Any-Precision Deep Neural Networks

Haichao Yu*[1], Haoxiang Li[2], Honghui Shi[1], Thomas S. Huang[1], Gang Hua[2]

[1]IFP Group, UIUC, [2]Wormpex AI Research

Abstract

*We present Any-Precision Deep Neural Networks (**Any-Precision DNNs**), which are trained with a new method that **empowers learned DNNs to be flexible in any numerical precision during inference**. The same model in runtime can be flexibly and directly set to different bit-width, by truncating the least significant bits, to support dynamic speed and accuracy trade-off. When all layers are set to low-bits, we show that the model achieved accuracy comparable to dedicated models trained at the same precision. This nice property facilitates flexible deployment of deep learning models in real-world applications, where in practice trade-offs between model accuracy and runtime efficiency are often sought. Previous literature presents solutions to train models at each individual fixed efficiency/accuracy trade-off point. But how to produce a model flexible in runtime precision is largely unexplored. When the demand of efficiency/accuracy trade-off varies from time to time or even dynamically changes in runtime, it is infeasible to re-train models accordingly, and the storage budget may forbid keeping multiple models. Our proposed framework achieves this flexibility without performance degradation. More importantly, we demonstrate that this achievement is agnostic to model architectures. We experimentally validated our method with different deep network backbones (AlexNet-small, Resnet-20, Resnet-50) on different datasets (SVHN, Cifar-10, ImageNet) and observed consistent results. Code and models will be available at here.*

1. Introduction

While state-of-the-art deep learning models can achieve very high accuracy on various benchmarks, runtime cost is another crucial factor to consider in practice. In general, the capacity of a deep learning model is positively correlated with its complexity. As a result, accurate models mostly run slower, consume more power, and have larger memory footprint as well as model size. In practice, it is inevitable to balance efficiency and accuracy to get a good trade-off when deploying any deep learning models.

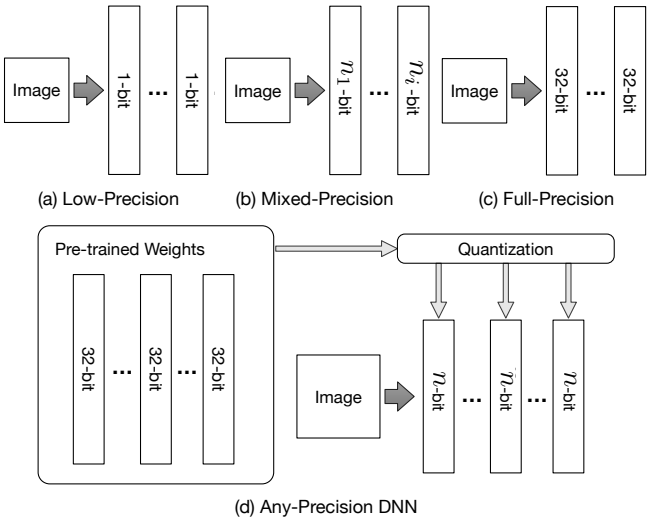

Figure 1. Illustrations of deep neural networks (DNN) in different numerical precisions: a) weights and activations of typical full-precision DNN are in 32-bit floating values; b) binary DNN with 1-bit weights and activations as an example low-precision DNN; c) different layers in mixed-precision DNN can be in arbitrary bit-width and n_i is fixed after training; d) the proposed Any-Precision DNN can have pre-trained weights in full-precision while in runtime the weights can be quantized into arbitrary bit-width n and the activations are set to be in the same bit-width accordingly.

To alleviate this issue, a number of approaches have been proposed to address it from different perspectives. We observe active researches [20, 2, 4] in looking for more efficient deep neural network architectures to support practical usage [15, 32, 26, 35]. People also consider to adaptively modify general deep learning model inference to dynamically determine the execution during the feed-forward pass to save some computation at the cost of potential accuracy drop [10, 28, 31, 29].

Besides these explorations, another important line of research proposes a low-level solution to use less bits to represent deep learning model and its runtime data to achieve largely reduced runtime cost. It has been shown in various literatures that full-precision is over-abundant in many applications that we can use 8-bit or even 4-bit models without

*This work was done during Yu's internship at Wormpex AI Research.

obvious performance degradation.

Some previous works went further in this direction. For example, BNN, XNOR-Net, and others [6, 25, 36] are proposed to use as low as 1-bit for both the weights and activations of the deep neural networks to reduce power-usage, memory-footprint, running time, and model size. However, ultra low-precision models always observe obvious accuracy drop [6]. While many methods have been proposed to improve accuracy of the low-precision models, so far we see no silver bullet. Stepping back from uniformly ultra-low precision models, mixed-precision models have been proposed to serve as a better trade-off [30, 9]. Effective ways have been found to train accurate models with some layers processing in ultra-low precision and some layers in high precision.We illustrate these different paradigms in Figure 1.

