Training for Multi-resolution Inference using Reusable Quantization Terms

Sai Qian Zhang
Harvard University, USA
zhangs@g.harvard.edu

Bradley McDanel
Franklin and Marshall College, USA
bmcdanel@fandm.edu

H. T. Kung
Harvard University, USA
kung@harvard.edu

Xin Dong
Harvard University, USA
xindong@g.harvard.edu

ABSTRACT

Low-resolution uniform quantization (e.g., 4-bit bitwidth) for both Deep Neural Network (DNN) weights and data has emerged as an important technique for efficient inference. Departing from conventional quantization, we describe a novel training approach to support inference at multiple resolutions by reusing a single set of quantization terms (the same set of nonzero bits in values). The proposed approach streamlines the training and supports dynamic selection of resolution levels during inference. We evaluate the method on a diverse range of applications including multiple CNNs on ImageNet, an LSTM on Wikitext-2, and YOLO-v5 on COCO. We show that models resulting from our multi-resolution training can support up to 10 resolutions with only a moderate performance reduction (e.g., $\leq 1\%$) compared to training them individually. Lastly, using an FPGA, we compare our multi-resolution multiplier-accumulator (mMAC) against other conventional MAC designs and evaluate the inference performance. We show that the mMAC design broadens the choices in trading off cost, efficiency, and latency across a range of computational budgets.

CCS CONCEPTS

• **Computer systems organization** → **Neural networks**; **Reconfigurable computing**; **Data flow architectures**; **Systolic arrays**; • **Computing methodologies** → **Artificial intelligence**.

KEYWORDS

Multi-resolution inference; deep neural networks; systolic arrays; quantization; joint-optimization training; co-design

ACM Reference Format:
Sai Qian Zhang, Bradley McDanel, H. T. Kung, and Xin Dong. 2021. Training for Multi-resolution Inference using Reusable Quantization Terms. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '21), April 19–23, 2021, Virtual, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3445814.3446741

ASPLOS '21, April 19–23, 2021, Virtual, USA

ACM ISBN 978-1-4503-8317-2/21/04…$15.00
https://doi.org/10.1145/3445814.3446741

1 INTRODUCTION

Deep Neural Network (DNN) quantization has received much attention in recent years due to its potential to address the growing computational costs of DNN training and inference. By quantizing a 32-bit full-precision models to a lower resolution (*e.g.,* 8-bit fixed-point), quantized DNNs can be implemented using more efficient hardware with less costly memory access, such as 8-bit Multiplier-Accumulator (MAC) units instead of floating-point units, leading to significant improvements in energy efficiency, latency, and storage requirements. However, there is generally a trade-off between the hardware cost and the performance (*e.g.,*classification accuracy) of a DNN under a fixed quantization regime. Due to this trade-off, quantization methods often present multiple distinct models trained at different precisions (*e.g.,* from 8-bit to 4-bit fixed-point bitwidths) that achieve varying degrees of performance/cost trade-off.

However, conventional hardware is typically designed for a single precision, such as 8-bit fixed-point, and therefore can not efficiently implement quantized models across a wide range of precisions to support such a performance/cost trade-off. For instance, while an 8-bit MAC can be used to multiply two 4-bit numbers, the upper-half of the MAC will only be multiplying zero bits.

To address this concern, we present a multi-resolution Multiplier-Accumulator (mMAC) design that allows a hardware platform to support efficiently multiple resolutions at runtime. The mMAC operates on only the nonzero power-of-two terms in a value. For example, for the value $20 = 00010100_2$, mMAC only operates on the 2^4 and 2^2 terms, corresponding to the two nonzero bits in the unsigned binary representation of the value. Our approach also generalizes to signed-digit representations (SDRs), such as Booth Encoding [4], which can have both positive and negative terms. Consequently, the mMAC takes fewer cycles to multiply lower-resolution numbers (*i.e.,* numbers with fewer nonzero terms) and more cycles to multiply higher-resolution numbers with more nonzero terms. Note that, unlike conventional uniform quantization where resolution is tied to the bitwidth of the representation, in this paper we generalize the notion of resolution of a value to be the allowed number of nonzero power-of-two terms in the value, regardless of their positions in the encoding.

To support multi-resolution inference, we have developed a multi-resolution DNN training approach that jointly optimizes many sub-models across a wide range of resolutions (illustrated on the left of Figure 1). The result of this joint-optimization training is a meta multi-resolution model capable of spawning sub-models at multiple resolutions at runtime. These sub-models have two novel

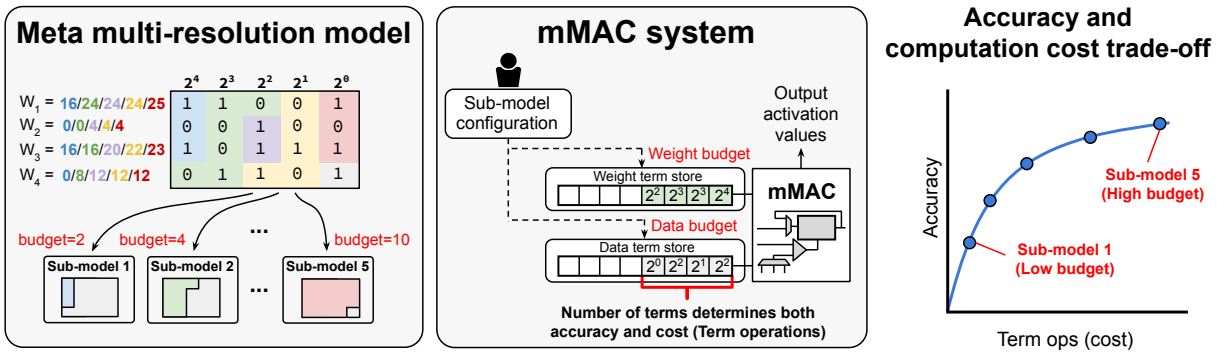

Figure 1: A multi-resolution model (left) contains multiple sub-models with varying power-of-two term budgets leading to different degrees of quantization. Here, a group of 4 weight values (25, 4, 23, 13) are quantized with 5 different term budgets. In a hardware deployment (middle) the multi-resolution model is implemented using a multi-resolution Multiplier-Accumulator (mMAC) which all sub-models can share. The proposed multi-resolution approach enables an efficient cost/performance trade-off to suit the current runtime conditions (right) by selecting the appropriate sub-model with a corresponding resolution.

properties: **storage sharing** across the sub-models, as the same nonzero terms (which need not be adjacent) for lower-resolution sub-models also appear in higher-resolution sub-models, and **computation sharing** as all sub-models can share the same mMAC computation engine. In both training and inference, for the same set of DNN weights, we simply adjust the number of leading nonzero terms to implement different quantization resolutions (*i.e.,* sub-models). Thus, the larger quantization terms in a weight are reused across multiple resolutions.

