Latency Prediction for Convolutional Neural Networks

Yuxin Yue
yyue1@ualberta.ca

December 22, 2021

Abstract

We analyze commonly used latency estimation methods in neural architecture search but find most of them don't perform well. In this paper, we implement the GCN-based method [1] to obtain accurate and fast latency prediction. We conduct experiments on multiple devices and achieve latency predictors with competitive performance in both accuracy and speed. We also verify that the GCN-based predictors perform much better than linear regression and XGBoost. The code is now available at https://github.com/YYue000/LatencyPrediction.

1 Introduction

With more and more neural network applications on real-world devices, accurate inference latency plays an important role in determining the performance of the neural architecture. Therefore, more recent works [2, 3] prefer to utilize accurate latency instead of proxy metrics like FLOPs and parameters to conduct neural architecture search. It's intuitive to utilize the inference time for reference.

However, the ground truth values of latency on real-world devices are difficult and expensive to obtain. Some models run very slowly on some mobile devices with relatively poor computation resources, and the total inference time is too long. In addition, the pre-processing and post-processing processes would also occupy a large amount of time. Models dependent on some backend(e.g. TensorRT) must be compiled and deployed before running. Even if the running time of the model is not long, it might also cost a lot to obtain the inference latency. Usually, each network would run many times, 10 or 100 for example, to obtain the stable latency value. Together with other factors, the measuring of the inference latency is not a free lunch, especially when handling tasks like NAS, which explores millions of models. Thus it's reasonable to build models to predict inference latency of neural networks.

We find that most of the previous methods for latency prediction are not competent to replace inference on the real device. In this paper, the GCN-based

method [1] is implemented to obtain accurate and fast latency prediction. As the graph topology of the neural architecture is informative for its inference performance, the GCN [4] can be employed to capture the graph representation of the models. We conducted experiments on multiple devices and implemented latency predictors with competitive performance in both accuracy and speed.

2 Related Work

Latency prediction with proxy metrics. Manually designed metrics like FLOPs and parameters are widely used as estimations of inference latency in designing and searching neural architectures [5, 6, 7, 8]. [9] utilizes the results of proxy metrics as feature inputs of regressors to obtain final latency predictions. In fact, the real inference latency does not always own linear correlations with the proxy metrics on most devices [1].

Latency prediction with runtime information. [10] comes up with a performance model on TPU using GNN with detailed kernel features and node features. Similarly, TVM [11] builds a cost model based on XGBoost [12] in order to guide the search of optimization. NN-meter [13] first splits the model into kernels by kernel detection and then fits linear regressors with the help of adaptive sampling. It's developed based on the insights of inference model behavior, i.e. kernel fusion for acceleration and sequential execution of kernels. These methods aim to obtain the runtime performance with fine-grained features, which might lead to better predictions. However, only a few inference frameworks are open-sourced. Even for those open-sourced ones, it requires lots of hardware expertise to conduct feature engineering of the predictors.

Layer-wise latency predictors. Paleo [14] and NeuralPower [15] sum up the predicted layer-wise or operator-wise latency values as the target model latency. ProxylessNAS [2] directly measures the layer-wise inference latency and sums them up. They fail to pay attention to the graph topology of the architectures and are inaccurate due to the ignorance of runtime graph optimization.

GCN-based latency predictions. BRP-NAS [1] employs the GCN to predict more accurate inference latency of cell-based architectures to improve neural architecture search. In [16], a GCN-based predictor is proposed to build a performance model for deep learning compliers. While BRP-NAS directly captures the information of the architecture, [16] utilizes schedule invariant and schedule dependent features for prediction.

3 Method

3.1 Analysis of Latency Predictors

Proxy metrics like FLOPs always act as an estimation of inference latency in the design and search of neural architectures. However, FLOPs and inference latency are not strongly correlated as shown in Figure 1. We attempted to add FLOPs as augments to boost the predictor but failed. One of the possible reasons is that models on these devices are memory-bounded instead of computation-bounded. Based on the roofline model [17] illustrated in Figure 2, the bottleneck of inference performance, i.e. inference latency, is memory bandwidth if the model's operational density is small. And the model is blocked by computation if the operational intensity is large. Considering that the models in NAS-Bench-201 [18] are relatively small, it's reasonable to infer that the models are memory-bounded on these devices. More experiments could be conducted if more detailed hardware properties and runtime information are available.