When we look at this spectrum of deep learning models in terms of its numerical precision, from full-precision at one end to low-precision at the other, and mixed-precision in between, we have to admit that efficiency/accuracy trade-off always exists in reality and to deploy a model in a specific application scenario we have to find the right trade-off point. Previous methods can provide a specific operating point but what if we demand flexibility as well? It would be a highly favorable property if we can dynamically change the efficiency/accuracy trade-off point given a single model. Preferably, we want to be able to adjust the model, without the need of re-training or re-calibration, to run in high accuracy mode when resources are sufficient and switch to low accuracy mode when resources are limited.

In this paper, we propose a method to train deep learning models to be flexible in numerical precision, namely Any-Precision deep neural networks. After training, we can freely quantize the model layers into various precision levels, without fine-tuning or calibration and without any data. When running in low-precision, full-precision or other precision levels in between, it achieves comparable accuracy to models specifically trained under the matched settings. Furthermore, given fixed computational budget, it can potentially find better operating point than one trained rigorously.

To summarize, our contributions are:

- We introduce the concept of Any-Precision DNN. In runtime we can quantize its layers into different bit-width. Its accuracy changes smoothly with respect to its precision level without drastic performance degradation;

- We propose a novel model-agnostic method to train Any-Precision DNN and validate its effectiveness with multiple widely used benchmarks and with multiple neural network architectures;

2. Related Work

Low-Precision Deep Neural Networks. Recent progresses in deep learning inference hardware motivate the research of using low-bit integer instead of float-point values to represent network weights and activations. Binarized Neural Networks [6] and XNOR-Net [25] are early works in this direction to use only 1-bit to represent the weights and activations in DNNs. When training these 1-bit networks, a float-point value copy of the parameters are maintained under the hood to calculate approximated gradients. Usually a *sign* function is used to quantize the float-point value copy to binary value in the feed-forward pass. Using only 1-bit numerical precision leads to obvious drop in accuracy in most scenarios, Zhou *et al*. [36] proposed DoReFa-Net to specifically train arbitrary bitwidth in weights, activations, and gradients. Since gradients are also in low-bits, proper implementation could accelerate both the forward and backward passes.

One of the essential problem in learning low-precision DNNs is the quantization operator. Quantization of the real-value parameters in the feed-foward pass and approximation of the gradients through the quantization operator in the backward pass heavily influence the final model accuracy. For example, the *sign* function adopted in Binarized NN [6] discards the value distribution variations across layers and hurt the performance. In XNOR-Net [25], a scaling factor is added to each layer to minimize the information loss. Choi *et al*. [5] proposed a parameterized clipping activation for quantization to support arbitrary bits quantization of activation. Zhang *et al*. [34] pointed out that having an uniform quantization pattern across layers is suboptimal and propose a learnable quantizer for each layer to improve the model accuracy.

In the backward pass, most prior works use the Straight-Through Estimator (STE) [1] to approximate the gradients over the quantizers. Cai *et al*. [3] proposed to use a half-wave gaussian quantization operator to replace the *sign* function for better learning efficiency and a piece-wise continuous function in the backpropagation step to alleviate the gradient mismatch issue in the prior design. Liu *et al*. [21] also attacked the gradient mismatch problem by introducing a piecewise polynomial function to approximate the *sign* function. Another interesting recent work from Ding *et al*. [8] addressed this problem by introducing a new loss function over the value distribution of layer activations.

Besides the performance gap to full-precision model, training binary networks have been reportedly to be unstable. Tang *et al*. [27] carefully analyzed the training process and concluded that using PReLu [11] activation function, a low learning rate, and the bipolar regularization on weights could lead to a more stable training process with better optimum. Zhuang *et al*. [38] looked at the overall training strategy and propose a progressive training process. They

suggested to first train the net with quantized weights and then quantized activations, first train with high-precision and then low-precision, and jointly train the low-bit model with the full-precision one.

A similar joint training strategy has been observed to be effective in this work as well. Since our work is along an orthogonal direction of low-precision DNNs training and design, our method can be complementary to train better and flexible DNNs.

Mixed-Precision Neural Networks. The accuracy drop of ultra low-bit models and the emergent new hardware designs motivate the research of training DNNs with mixed-precision. Although most of the knowledge from training low-precision DNNs can be transfered to mixed-precision training as well, an open question is how to specify the bit-width of each layer for both weights and activations. Given fixed computational budget, number of potential configurations are exponentially large.

Zhou *et al*. [37] proposed to find the configuration by solving an optimization problem where the prospective accuracy drop is added as the constraint. They revealed how noise on the feature map related to accuracy degradation, then estimated the effect of parameter quantization errors in individual layers on the overall model prediction accuracy. Wang *et al*. [30] proposed to use reinforcement learning to determine the quantization policy. The policy takes in layer configuration and stats as input to predict the bit-width of weights and activations. When learning the policy, the feedback from the hardware is taken into consideration through a hardware simulator generating the latency and energy signals. Dong *et al*. [9] presented an novel second-order quantization method to select the bit-width of each layer as well as the fine-tuning order of layers, based on the layer's Hessian spectrum.

