EMS: End-to-End Model Search for Network Architecture, Pruning and Quantization

Anonymous authors
Paper under double-blind review

Abstract

We present an end-to-end design methodology for efficient deep learning deployment. Unlike previous methods that separately optimize the neural network architecture, pruning policy, and quantization policy, we jointly optimize them in an end-to-end manner. To deal with the larger design space it brings, we train a quantization-aware accuracy predictor that fed to the evolutionary search to select the best fit. We first generate a large dataset of ⟨NN architecture, ImageNet accuracy⟩ pairs without training each architecture, but by sampling a unified supernet. Then we use these data to train an accuracy predictor without quantization, further using *predictor-transfer technique* to get the quantization-aware predictor, which reduces the amount of post-quantization fine-tuning time. Extensive experiments on ImageNet show the benefits of the end-to-end methodology: it maintains the same accuracy (75.1%) as ResNet34 float model while saving **2.2×** BitOps comparing with the 8-bit model; we obtain the same level accuracy as MobileNetV2+HAQ (Sandler et al., 2018; Wang et al., 2019) while achieving **2×/1.3×** latency/energy saving; the end-to-end optimization outperforms separate optimizations using ProxylessNAS+AMC+HAQ (Cai et al., 2019b; He et al., 2018; Wang et al., 2019) by **2.3**% accuracy while reducing **orders of magnitude** GPU hours and CO_2 emission.

1 Introduction

Deep learning has prevailed in many real-world applications like autonomous driving, robotics, and mobile VR/AR, while efficiency is the key to bridge research and deployment. Given a constrained resource budget on the target hardware (*e.g.*, latency, model size, and energy consumption), it requires an elaborated design of network architecture to achieve the optimal performance within the constraint. Traditionally, the deployment of efficient deep learning can be split into model architecture design and model compression (pruning and quantization). Some existing works (Han et al., 2016b; 2019) have shown that such a sequential pipeline can significantly reduce the cost of existing models. Nevertheless, careful hyper-parameter tuning is required to obtain optimal performance (He et al., 2018). The number of hyper-parameters grows exponentially when we consider the three stages in the pipeline together, which will soon exceed acceptable human labor bandwidth.

To tackle the problem, recent works have applied AutoML techniques to automate the process. Researchers proposed Neural Architecture Search (NAS) (Zoph & Le, 2017; Zoph et al., 2018; Real et al., 2018; Liu et al., 2018a;b; Zhong et al., 2018; Elsken et al., 2018; 2017; Cai et al., 2018a;b; Luo et al., 2018; Kamath et al., 2018) to automate the model design, outperforming the human-designed models by a large margin. Based on a similar technique, researchers adopt reinforcement learning to compress the model by automated pruning (He et al., 2018) and automated quantization (Wang et al., 2019). However, optimizing these three factors in separate stages will lead to sub-optimal results: *e.g.*, the best network architecture for the full-precision model is not necessarily the optimal one after pruning and quantization. Besides, this three-step strategy also requires considerable search time and energy consumption (Strubell et al., 2019). Therefore, we need a joint, end-to-end solution to optimize the deep learning model for a certain hardware platform.

However, directly extending existing AutoML techniques to our end-to-end model optimization setting can be problematic. Firstly, the joint search space is cubic compared to stage-wise search, making the search difficult. Introducing pruning and quantization into the pipeline will also greatly

increase the total search time, as both of them require time-consuming post-processing (*e.g.*, fine-tuning) to get accuracy approximation (Wang et al., 2019; Yang et al., 2018). Moreover, the search space of each step in pipeline is hard to be attested to be disentangle, and each step has its own optimization objective (eg. acc, latency, energy), so that the final policy of the pipeline always turns out to be sub-optimal.

To this end, we proposed *EMS*, an end-to-end design method to solve this problem. Our approach is derived from one-shot NAS (Guo et al., 2019; Brock et al., 2018; Pham et al., 2018; Bender et al., 2018; Liu et al., 2019a; Yu & Huang, 2019). We reorganize the traditional pipeline of "model design→pruning→quantization" into "architecture search + mixed-precision search". The former consists of both coarse-grained architecture search (topology, operator choice, *etc.*) and fine-grained channel search (replacing the traditional channel pruning (He et al., 2017)). The latter aims to find the optimal mixed-precision quantization policy trading off between accuracy and resource consumption. We work on both aspects to address the search efficiency. For architecture search, we proposed to train a highly flexible super network that supports not only the operator change but also fine-grained channel change, so that we can perform joint search over architecture and channel number. For the mixed-precision search, since quantized accuracy evaluation requires time-consuming fine-tuning, we instead use a predictor to predict the accuracy after quantization. Nevertheless, collecting data pairs for predictor training could be expensive (also requires fine-tuning). We proposed *Predictor-Transfer Technique* to dramatically improve the sample efficiency. Our quantization-aware accuracy predictor is transferred from full-precision accuracy predictor, which is firstly trained on cheap data points collected using our flexible super network (evaluation only, no training required). Once the predictor $P(arch, prune, quantization)$ is trained, we can perform search at ultra fast speed just using the predictor. With the above design, we are able to efficiently perform joint search over model architecture, channel number, and mixed-precision quantization. The predictor can also be used for new hardware and deployment scenarios, without training the whole system again.

