QoS-aware Neural Architecture Search

An-Chieh Cheng[†]
accheng.tw@gmail.com

Chieh Hubert Lin[†]
hubert052702@gmail.com

Da-Cheng Juan[‡]
dacheng@google.com

Wei Wei[‡]
wewei@google.com

Min Sun[†]
sunmin@ee.nthu.edu.tw

[†] National Tsing-Hua University, Hsinchu, Taiwan
[‡] Google Research, Mountain View, USA

Abstract

The execution time of a real-world application varies substantially due to both system variations (*e.g*. CPU performance) and hardware states (*e.g*. energy efficiency). A neural architecture that can be operated effectively under certain conditions (*e.g*. low energy regime) may not be suitable for others, which can cause long latency that leads to poor user satisfactory. In this paper, we present QoS-NAS, a Quality-of-Service-aware neural architecture search that automatically searches for a neural network to be executed efficiently at each frame rate condition. At runtime, the controller of QoS-NAS offers trade-offs between accuracy and efficiency by transforming its downstream architectures at almost no additional latency cost. Experimental results confirm the effectiveness of QoS-NAS and show that QoS-NAS significantly outperform MobileNetV2 in terms of QoS satisfactory.

1 Introduction

Neural Architecture Search (NAS) has become an effective and promising approach to automate the design of deep learning models. It aims at finding the optimal model architectures based on their performances on evaluation metrics such as accuracy (Zoph and Le, 2017).

A common characteristic of the models searched by classic NAS methods is that a single model is typically considered as the optimal architecture. However, such an assumption turn out not to be ideal for real-world setting. In real-world, the execution times of an application vary substantially(Gaudette et al. (2016)), due to both system variations (*e.g*. CPU performance) and hardware states (*e.g*. energy efficiency). When the system workload is too high or battery power is too low, a neural architecture may have long latency and thus leads to poor user satisfactory. Intuitively, the architecture entails the ability to adapt to the change of both system and hardware states; for instance, we prefer the model to become more light-weighted while encountering low battery condition. Taking these conditions into consideration, we develop Qos-NAS, a NAS framework which offers a single neural network executable at a different level of QoS parameter requests. For each input pair of image sample and QoS parameter request, the controller is trained to select a best-suited architecture from the architecture distribution. With sub-modules being shared across different architectures, weights can be re-used for architectures that have never been selected before. Our QoS-NAS framework allows every sample and QoS parameter request pair to have their specifically tailored architectures.

QoS-NAS also aligns with the concept of conditional computing (Bengio et al., 2015; Kuen et al., 2018; Liu and Deng, 2017; Teja Mullapudi et al., 2018; Wu et al., 2018; Véniat and Denoyer, 2018)) since the instance-level architecture depends on the given input sample. However, these methods mentioned above assume their base model to be optimal across all samples, then perform their

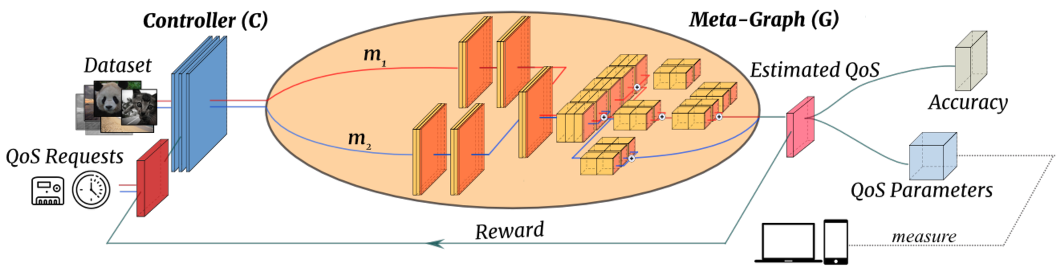

Figure 1: For each input instance, the controller of QoS-NAS learns to select an expert child architecture from the expressive meta-graph while considering the input image instance and QoS parameter request.

algorithm as a post-processing method to further reduce computational costs. In contrast, QoS-NAS is a neural architecture search framework with built-in instance awareness during architecture design and inference time decisions. Nevertheless, when the system states vary, these methods cannot adapt the network architecture to meet QoS requirements accurately. QoS-NAS elegantly combines the ideas of NAS and conditional computing by learning a controller to search and select architectures while considering both the data variation and different level of QoS parameter requests.