Layer-wise predictors proposed by [2] are also widely utilized to estimate the latency of the architectures. The layer-wise approaches sum up the inference time of all layers but don't perform well according to [1]. Possible reasons for inaccurate predictions include the additional cost of actions before and after network inference, large variance for the value of layer-wise latency, and etc. For instance, on the Atlas chip with Ascend backend, it's difficult to remove the impact of the memory-copying time if the layer-wise time is summed up directly. Moreover, they fail to take runtime acceleration commonly used on devices like operator fusion [19] into consideration. As can be observed in Figure 1, layer-wise prediction doesn't perform well either.

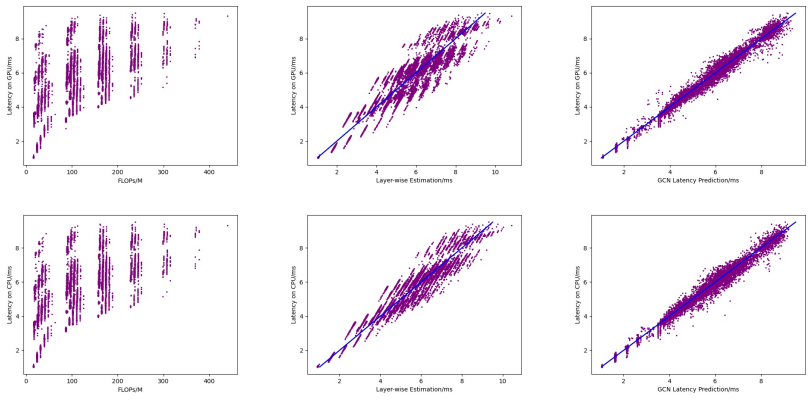

Figure 1: Comparison between latency truth w.r.t proxy (left), layer-wise prediction (middle), and GCN prediction (right) on desktop-GPU(the fist row) and desktop-CPU(the second row).

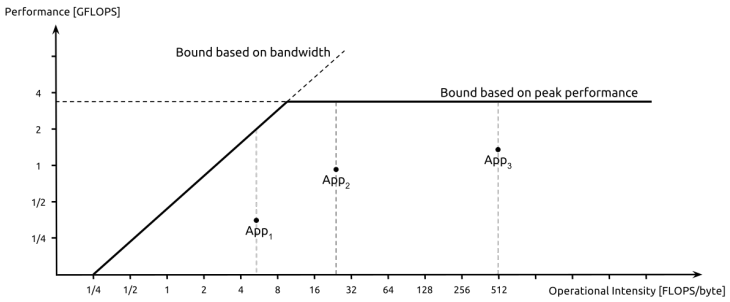

Figure 2: Illustration of Roofline Model [17].

3.2 End-to-end GCN-based Latency Predictor

Graphs present a natural and intuitive way to model deep-learning networks [16]. BRP-NAS [1] analyses the performance and limitations of existing latency predictors and proposes a novel latency predictor composed by a GCN to capture the graph-structure data. Given a graph $g = (V, E)$, where V is the set of N nodes with D features and E is the set of edges, a GCN takes feature matrix $X \in \mathbb{R}^{N \times D}$ and the graph adjacency matrix $A \in \mathbb{R}^{N \times N}$ as input and generates the predicted latency as output. The layer-wise propagation rule for an L-layer GCN is described as [1]

$$H^{l+1} = f(H^l, A) = \sigma(AH^lW^l) \tag{1}$$

where H^l and W^l are the feature map and weights at the l-th layer, and σ is the non-linear activation function. It's an equivalent version of the aggregation-update operation as following [16]:

$$H^{l+1} = \sigma(W^l_{self}H^l + AE^lW^l_{agg}) \tag{2}$$

where H^l is the feature of lth layer. W^l_{self} and W^l_{agg} are the corresponding weights used for the node features and their neighbors. A represents the adjacency matrix.

