On-Device Image Classification with Proxyless Neural Architecture Search and Quantization-Aware Fine-tuning

Han Cai, Tianzhe Wang, Zhanghao Wu, Kuan Wang, Ji Lin, Song Han
Massachusetts Institute of Technology
{hancai, usedtobe, zhwu, kuanwang, jilin, songhan}@mit.edu

Abstract

It is challenging to efficiently deploy deep learning models on resource-constrained hardware devices (e.g., mobile and IoT devices) with strict efficiency constraints (e.g., latency, energy consumption). We employ Proxyless Neural Architecture Search (ProxylessNAS) [4] to auto design compact and specialized neural network architectures for the target hardware platform. ProxylessNAS makes latency differentiable, so we can optimize not only accuracy but also latency by gradient descent. Such direct optimization saves the search cost by 200× compared to conventional neural architecture search methods. Our work is followed by quantization-aware fine-tuning to further boost efficiency. In the Low Power Image Recognition Competition at CVPR'19, our solution won the 3rd place on the task of Real-Time Image Classification (online track).

1. Overview

1.1. Introduction

Deep Neural Networks (DNNs) have achieved great success in many machine learning applications. However, it remains very challenging to efficiently deploy these deep learning models on resource-constrained edge devices, since they have to meet *strict* efficiency constraints (e.g., latency, energy consumption, throughput, etc.). Given the target hardware, researchers either design efficient neural network architectures specialized for the target hardware [13, 8, 10] or apply model compression techniques [5, 6, 7, 14] to improve the efficiency of existing models. Both approaches require to search in a vast design space, making it difficult to rely on domain experts to manually design compression strategies or neural network architectures for each case [2]. Therefore, there is an increasing interest in automating the design process, including neural architecture search (NAS) [16, 17, 1], auto channel pruning [7, 15], and auto model quantization [14].

Our solution incorporates Proxyless Neural Architecture Search (ProxylessNAS) [4] to auto design specialized neural network architectures, aiming to fit the given latency constraints on the target hardware platform while optimizing the accuracy performances. Unlike conventional NAS methods [16, 12, 3, 11] that search on the small proxy dataset (e.g., CIFAR-10) and do not take hardware efficiency into consideration, ProxylessNAS directly search for hardware-efficient neural network architectures on the target task and target hardware. It is achieved by the path-level pruning and path-level binarization techniques that cut the search cost by more than two orders of magnitude. Additionally, a latency model is built to predict each operator's latency, and the network latency is modeled as the weighted sum of each operator's latency where the weights are probabilities of choosing each operation. This network latency is incorporated into the loss function as a regularization term, thereby making latency differentiable. Benefiting from the directness and specialization, ProxylessNAS-searched model runs 1.8× faster than state-of-the-art human-designed neural networks [13] on mobile while maintaining similar accuracy, as shown in Figure 2.

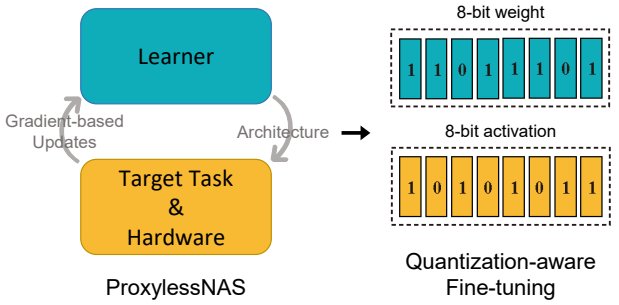

Figure 1. Our solution includes two parts: 1. search a high performance architecture under the latency constraint with *ProxylessNAS* and train the full-precision model; 2. quantize the model to 8-bit with quantization-aware finetuning.

After getting a specialized model using ProxylessNAS, we apply network quantization to further improve efficiency (Figure 1, right). Specifically, by adding fake-quantization

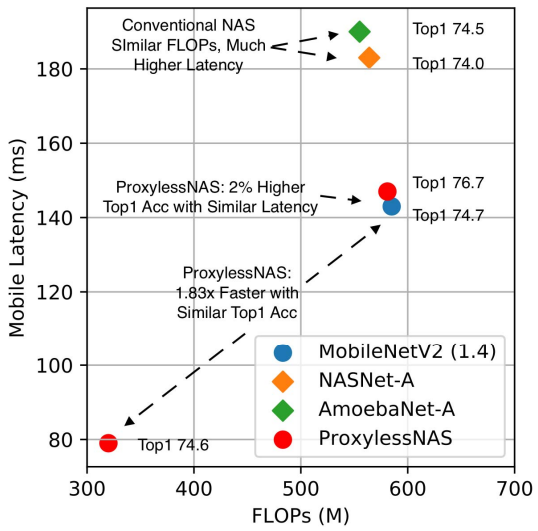

Figure 2. The performances of models searched by conventional NAS and ProxylessNAS. Previous work attains low FLOPs with the cost of high latency. Also, the models found by previous work are not hardware-aware. In contrast, our ProxylessNAS directly searches for a model on the target hardware, which takes latency constraint into consideration. Thereby, our model has lower latency and FLOPs while maintaining competitive accuracy, or even higher, with the same latency and FLOPs constraint.

