Real-Time Image Classification with Proxyless Neural Architecture Search And Quantization-Aware Finetuning

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 device) with strict efficiency constraints (e.g., latency, energy consumption). We employ Proxyless Neural Architecture Search (ProxylessNAS) [1] 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 finetuning to further boost the efficiency. In Low Power Image Recognition Competition, CVPR'19, our solution won the 3rd place on the task of Real-Time Image Classification (online track).

1. Overview

1.1. Introduction

Designing a specialized neural network architecture tends to be difficult. First, there are limited heuristics. Second, the computation cost used to be prohibitive: even though many former works search a model on a proxy dataset (e.g., CIFAR-10) and then transfer the model found to the target dataset (e.g., Imagenet), the searching procedure can still take 10^4 GPU hours [4, 5]. To alleviate this huge cost, we proposed a method called *ProxylessNAS* [1], which leverages some special techniques to efficiently search the best architecture directly on target dataset without proxy under certain constraints, as shown in the left part of Figure 1.

Specifically, we cut the search cost by more than two orders of magnitude, mainly by two techniques: path-level pruning and path-level binarization, which saves both GPU hours and GPU memory. Reducing the search cost enables us to directly design specialized model for the target task and target hardware. On mobile phones, our searched model runs 1.8× faster than the best human-designed model [3].

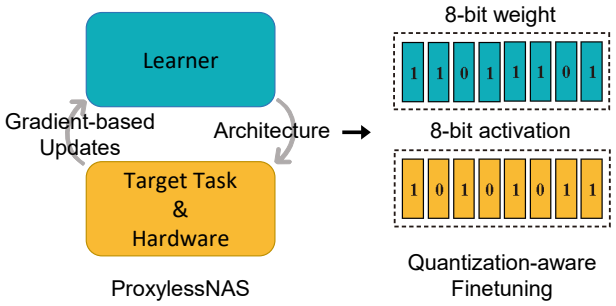

Figure 1. Pipeline of our solution. The whole process includes two parts: 1. find a well-performed architecture under specific latency constraint with *ProxylessNAS* and train the full-precision model; 2. quantize the model to 8-bit with quantization-aware finetuning.

Though it is quite straight forward to employ latency constraint as a discrete bound used for neural achitecture search, we can only use non-differentiable method (e.g., Reinforcement learning and evolution algorithm) to get it work. Moreover, it is always harder to train a non-differentiable model than a differentiable one: we cannot directly use the non-differentiable constraint (e.g., latency) to build our neural architecture search model, since the latency measurement as a whole is not differentiable given a specific architecture. Thus, we are inspired to make latency constraint differentiable by measuring each operator's latency and sum them up to approximate the real latency of the searched architecture. After that, we can directly use some gradient-based method to directly search the best architecture on ImageNet dataset under certain constraint.

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. Use *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 finetuning to the full-precision model, with fake-quantization nodes, converting the full-precision model to 8-bit model.

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 [3]), 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 [3] 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 [4, 5, 2] 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 [5]. 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).

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. 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.

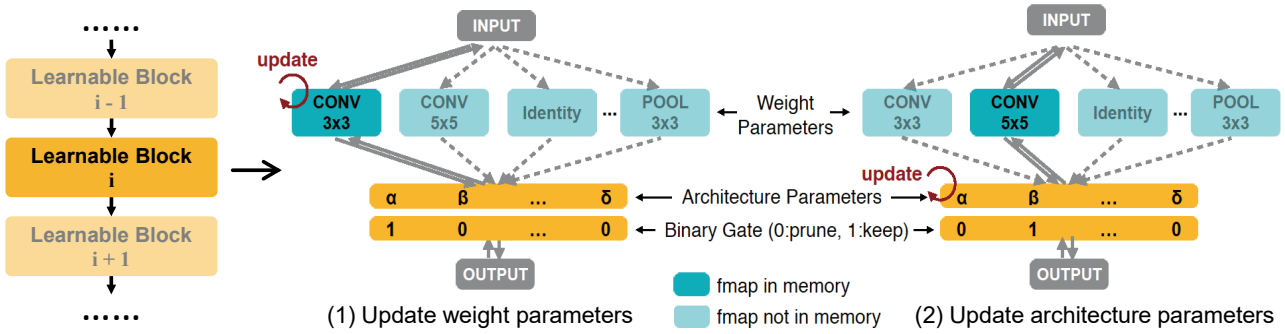

Figure 2. The illustration of ProxylessNAS, which learns both weight parameters and binarized architecture parameters.

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

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

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

where α and β are hyper-parameters controlling the accuracy-latency trade-off, and $\mathrm{LAT_{ref}}$ 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, while in this competition, we are required to provide a model with 1001 predictions, including a extra class for background. To adapt our model for the task, we change 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%(-)	28ms
MobileNetV2	192-0.75	67.4%(-)	36ms
MobileNetV2	160-1.0	67.4%(-)	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 original full-precision model's accuracy on test set.

3.2. Quantization-Aware Finetuning

In order to quantize the model found by ProxylessNAS without sacrifice the accuracy, we adopt quantization-aware finetuning. The process mainly contains two parts: constructing the fake-quantization graph and finetuning. 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 finetuning, 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. We adopt the TF-slim for the fake-quantization graph construction process. 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 finetuning, we can convert the model to 8-bit without much accuracy degradation.

3.3. 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.

4. Acknowledgement

We thank Bo Chen and his team for technical support of Tensorflow Lite Exportation.

References

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

[2] 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.

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

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

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