The multi-resolution model is deployed on the proposed mMAC system, shown in the middle of Figure 1. A user (or other selection mechanism) can select which sub-model to use based on the current resource constraints in the performance/cost trade-off space (right of Figure 1). Configuring the system to use a low-budget sub-models is achieved by simply dropping more low-order power-of-two terms from the meta multi-resolution model. Since a low-resolution sub-model has fewer terms, the hardware system using mMACs will perform the inference computation at a lower computation cost, thereby at an increased computation rate. The multi-resolution joint-optimization training procedure is critical in speeding up the training of sub-models as well as ensuring low hardware cost and high accuracy of the resulting sub-models. The main contributions of the paper are:

- A *multi-resolution training paradigm* that supports efficient joint-optimization training of multiple sub-models that share power-of-two terms. The method uses a teacher-student learning approach to train only two of the target sub-models at each iteration. The resulting sub-models achieve a trade-off in accuracy and number of term operations (cost).
- *Sub-model configuration* at inference to meet the current resource constraints at deployment which translates to simply adjusting the number of leading terms to use in learned weights. The multi-resolution model has a low memory footprint, as all sub-model instances of a large model share the same leading power-of-two terms of the latter across the model weights. This also enables efficient memory access of only the currently required terms given a specified term budget.
- A multi-resolution hardware system with mMAC for supporting field-configurable multi-resolution DNN inference. The mMAC computes dot products by processing only the nonzero terms in weight and data values.

The remainder of the paper is organized as follows: Section 2 presents the background and related work on DNN pruning, DNN quantization and binary encoding schemes. Section 3 provides an overview of term quantization and its application for dot-product computation. In Section 4, we describe the joint-optimization training strategy to produce a meta multi-resolution DNN model. The hardware design of mMAC for multi-resolution inference system is described in Section 5. The evaluation results on the accuracy performance of the multi-resolution model are given in Section 6, followed by the hardware evaluation in Section 7. We have released the code for multi-resolution training and its implementation at https://github.com/saizhang0218/Multi-resolution-Inference.

2 BACKGROUND AND RELATED WORK

In Section 2.1, we review related work on DNNs that can dynamically reduce computation costs under a small or negligible decrease of model performance (*e.g.,* accuracy). After that, we discuss various pruning techniques in Section 2.2 and quantization techniques in Section 2.3 used for DNNs acceleration. In Section 2.4, we provide an overview of the Signed-digit Representation (SDR), which we use instead of the conventional Unsigned Binary Representation (UBR), and review specialized hardware accelerators which employ SDR for DNN inference. Finally, in Section 2.5, we illustrate how matrix-matrix multiplication is performed with those systolic arrays which are used in evaluating the multi-resolution inference method of this paper.

2.1 DNNs with Performance/Cost Trade-off

In recent years, there has been a trend towards designing dynamic neural network to achieve an on-demand performance/cost trade-off. Many approaches achieve this trade-off by skipping parts of the

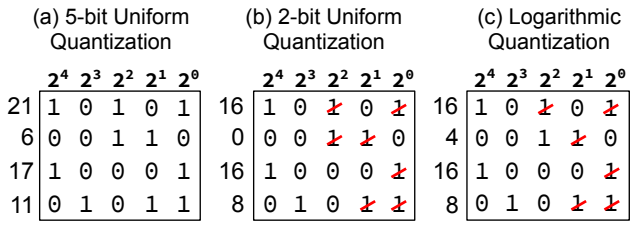

Figure 2: (a) Four values under 5-bit uniform quantization. (b) 2-bit uniform quantization takes only the two leading terms (2^4 and 2^3) of the 5-bit uniform quantization in (a). (c) Logarithmic quantization uses only the largest term in each value.

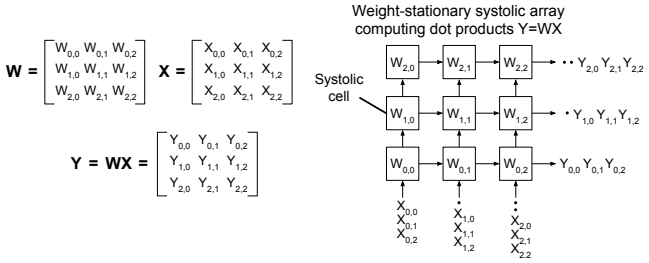

Figure 3: Weight-stationary systolic array for matrix-matrix multiplication.

inference computation based on the complexity of the input sample. In [27, 43, 53], the authors add early-exit branches to a DNN, so that easier input samples can exit at an earlier point in the network with high confidence. BlockDrop [57] and SkipNet [55] selectively drop the convolutional blocks in ResNet architecture [23] on a per-sample basis.

In [58, 59], the authors propose approaches to train a single model which can generate sub-models with varying number of channels in each convolutional layer. During runtime, the number of activated channels in a convolutional layer can be adjusted dynamically based on the on-device resource budget. Once-For-All [5] allows for a significantly larger number of DNN architectural settings by exploring a greater design space (*e.g.,* depth, width, kernel size, and resolution) and using a teacher-student training paradigm. In contrast to these works, where the derived sub-models share **weight values**, our work proposes a multi-resolution DNN approach by sharing **weight terms**. Our approach allows for greater sharing flexibility in weight representations and therefore a better performance/cost trade-off.

2.2 Pruning Techniques

There have been significant research efforts on exploiting sparsity presented in DNN weights and activations [9, 21, 22, 41] as performing multiplication with zero operands can be viewed as wasted computation. Some of these methods take advantage of the latent sparsity (*e.g.,* in CNN activations) of pre-trained models [2, 7, 20, 21, 41, 47]. Other methods introduce mechanisms at training time to make the computation sparser [22, 24, 28, 40, 56], and for more efficient deployment in hardware [18, 46, 49]. For instance, column combining [35] performs pruning on groups of values in order to achieve denser representations when deployed in systolic arrays. Regardless of the approach, these methods work at the *value level*: either a value is nonzero, requiring a fixed amount of computation time to process, or the value is zero, meaning that the associated computation can be skipped.