Extensive experiment shows the superiority of our method: while maintaining the same level of accuracy (75.1%) with ResNet34 float model, we achieve **2.2×** reduction in BitOps compared to the 8-bit version; we obtain the same level accuracy as MobileNetV2+HAQ, and achieve **2×/1.3×** latency/energy saving; our models outperform separate optimizations using ProxylessNAS+AMC+HAQ by **2.3**% accuracy under same latency constraints, while reducing **orders of magnitude** GPU hours and CO_2 emission.

The contributions of this paper are:

- We devise an end-to-end methodology *EMS* to jointly perform NAS-pruning-quantization, thus unifying the conventionally separated stages into an integrated solution.

- We propose a *predictor-transfer* method to tackle the high cost of the quantization-aware accuracy predictor's dataset collection ⟨NN architecture, quantization policy, accuracy⟩.

- Such end-to-end method can efficiently search efficient models. With the supernet and the quantization-aware accuracy predictor, it only takes minutes to search a compact model for a new platform, enabling automatic model adjustment in diverse deployment scenarios.

2 BACKGROUND

Researchers have proposed various methods to accelerate the model inference, including architecture design (Howard et al., 2017; Sandler et al., 2018), network pruning (Han et al., 2015; Liu et al., 2017) and network quantization (Han et al., 2016b).

Neural Architecture Search. Tracing back to the development of NAS, one can see the reduction in the search time. Former NAS (Zoph et al., 2018; Real et al., 2018) use an RL agent to determine the cell-wise architecture. To efficiently search for the architecture, many later works viewed architecture searching as a path finding problem (Liu et al., 2019a; Cai et al., 2019b), it cuts down the search time by jointly training rather than iteratively training from scratch. Inspired by the path structure, some one-shot methods (Guo et al., 2019) have been proposed to further leverage the network's weights in training time and begin to handle mixed-precision case for efficient deployment. Another line of works tries to grasp the information by a performance predictor (Luo et al., 2018; Dai et al., 2019), which reduces the frequent evaluation for target dataset when searching for the optimal.

	ProxylessNAS	ChamNet	SPOS	AMC	HAQ	**EMS**
Hardware-aware	✓	✓	✓	✓	✓	✓
No extra training during searching		✓	✓			✓
No extra inference during searching		✓				✓
Channel pruning				✓		✓
Handling mixed-precision case			✓		✓	✓

Table 1: Comparisons of architecture search approaches for efficient models (ProxylessNAS (Cai et al., 2019b), SPOS: Single Path One-Shot (Guo et al., 2019), ChamNet (Dai et al., 2019), AMC (He et al., 2018), HAQ(Wang et al., 2019) and EMS (Ours). EMS distinguishes from other works by directly searching mixed-precision architecture without extra interaction with target dataset.

Pruning.   Extensive works show the progresses achieved in pruning: in early time, researchers proposed fine-grained pruning (Han et al., 2015; 2016b) by cutting off the connections (*i.e.*, elements) within the weight matrix. However, such kind of method is not friendly to the CPU and GPU and requires dedicated hardware (that supports sparse matrix multiplication) to perform the inference. Later, some researchers proposed channel-level pruning (He et al., 2017; Liu et al., 2017; Lin et al., 2017; Molchanov et al., 2016; Anwar & Sung, 2016; Hu et al., 2016; Polyak & Wolf, 2015) by pruning the entire convolution channel based on some importance score (*e.g.*, L1-norm) to enable acceleration on general-purpose hardware. However, both fine-grained pruning and channel-level pruning introduces an enormous search space as different layer has different sensitivities (*e.g.*, the first convolution layer is very sensitive to be pruned as it extracts important low-level features; while the last layer can be easily pruned as it's very redundant). To this end, recent researches leverage the AutoML techniques (He et al., 2018; Yang et al., 2018) to automate this exploration process and surpass the human design.

Quantization.   Quantization is a necessary technique to deploy the models on hardware platforms like FPGAs and mobile phones. (Han et al., 2016a) quantized the network weights to reduce the model size by grouping the weights using k-means. (Courbariaux et al., 2016) binarized the network weights into $\{-1, +1\}$; (Zhou et al., 2016) quantized the network using one bit for weights and two bits for activation; (Rastegari et al., 2016) binarized each convolution filter into $\{-w, +w\}$; (Zhu et al., 2017) mapped the network weights into $\{-w_N, 0, +w_P\}$ using two bits with a trainable range; (Zhou et al., 2018) explicitly regularized the loss perturbation and weight approximation error in a incremental way to quantize the network using binary or ternary weights. (Jacob et al., 2018) used 8-bit integers for both weights and activation for deployment on mobile devices. Some existing works explored the relationship between quantization and network architecture. HAQ (Wang et al., 2019) proposed to leverage AutoML to determine the bit-width for a mixed-precision quantized model. A better trade-off can be achieved when different layers are quantized with different bits, showing the strong correlation between network architecture and quantization.