Post-training Quantization. Quantization of a pre-trained model with fine-tuning or calibration on a dataset is another related research topic in the area. Although methods in this area are working on a different problem from ours, we partially share the motivation to have the flexibility of quantization control in the runtime. Without special treatment, many models collapse even in 8-bit precision in post-training quantization. One recent work from Nagel *et al*. [22] identified two issues leading to the large accuracy drop, the large variation in the weight ranges across channels and biased output errors due to quantization errors affecting following layers. With their method, they are able to alleviate the bias and equalize the weight ranges by rescaling and reparameterization. In this paper, the model we produced can be readily quantized into lower precision without further process.

In the research area of deep neural networks architecture search, the slimmable neural networks by Yu *et al*. [33] is related to ours in terms of methodology. They presented method to train a single neural network with adjustable number of channels in each layer at runtime. Their exploration is limited to the search space of network architecture instead of weights.

3. Any-Precision Deep Neural Networks

3.1. Overview

Neural networks are generally constructed layer by layer. We denote input to the i-th layer in a neural network model as $\mathbf{x_i}$, the weights of the layer as $\mathbf{w_i}$ and the biases as $\mathbf{b_i}$. The output from this layer can be calculated as

$$\mathbf{y_i} = \mathcal{F}(\mathbf{x_i}|\mathbf{w_i}, \mathbf{b_i}). \tag{1}$$

Without loss of generality, we take one channel in a fully-connected layer as a concrete example in the following description and drop the subscript i for simplicity, i.e.,

$$\mathbf{y} = \mathbf{w} \cdot \mathbf{x} + b, \tag{2}$$

where $\mathbf{y}, \mathbf{w} \in \mathbb{R}^D$ and b is a scalar.

For better computation efficiency, we would like to avoid the float-value dot product of D-dimensional vectors. Instead we use N-bit fixed-point integers to represent the weights as $\mathbf{w_Q}$ and input activations as $\mathbf{x_Q}$. Hereafter, we assume $\mathbf{w_Q}$ and $\mathbf{x_Q}$ are stored as signed integers in its bit-wise format. Note that in some related works [6], elements of $\mathbf{w_Q}$ and $\mathbf{x_Q}$ could be represented as vectors of $\{-1, 1\}$ and the conversion between these two formats are trivial. With N-bit integers weights and activations, as discussed in prior arts [36, 25], the computation can be accelerated by leveraging bit-wise operations (*and*, *xnor*, *bit-count*), or even dedicated DNN hardwares.

Early works [27] show that by adding a layer-wise real-value scaling factor s could largely help reduce the output range variation and hence achieve better model accuracy. Since the scaling factor is shared across channels within the same layer, the computational cost is fractional. Following this setting, with the quantized weights and inputs, we have

$$\mathbf{y}' = s * (\mathbf{w_Q} \cdot \mathbf{x_Q}) + b. \tag{3}$$

The activations $\mathbf{y}'$ are then quantized into N-bit fixed-point integers as the input to the next layer.

3.2. Inference

We will discuss our quantization functions in details in the next section. Here we describe the runtime of a trained Any-Precision DNN.

Once training is finished, we can keep the weights at a higher precision level for storage, for example, at 8-bit. As

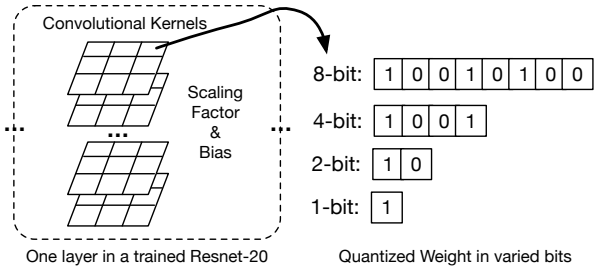

Figure 2. Quantization of a kernel weight in the trained model into different precision levels: since we follow an uniform quantization pattern, when representing weight values in signed integers, the quantization can be implemented as simple bit-shift.

shown in Figure 2, we can simply quantize the weights into lower bit-width by bit-shifting. We experimentally observe that with the proposed training framework, the model accuracy changes smoothly and consistently on-par or even outperform dedicated models trained at the same bit-width.

3.3. Training

A number of quantization functions have been proposed in the literature for weights and activations respectively. Given a pre-trained DNN model, one can quantize its weights into low-bit and apply certain quantization function to activations accordingly. However, when the number of bits gets smaller, the accuracy quickly drops due to the rough approximation in weights and large variations in activations. The most widely adopted framework to obtain low-bit model is quantization-aware training. The proposed method follows the quantization-aware training framework.

We take the same fully-connected layer as an example. In training, we maintain the float-point value weights $\mathbf{w}$ for the actual layer weights $\mathbf{w_Q}$. In the feed-forward pass, given input $\mathbf{x_Q}$, we follow Equation 3 to compute the raw output $\mathbf{y}'$. Prior arts show the importance of the batch normalization (BN) [16] layer in low-precision DNN training and we follow accordingly. $\mathbf{y}'$ is then passed into a BN layer and then quantized into $\mathbf{y^Q}$ as the input to the next layer.