2.3 Quantization Techniques

Quantization [11, 12, 19, 29, 33, 48, 62] has been studied extensively for reducing the associated storage, memory access, and computation costs of DNNs. Several post-training quantization methods have been proposed to quantize floating-point weights after training, using 16-bit and 8-bit uniform quantization (UQ), without dramatically impacting classification accuracy [15, 19]. Recently, low-precision UQ approaches (*e.g.,* with less than 4 bits per value) have also been studied. To mitigate the accuracy degradation caused by low-precision weight and data, additional quantization-aware training [30] is required to fine-tune the model weights. Figure 2(a)-(b) show 5-bit UQ and 2-bit UQ applied on four values.

Logarithmic quantization (LQ) is a more aggressive form of quantization that works by rounding each value to the nearest power-of-two term as shown in Figure 2(c) [45, 61]. This allows for significantly more efficient inference, as fixed-point multiplication in UQ can be replaced with bit shift operations as each value has only a single power-of-two term. However, due to the aggressive form of quantization, LQ typically suffers from larger accuracy degradation than UQ, as the resolution decreases exponentially when the values gets larger.

Power-of-two **term quantization** (TQ) relaxes LQ by allowing a term budget of one or more terms for values [36]. Unlike prior work that applied TQ in a post-training quantization fashion [36], in this work, we propose a multi-resolution training paradigm using TQ as our quantization function for weights and data. We discuss TQ in greater detail in Section 3, as it is the quantization scheme used throughout this paper.

2.4 Techniques of Reducing Term Operations

Unsigned Binary Representations (UBRs) are a commonly used positional encoding system, where each position is either 0 or 1. By comparison, in Signed-digit Representations (SDRs), each position can have a coefficient of $\{-1, 0, 1\}$. Note that this means each digit in an SDR requires two bits: a sign bit and a $0/1$ bit. Allowing each position to have a negative coefficient leads to representations with fewer nonzero digits. For instance, 27 (11011 in UBR with four nonzero digits) can be represented as $100\bar{1}0\bar{1}$ in SDR (three nonzero digits).[1] SDRs have been extensively studied in the past [3, 4, 16], but have received less attention in relation to the literature of DNN hardware accelerators [6–8, 17, 21, 35, 42, 47, 60].

Some recent works [1, 13, 36, 51] have observed that the distribution of DNN weight values, leading to most values having few nonzero terms, can be exploited to reduce the computational cost of DNN inference. Specifically, multiplication between a weight

[1]Here, $\bar{1}$ represents a negative coefficient (*e.g.,* -2^2 or -2^0 in the example).

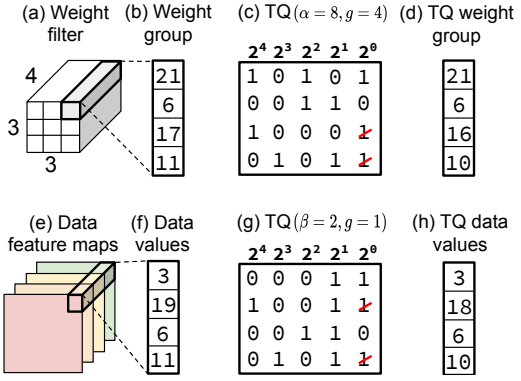

Figure 4: Term quantization (TQ) applied to a group of size $g = 4$ filter weights (top) and individual data values (bottom).

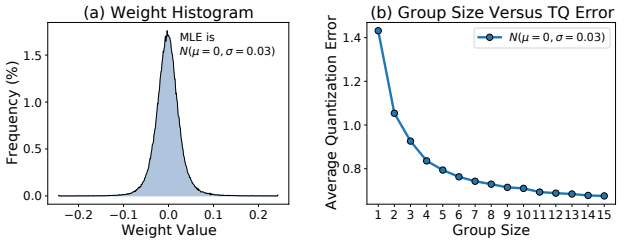

Figure 5: (a) The distribution of weights in the 13th convolutional layer in ResNet-18 [23]. (b) The quantization error using Term Quantization (TQ) with an average term budget of one term per value as the group size varies from 1 to 15.

and data value can be decomposed into exponent additions between power-of-two term-pairs in the two values. Through this view, SDRs are an attractive representation as they reduce the number of terms in each value, leading to fewer term-pairs. (In fact, it is known in the literature that SDR can achieve the minimum number of terms [31]). Our proposed multi-resolution hardware architecture (Section 5) supports weight and data values encoded in SDRs.

2.5 Systolic Arrays for Matrix-Matrix Multiplication

The majority of computational workload of DNN inference involves matrix-matrix multiplication between a trained weight matrix and input data matrix. Systolic architectures are known to efficiently implement the matrix-matrix multiplication due to their regular structures, dataflow architectures and reduced memory access [34]. In the evaluation, we adopt a *weight-stationary* systolic array for matrix-matrix multiplication computation, where the weight matrix W is pre-stored in systolic cells of the systolic array before the operation begins. During the operation, the input data are passed into the systolic array from bottom in a skewed fashion. The matrix product will be produced at the right end of each row of the systolic array (Figure 3). In Section 5, we describe a DNN inference system that uses a systolic array of mMAC systolic cells.

3 TERM QUANTIZATION

In this section, we provide an overview of term quantization (TQ), which we use in training a multi-resolution DNN described in Section 4. First, in Section 3.1, we show how we apply TQ to group of weights in a DNN and discuss the associated quantization error based on the distribution of weights. Then, in Section 3.2, we illustrate how we apply TQ to individual data values. Finally, in Section 3.3, we discuss the computational motivation for bounding the number of terms in weight and data values.

3.1 Term Quantization on Weight Groups

TQ is a new quantization technique, proposed in [36], applicable across a group of values (with group size g) by keeping the leading α terms across all values in the group. Figure 4(a)-(b) show how four weights in a convolutional weight filter can form a group to be processed by TQ. After applying TQ with a term budget of $\alpha = 8$ (Figure 4(c)), two of the 2^0 terms are dropped such that only 8 nonzero terms remain across the values in the group. Figure 4(d) shows the weight group values after TQ.

Compared with rigid quantization schemes, such as uniform quantization, where the terms can only be placed at certain digit location, TQ allows for a much greater flexibility in allocating terms across the group. By allocating more terms to larger values in the group and fewer terms to smaller values in the group, TQ achieves a much lower quantization error. This term allocation property is especially useful for DNN weights, when they are well approximated by a normal distribution. Figure 5(a) shows a histogram of the weights in the 13th convolution layer in ResNet-18 [23]. Applying a Maximum Likelihood Estimate (MLE) of a 1D normal distribution gives $N(0, 0.03)$. Other layers in ResNet-18 follow a similar distribution with the standard deviation σ between 0.01 and 0.04. Figure 5(b) shows the average quantization error due to TQ for samples drawn from a zero-mean normal distribution, with σ of 0.03, as a function of the TQ group size. The quantization error rapidly decreases as the group size goes from 1 to 4 and becomes flatter as the group size approaches 15. Based on this analysis, we use a group size of 16 for the rest of the paper, as it achieves most of the benefit from weight grouping. As discussed later in Section 5, there is a significant hardware cost associated with in making the group too large due to the increasing size of a multiplexer in the mMAC based on the group size.