2 QoS-NAS: QoS-aware Neural Architecture Search

QoS-NAS inherits the design paradigm of InstaNAS (Cheng et al., 2018) on building instance-awareness into NAS, which incorporates a weight-sharing meta-graph (Bender et al., 2018; Pham et al., 2018; Liu et al., 2018) and a controller (Zoph and Le, 2017; Pham et al., 2018) responsible for architecture search sampling.

Similar to InstaNAS, QoS-NAS contains two major components: a meta-graph and a controller. The meta-graph is a directed acyclic graph (DAG), which each node represents the weights of a convolutional module and the edges represent the output of a module is taken as the input of the other. As a result, selecting a sequence of nodes within the meta-graph forms a valid architecture. Furthermore, the meta-graph can be treated as a set that represents all the possible combinations of child architectures.

On the other hand, the controller of InstaNAS is designed to be *instance-aware*. Different to InstaNAS, it consumes not only the input image instance, but also a QoS parameter request (*e.g.* a certain frame rate) in advance to the meta-graph. The controller then predicts a probability vector $\boldsymbol{p}$. Such a $\boldsymbol{p}$ corresponds to a child architecture within the meta-graph that is specifically designed for the input instance and QoS parameter request. During the architecture search, which involves reinforcement learning, we sample child architectures with respect to $\boldsymbol{p}$. We use policy gradient to train the controller with a multiple-objective reward:

$$\mathcal{R} = \mathcal{R}_{Accuracy} \cdot \mathcal{R}_{QoS}, \tag{1}$$

, which $R_{Accuracy}$ is accuracy reward and R_{QoS} is QoS parameter reward that are both normalized to $[0, 1]$. In our experiments, we use frame rate as the QoS parameter in R_{QoS} as a case study. The R_{QoS} is calculated as:

$$\mathcal{R}_{QoS} = \begin{cases} 1, & \text{if QoS parameter request satisfied.} \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Such a reward function design treats $R_{Accuracy}$ as a preference that is only considered while hard constraint R_{QoS} is satisfied.

QoS-NAS is trained with three stages: (a) "pre-train" the meta-graph, (b) "search" for architectures while jointly fine-tunes the meta-graph, and (c) "fine-tune" the meta-graph. In the pre-training stage, we use the random drop-path training paradigm (Bender et al., 2018), which makes any randomly sampled child architecture a valid architecture with decent accuracy. Such a pre-training is required, since, as described earlier, the search stage includes random sampling architectures. Otherwise, the sampled architectures will produce incorrect and noisy rewards. The second stage aims to search for architectures and train the controller as well. However, we observe that the controller tends to

	uniform	***beta1***	***beta2***	***beta3***	***beta4***
beta5					
MobileNetV2 1.4×	8.2	18.0	34.59	0	1.9
MobileNetV2 1.0×	11.9	21.8	46.44	0.1	4.0
MobileNetV2 0.4×	12.6	22.2	48.48	0.2	4.7
MobileNetV2$_{[0.4,1.0,1.4]}$	12.6	23.1	49.31	0.2	5.0
InstaNAS_$C10$_A	13.1	23.0	50.1	0.2	4.8
InstaNAS_$C10$_B	20.8	29.5	67.7	0.9	11.5
InstaNAS_$C10$_C	24.3	32.1	73.4	1.5	15.5
InstaNAS_$C10$_D	35.0	39.2	84.8	4.7	29.2
InstaNAS_$C10$_E	71.5	63.2	91.8	43.4	80.3
InstaNAS$_{[A,B,C,D,E]}$	72.5	64.6	**94.9**	43.4	81.2
QoS-NAS	**82.7**	**83.8**	86.4	**79.8**	**82.3**

Table 1: Cifar10 QoS performances in terms of satisfied (both frame rate and accuracy) requests percentage (%). QoS-NAS out-performs MobileNetV2 on five simulated frame rate distribution. Note that the controller latency is already included in the reported number.

exploit R_{QoS} easily, comparing to $R_{Accuracy}$. In other words, it is easy for the controller to select extremely-shallow architectures that have high R_{QoS} (*e.g.*, high frame rate) but low $R_{Accuracy}$ (*e.g.*, low accuracy). As a solution, we set the QoS parameter request range related to the number of the current epoch. This design shares a similar concept with curriculum learning (Bengio et al., 2009) which aims to gradually increase the task difficulty to avoid sudden collapsing. To be more specific, we enforce the controller to prefer low frame rate architectures in the early stage, then steadily scale up the targeting frame rate. In the third stage, we fine-tune the meta-graph with accuracy only while setting the weights of the controller fixed. This stage aims to make the meta-graph fully dedicating into the current controller policy and to reveal its best performance. Generally, the input QoS parameter requests during fine-tune stage can be in any distribution (e.g., uniform, normal) according to the real-world use case. In our experiment, we simply assume uniform distribution when fine-tuning the meta-graph.