Weights. We use a uniform quantization strategy similar to Zhou *et al*. [36] with a scaling factor to approximate the weights. Given the floating point weight $\mathbf{w}$, we first apply the *tanh* function to normalize it into $[-1, 1]$ and then transform it into $\mathbf{w}' \in [0, 1]$, i.e.,

$$\mathbf{w}' = \frac{\tanh(\mathbf{w})}{2\max(|\tanh(\mathbf{w})|)} + 0.5. \tag{4}$$

We then quantize normalized value into N-bit integers $\mathbf{w_Q}'$ and scaling factor s, where

$$\begin{aligned} \mathbf{w_Q}' &= \mathrm{INT}(\mathrm{round}(\mathbf{w}' * \mathrm{MAX_N})), \\ s' &= 1 \,/\, \mathrm{MAX_N}. \end{aligned} \tag{5}$$

Hereafter $\mathrm{MAX_N}$ denotes the upper-bound of N-bit integer and $\mathrm{INT}(\cdot)$ converts a floating point value into an integer.

Finally the values are re-mapped back to approximate the range of floating point values to obtain

$$\begin{aligned} \mathbf{w_Q} &= 2 * \mathbf{w_Q}' - 1, \\ s &= \mathbb{E}(|\mathbf{w}|) \,/\, \mathrm{MAX_N}, \end{aligned} \tag{6}$$

where $\mathbb{E}$ is the mean of absolute value of all floating-valued weights in the same layer. Eventually, we approximate $\mathbf{w}$ with $s * \mathbf{w_Q}$ and execute the feed-forward pass with the quantized weights as shown in Equation 3, the scaling factor can be applied after the dot-product of all integers vectors.

In the backward pass, gradients are computed with respect to the underlying float-value variable $\mathbf{w}$ and updates are applied to $\mathbf{w}$ as well. In this way, the relatively unreliable and nuance signals would be accumulated gradually and hence this will stabilize the overall training process. Since not all operations involved are nice smooth functions to support back-propagation, we use the straight through estimator (STE) [13] to approximate the gradients. For example, the *round* operation in Equation 5 has zero derivative almost everywhere. With STE, we assign $\partial \mathrm{round}(x) \,/\, \partial x \coloneqq 1$.

Activations. For activation quantization in the feed-forward pass, we obtain the N-bit fixed-point representation by first clipping the value to be within $[0, 1]$ and then

$$\begin{aligned} \mathbf{y_c}' &= \mathrm{clip}(\mathbf{y}', 0, 1), \\ \mathbf{y_Q} &= \mathrm{INT}(\mathrm{round}(\mathbf{y_c}' * \mathrm{MAX_N})) * \frac{1}{\mathrm{MAX_N}}, \end{aligned} \tag{7}$$

In practice, we only calculate the integer part as $\mathbf{y_Q}$ and absorb the constant scaling factor into the persistent network parameters in the next layer.

Let L denote the final loss function and the gradient with respect to the activation $\mathbf{y_Q}$ is then approximated to be

$$\frac{\partial L}{\partial \mathbf{y_Q}} \approx \frac{\partial L}{\partial \mathbf{y_c}'}, \tag{8}$$

where

$$\frac{\partial L}{\partial \mathbf{y_c}'} = \begin{cases} \frac{\partial L}{\partial \mathbf{y}'}, & \text{if } 0 \le \mathbf{y}' \le 1 \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

The gradient of the *round* function is approximated with STE to be 1.

Dynamic Model-wise Quantization. In prior low-precision models, the bit-width N is fixed during the training process. In runtime, if we alter N the model accuracy drops drastically. To encourage flexibility in the produced model, here we propose to dynamically change N within

the training stage to align the training and inference process. However, the distribution of activations varies under different bit-width N, especially when N is small (e.g., 1-bit), as shown in Figure 3. As a result, without special treatment, the dynamically changed N creates conflicts in learning the model that it fails to converge in our experiments.

One of the widely adopted technique to adjust internal feature/activation distribution is Batch Normalization (BatchNorm) [16]. It works by normalizing layer output across batch dimension as following

$$\hat{x_i} = \gamma \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta, \quad i = 1..B \tag{10}$$

where B is the batch size, i denotes the index within current batch, ϵ is a small value added to avoid numerical issue. μ and σ^2 are mean and variance respectively defined as

$$\mu = \sum_{i=1}^{B} x_i \text{ and } \sigma^2 = \frac{1}{B} \sum_{i=1}^{B} (x_i - \mu)^2 \tag{11}$$

During training, BatchNorm layer keeps calculating running averages for μ and σ^2, i.e.,

$$\begin{aligned} \mu &= \lambda\mu + (1-\lambda)\mu_t, \\ \sigma^2 &= \lambda\sigma^2 + (1-\lambda)\sigma_t^2, \end{aligned} \tag{12}$$

where μ_t and σ_t^2 are the values before the current update, the decay rate λ is a hyper-parameter set a-prior. But even with the BatchNorm layer, dynamically changed N will lead to failure of convergence in training due to the value distribution variations shown in the toy example in Figure 3.

In our proposed framework, we adopted **dynamically changed BatchNorm layer** to work with different N in training. More specifically, assume we have a list of bit-width candidates $\{n_k\}_{k=1}^K$, we keep $|K|$ copies of BatchNorm layer parameters and internal states $\Phi_{k=1}^K$. When the current training iteration works with $N = n_k$, we reset the BatchNorm layers with data from Φ_k to use and update the corresponded copy.