nodes [9] to simulate the low bit calculation, quantization-aware finetuning preserves end-to-end model accuracy post quantization.

1.2. Solution Pipeline

The Real-Time Image Classification task focuses on building ImageNet classification models with tight latency constraint on a Pixel phone (24∼36ms). We use TensorflowLite for deploying our models.

The pipeline of our solution, shown in Figure 1, mainly consists of two parts:

1. Adopt *ProxylessNAS* to design a specialized neural network architecture under the latency constraint (i.e. 30ms) and train the model with full-precision.

2. Apply quantization-aware fine-tuning to the full-precision model, with fake-quantization nodes, converting the full-precision model to 8-bit one.

2. Proxyless Neural Architecture Search

Under strict efficiency constraints, it is critical to specialize neural network architectures to best fit the target hardware. To achieve this, instead of using existing human-designed neural network architectures (e.g., MobileNetV2 [13]), we consider searching for the best neural network architecture in a large design space. Specifically, for each block in the CNN model, we allow a set of candidate operations with various kernel sizes and widths:

- `ConvOp`: mobile inverted bottleneck conv [13] with various kernel sizes and expansion ratios
  - Kernel size: {3×3, 5×5, 7×7}
  - Expansion ratio: {3, 6}
- `ZeroOp`: if `ZeroOp` is chosen at i^{th} block, it means the block is skipped.

Moreover, to enable a direct trade-off between width and depth, we add the "ZeroOp" to candidate set. With a limited efficiency budget, the network can either choose to be shallower and wider by skipping more blocks and using larger ConvOps or choose to be deeper and thinner by keeping more blocks and using smaller ConvOps.

Therefore, the number of possible architectures in the design space is $[\underbrace{(3 \times 2)}_{\texttt{ConvOp}} + \underbrace{1}_{\texttt{ZeroOp}}]^N = 7^N$ where N is the number of blocks (21 in our experiments).

Given the vast design space, it is infeasible to rely on domain experts to manually design the CNN model for each hardware platform. So we need to employ neural architecture search (NAS) techniques to automatically design specialized CNN models for different target hardware platforms.

However, there are several challenges of applying conventional NAS methods in this case. First, conventional NAS methods [16, 17, 12] are very expensive to run (*e.g.*, 10^4 GPU hours) since they need to iteratively sample an architecture, train it from scratch and update the meta-controller. It typically requires tens of thousands of networks to be trained to find a good neural network architecture. Therefore, it is computationally difficult to apply them to large-scale datasets (e.g., ImageNet). Though we can alleviate this problem by first learning on a proxy dataset then transferring learned neural network architectures to the target dataset, it will lead to sub-optimal results for the target dataset. Second, they only optimize for the trade-off between accuracy and FLOPs [17]. However, FLOPs is a proxy metric of hardware efficiency. It does not directly translate to low latency on the hardware.

To address these challenges, we adopt a different approach to improve the efficiency of model specialization. We first build a super network that comprises all candidate architectures, which has a similar structure to a CNN model in the design space except that each specific operation is replaced with a mixed operation that has n parallel paths. Each path in a mixed operation corresponds to a candidate operation $o_i(\cdot)$, and we introduce an architecture parameter α_i to each path to learn which paths are redundant and thereby can be pruned (i.e. path-level pruning).