3 Experiments

In this section, we explain and analyze the building blocks of QoS-NAS. We specify our main task to be image classification, though QoS-NAS is expected to work for most vision tasks. We choose frame rate as our QoS parameter, which is a very important factor to satisfy user experience.

Search Space. We follow the search space design proposed by (Cheng et al., 2018), which consists of 17 layers and each layer is consisted of 5 choices of modules: one basic convolution *(BasicConv)* and four mobile inverted bottleneck convolution *(MBConv)* layers with different kernel sizes {3, 5} and filter expansion ratios {3, 6}. We formulate the actions of the controller in each layer to be binary options among the five module choices. As a result, each layer has $2^5 = 32$ combinations of option, and approximately $32^{17} \simeq 10^{25}$ combinations of a complete network within the whole meta-graph.

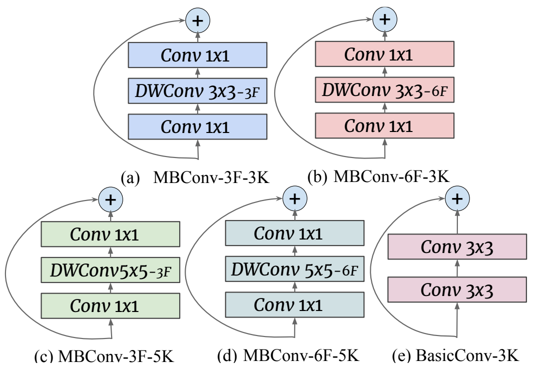

Figure 2: The five module options in each cell of QoS-NAS including basic convolution and mobile inverted bottleneck convolution with different expansion ratios (F) and kernel sizes (K). Note that DWConv stands for depthwise convolution.

Module Latency Profiling. The reward computation is especially challenging for QoS-NAS in practice. The reward of QoS-NAS considers the frame rate of each child architecture sampled in an instance-wise manner, which becomes extremely time-consuming, thus impractical to measure the frame rate for each architecture. Therefore,

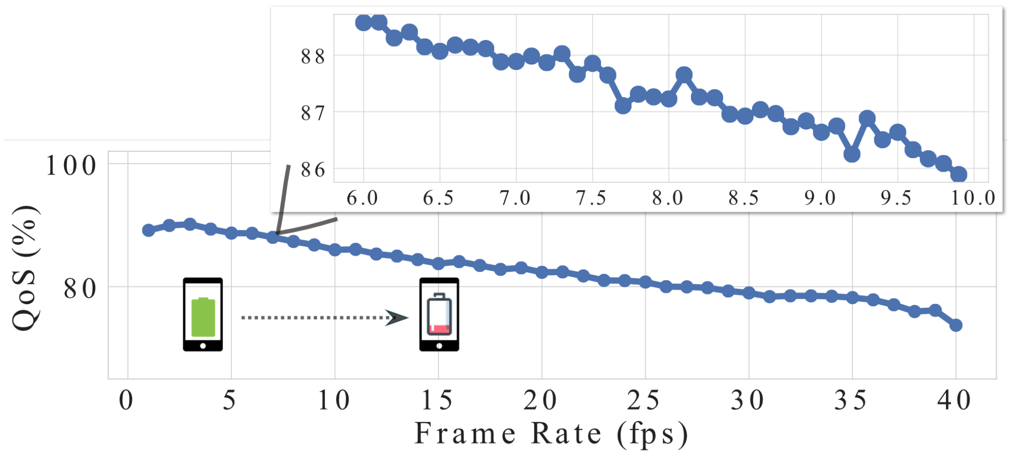

Figure 4: Experimental results confirm the effectiveness of QoS-NAS. At runtime, the controller of QoS-NAS can offer diverse trade-offs among accuracy and efficiency by transforming its downstream architecture.

to alleviate this problem, we adopt a strategy similar to (Yang et al., 2018; Cheng et al., 2018). We first profile the latency of each module in each layer and record the values within a look-up table. During the search phase, with each sampled child architecture, we check out the latency cost of each module and accumulate the values to form an estimated latency for the child architecture. To demonstrate the estimated latency is a good proxy for the real latency, we measure the correlation coefficient between estimated and real latencies for input resolution 32, 64 and 224 settings, which yield consistently high correlation coefficients of 0.97, 0.98 and 0.97, respectively.