Similar technique has been adopted by Yu *et al.* [32] when dealing with varied network architectures. Parameters of all BatchNorm layers are kept after training and used in inference. Note that compared with the total number of network parameters, the additional amount from BatchNorm layers is negligible. We summarize the proposed method in Algorithm 1. With the proposed algorithm, we can train DNN being flexible for runtime bit-width adjustment.

Another optional component in our method is **adding knowledge distillation [14] in training**. Knowledge distillation works by matching the outputs of two networks. In training a network, we can use a more complicated model or an ensemble of models to produce soft targets by adjusting the temperature of the final softmax layer and then use the soft targets to guide the network learning.

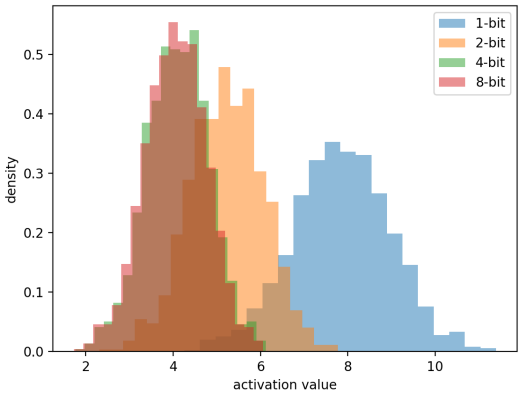

Figure 3. Activations Distributions under different bit-width for weights and inputs: in this toy example, we randomly generate a single-channel fully-connected layer and 1000 16-dimensional inputs; we then quantize the weights and inputs into $1, 2, 4, 8$ bits respectively and summarize the distributions of activations under different bit-width; as observed in the figure, the 1-bit quantization leads to significant distribution shift compared to 8-bit model and the discrepancy under 2-bit is also obvious.

Algorithm 1 Training of the proposed Any-Precision DNN

Require: Given candidate bit-widths $P \leftarrow \{n_k\}_{k=1}^K$
Initialize the model $\mathcal{M}$ with floating-value parameters
Initialize K BatchNorm layers: $\Phi_{k=1}^K$
for $t = 1, ..., T_{iters}$ **do**
  Sample data batch (x, y) from train set D_{train}
  for n_p in P **do**
    Set quantization bit-width $N \leftarrow n_p$
    Feed-forward pass: $y_{n_p} \leftarrow \mathcal{M}(x)$
    Set BatchNorm layers: $\mathcal{M}$.replace(Φ_p)
    $L \leftarrow L + loss(y_{n_p}, y_{gt})$
  end for
  Back-propagate to update network parameters
end for

In our framework, we apply this idea by generating soft targets from a high-precision model. More specifically, in each training iteration, we first set the quantization bit-width to the highest candidate n_K and run feed-forward pass to obtain soft targets y_{soft}. Later, instead of accumulating cross-entropy loss for each precision candidate, we use KL divergence of the model prediction and y_{soft} as the loss. In our experiments, we observe that in general knowledge distillation leads to better performance at 1-bit precision level.

4. Experiments

We validate our method with several network architectures and datasets. These networks include a 8-layer CNN

Dataset	Class Number	Image Number (Train/Test)
Cifar-10 [18]	10	50k/10k
SVHN [23]	10	604k/26k
ImageNet [7]	1000	1.3M/50k

Table 1. Details of datasets used in our experiments.

(a) Cifar-10 (b) SVHN (c) ImageNet

Figure 4. Example images from datasets used in our experiments.

(named Model C in [36]), AlexNet [19], Resnet-8 [12], Resnet-20, and Resnet-50 [12]. The datasets include Cifar-10 [18], Street View House Numbers (SVHN) [23], and ImageNet [7]. In Table 1 and Figure 4, we show details of these datasets.

4.1. Detailed Settings

We implement the whole framework in PyTorch [24]. On Cifar-10, we train AlexNet and Resnet-20 models for 200 epochs with initial learning rate 0.001 and decayed by 0.1 at epochs $\{100, 150, 280\}$. On SVHN, the 8-layer CNN and Resnet-8 models are trained for 100 epochs with initial learning rate 0.001 and decayed by 0.1 at epochs $\{50, 75, 90\}$. We combine the training and extra training data on SVHN as our training dataset. All models on Cifar-10 and SVHN are optimized with the Adam optimizer [17] without weight decay. On ImageNet, we train Resnet-50 model for 120 epochs with initial learning rate 0.1 decayed by 0.1 at epochs $\{30, 60, 85, 95, 105\}$ with SGD optimizer.

For all models, following Zhou *et al*. [36] we keep first and last layer be real-valued. In training, we train the networks with bit-width candidates $\{1, 2, 4, 8, 32\}$. Note that when the bit-width is set to 32, it is a full-precision model that we use floating-valued weights and activations. In testing, we evaluate the model runs at each bit-width in the list respectively. By default, we add knowledge distillation (KD) in training, we use full-precision model to get soft targets as supervision in the low-precision iterations.