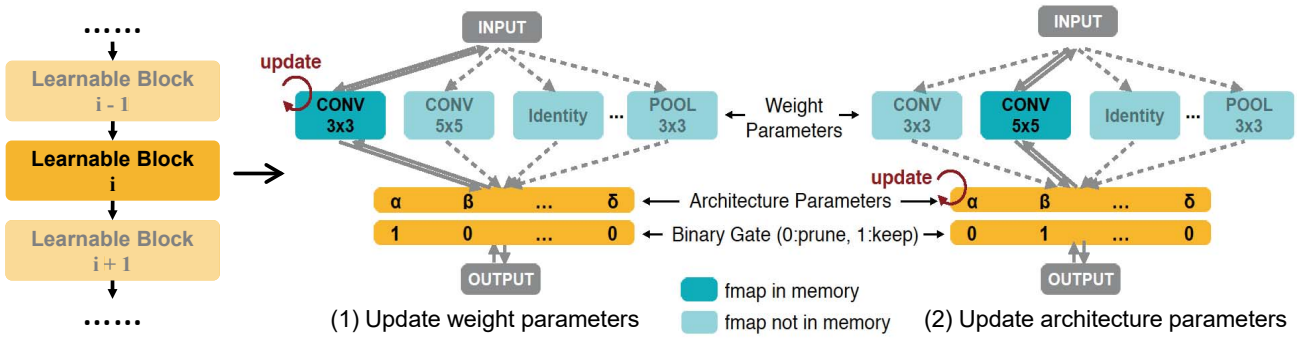

Figure 3. The illustration of ProxylessNAS, which learns both weight parameters and binarized architecture parameters.

In the forward step, to save GPU memory, we allow only one candidate path to actively reside in the GPU memory. This is achieved by hard-thresholding the probability of each candidate path to either 0 or 1 (*i.e.*, path-level binarization). As such the output of a mixed operation is given as

$$x_l = \sum_i g_i o_i(x_{l-1}) \tag{1}$$

where g_i is sampled according to the multinomial distribution derived from the architecture parameters, *i.e.*, $\{p_i = \text{softmax}(\alpha_i; \alpha) = \exp(\alpha_i)/\sum_i \exp(\alpha_i)\}$.

In the backward step, we update the weight parameters of active paths using standard gradient descent. Since the architecture parameters are not directly involved in the computational graph (Eq. 1), we use the gradient w.r.t. binary gates to update the corresponding architecture parameters:

$$\frac{\partial L}{\partial \alpha_i} = \sum_{j=1} \frac{\partial L}{\partial p_j} \frac{\partial p_j}{\partial \alpha_i} \approx \sum_{j=1} \frac{\partial L}{\partial g_j} \frac{\partial p_j}{\partial \alpha_i}.$$

This differentiable architecture learning process reduces the cost of NAS to the same level of training a regular neural network (as shown in Figure 4). Thereby, we can afford to directly search for the optimal neural network architecture on the target dataset without any proxy.

In order to specialize the model for hardware, we need to take the latency running on the hardware as a design reward. However, directly measuring the inference latency suffers from (i) slow (ii) high variance due to different battery condition and thermal throttling (iii) latency is non-differentiable and can't be directly optimized. To address these, we present our latency prediction model and hardware-aware loss.

To build the latency model we pre-compute the latency of each operator with all possible inputs. We query the lookup table during the searching process. The overall latency of i^{th} block is the weighted sum of the latency of each operator.

$$\begin{aligned} \mathbb{E}[\text{LAT}_\text{i}] = \; & \alpha \times F(\text{mb3\_3x3}) + \\ & \beta \times F(\text{mb3\_5x5}) + \\ & \sigma \times F(\text{identity}) + \\ & \ldots\ldots \\ & \zeta \times F(\text{mb6\_7x7}) \\ \mathbb{E}[\text{LAT}] = \; & \sum_i^N \mathbb{E}[\text{LAT}_\text{i}] \end{aligned} \tag{2}$$

Then we combine the latency and training loss (e.g. cross-entropy loss) using the following formula

$$\mathcal{L} = \mathcal{L}_\text{CE} \times \alpha \log \left(\frac{\mathbb{E}[\text{LAT}]}{\text{LAT}_\text{ref}} \right)^\beta, \tag{3}$$

where α and β are hyper-parameters controlling the accuracy-latency trade-off, and LAT_ref is the target latency. Note our formulation not only provides a fast estimation of the searched model but also makes the search process fully differentiable.

3. Post Quantization-Aware Procedure

3.1. Model Conversion

Many of the proposed models designed for ImageNet have only 1000 classes in total, in order to meet the requirement of the task with 1001 predictions, including an extra class for background that will never be used, we adjust the number of classes from 1000 to 1001 by assigning some specific value to the weight and bias. Explicitly, we set the bias of the background class to the minimum one among the original 1000 classes and assign the randomly chosen weight from the other 999 classes to this class. In that way, it can be proved that the background class will never be the prediction for any input.