3.2 Term Quantization for Data Values

TQ can also be applied to an individual value by keeping the leading β power-of-two terms in it. In this paper, we make a simplifying assumption that when applying TQ to data values, we use group size $g = 1$. That is, for data values we apply TQ to individual data values. The bottom of Figure 4(e)-(h) illustrate how TQ is applied to values in data feature maps of a CNN with a data term budget $\beta = 2$. As shown in the figure, for the value 19 (10011), TQ with a term budget $\beta = 2$ would quantize it to 18 (10010) by dropping the smallest 2^0 term (denoted by the red slash). Unlike UQ, which always truncates the low-order terms (*e.g.,* 2^0) by reducing the bitwidth, TQ maintains a larger bitwidth but reduces the number of 'active' terms in each value up to the term budget β. In Section 3.3,

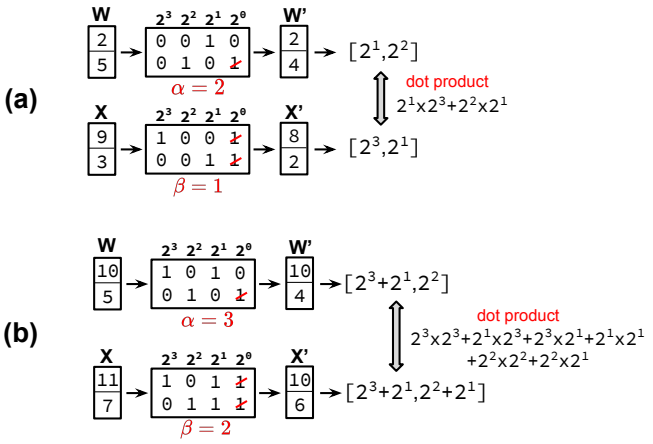

Figure 6: (a) and (b) show two examples on the term operations to compute the dot product between a group of two term-quantized weights and data. In (a), the weight term budget α and data term budget are set to $\alpha = 2$ and $\beta = 1$, resulting $\gamma = \alpha \times \beta = 2$ term-pair multiplications. In (b), the weight term budget α and data term budget are set to $\alpha = 3$ and $\beta = 2$, resulting $\gamma = \alpha \times \beta = 6$ term-pair multiplications.

we discuss the computation implications of limiting the number of nonzero terms in data values.

3.3 Term-Pair Multiplication under TQ

Suppose we consider the dot-product computation between a group of weights and a group of data values. We use the following terminologies and notations throughout the paper:

- *group size* (g)
- *weight term budget* (α) for a group of weights
- *data term budget* (β) for each data item (*i.e.,* $g = 1$)
- *term-pair budget* (γ) is $\alpha \times \beta$

We note that the dot-product computation between weights and data under TQ involves γ *term-pair multiplications*. Each term-pair multiplication amounts to an addition of the exponents in the two terms. Two examples of such dot-product computation are given in Figure 6. In Figure 6 (a), assume TQ with a weight term budget $\alpha = 2$ and data term budget $\beta = 1$ is applied on a weight group $\mathbf{W} = [w_1, w_2] = [2, 5] = [2^1, 2^2 + 2^0]$ and data values $\mathbf{X} = [x_1, x_2] = [9, 3] = [2^3 + 2^0, 2^1 + 2^0]$, respectively. This produces the term-quantized weights $\mathbf{W'} = [2, 4] = [2^1, 2^2]$ and the term-quantized data $\mathbf{X'} = [8, 2] = [2^3, 2^1]$. The dot product between $\mathbf{W'}$ and $\mathbf{X'}$ is then computed by summing the following term-pair multiplication results: $2^1 \times 2^3 + 2^2 \times 2^1 = 2^4 + 2^3 = 24$. This dot product requires $\gamma = \alpha\beta = 2$ term-pair multiplications. Figure 6 (b) depicts another example with a weight term budget $\alpha = 3$, data term budget $\beta = 2$ and term-pair budget of $\gamma = 6$. By limiting the number of leading terms in weights and data to a fixed group budget via TQ, we can bound the term-pair computations needed for each value-level multiplication in a convolutional layer. We utilize this term-pair budget $\gamma = \alpha\beta$ in the design of our multi-resolution Multiplier-Accumulator (mMAC) discussed in Section 5.

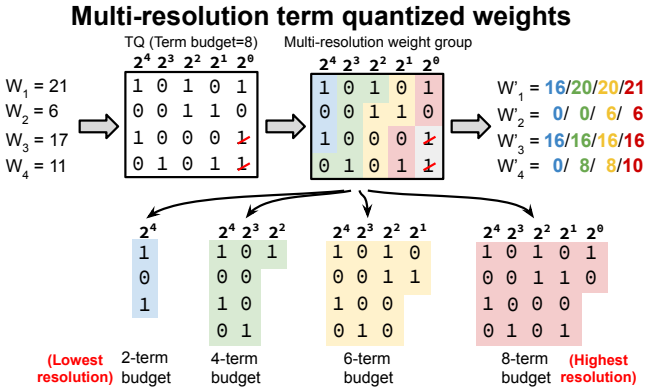

Figure 7: A multi-resolution weight group of size $g = 4$ with four different weight term budgets α: 2 terms (blue), 4 terms (green), 6 terms (yellow), and 8 terms (red). Smaller term budgets share terms with all larger term budget.

4 META MULTI-RESOLUTION DNN

In this section, we illustrate how we can train a single meta DNN model to support multiple resolutions (*i.e.,* term budgets) through the use of TQ. First, in Section 4.1, we show how we can share the terms in a group of weights across multiple resolutions. Then, in Section 4.2, we present a teacher-student training algorithm to learn a meta multi-resolution DNN that supports multiple resolutions via term sharing.