4.2. Comparison to Baseline Methods

We compare our method to very competitive baseline models at each precision level. For each bit-width we tested, we dedicatedly train a low-precision model following the same training pipeline with fixed bit-width for weights and activations. We compare the accuracy we obtained from our dedicated low-bit models to other recent works in this field to make sure the baseline models are competitive. For example, on Cifar-10, our 1-bit model achieves an accuracy of 89.99% while the recent work from Ding *et al*. [8] reports 89.90%.

Cifar-10					
Models	1 bit	2 bit	4 bit	8 bit	FP32
Resnet-20	89.99	92.66	92.69	93.20	92.92
Ours	90.20	92.21	92.35	92.28	92.27
AlexNet	92.56	94.06	94.02	93.82	93.74
Ours	93.00	94.08	94.26	94.24	94.22
SVHN					
Models	1 bit	2 bit	4 bit	8 bit	FP32
Resnet-8	92.94	95.91	95.15	94.64	94.60
Ours	91.65	94.78	95.46	95.39	95.36
8-layer CNN	90.74	96.24	97.03	97.03	97.10
Ours	88.72	95.17	96.43	96.56	96.46
ImageNet					
Models	1 bit	2 bit	4 bit	8 bit	FP32
Resnet-50	55.71	65.70	72.17	74.31	76.38
Ours	55.3	65.20	71.33	73.50	75.21

Table 2. Comparison of the proposed Any-Precision DNN to dedicated models: the proposed method achieved the strong baseline accuracy in most cases, even occasionally outperforms the baselines in low-precision. We hypothesize that the gain is mainly from the knowledge distillation from high-precision models in training.

As shown in Table 2, on all three datasets, the proposed Any-Precision DNN achieves comparable performance to the competitive dedicated models.

4.3. Post-Training Quantization Methods

We compare our method to alternative post-training quantization methods. We experiment with Resnet-20 on Cifar-10. We evaluated two post-training strategies.

The first one directly quantizes dedicated models with bit-shifting. In other words, to obtain an $(n-1)$-bit model from a trained n-bit model, as what we do with Any-Precision DNN shown in Figure 2, we simply drop the least-significant bit of all weights. With no surprise, this strategy fails dramatically on challenging large-scale benchmark as shown in Table 4. On smaller dataset Cifar-10, when quantizing models into very low bit-width accuracy drops a lot but when the target runtime bit-width is higher than 3, the simple strategy shows to be effective as well (Table 3). We argue that this is because Resnet-20 has a relatively large capacity on Cifar-10 that rough numerical precision works as well.

The second strategy follows the same bit-shifting to drop bit with an added BatchNorm calibration process. In the calibration process, BatchNorm statistics will be re-calculated by feed-forwarding a number of training samples. As shown in Table 3 and Table 4, the BatchNorm cal-

Methods \ Runtime bit-width	1 bit	2 bit	3 bit	4 bit	5 bit	6 bit	7 bit	8 bit	FP32
Ours	89.69	92.75	93.13	93.16	92.97	92.90	92.99	92.91	93.04
Quantize Dedicated Models with bit-shifting									
From 1-bit	89.79	-	-	-	-	-	-	-	-
From 2-bit	10.32	92.79	-	-	-	-	-	-	-
From 4-bit	10.87	64.76	92.08	92.81	-	-	-	-	-
From 8-bit	10.50	30.61	92.00	92.70	92.92	92.88	92.88	92.97	-
From FP32	10.92	36.38	92.19	92.73	93.10	92.99	93.01	93.04	93.20
Quantize Dedicated Models with bit-shifting and BatchNorm calibration									
From 1-bit	89.79	-	-	-	-	-	-	-	-
From 2-bit	79.11	92.79	-	-	-	-	-	-	-
From 4-bit	54.97	89.35	92.64	92.81	-	-	-	-	-
From 8-bit	50.76	88.35	92.56	92.80	92.94	93.00	93.01	92.97	-
From FP32	51.0	88.26	92.72	92.84	92.95	92.95	92.96	93.04	93.20

Table 3. Comparison to other post-training quantization methods: All models are Resnet-20 trained on Cifar-10. When bit-width drops from their original training setting, our method consistently outperforms them.

Runtime bit-width	1 bit	2 bit	4 bit	8 bit
Ours	55.3	65.20	71.33	73.50
Quantize Dedicated Models with Bit-shifting				
From 8-bit model	0.104	0.116	0.436	74.3
After BatchNorm Calibration				
From 8-bit model	0.164	0.126	6.91	74.3

Table 4. Comparison to other post-training quantization methods: All models are Resnet-50 trained on ImageNet. When bit-width drops from their original training setting, our method consistently outperform them.

Test \ Train	1,2,4,8	1,8	2,8	4,8
1-bit	90.51	90.05	75.00	53.96
2-bit	92.67	91.24	92.64	90.05
3-bit	92.90	92.33	92.87	92.77
4-bit	93.00	92.42	93.1	93.19
5-bit	93.18	93.31	93.15	93.04
6-bit	93.13	92.48	93.01	93.07
7-bit	93.09	92.45	93.17	93.08
8-bit	93.18	92.42	93.15	93.01

Table 5. Classification accuracy of Resnet-20 with different bit-width combinations in training on Cifar-10.

ibration helps a lot in low-bit settings. However, the accuracy is still much lower than the ones from our method. We can leverage this post-training calibration technique with the proposed framework to fill-in the gaps of training candidate bit-width list, i.e., after training for 1,2,4,8-bits precision levels, we can further calibrate the model under the remaining 3,5,6,7-bit settings to get the missed copies of BatchNorm layer parameters. So that, in runtime, we can freely choose any precision level from 1 to 8 bits.