Multi-Stage Optimization.   Above methods are orthogonal to each other and a straightforward combination approach is to apply them sequentially in multiple stages *i.e.* NAS+Pruning+Quantization:

- In the first stage, we can search the neural network architecture with the best accuracy on the target dataset (Tan et al., 2018; Cai et al., 2019b; Wu et al., 2019a):

$$\mathcal{A}^*_{\text{NAS}}, w^*_{\text{NAS}} = \arg\max_{\mathcal{A},w} ACC_{val}(\mathcal{A}, w). \tag{1}$$

- In the second stage, we can prune the channels in the model automatically (He et al., 2018):

$$\mathcal{A}^*_P, w^*_P = \arg\max_{P} ACC_{val}(P(\mathcal{A}^*_{\text{NAS}}, w^*_{\text{NAS}})). \tag{2}$$

- In the third stage, we can quantize the model to mixed-precision to make full use of the emerging hardware architecture (Wang et al., 2019):

$$\mathcal{A}^*, w^* = \arg\max_{Q} ACC_{val}(Q(\mathcal{A}^*_P, w^*_P)) \tag{3}$$

However, this *separation* usually leads to a sub-optimal solution: *e.g.*, the best neural architecture for the floating-point model may not be optimal for the quantized model. Moreover, frequent evaluations on the target dataset make such kind of methods time-costly: *e.g.*, a typical pipeline as above can take about 300 GPU hours, making it hard for researchers with limited computation resources to do automatic design.
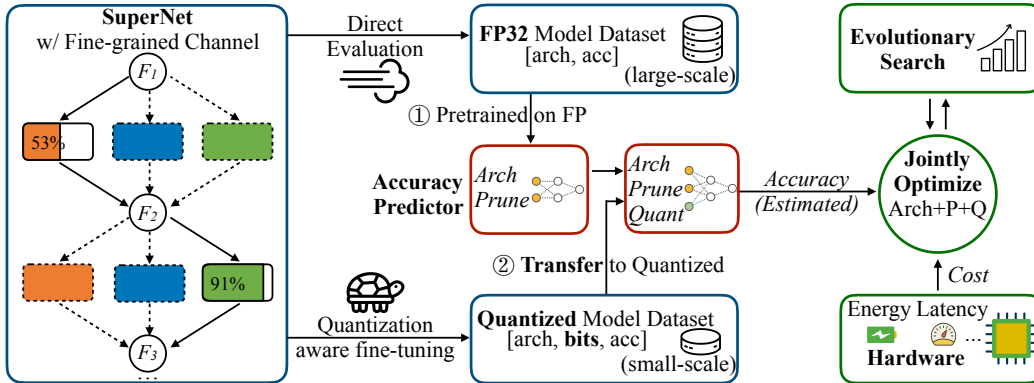
Figure 1: An overview of our end-to-end design methodology. We first train an accuracy predictor for the full precision NN, then incrementally train an accuracy predictor for the quantized NN (predictor-transfer). Finally, evolutionary search is performed to find the specialized NN architecture that fits hardware constraints.

Joint Optimization. Instead of optimizing NAS, pruning and quantization independently, joint optimization aims to find a balance among these configurations and search for the optimal strategy. To this end, the joint optimization objective can be formalized into:

$$\mathcal{A}^* = \underset{\mathcal{A}, w, P, Q}{\arg \max} \, ACC_{val}\big(Q(P(\mathcal{A}, w))\big), \tag{4}$$

However, the search space of this new objective is tripled as original one, so it becomes challenging to perform joint optimization. We endeavor to unify NAS, pruning and quantization as joint optimization. The outline is: 1. Train a super network that covers a large search space and every sub-network can be directly extracted without re-training. 2. Build a quantization-aware accuracy predictor to predict quantized accuracy given a sub-network and quantization policy. 3. Construct a latency/energy lookup table and do resource constrained evolution search. Thereby, this joint optimization problem can be tackled in an end-to-end manner.

Comparison with Recent Methods. The search space is quadratic when comparing to (Wu et al., 2019b), since we need to take care of both architecture configuration and quantization policy rather than quantization policy only. Unlike (Dai et al., 2019), whose predictor only use full precision (FP) data to train, we face an unbalance ratio in full precision (FP) and mixed-precision (MP) data. Also, architecture configuration and quantization policy are orthogonal to each other, simply treating this problem as before will lead to a significant performance drop when training the predictor. Different from (Guo et al., 2019), which uses ResNet as super network's backbone and cannot handle a more efficient scenario when changing backbone into MobileNet due to large accuracy drop after quantization, our super network provides a more stable accuracy statistic after sampling network to do quantization. This gives us the opportunity to acquire quantization data simply by extract a sub-network and do quantization.

3 END-TO-END DESIGN METHODOLOY

The overall framework of our end-to-end design framework is shown in Figure 1. It consists of a highly flexible super network with fine-grained channels, an accuracy predictor, and evolution search jointly optimizing architecture, pruning, and quantization.