4.1 Multi-resolution Weight Groups

In Section 3, we discussed how a group of g weights can be quantized to meet a specific weight term budget α by dropping the low-order power-of-two terms in the group. Here, we extend this notion such that a single group of weights can support multiple weight term budgets. Figure 7 depicts a multi-resolution weight group for the same group of values in Figure 4, under four weight term budgets: 2, 4, 6, and 8 terms. When a lower-resolution term budget of $\alpha = 2$ is selected, the top two leading terms across the four values are kept (shown in the blue region), resulting in the set of term-quantized values of $[16, 0, 16, 0]$. In contrast, when a higher-resolution term budget (*e.g.,* $\alpha = 8$) is selected, the values in the group have less quantization error (*e.g.,* $[21, 6, 16, 10]$). This group-based multi-resolution approach can be applied to the weight across all layers of the DNN. Similarly, this multi-resolution paradigm can also apply to the data by dynamically selecting the data term budget β.

Under this multi-resolution paradigm, during inference runtime, the term budgets α, β can be selected to accommodate the current hardware resource constraint (*e.g.,* processing time or energy budget) at hand. For instance, when there is less processing demand, larger term budgets can be applied to the DNN weights, producing a DNN model with both a better performance (*e.g.,* classification accuracy) and higher computation cost, and vice versa. Additionally, since the terms in a weight group are shared across all resolutions, they only need to be stored a single time while still supporting all resolutions. Specifically, the terms for the lower-resolution weight group can be derived by applying TQ with a corresponding term

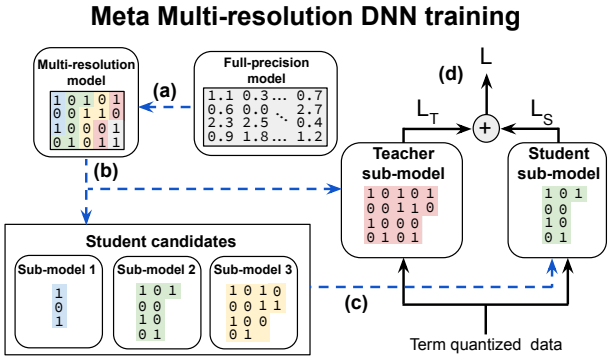

Figure 8: Meta multi-resolution DNN training procedure to support sub-models with different term budgets at inference. At each training iteration, the full-precision model is updated via (a)-(d). The highest resolution sub-model is always selected as the teacher sub-model and the student sub-model is randomly selected from the other models. Note that the terms used in a sub-model are nested in the terms of any larger sub-model.

budget. We call the resulting DNN model corresponding to a specific term budget pair (α, β) a *sub-model*.

4.2 Meta Multi-resolution Model Training

Training a meta multi-resolution DNN to support multiple term budgets (*i.e.,* sub-models) during inference requires special considerations. As we show later in Section 6.3, a simple post-training quantization approach [38] applied to a DNN trained without multi-resolution considerations does not achieve good performance for the low-resolution settings with smaller term budgets. Therefore, we propose to train the multi-resolution DNN such that the quantization error introduced by TQ during inference can be accounted for during training in a similar manner to other quantization-aware training approaches [30]. The resulting meta model generated from the multi-resolution training enables superior prediction accuracies for each sub-model compared to post-training quantization.

A straightforward training strategy is to jointly train all the possible sub-models by minimizing the sum of the losses derived from each sub-model. However, this formulation causes the training runtime and memory to grow rapidly as the number of sub-models increases, making training with more than a few sub-models impractical. To mitigate these training challenges, we propose a knowledge distillation mechanism which optimizes only a few sub-models per iteration. Specifically, in this work, we optimize two sub-models at a time: a higher-resolution teacher sub-model and a lower-resolution student sub-model. To speed up convergence of the multi-resolution training, we use a pre-trained full-precision 32-bit floating-point model trained on the same training dataset.

Figure 8 provides an overview of this teacher-student training procedure. At each iteration, the full-precision model is first quantized by UQ to produce a multi-resolution model (step a) which is used to generate both teacher and student sub-models (step b). The sub-model with the largest weight term budget is always used as the teacher network. The student network is randomly selected at each iteration from the remaining sub-models (step c). The same DNN topology is used for all sub-models. The loss is computed using the knowledge distillation technique [25], which combines loss terms using both the real labels (L_T) and the soft labels generated by the teacher network (L_S) (step d). The resulting gradients are applied to the full-precision model. No quantization is performed during backward propagation.

As this training is run over many iterations (*e.g.,* 50000), each student model will receive approximately the same number of updates via random selection. During each forward propagation, we apply SDR to reduce the weight and data terms in the multi-resolution models. Additionally, we adopt the techniques in [10] to learn the clipping parameters for both weight and data. Algorithm 1 provides a detailed summary of this training procedure. Step 2 and 4 of the algorithm generate the weight terms for the teacher and student sub-models, respectively. Due to the fact that the sub-models use nested terms as described in Figure 7, all weight terms that appear in the student sub-model are also present in the teacher sub-model.

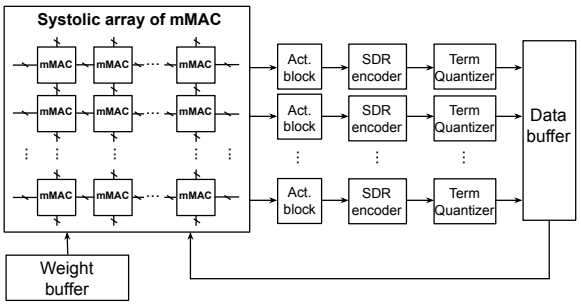

Figure 9: The mMAC system design.

5 MULTI-RESOLUTION INFERENCE SYSTEM

In this section, we describe the design of the mMAC system which can perform efficient inference for any of the sub-models generated by Meta Multi-resolution DNN training. To allow for a simple system design, we use a 2D systolic array [34] for the implementation of the computation engine. However, we note that our multi-resolution paradigm can also supports other computation engine designs.

The system has the following components (depicted in Figure 9): (1) a weight buffer that stores the weights of the largest sub-model, (2) a systolic array composed of multi-resolution MAC (mMAC) cells (described in Section 5.2), (3) activation blocks which apply nonlinear activation functions (*e.g.,* ReLU) on the systolic array output, (4) SDR encoders (Section 5.3) which convert unsigned input values into signed-digit representation (Section 2.4), (5) term quantizers (Section 5.3) which select the top β terms (data term budget) from the outputs of the SDR encoders, and (6) a data buffer which saves all the intermediate data for subsequent processing.