4.4. Dynamically Changed BatchNorm Layers

To understand how the dynamically changed BatchNorm layers help in our framework, we visualize the activation value distributions of several layers of an Any-Precision AlexNet. More specifically, we look at how activation value distribution changes from the 2nd, 4th, to the 6th convolutional layers and the BatchNorm layers after them when the runtime precision level is set to 1,2,4,8-bit respectively. As shown in Figure 5, when running at 1-bit precision, the activation distribution is obviously off from others after the convolutional layers; the followed BatchNorm layer rectifies the distributions; then the next convolutional layer would create this distribution variation again. It is very clear that by keeping multiple copies of the BatchNorm layer parameters for different bit-width, we can minimize input variations to the convolutional layers and hence have the same set of convolutional layer parameters to support Any-Precision in runtime.

4.5. Ablation Studies

Candidate bit-width List. We study how the candidate bit-width list used in training the Any-Precision DNN influence the testing performance on other bit-widths.

Table 5 shows test accuracy of models trained under different bit-width combinations. We observe that training with more candidate bit-width generally lead to better generalization to the others and the candidate bit-width list is better to cover the extreme cases in runtime. For example, the 1,8-bits combination performs more stable across different runtime bit-width compared with 2,8-bits and 4,8-bits combination. Since better coverage in training takes longer for the model to converge, this observation can guide the bit-width selection under limited training resources.

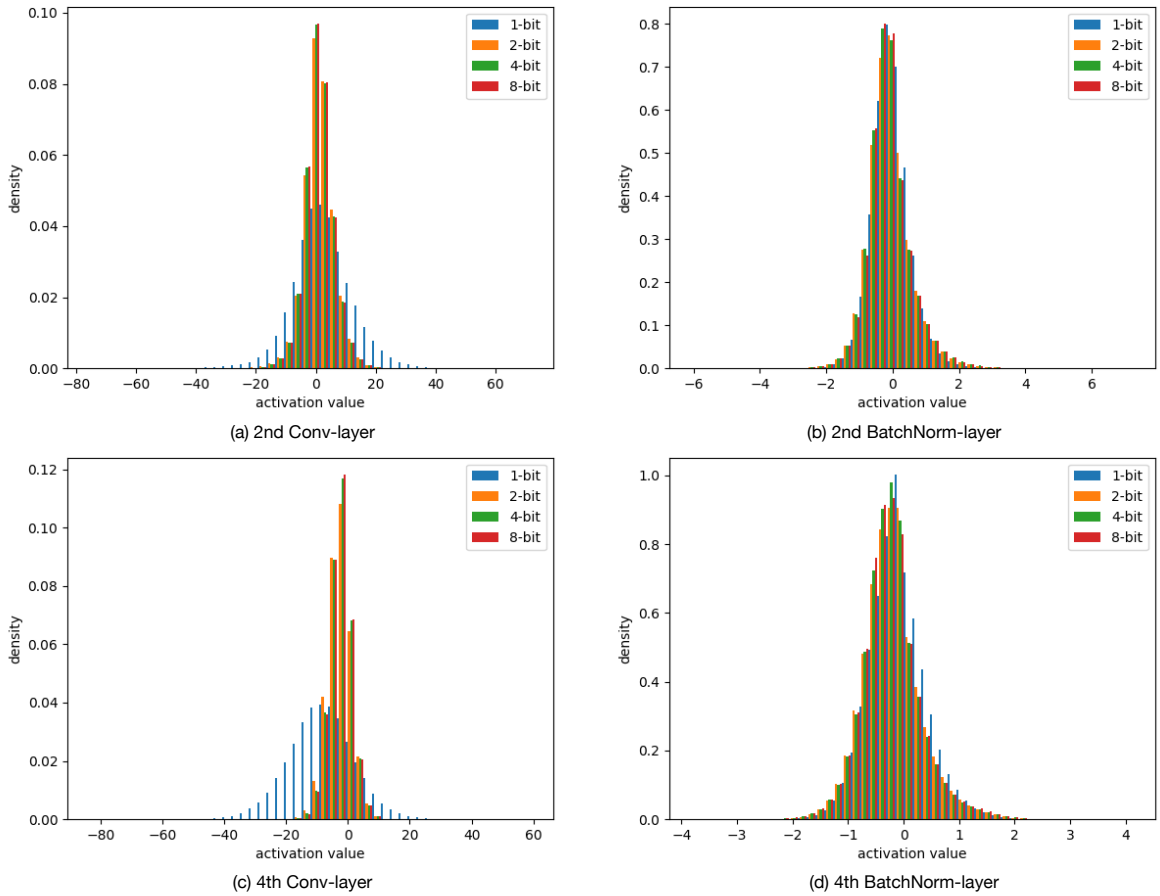

Figure 5. Activation value distributions of several layers in an Any-Precision AlexNet: low-bit quantization leads to value distribution different from others after convolutional layers but accordingly changed BatchNorm layer could rectify the mis-match.