Architecture. In BRP-NAS [1], the predictor consists of several layers of GCNs followed by a fully connected layer generating the prediction values of latency. The models in the experimental dataset come from NAS-Bench201 [18] and each model is comprised of several cells in a well-designed macro-structure. The adjacency matrix A of the GCNs captures the topology of the neural cells and a global node that is connected to all the other nodes is introduced to obtain global information. The feature matrix X is a one-hot encoding of the nodes' type.

4 Experiments

4.1 Implementation Details

Architecture and training details. Following [1], we implement the graph convolutional neural network with a sequence of four GC-Norm-ReLU-Dropout operations and a fully-connected layer as the output. The number of hidden layers in each graph convolution block is 600 and the normalization operators are layer normalizations. The dropout ratio is 0.002. The criterion is L1 loss and we attempted MAPE loss but failed to obtain significant gain in performance. The networks are trained for 250 epochs with AdamW optimizer. An early stopping scheme is applied if there is no improvement within 35 epochs. The initial learning rate is 0.0008 for batch size 10 (configuration in the supplementary material of the paper [1]) and 0.0004 for batch size 1 (configuration in the code released by the authors). The learning rate schedule is the plateau, i.e., half the learning rate if no improvement in validation loss is seen for 10 epochs. L2 weight decay is 0.0005.

Evaluation metrics. We employ the same evaluation metrics as [1]. The values are the percentage of predicted latency within the corresponding error bound relative to the measured latency. Reported error bounds are 1%, 5%, and 10% as the original paper.

Dataset. We focus on Nas-Bench-201 [18], which include 15625 models. Following the same setting in [1], 900 models are randomly sampled as the training data. 100 random models are used for validation and the other 14k are for testing.

We conduct most of our experiments on desktop GPU - NVIDIA GTX 1080TI, and some additional experiments on desktop CPU - Intel Core i7-7820X, mobile DSP - Qualcomm Hexagon 690 DSP, and mobile GPU - Qualcomm Adreno 612 GPU. The values of inference latency come from the benchmark provided by [1].

Since it is noticed that both training loss and validation loss are very close to 0 at the end of the training, we consider increasing the size of training data. As we can observe in Table 1, using 900 random samples for training would result in similar performances in 14k test set and 13k dataset. However, increasing the training dataset from 900 to 2000 would come up with significant gains in the values of accuracy under all error bounds. It's not clear why the authors of [1] utilize only 900 samples as training data. We perform most of our experiments based on the same configuration of dataset split as the original paper.

Graph representation. In NAS-Bench-201 [18], each cell is composed of 4 nodes connected with the others by six operators. Different from representations with operators on the edge, BRP-NAS [1] deploy operators to the nodes of the graphs, as shown in Figure 3. Besides, input, output, and global nodes are added to the graph. The graph ends with nine nodes in total. The adjacency

matrix and one-hot features of operators are utilized as the input of the GCN. The blank operators, namely zero and skip-connection, are removed from the adjacency matrix. The features of the global node are used to predict the inference latency.

training data	test data	acc@error bound=[1%, 5%, 10%]
900^{+}	$14k^{+}$	36.7 ± 4.0, 85.9 ± 1.9, 96.9 ± 0.8
900	14k	38.5, 85.8, 97.1
2k	13k	43.5, 90.0, 97.9
900 from 2k	13k	37.0, 85.2, 96.6

Table 1: We implement the GCN and reproduce the performance of the original paper [1] on desktop GPU. The results with $^{+}$ are reported in the paper, which repeats the experiments for 100 times. We split the training and test dataset in another way. It can be observed that decreasing the test dataset size from 14k to 13k would result in similar accuracies. But increasing the training set from 900 to 2k would lead to significant gain in performance.

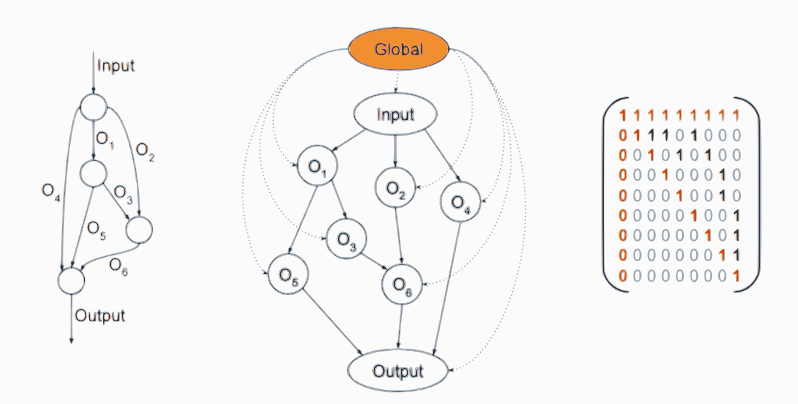

Figure 3: The figure [1] shows differences between graph representation in NAS-Bench-201 and BRP-NAS.