5.1 Multi-resolution Model Deployment

Given a multi-resolution DNN model trained via Algorithm 1, the terms in each weight group of the largest sub-model are sorted from largest to smallest and stored in memory. Before inference, a chosen weight term budget α determines the number of leading

Algorithm 1: Meta Multi-resolution DNN Training

Input: W_l is the full-precision weights at layer l.
X_l^T is input data for the teacher sub-model at layer l.
X_l^S is input data for the student sub-model at layer l.
K and A are sets of weight and data term budgets.
S is the set of term budget pairs,
$S = \{(\alpha, \beta) | \alpha \in K, \beta \in A\}$.
b is the bitwidth of the multi-resolution model.
I is the number of training iterations (*i.e.,* steps).
L is the total number of DNN layers.
g is the TQ weight group size (static across sub-models).

Output: The teacher model $W_l^{tq,T}$.

for $i \leftarrow 1$ *to* I **do**
  for $l \leftarrow 1$ *to* L **do**
    ▷ **Step 1** Apply b-bit uniform quantization on W_l, X_l^S and X_l^T to produce the result quantized model W_l^q and quantized data $X_l^{q,S}$, $X_l^{q,T}$.
    ▷ **Step 2** Let α_T and β_T denote the maximum possible term budgets for weight and data, respectively. Apply term quantization on SDR-encoded W_l^q with a budget α_T and a group size g, generating the largest sub-model weight matrix $W_l^{tq,T}$.
    ▷ **Step 3** Apply term quantization on $X_l^{q,T}$ with a budget β_T and a group size 1 to produce TQ data $X_l^{tq,T}$.
    ▷ **Step 4** Randomly select a pair of term budgets (α_S, β_S) from S, apply term quantization on SDR-encoded W_l^q under a budget α_S and a group size g to generate $W_l^{tq,S}$.
    ▷ **Step 5** Apply term quantization on $X_l^{q,S}$ with a budget β_S to generate the term-quantized data $X_l^{tq,S}$.
    ▷ **Step 6** Perform the forward pass (*e.g.,* convolution) for $(W_l^{tq,T}, X_l^{tq,T})$ and $(W_l^{tq,S}, X_l^{tq,S})$ to produce Y_{out}^T, Y_{out}^S.
    ▷ **Step 7** Perform additional operations (*e.g.,* non-linear activation, batch normalization) on Y_{out}^T and Y_{out}^S to produce X_{l+1}^T and X_{l+1}^S.
  ▷ **Step 8** Compute the loss L_T, L_S for teacher and student network.
  ▷ **Step 9** Compute the gradient, update W_l and the corresponding clipping parameters for each layer $l \in L$.

terms for each group to load from memory into each mMAC. During inference, the data terms are first quantized under the data term budget β before entering the mMAC. These term budgets lead to a processing latency that is directly proportional to the term-pair budget γ, which is $\alpha \times \beta$. An example is given in Figure 10, for a weight term budget of $\alpha = 8$ (shown in red), 8 weight terms are loaded from memory to mMAC, which together with $\beta = 1$ leads to a processing time of $\gamma = 8$. In contrast, a lower term budget ($\alpha = 4$ shown in green) would require a subset of these terms to be loaded from memory, giving a lower processing time ($\gamma = 4$) and memory access cost. More details on term-level memory access are provided in Section 5.4.

5.2 Multi-resolution MAC (mMAC) Design

The multi-resolution MAC (mMAC) performs term-pair multiplication via exponent addition. The hardware design of a mMAC

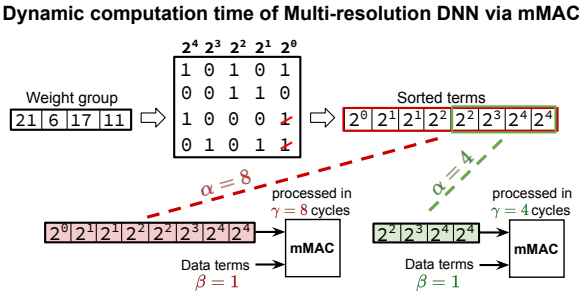

Figure 10: The mMAC can process multiple term budgets by varying the processing time (*e.g.,* a 4-term budget in 4 cycles and a 8-term budget in 8 cycles).

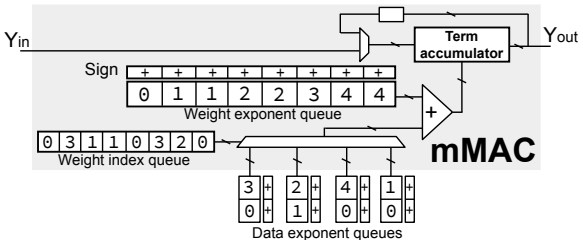

Figure 11: Multi-resolution MAC design.

for a group size $g = 4$ and term budgets $\alpha = 8, \beta = 2$ is shown in Figure 11. The exponents for term-pairs are stored in weight and data exponent queues, with the sign of each term stored in a separate queue with one bit per term. For example, the term -2^3 would save the exponent 3 in the exponent queue and a minus (−) in the sign queue. Figure 11 uses the same weight terms as depicted in Figure 10 for illustration clarity. Each cycle, the next weight exponent is pulled from the weight exponent queue, and a data exponent is selected for exponent addition using the next index from the weight index queue. The adder then computes the sum of these exponents, sets the sign, and delivers the sum to the term accumulator. Processing a group with 8 term-pairs takes 8 cycles in total. To support the multi-resolution term operation, the weight exponent queue and weight index queue are designed to fit the terms of largest budget.

Figure 12 depicts the operation of mMAC under a weight term budget $\alpha = 4$ across multiple cycles ($T = 0, \ldots, 5$). Only the four leading weight terms and the corresponding weight indexes are loaded into the weight exponent and weight index queues. At each cycle, a pair of weight and data exponents are processed by the adder, with the resulting signed exponent sent to the term accumulator. The term accumulator converts the signed exponent to a value and then adds it with the accumulation input. The term accumulator output will loop back to be used again in the next cycle while there are remaining term-pairs to be processed before being passed to the neighboring systolic cell. To support the processing of additional term-pairs across multiple cycles, the weight exponent and index queues are implemented using linear feedback shift registers.

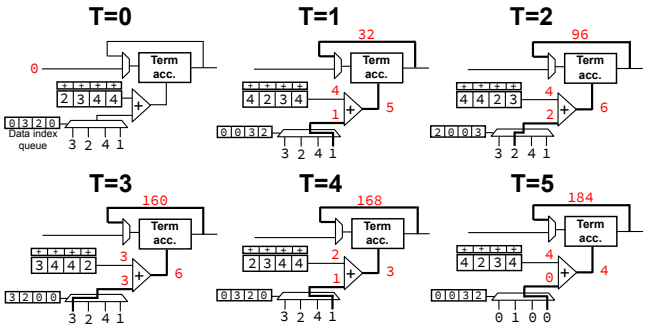

Figure 12: An example of mMAC operation.