3.1 SUPER NETWORK WITH FINE-GRAINED CHANNELS

Neural architecture search aims to find a good sub-network from a large search space. Traditionally, each sampled network is trained to obtain the actual accuracy (Zoph & Le, 2017), which is time consuming. Recent one-shot based NAS (Guo et al., 2019) first trains a large, multi-branch network. At each time, a sub-network is extracted from the large network to directly evaluate the approximated accuracy. Such a large network is called *super network*. Since the choice of different layers in a deep

neural network is largely independent, a popular way is to design multiple choices (*e.g.*, kernel size, expansion ratios) for each layer.

In this paper, we used a super network that supports different kernel sizes (*i.e.* 3, 5, 7) and channel number (*i.e.* $4\times\mathcal{B}$ to $6\times\mathcal{B}$, 8 as internal, $\mathcal{B}$ is the base channel number in that block) in block level, and different depths (*i.e.* 2, 3, 4) in stage level. The combined search space contains more than 10^{35} sub-networks, which is large enough to conduct neural architecture search.

Properties of the Super Network. We also followed the one-shot setting to first build a super network and then perform search on top of it. To ensure efficient architecture search, we find that the super network needs to satisfy the following properties: (1) For every extracted sub-network, the performance could be directly evaluated without re-training, so that the cost of training only need to be paid once. (2) Support an extremely large and fine-grained search space to support channel number search. As we hope to incorporate pruning policy into architecture space, the super network not only needs to support different operators, but also fine-grained channel numbers (8 as interval). Thereby, the new space is significantly enlarged (nearly quadratic from 10^{19} to 10^{35}).

However, it is hard to achieve the two goals at the same time due to the nature of super network training: it is generally believed that if the search space gets too large (*e.g.*, supporting fine-grained channel numbers), the accuracy approximation would be inaccurate (Liu et al., 2019b). A large search space will result in high variance when training the super network. To address the issue, We adopt progressive shrinking (PS) algorithm (Cai et al., 2019a) to train the super network. Specifically, we first train a full sub-network with largest kernel sizes, channel numbers and depths in the super network, and use it as a teacher to progressively distill the smaller sub-networks sampled from the super network. During distillation, the trained sub-networks still update the weights to prevent accuracy loss. The PS algorithm effectively reduce the variance during super network training. By doing so, we can assure that the extracted sub-network from the super network preserves competitive accuracy without re-training.

3.2 QUANTIZATION-AWARE ACCURACY PREDICTOR

To reduce the cost for designs in various deployment scenarios, we propose to build a quantization-aware accuracy predictor P, which predicts the accuracy of the mixed-precision (MP) model based on architecture configurations and quantization policies. During search, we used the predicted accuracy $\overline{acc} = P(\text{arch}, \text{prune}, \text{quantize})$ instead of the measured accuracy. The input to the predictor P is the encoding of the network architecture, the pruning strategy, and the quantization policy.

Architecture and quantization policy encoding. We encode the network architecture block by block: for each building block (*i.e.* bottleneck residual block like MobileNetV2 (Sandler et al., 2018)), we encode the kernel size, channel numbers, weight/activation bits for pointwise and depthwise convolutions into one-hot vectors, and concatenate these vectors together as the encoding of the block. For example, a block has 3 choices of kernel sizes (*e.g.* 3,5,7) and 4 choices of channel numbers(*e.g.* 16,24,32,40), if we choose kernel size=3 and channel numbers=32, then we get two vectors [1,0,0] and [0,0,1,0], and we concatenate them together and use [1,0,0,0,0,1,0] to represent this block's architecture. Likewise, we also use one-hot vectors to denote the choice of bitwidth for certain weights/activation of pointwise and depthwise layers, *e.g.* suppose weight/activation bitwidth choices for pointwise/depthwise layer are 4 or 8, we use [1,0,0,1,0,1,1,0] to denote the choice (4,8,8,4) for quantization policy. If this block is skipped, we set all values of the vector to 0. We further concatenate the features of all blocks as the encoding of the whole network. Then for a 5-layer network, we can use a 75-dim($5\times(3+4+2\times4)$=75) vector to represent such an encoding. In our setting, the choices of kernel sizes are [3,5,7], the choices of channel number depend on the base channel number for each block, and bitwidth choices are [4,6,8], there are 21 blocks in total to design.

Accuracy Predictor. The predictor we use is a 3-layer feed-forward neural network with each embedding dim equaling to 400. As shown in the left of Figure 2, the input of the predictor is the one-hot encoding described above and the output is the predicted accuracy. Different from existing methods (Liu et al., 2019a; Cai et al., 2019b; Wu et al., 2019a), our predictor based method does not require frequent evaluation of architecture on target dataset in the search phase. Once we have the predictor, we can integrate it with any search method (*e.g.* reinforcement learning, evolution, bayesian optimization, etc.) to perform end-to-end design over architecture-pruning-quantization at a

negligible cost. However, the biggest challenge is how to collect a [architecture, quantization policy, accuracy] dataset to train the predictor for quantized models due to: 1) collecting quantized model's accuracy is time-consuming: fine-tuning is required to recover the accuracy after quantization, which takes about 0.2 GPU hours per data point. In fact, we find that 80k data pairs is a suitable size to train a good full precision accuracy predictor. If we collect a quantized dataset with the same size as the full-precision one, it can cost 16,000 GPU hours, which is far beyond affordable. 2) The quantization-aware accuracy predictor is harder to train than a traditional accuracy predictor on full-precision models: the architecture design and quantization policy affect network performance from two separate aspects, making it hard to model the mutual influence. Thus using traditional way to train quantization-aware accuracy predictor can result in a significant performance drop (Table 2).