Model	Setting	Accuracy	Latency
MoblieNetV2	224-0.5	63.7%(65.4%)	28ms
MobileNetV2	192-0.75	67.4%(68.7%)	36ms
MobileNetV2	160-1.0	67.4%(68.8%)	31ms
ProxylessNAS	224-0.5	65.7%(67.0%)	31ms
ProxylessNAS	160-1.0	**69.2%**(70.3%)	35ms

Table 1. Results of 8-bit model using different preprocessing, the number in the bracket denotes the full-precision model's top-1 accuracy on ImageNet The latency is directly measured on Google Pixel 2. It takes only 200 GPU hours to find the specialized model with ProxylessNAS in the table.

3.2. Quantization-Aware fine-tuning

In order to quantize the model found by ProxylessNAS without sacrifice the accuracy, we adopt quantization-aware fine-tuning. The process mainly contains two parts: constructing the fake-quantization graph and fine-tuning. The graph is based on the computational graph of the original model with extra fake-quantization nodes that simulate the effect of quantization in the forward and backward passes. During fine-tuning, the nodes also collects min-max information for the activations which allows the model to be quantized to fixed-point inference model easily without a separate calibration step.

4. Experiments

4.1. Training Details

For architecture searching, following the settings in [4], we randomly sample 5,000 images from the training set as a validation set for learning architecture parameters which are updated using the Adam optimizer with an initial learning rate of 0.006 for the gradient-based algorithm. After the training process of the over-parameterized network completes, a compact network is derived according to the architecture parameters. Next, we train the compact network using the same training settings except that the number of training epochs increases from 200 to 300. It takes 200 GPU hours to find the specialized model in the architecture search phase.

We adopt the TF-slim for the fake-quantization graph construction process in the quantization phase. However, the automatic fake-quantization node adding process can leave out some nodes, such as some adding and convolutional operators. Therefore, we visualize model and manually add those nodes into it. After quantization-aware fine-tuning, we can convert the model to 8-bit without much accuracy degradation.

4.2. Results

The final submission is 160-1.0 model found by ProxylessNAS (160 for the input size and 1.0 for the width multiplier), which outperforms the published mobilenetv2 192-0.75 about 2% accuracy, as shown in 1. Also, it is worth noticing that under the same latency constraint, the wider model (160-1.0) outperforms the deeper one (224-0.5) by about 3% accuracy.

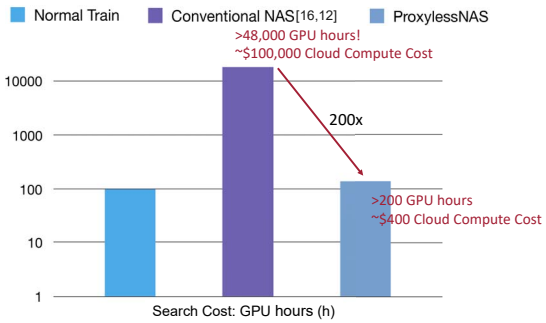

Figure 4. The comparison of GPU hours between conventional NAS and ProxylessNAS. ProxylessNAS can save two orders of magnitude for search cost, which is the same level as normal training.

5. Conclusion

We introduced ProxylessNAS that can directly learn neural network architectures on the target task and target hardware without any proxy. Benefiting from the direct search, we achieve strong empirical results on ImageNet. Moreover, we allow specializing network architectures for different platforms by directly incorporating the measured hardware latency into optimization objectives. Making the latency differentiable and using path binarization, we further reduce the searching cost for a specific target device to 200 GPU hours, which is the same level with normal training.

6. Acknowledgement

We thank Bo Chen, Bill Mark and the mobile vision team for the technical support on Tensorflow Lite and the mobile phone donation.

References

[1] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang. Efficient architecture search by network transformation. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[2] H. Cai, C. Gan, and S. Han. Once for All: Train One Network and Specialize it for Efficient Deployment. *arXiv preprint arXiv:1908.09791*, 2019.

[3] H. Cai, J. Yang, W. Zhang, S. Han, and Y. Yu. Path-level network transformation for efficient architecture search. In *ICML*, pages 677–686, 2018.

[4] H. Cai, L. Zhu, and S. Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.

[5] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ICLR*, 2016.

[6] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[7] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han. AMC: AutoML for Model Compression and Acceleration on Mobile Devices. In *ECCV*, 2018.

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

[9] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *CVPR*, 2018.

[10] Z. Liu, H. Tang, Y. Lin, and S. Han. Point-Voxel CNN for Efficient 3D Deep Learning. *arXiv*, 2019.

[11] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning*, pages 4092–4101, 2018.

[12] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4780–4789, 2019.

[13] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

[14] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han. HAQ: Hardware-Aware Automated Quantization with Mixed Precision. In *CVPR*, 2019.

[15] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 285–300, 2018.

[16] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

[17] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018.