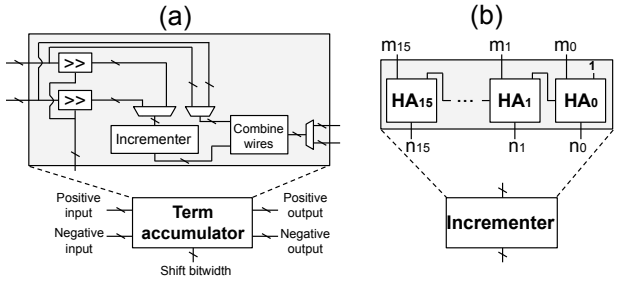

Figure 13: The term accumulator and incrementer.

5.2.1 Term Accumulator. A simple implementation of the term accumulator would convert the signed exponent into the corresponding binary value, and sum the intermediate result with the input accumulation using a parallel adder. However, since the accumulation must have a high precision (*e.g.,* a 32-bit integer), a parallel adder would be expensive to implement. Instead, we leverage the property that each exponent addition results in a power-of-two value to reduce the hardware implementation cost. Figure 13(a) illustrates the design of the term accumulator. Using this design, we can add 4 $(0100)_2$ to the accumulator with value 9 $(1001)_2$ by right shifting both by 2 places (corresponding to the 4), adding the resulting two numbers with the incrementer, and finally left shifting back the result. Exploiting the fact that exponent addition results only in values with a single nonzero term enables the increments using half adders (HA), as opposed to full adders, as shown in Figure 13(b).

One additional consideration is that under SDR a term can be negative, meaning that the accumulator must support both increment and decrement operations. To support this requirement, we use two accumulations to accumulate the positive sums and negative sums separately. A single parallel adder is used to perform subtraction between the positive and negative sums at the end of each row of systolic array to produce the final result.

5.3 Encoding and Term Quantization

The SDR encoder (Figure 14) takes outputs from the activation block and produces SDRs with the minimum number of terms, as described in Section 2.4. It implements the encoding scheme proposed by [36] using a Finite-state machine (FSM). The FSM

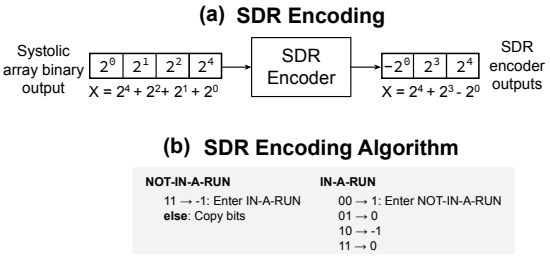

Figure 14: (a) SDR encoder operation. (b) SDR encoding algorithm.

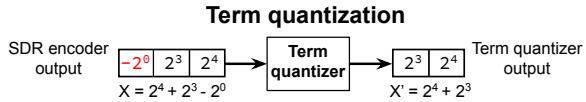

Figure 15: Term quantization with $\beta = 2$.

looks at only two consecutive bits in a binary input stream each cycle to generate the corresponding SDR.

The term quantizer selects the top β terms for each data value. Figure 15 demonstrates this process for an input $x = 23 = 2^4 + 2^3 - 2^0$. One term of x is delivered to the term quantizer every cycle, which counts the total number of the observed terms and sets terms to 0 once the budget β has been reached (*e.g.,* 2 in this example). The outputs from the term quantizer will be saved in the data buffer.

5.4 Efficient Storage of Multi-resolution Model

A key to our proposed multi-resolution approach is the sharing of terms across sub-models. Via our meta multi-resolution training regime (Algorithm 1), the weight terms for all lower-resolution sub-models are shared with higher-resolution sub-models so that it is sufficient to store only the largest sub-model. Based on this term sharing, we have developed a compact format for the storage of weight and data terms in memory. Figure 16(a) shows an example for the encoding of a group of 4 terms 2^4, 2^4, -2^3 and 2^1, where each term is encoded with 4 bits. The first three bits represent the exponent of the term, and the forth bit indicates the sign. Figure 16(b) shows encoding table for 5-bit TQ.

Under this storage scheme, each data value will be represented by 4β bits, given that β is the number of data terms in the value and each term requires 4 bits.

Similarly, storing the weight terms in a group of g values requires 4α bits per weight value. Besides storing the terms, we must also store the *weight index* for each weight term in a group of weight values. Each weight index indicates which value a given weight term belongs to. Storing weight indexes for each group requires $\alpha log_2(g)$ bits, where α is the number of weight terms per group and $log_2(g)$ is the number of bits to store each weight index. This leads to an average of $\frac{4\alpha+\alpha \log_2(g)}{g}$ bits per weight value.

In our multi-resolution training scheme, the weight budget α of the highest resolution sub-model is selected to be close to g. For instance, in the multi-resolution training of ResNet18, 8 sub-models are trained jointly with a group size of $g = 16$ and a group

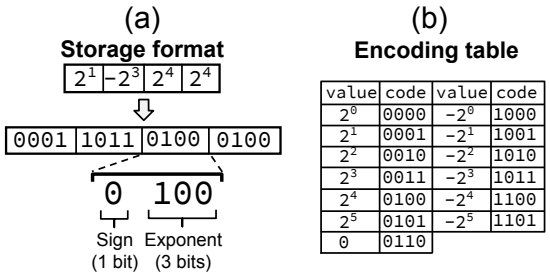

Figure 16: (a) Terms are converted into a packed format for efficient storage. (b) The encoding table for storage.

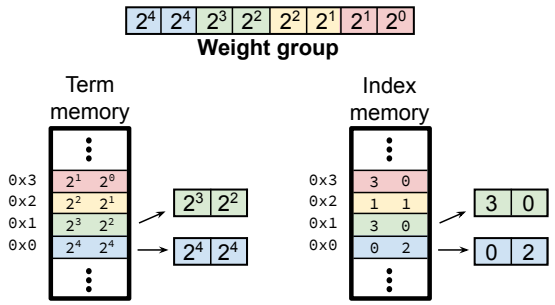

Figure 17: The power-of-two weight terms are stored in two-term increments. In this example, the same group of weight values in Figure 7 are illustrated with four weight term budgets $\alpha = 2$ (blue), $\alpha = 4$ (green), $\alpha = 6$ (yellow) and $\alpha = 8$ (red). When $\alpha = 4$ (shown in this figure), both addresses 0x0 and 0x1 will be accessed from the term memory and index memory. When $\alpha = 2$, only address 0x0 will be accessed.

budget of $\alpha = 1.25 \times g = 20$ for the highest resolution sub-model. Storing a group of $g = 16$ weights requires $4 \times \alpha + \alpha \times log_2(g) = 4 \times 20 + 20 \times log_2(16) = 160$ bits, which gives an average of 10 bits per weight value. Since all the 8 sub-models share the weights, this allows for an average of $\frac{10}{8} = 1.25$ bits per weight values for each sub-model.