Transfer Predictor to Quantized Models. Collecting a quantized NN dataset for training the predictor is difficult (needs finetuning), but collecting a full-precision NN dataset is easy: we can directly pick sub-networks from the super net and measure its accuracy. We propose the *predictor-transfer* technique to increase the sample efficiency and make up for the lack of data. As the order of accuracy before and after quantization is usually preserved, we first pre-train the predictor on a large-scale dataset to predict the accuracy of full-precision models, then transfer to quantized models. The quantized accuracy dataset is much smaller and we only perform short-term fine-tuning. As shown in Figure 2, we add the quantization bits (weights& activation) of the current block into the input embedding to build the quantization-aware accuracy predictor. We then further fine-tune the quantization-aware accuracy predictor using pre-trained FP predictor's weights as initialization. Since most of the weights are inherited from the full-precision predictor, the training requires much less data compared to training from scratch.
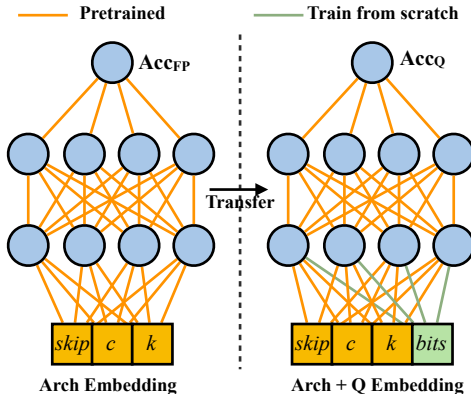

Figure 2: Predictor-transfer technique. We start from a pre-trained full-precision predictor and add another input head (green square at bottom right) denoting quantization policy. Then fine-tune the quantization-aware accuracy predictor.

3.3 HARDWARE-AWARE EVOLUTIONARY SEARCH

As different hardware might have drastically different properties (*e.g.*, cache size, level of parallelism), the optimal network architecture and quantization policy for one hardware is not necessarily the best for the other. Therefore, instead of relying on some indirect signals (*e.g.*, BitOps), our optimization is directly based on the measured latency and energy on the target hardware.

Measuring Latency and Energy. Evaluating each candidate policy on actual hardware can be very costly. Thanks to the sequential structure of neural network, we can approximate the latency (or energy) of the model by summing up the latency (or energy) of each layer. We can first build a lookup table containing the latency and energy of each layer under different architecture configurations and bit-widths. Afterwards, for any candidate policy, we can break it down and query the lookup table to directly calculate the latency (or energy) at negligible cost. In practice, we find that such practice can precisely approximate the actual inference cost.

Resource-Constrained Evolution Search. We adopt the evolution-based architecture search (Guo et al., 2019) to explore the best resource-constrained model. Based on this, we further replace the evaluation process with our *quantization-aware accuracy predictor* to estimate the performance of each candidate directly. The cost for each candidate can then be reduced from N times of model inference to only **one time** of predictor inference (where N is the size of the validation set). Furthermore, we can verify the resource constraints by our *latency/energy lookup table* to avoid the direct interaction with the target hardware. Given a resource budget, we directly eliminate the candidates that exceed the constraints.

Model	ImageNet Top1 (%)	Latency (ms)	Energy (mJ)	BitOps (G)	Search cost (GPU hours)	CO$_2$e (marginal)	Cloud compute cost (marginal)
MobileNetV2 - 8bit	71.8	9.10	12.46	19.2	-	-	-
ProxylessNAS - 8bit	74.2	13.14	14.12	19.5	200N	56.72	$148 – $496
ProxylessNAS + AMC - 8bit	73.3	9.77	10.53	15.0	204N	57.85	$151 – $506
MobileNetV2 + HAQ	71.9	8.93	11.82	-	96N	27.23	$71 – $238
ProxylessNAS + AMC + HAQ	71.8	8.45	8.84	-	300N	85.08	$222 – $744
DNAS	74.0	-	-	57.3	40N	11.34	$30 – $99
SinglePathNAS	74.6	-	-	51.9	288 + 24N	6.81	$18 – $60
Ours-A (w/o transfer)	72.1	8.85	11.79	13.2	1200 + 0.5N	0.14	**$0.4 – $1.2**
Ours-B (w/ transfer)	**74.1**	8.42	12.11	16.5	1200 + 0.5N	0.14	**$0.4 – $1.2**
Ours-C (w/ transfer)	**75.1**	12.17	14.14	23.6	1200 + 0.5N	0.14	**$0.4 – $1.2**

Table 2: Comparison with state-of-the-art efficient models for hardware with fixed quantization or mixed precision. Our method cuts down the marginal search time by two-order of magnitudes while achieving better performance than others. The marginal CO$_2$ emission (lbs) and cloud compute cost ($) (Strubell et al., 2019) is negligible for search in a new scenario.

