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46493-0_38
18. Howard, A., et al.: Searching for mobilenetv3. In: IEEE/CVF CVPR, pp. 1314–1324 (2019)
19. Jin, H., Song, Q., Hu, X.: Auto-keras: An efficient neural architecture search system. In: ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1946–1956. ACM (2019)
20. Krizhevsky, A.: Learning multiple layers of features from tiny images (2009)
21. Lee, H., Lee, S., Chong, S., Hwang, S.J.: HELP: Hardware-adaptive efficient latency prediction for NAS via meta-learning. In: Advances in Neural Information Processing Systems (NeurIPS) (2021)
22. Li, C., et al.: HW-NAS-Bench: Hardware-aware neural architecture search benchmark. In: ICLR (2021)
23. Li, W., Liewig, M.: A survey of ai accelerators for edge environment. In: Rocha, Á., Adeli, H., Reis, L.P., Costanzo, S., Orovic, I., Moreira, F. (eds.) WorldCIST 2020. AISC, vol. 1160, pp. 35–44. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45691-7_4
24. Moons, B., et al.: Distilling optimal neural networks: Rapid search in diverse spaces. In: IEEE/CVF CVPR, pp. 12229–12238 (2021)

25. Roijers, D.M., Zintgraf, L.M., Nowé, A.: Interactive Thompson sampling for multi-objective multi-armed bandits. In: Rothe, J. (ed.) ADT 2017. LNCS (LNAI), vol. 10576, pp. 18–34. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67504-6_2
26. Shaw, A., Hunter, D., Iandola, F., Sidhu, S.: Squeezenas: Fast neural architecture search for faster semantic segmentation. arXiv preprint arXiv:1908.01748 (2019)
27. Stamoulis, D., et al.: Single-path NAS: Device-aware efficient convnet design. In: Joint Workshop on On-Device Machine Learning & Compact Deep Neural Network Representations with Industrial Applications (ODML-CDNNRIA) at ICML (2019)
28. Tan, M., Chen, B., Pang, R., Vasudevan, V., Le, Q.V.: Mnasnet: Platform-aware neural architecture search for mobile. IEEE/CVF CVPR, pp. 2815–2823 (2019)
29. Tsai, H., Ooi, J., Ferng, C.S., Chung, H.W., Riesa, J.: Finding fast transformers: One-shot neural architecture search by component composition. arXiv preprint arXiv:2008.06808 (2020)
30. Vanschoren, J.: Meta-Learning, pp. 35–61. Springer International Publishing (2019)
31. Wistuba, M., Rawat, A., Pedapati, T.: A survey on neural architecture search. arXiv preprint arXiv:1905.01392 (2019)
32. Wu, B., et al.: FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search. In: IEEE/CVF CVPR, pp. 10726–10734 (2019)
33. Wu, J., et al.: Weak NAS predictors are all you need. arXiv preprint arXiv:2102.10490 (2021)
34. Yang, T.-J., et al.: NetAdapt: platform-aware neural network adaptation for mobile applications. In: Ferrari, V., Hebert, M., Sminchisescu, C., Weiss, Y. (eds.) ECCV 2018. LNCS, vol. 11214, pp. 289–304. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01249-6_18