4.2 Experimental Results

We reproduce the results reported in the paper [1] on desktop GPU, desktop CPU, and mobile DSP as shown in Table 2.

Comparison with other methods. To compare with other schemes, we also conducted experiments on linear regressor and XGBoost regressor with the one-hot embedding of operators as input features, similar to GCN. With the help of Table 3, we can easily find that GCN outperforms all the other methods.

The commonly used layer-wise method performs poorly and even the simplest linear regression shows better performance. GCN has a great advantage over the popular tree-based model XGBoost [12].

Impact of training batch size. Most of our experiments follow the settings of the original paper. In order to study the impact of hyper-parameters, we perform some experiments by changing the training batch size. Mini-batch training can be treated as the tradeoff between stochastic gradient descent and full-batch training. A slightly larger batch would speed up model training effectively. However, large batch training would also result in performance degradation of accuracy as noted in [20]. We study the impact of batch size on accuracy on the dataset of desktop GPU. As shown in Table 6, degradation on accuracy would appear if the batch size is 8, and the accuracy drops dramatically if the batch size equals 64. In fact, the authors of [1] also use large loss weight and a different learning rate when the batch size is 10.

Ablations on adjacency matrix and features. In BRP-NAS [1], no detailed study was carried out for the influence of the adjacency matrix and the one-hot embedding. In Table 4, we verify the importance of adjacency matrix by replacing it with an identity matrix. And we ablate one-hot embedding features with random features. In both cases, the performances drop significantly. The results of linear regression can also be treated as another reference for the influence of the adjacency matrix.

Inference time. To obtain practical latency predictors, the inference time of the predictor must be less than the true inference time. In Table 5, results are reported on the test dataset of desktop GPU. We compute the summation of the latency values as the inference cost on the real device. We test the inference cost of our latency predictor across the whole test dataset on Intel(R) Xeon(R) Platinum 8163 CPU @ 2.50GHz (the same platform where the predictor is trained). The speed of the latency predictor is reported on PyTorch backend and includes the whole pipeline instead of only network inference. The value might be even smaller if the predictor is deployed with Caffe backend. The inference batch is set as 128, which gives the best performance. Since the value of inference batch size doesn't have an impact on the performance, we test across a range of values on batch size and the total inference costs don't vary much. The total inference cost of our latency predictor is about 10 times smaller than that on the desktop GPU. Since our latency predictor owns both high accuracy and much faster speed, it is a great substitute for inference on real devices.

5 Conclusion and Future Works

In this paper, we analyze commonly used latency estimation methods in neural architecture search. The proxy metrics like FLOPs are not good estimations

platform	acc@error bound 1%	acc@error bound 5%	acc@error bound 10%
desktop GPU	36.7 ± 4.0	85.9 ± 1.9	96.9 ± 0.8
desktop GPU*	38.5	85.8	97.1
desktop CPU	36.0 ± 3.5	85.2 ± 1.8	96.4 ± 0.7
desktop CPU*	34.5	84.1	96.1
mobile DSP	21.3 ± 1.9	77.5 ± 2.6	94.2 ± 0.4
mobile DSP*	23.8	76.9	90.1

Table 2: Results of performance reported in [1] and our implementation (those with *).

method	acc@error bound 1%	acc@error bound 5%	acc@error bound 10%
Layer-wise$^+$	4.2 ± 0.2	18.2 ± 1.5	29.6 ± 1.1
Linear Regression	6.0	30.0	54.5
XGBoost	11.2	50.4	78.7
GCN	38.5	85.8	97.1

Table 3: Layer-wise results are provided by BRP-NAS [1]. GCN shows great advantage over the other methods.