Models	1 bit	2 bit	4 bit	8 bit	FP32
Dedicated	92.94	95.91	95.15	94.64	94.60
w/o KD	89.12	95.15	94.90	94.88	94.89
KD	91.65	94.78	95.46	95.39	95.36
KD recursive	91.50	95.17	95.38	95.32	95.30

Table 6. Classification accuracy of Resnet-8 with different knowledge distillation on SVHN test set.

Knowledge Distillation. We study the influence of different knowledge distillation strategies:

- *w/o KD*: no KD is used as shown in Algorithm 1;
- *KD*: the highest bit-width outputs are supervised by groundtruth. The others are supervised by the soft targets from the highest bit-width.
- *KD recursive*: the highest bit-width outputs are supervised by groundtruth. Then every other bit-width outputs are supervised by the soft targets from the nearest superior bit-width.

In Table 6, we observed in general KD helps improve 1-bit performance and it is better than the one *w/o KD*. An interesting observation is that KD variations even slightly outperform the dedicated models in 4,8-bit and full-precision. The hypothesis is that by jointly training, the KD losses from low-bit regularizes the training to avoid overfitting.

5. Conclusion

In this paper, we introduce Any-Precision DNN to address the practical efficiency/accuracy trade-off dilemma from a new perspective. Instead of seeking for a better operating point, we enable runtime adjustment of model precision-level to support flexible efficiency/accuracy trade-off without additional storage or computation cost. The model can be stored at 8-bit or higher and run in lower bit-width such as 1-bit or 2-bit. The model accuracy drops gracefully when bit-width gets smaller. To train an Any-Precision DNN, we propose to have dynamic model-wise quantization in training and employ dynamically changed BatchNorm layers to align activation distributions across different bit-width. We evaluate our method on three major image classification datasets with multiple network architectures. When running in low-bit by simply bit-shifting the pre-trained weights and quantizing the activations, our model achieves comparable accuracy to dedicatedly trained low-precision models.

References

[1] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 2

[2] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018. 1

[3] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. Deep learning with low precision by half-wave gaussian quantization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5918–5926, 2017. 2

[4] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. *arXiv preprint arXiv:1904.12760*, 2019. 1

[5] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018. 2

[6] 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 preprint arXiv:1602.02830*, 2016. 2, 3

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009. 6

[8] Ruizhou Ding, Ting-Wu Chin, Zeye Liu, and Diana Marculescu. Regularizing activation distribution for training binarized deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11408–11417, 2019. 2, 6

[9] Zhen Dong, Zhewei Yao, Amir Gholami, Michael Mahoney, and Kurt Keutzer. Hawq: Hessian aware quantization of neural networks with mixed-precision. *arXiv preprint arXiv:1905.03696*, 2019. 2, 3

[10] Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry Vetrov, and Ruslan Salakhutdinov. Spatially adaptive computation time for residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1039–1048, 2017. 1

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. 2

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 6

[13] Geoffrey Hinton. Neural networks for machine learning, 2012. 4

[14] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 5

[15] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. *arXiv preprint arXiv:1905.02244*, 2019. 1

[16] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 4, 5

[17] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6

[18] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. 6

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 6

[20] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018. 1

[21] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 722–737, 2018. 2

[22] Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. Data-free quantization through weight equalization and bias correction. *arXiv preprint arXiv:1906.04721*, 2019. 3

[23] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011. 6

[24] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017. 6

[25] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016. 2, 3

[26] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019. 1

[27] Wei Tang, Gang Hua, and Liang Wang. How to train a compact binary neural network with high accuracy? In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017. 2, 3

[28] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from

deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469. IEEE, 2016. 1

[29] Andreas Veit and Serge Belongie. Convolutional networks with adaptive inference graphs. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–18, 2018. 1

[30] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8612–8620, 2019. 2, 3

[31] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8817–8826, 2018. 1

[32] Jiahui Yu and Thomas Huang. Universally slimmable networks and improved training techniques. *arXiv preprint arXiv:1903.05134*, 2019. 1, 5

[33] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. *arXiv preprint arXiv:1812.08928*, 2018. 3

[34] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 365–382, 2018. 2

[35] Xiaofan Zhang, Haoming Lu, Cong Hao, Jiachen Li, Bowen Cheng, Yuhong Li, Kyle Rupnow, Jinjun Xiong, Thomas Huang, Honghui Shi, et al. Skynet: a hardware-efficient method for object detection and tracking on embedded systems. *arXiv preprint arXiv:1909.09709*, 2019. 1

[36] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016. 2, 3, 4, 6

[37] Yiren Zhou, Seyed-Mohsen Moosavi-Dezfooli, Ngai-Man Cheung, and Pascal Frossard. Adaptive quantization for deep neural network. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 3

[38] Bohan Zhuang, Chunhua Shen, Mingkui Tan, Lingqiao Liu, and Ian Reid. Towards effective low-bitwidth convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7920–7928, 2018. 2