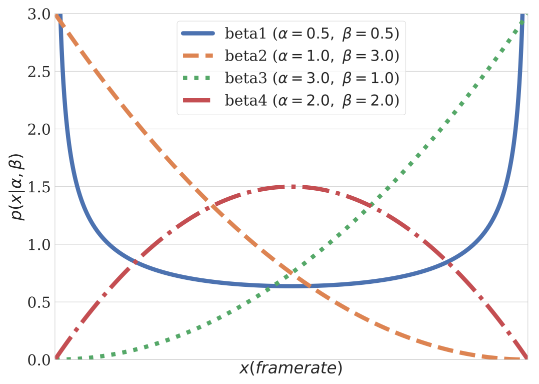

Figure 3: Visualization of the four beta distributions with different α and β values.

Quantitative Results We validate QoS-NAS on CIFAR-10 dataset with 5 simulated input frame rate distributions as the QoS parameter requests. The five distributions include uniform distribution and four beta distributions with different α and β values as visualized in Figure 3. Our experimental results in Table 1 show 70.1%, 60.7%, 37.0% , 79.6% and 77.3% of QoS satisfactory improvement comparing to MobileNetV2 on the five input distributions, respectively. We also compare with the model found by InstaNAS (Cheng et al. (2018)). InstaNAS can be considerate as a special case of QoS-NAS which the input QoS parameter requests are constant. As shown in Table 1, InstaNAS can find extremely architectures with high frame rate, which is effective for some specific input frame rate distributions (*e.g.* beta2). However, when considering a more diverse range of QoS requests, QoS-NAS performs more consistent stable than InstaNAS.

We also plot the accuracy of QoS-NAS at different frame rate request in Figure 4. At runtime, the controller of QoS-NAS can instantly offer diverse trade-offs among accuracy and efficiency even in finer-granularity of frame rate request.

4 Conclusion and Discussion

We instroduce QoSNAS, a novel neural architecture search approach with build-in Quality-of-Service-awareness. QoS-NAS outperforms MobileNetV2 on five distributions of QoS parameter requests with a large margin. More future research on extending this framework to other QoS parameters (*e.g.*, power consumption) or vision tasks (*e.g.*, object detection) may further exploit the benefits of QoS-NAS.

References

Bender, G., Kindermans, P.-J., Zoph, B., Vasudevan, V., and Le, Q. (2018). Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning*, pages 549–558.

Bengio, E., Bacon, P.-L., Pineau, J., and Precup, D. (2015). Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM.

Cheng, A.-C., Lin, C. H., Juan, D.-C., Wei, W., and Sun, M. (2018). Instanas: Instance-aware neural architecture search. *arXiv preprint arXiv:1811.10201*.

Gaudette, B., Wu, C.-J., and Vrudhula, S. (2016). Improving smartphone user experience by balancing performance and energy with probabilistic qos guarantee. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 52–63. IEEE.

Kuen, J., Kong, X., Lin, Z., Wang, G., Yin, J., See, S., and Tan, Y.-P. (2018). Stochastic downsampling for cost-adjustable inference and improved regularization in convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7929–7938.

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

Liu, L. and Deng, J. (2017). Dynamic deep neural networks: Optimizing accuracy-efficiency tradeoffs by selective execution. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. (2018). Efficient neural architecture search via parameter sharing. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 4092–4101.

Teja Mullapudi, R., Mark, W. R., Shazeer, N., and Fatahalian, K. (2018). Hydranets: Specialized dynamic architectures for efficient inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8080–8089.

Véniat, T. and Denoyer, L. (2018). Learning time/memory-efficient deep architectures with budgeted super networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Wu, Z., Nagarajan, T., Kumar, A., Rennie, S., Davis, L. S., Grauman, K., and Feris, R. (2018). Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8817–8826.

Yang, T.-J., Howard, A., Chen, B., Zhang, X., Go, A., Sandler, M., Sze, V., and Adam, H. (2018). Netadapt: Platform-aware neural network adaptation for mobile applications. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part X*, pages 289–304.

Zoph, B. and Le, Q. V. (2017). Neural architecture search with reinforcement learning.