method	acc@error bound 1%	acc@error bound 5%	acc@error bound 10%
identity matrix	3.3	10.25	18.9
random features	3.1	16.0	31.7
adjacency matrix and one-hot features	38.5	85.8	97.1

Table 4: Ablation on GCN.

method	inference cost
true inference on desktop GPU	69.89s
GCN latency predictor	6.74s

Table 5: Comparison of inference cost between inference on the real device and our latency predictor.

batch size	acc@error bound 1%	acc@error bound 5%	acc@error bound 10%
1	38.5	85.8	97.1
2	39.4	86.7	97.2
4	37.5	86.7	97.0
8	30.6	80.5	94.8
64	7.2	35.4	60.8

Table 6: Accuracy impact of training batch size.

for inference time on real devices. The layer-wise prediction method is not accurate enough. We implement the latency prediction method by graph neural networks proposed by BRP-NAS [1]. The GCN predictors show outstanding performance in both accuracy and speed across multiple devices. We only conducted experiments on NAS-Bench-201 [18] and more complex architecture can be utilized in further works. Besides, the latency predictors used in this paper are very simple. Better designs would be needed if the predictors are applied to more complex architectures. In addition, more detailed information of the operators can be embedded into the features with the increase of the operator type in the models. We planned to perform experiments on extended features. However, additional information is limited because there are only five kinds of operators in NAS-Bench-201. It might boost the predictors much if there are plenty of operators.

References

[1] Lukasz Dudziak, Thomas C. P. Chau, Mohamed S. Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas D. Lane. BRP-NAS: prediction-based NAS using gcns. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[2] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[3] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[4] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[5] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 4510–4520. Computer Vision Foundation / IEEE Computer Society, 2018.

[6] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations*, 2018.

[7] 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.

[8] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[9] Stefan Reif, Judith Hemp Benedict Herzog, Timo Hönig, and Wolfgang Schröder-Preikschat. Precious: Resource-demand estimation for embedded neural network accelerators. In *First International Workshop on Benchmarking Machine Learning Workloads on Emerging Hardware*, 2020.

[10] Sam Kaufman, Phitchaya Phothilimthana, Yanqi Zhou, Charith Mendis, Sudip Roy, Amit Sabne, and Mike Burrows. A learned performance model for tensor processing units. *Proceedings of Machine Learning and Systems*, 3, 2021.

[11] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Q. Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. TVM: an automated end-to-end optimizing compiler for deep learning. In Andrea C. Arpaci-Dusseau and Geoff Voelker, editors, *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, pages 578–594. USENIX Association, 2018.

[12] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 785–794. ACM, 2016.

[13] Li Lyna Zhang, Shihao Han, Jianyu Wei, Ningxin Zheng, Ting Cao, Yuqing Yang, and Yunxin Liu. nn-meter: towards accurate latency prediction of deep-learning model inference on diverse edge devices. In Suman Banerjee, Luca Mottola, and Xia Zhou, editors, *MobiSys '21: The 19th Annual International Conference on Mobile Systems, Applications, and Services, Virtual Event, Wisconsin, USA, 24 June - 2 July, 2021*, pages 81–93. ACM, 2021.

[14] Hang Qi, Evan R. Sparks, and Ameet Talwalkar. Paleo: A performance model for deep neural networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[15] Ermao Cai, Da-Cheng Juan, Dimitrios Stamoulis, and Diana Marculescu. Neuralpower: Predict and deploy energy-efficient convolutional neural networks. In *Asian Conference on Machine Learning*, pages 622–637. PMLR, 2017.

[16] Shikhar Singh, Benoit Steiner, James Hegarty, and Hugh Leather. Using graph neural networks to model the performance of deep neural networks. *CoRR*, abs/2108.12489, 2021.

[17] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

[18] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2020.

[19] Yuan Wen, Andrew Anderson, Valentin Radu, Michael F. P. O'Boyle, and David Gregg. TASO: time and space optimization for memory-constrained DNN inference. In *32nd IEEE International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2020, Porto, Portugal, September 9-11, 2020*, pages 199–208. IEEE, 2020.

[20] Chao Peng, Tete Xiao, Zeming Li, Yuning Jiang, Xiangyu Zhang, Kai Jia, Gang Yu, and Jian Sun. Megdet: A large mini-batch object detector. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6181–6189, 2018.