4 IMPLEMENTATION DETAILS

Data Preparation for Quantization-aware Accuracy Predictor. We generate two kinds of data (2,500 for each): 1. random sample both architecture and quantization policy; 2. random sample architecture, and sample 10 quantization policies for each architecture configuration. To speed up the data collection process, we use ImageNet-100 dataset. We mix the data for training the quantization-aware accuracy predictor, and use full-precision pretrained predictor's weights to transfer. The number of data to train a full precision predictor is 80,000. In that way, our quantization accuracy predictor can have the ability to generalize among different *architecture/quantization policy* pairs and learn the mutual relation between architecture and quantization policy.

Evolutionary Architecture Search. For evolutionary architecture search, we set the population size to be 100, and choose Top-25 candidates to produce the next generation (50 by mutation, 50 by crossover). The mutation rate is 0.1, which is the same as that in (Guo et al., 2019). We set max iterations to 500, and choose the best candidate among the final population.

Quantization. We follow the implementation in (Wang et al., 2019) to do quantization. Specifically, we quantize the weights and activations with the specific quantization policies. For each layer with weights w with quantization bit b, we linearly quantize it to $[-v, v]$, the quantized weight is:

$$w' = \max(0, \min(2v, round(\frac{2w}{2^b - 1}) \cdot v)) - v \qquad (5)$$

We set choose different v for each layer that minimize the KL-divergence $\mathcal{D}(w||w')$ between origin weights w and quantized weights w'. For activation weights, we quantize it to $[0, v]$ since the value is non-negative after ReLU6 layer.

5 EXPERIMENTS

To verify the effectiveness of our methods, we conduct experiments that cover two of the most important constraints for on-device deployment: *latency* and *energy consumption* in comparison with some state-of-the-art models using neural architecture search. Besides, we compare BitOps with some multi-stage optimized models.

Dataset, Models and Hardware Platform. The experiments are conducted on ImageNet dataset. We compare the performance of our end-to-end designed models with mixed-precision models searched by (Wang et al., 2019; He et al., 2018; Cai et al., 2019b) and some SOTA fixed precision 8-bit models. The platform we used to measure the resource consumption for mixed-precision model is BitFusion (Sharma et al., 2018), which is a state-of-the-art spatial ASIC design for neural network accelerator. It employs a 2D systolic array of Fusion Units which spatially sum the shifted partial products of two-bit elements from weights and activations.
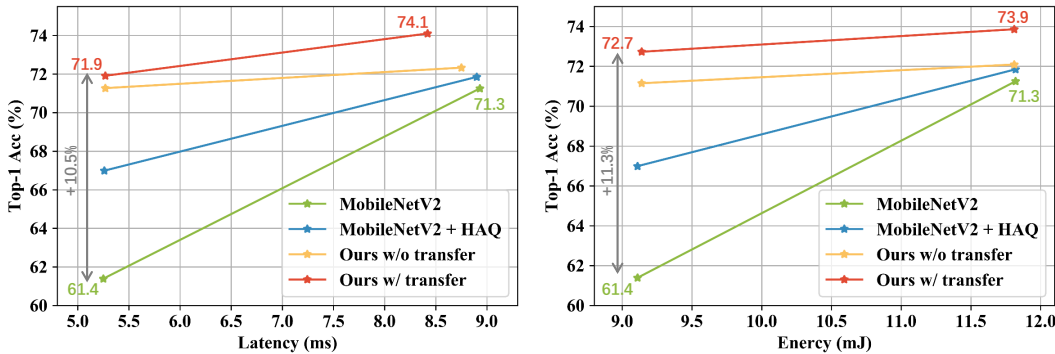
Figure 3: Comparison with mixed-precision models searched by HAQ (Wang et al., 2019) under latency/energy constraints. When the constraint is strict, our model can outperform fixed precision model by more than 10% accuracy, and 5% compared with HAQ. Such performance boost may benefit from the dynamic architecture search space rather than fixed one as MobileNetV2.
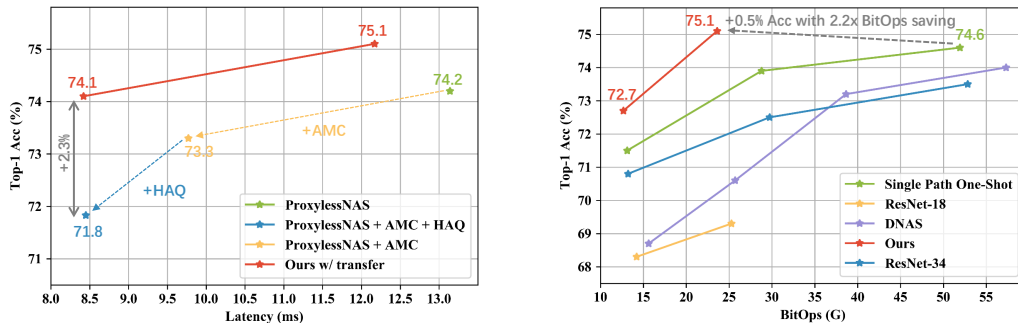

Figure 4: Comparison with *sequentially designed* mixed-precision models searched by AMC and HAQ (Cai et al., 2019b; He et al., 2018; Wang et al., 2019) under latency constraints. Our end-to-end designed model while achieving better accuracy than sequentially designed models.

Figure 5: Comparison with quantized model under BitOps constraint. Our model achieves 0.5% accuracy boost (from 74.6% to 75.1%) at half BitOps compared to single path one-shot. Also note that accuracy of our model is the same level as float version ResNet-34 model (75.0%).

5.1 COMPARISON WITH STATE-OF-THE-ART EFFICIENT MODELS

Table 2 presents the results for different efficiency constraints. As one can see, our model can consistently outperform state-of-the-art models with either fixed or mixed-precision. Specifically, our small model (Ours-B) can have 2.2% accuracy boost than mixed-precision MobileNetV2 search by HAQ (from 71.9% to 74.1%); our large model (Ours-C) attains better accuracy (from 74.6% to 75.1%) while only requires half of BitOps. When applied with transfer technology, it does help for the model to get better performance (from 72.1% to 74.1%). It is also notable that the marginal cost for cloud computer and CO_2 emission is two orders of magnitudes smaller than other works.

5.2 EFFECTIVENESS OF END-TO-END DESIGN

Comparison with MobileNetV2+HAQ. Figure 3 show the results on the BitFusion platform under different latency constraints and energy constraints. Our end-to-end designed models consistently outperform both mixed-precision and fixed precision SOTA models under certain constraints. It is notable when constraint is tight, our models have significant improvement compared with state-of-the-art mixed-precision models. Specifically, with similar efficiency constraints, we improve the ImageNet top1 accuracy from the MobileNetV2 baseline 61.4% to 71.9% (+10.5%) and 72.7% (+11.3%) for latency and energy constraints, respectively. Moreover, we show some models searched by our quantization-aware predictor without predictor-transfer technique. With this technique applied,
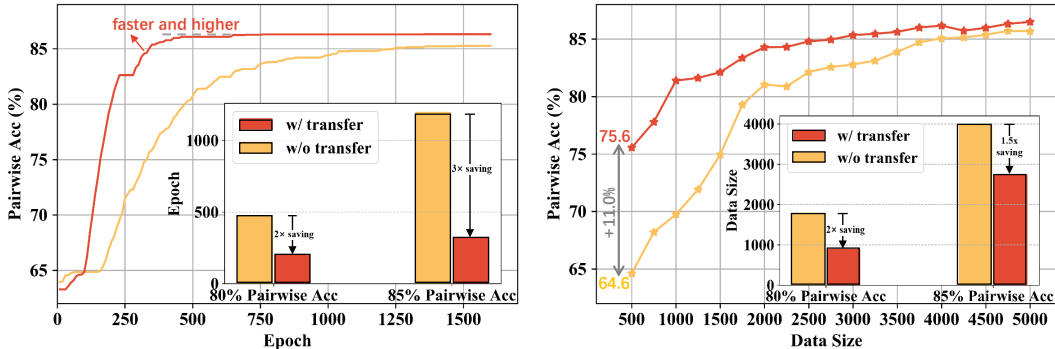
Figure 6: Illustration of the performance w/ or w/o predictor-transfer technique. Pairwise accuracy is a metric that measures the relative relationship between each two architectures. Left graph shows that the quantization-aware predictor could attain a faster and higher convergence compared w/o transfer. Right graph shows that when data is limited, predictor-transfer technique could largely improve the pairwise accuracy (from 64.6% to 75.6%). Using predictor-transfer technique, we can achieve 85% pairwise accuracy using less than 3k data points, while at least 4k data will be required without this technique.

the accuracy can consistently have an improvement, since the non-transferred predictor might loss some mutual information between architecture and quantization policy.

Comparison with multi-stage optimized Model. Figure 4 compares the multi-stage optimization with our joint optimization results. As one can see, under the same latency/energy constraint, our model can attain better accuracy than the multi-stage optimized model (74.1% vs 71.8%). This is reasonable since the per-stage optimization might not find the global optimal model as end-to-end design does.

Comparison under Limited BitOps. Figure 5 reports the results with limited BitOps budget. As one can see, under a tight BitOps constraint, our model improves over 2% accuracy (from 71.5% to 73.9%) compared with searched model using (Guo et al., 2019). Moreover, our models achieve the same level accuracy (75.1%) as ResNet34 full precision model while only consumes half of the BitOps as 4-bit version (from 52.83G to 25.69G).

5.3 EFFECTIVENESS OF PREDICTOR-TRANSFER

Figure 6 shows the performance of our predictor-transfer technique compared with training from scratch. For each setting, we train the predictor to convergence and evaluate the pairwise accuracy (*i.e.* the proportion that predictor correctly identifies which is better between two randomly selected candidates from a held-out dataset), which is a measurement for the predictor's performance. As shown, the transferred predictor have a higher and faster pairwise accuracy convergence. Also, when the data is very limited, our method can have more than 10% pairwise accuracy over scratch training.

6 CONCLUSION

We propose *EMS*, an end-to-end design method for architecting mixed-precision model. Unlike former works that decouple into separated stages, we directly search for the optimal mixed-precision architecture without multi-stage optimization. We use predictor-base method that can have no extra evaluation for target dataset, which greatly saves GPU hours for searching under an upcoming scenario, thus reducing marginally CO_2 emission and cloud compute cost. To tackle the problem for high expense of data collection, we propose predictor-transfer technique to make up for the limitation of data. Comparisons with state-of-the-art models show the necessity of joint optimization and prosperity of our end-to-end design method.

REFERENCES

Sajid Anwar and Wonyong Sung. Compact deep convolutional neural networks with coarse pruning, 2016. 3

Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *ICML*, 2018. 2

Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks. In *ICLR*, 2018. 2

Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *AAAI*, 2018a. 1

Han Cai, Jiacheng Yang, Weinan Zhang, Song Han, and Yong Yu. Path-level network transformation for efficient architecture search. In *ICML*, 2018b. 1

Han Cai, Chuang Gan, and Song Han. Once for all: Train one network and specialize it for efficient deployment, 2019a. 5

Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *ICLR*, 2019b. 1, 2, 3, 5, 7, 8

Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *arXiv*, 2016. 3

Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, et al. Chamnet: Towards efficient network design through platform-aware model adaptation. *CVPR*, 2019. 2, 3, 4

Thomas Elsken, Jan-Hendrik Metzen, and Frank Hutter. Simple and efficient architecture search for convolutional neural networks. *arXiv preprint arXiv:1711.04528*, 2017. 1

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018. 1

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. *arXiv preprint arXiv:1904.00420*, 2019. 2, 3, 4, 6, 7, 9

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *NeurIPS*, 2015. 2, 3

Song Han, Huizi Mao, and William Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *ICLR*, 2016a. 3

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR*, 2016b. 1, 2, 3

Song Han, Han Cai, Ligeng Zhu, Ji Lin, Kuan Wang, Zhijian Liu, and Yujun Lin. Design automation for efficient deep learning computing. *arXiv preprint arXiv:1904.10616*, 2019. 1

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. *ICCV*, 2017. 2, 3

Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *ECCV*, 2018. 1, 3, 7, 8

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 2

Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures, 2016. 3

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *CVPR*, 2018. 3

Purushotham Kamath, Abhishek Singh, and Debo Dutta. Neural architecture construction using envelopenets. *arXiv preprint arXiv:1803.06744*, 2018. 1

Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime Neural Pruning. In *NIPS*, 2017. 3

Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *ECCV*, 2018a. 1

Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. In *ICLR*, 2018b. 1

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *ICLR*, 2019a. 2, 5

Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Tim Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. *arXiv preprint arXiv:1903.10258*, 2019b. 5

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, 2017. 2, 3

Renqian Luo, Fei Tian, Tao Qin, and Tie-Yan Liu. Neural architecture optimization. In *NeurIPS*, 2018. 1, 2

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference, 2016. 3

Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *ICML*, 2018. 2

A. Polyak and L. Wolf. Channel-level acceleration of deep face representations. *IEEE Access*, 2015. 3

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net - ImageNet Classification Using Binary Convolutional Neural Networks. In *ECCV*, 2016. 3

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2018. 1, 2

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018. 1, 2, 5

Hardik Sharma, Jongse Park, Naveen Suda, Liangzhen Lai, Benson Chau, Vikas Chandra, and Hadi Esmaeilzadeh. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network. *ISCA*, Jun 2018. 7

Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *ACL*, 2019. 1, 7

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. *arXiv preprint arXiv:1807.11626v1*, 2018. 3

Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization. In *CVPR*, 2019. 1, 2, 3, 7, 8

Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*, 2019a. 3, 5

Bichen Wu, Yanghan Wang, Peizhao Zhang, Yuandong Tian, Peter Vajda, and Kurt Keutzer. Mixed precision quantization of convnets via differentiable neural architecture search, 2019b. URL https://openreview.net/forum?id=BJGVX3CqYm. 4

Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. *Lecture Notes in Computer Science*, 2018. 2, 3

Jiahui Yu and Thomas Huang. Universally slimmable networks and improved training techniques. *arXiv preprint arXiv:1903.05134*, 2019. 2

Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. In *CVPR*, 2018. 1

Aojun Zhou, Anbang Yao, Kuan Wang, and Yurong Chen. Explicit loss-error-aware quantization for low-bit deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9426–9435, 2018. 3

Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. DoReFa-Net - Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *arXiv*, 2016. 3

Chenzhuo Zhu, Song Han, Huizi Mao, and William Dally. Trained Ternary Quantization. In *ICLR*, 2017. 3

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017. 1, 4

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018. 1, 2