In addition to the reduction in storage cost of weights shared across multiple models, our multi-resolution approach also allows for reduced memory access. Consider a multi-resolution weight group depicted in Figure 7, which supports four resolutions associated with 2-term, 4-term, 6-term, and 8-term budgets for a group of $g = 4$ values. In this case, each resolution increase amounts to adding a two-term increment to the group. For example, in increasing the 6-term budget to the 8-term budget resolution, we use a two-term increment composed of 2^0 and 2^1 for w_1 and w_4, respectively. Based on this, we store each of these two-term increments in a memory entry, in a manner that the two-term increments of a larger sub-model always follow those of a smaller sub-models. When implementing the next higher resolution, we access one more entry to obtain the additional terms. Therefore, this memory layout reduces the number of memory accesses for the low-resolution sub-models (see Figure 17). In addition, the indices for the weight terms are stored separately in a similar manner. The loaded indices are used to position the terms within a group (Figure 18).

In general, an increment may have one, two, or more terms, depending on the target multiple resolutions. We store several increments for multiple groups contiguously in order to make full use of the memory width. For example, for a 16-bit wide memory with each term requiring 4 bits (Figure 16), we store two two-term increments for two groups in each memory entry.

6 PERFORMANCE EVALUATION

In this section, we evaluate the performance (*e.g.,* classification accuracy, perplexity, or mean Average Precision) of DNNs trained under the multi-resolution paradigm proposed in Section 4 on a diverse range of applications including multiple CNNs (ResNet-18 [23], ResNet-50 [23] and MobileNet-V2 [50]) on ImageNet [14], an LSTM [26] on Wikitext-2 [44], and YOLO-v5 [32] on COCO [39]. We use pre-trained full-precision models as the initial models for the proposed multi-resolution paradigm discussed in Section 4.2. These models come from the PyTorch torchvision for ResNet-18, ResNet-50, and MobileNet-v2. We train a full-precision LSTM ourselves using the PyTorch language model example. For YOLO-v5, we use the pre-trained small model provided by the official repository (https://github.com/ultralytics/yolov5).

We use these pre-trained models to initialize the training procedure described in Algorithm 1. **For all settings, we use a weight group size of $g = 16$.** To perform the training efficiently, we have implemented a custom CUDA kernel to perform TQ and SDR encoding during the forward pass of training. Full details of the training hyper parameters can be found in the appendix. This section answers the following questions on multi-resolution performance:

- (Section 6.1) How much performance is lost by enforcing term sharing instead of training each sub-model separately?
- (Section 6.2) How does the distribution of weight values change across sub-models as a function of the the weight term budget?
- (Section 6.3) How much performance is gained via the multi-resolution training approach (Algorithm 1) as opposed to a post-training term quantization approach as in [36]?
- (Section 6.4) How does UQ (with varying bitwidths) compare to TQ (with varying term budgets) under a bit or term sharing regime in terms of performance?
- (Section 6.5) How long does multi-resolution training (Algorithm 1) take to perform for each of the evaluated models?
- (Section 6.6) What is the impact of the group size of TQ on classification accuracy under multi-resolution training?
- (Section 6.7) How well does the multi-resolution training approach scale as the number of sub-models is increased?

6.1 Impact of Term Sharing on Performance

In order to use the multi-resolution paradigm discussed in Section 4, it is required that the weight values must be shared across all sub-models. Therefore, it is important to investigate the impact in performance of enforcing this weight sharing across the different sub-models. Figure 19 shows the number of term-pair multiplications and classification accuracy for ResNet-18 models trained on ImageNet. The dark green points represent 8 models trained individually using different TQ settings, such as $(\alpha = 10, \beta = 2)$.

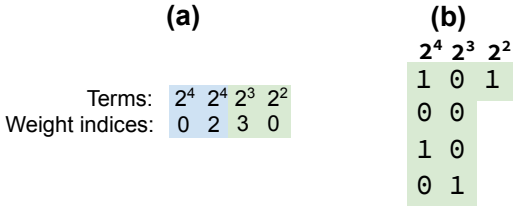

Figure 18: (a) An illustration of a term usage table for a group of four weights. For each term used by a weight, the index of the weight is shown. Suppose that the first, second, third and fourth weights are indexed by 0, 2, 3 and 0, respectively. Then the table states that the 2^4, 2^3 and 2^2 terms are used by both the first and third weights, the fourth weight, and the first weight, respectively. (b) The corresponding terms used by each of the four weights are shown.

By comparison, the light green points show the corresponding performance for the proposed multi-resolution model with 8 sub-models using the same TQ settings. Generally, we see that the multi-resolution model is 0.25% to 1.25% worse than each point trained individually, with the largest gap being for the most aggressive setting ($\alpha = 8, \beta = 2$). The performance degradation is caused by potentially inconsistent gradient updates produced by the teacher and student. These conflicting gradient updates lead to a slight accuracy degradation for the jointly-trained model as compared to individually-trained models.

6.2 Multi-resolution Weight Distributions

As discussed in the previous section, the terms in a multi-resolution DNN are shared across all sub-models, meaning that the weight values change depending on the number of allocated terms for the sub-model. Figure 20 shows a histogram of the frequency of weight values for three different sub-models from a multi-resolution DNN and a 5-bit UQ setting. Interestingly, for the low-resolution sub-model setting of ($\alpha = 8, \beta = 2$), the weights are concentrated mostly at values that can be represented with a single power-of-two (*e.g.*, 2, 4, and 8) and almost 50% of values are 0. For the more high-resolution sub-model setting of ($\alpha = 20, \beta = 3$), the distribution of values closely follows the 5-bit UQ model.

In this way, the proposed multi-resolution DNN can be viewed as interpolating between logarithmic quantization for the low-resolution sub-model and 5-bit uniform quantization for the high-resolution sub-model. By training many sub-models between these two extreme settings, the multi-resolution model is able to gradually trade-off computation (in the number of term operations per sample) for performance (*e.g.*, classification accuracy) as depicted earlier in Figure 19.

6.3 Comparison to Post-training Quantization

Instead of training a multi-resolution DNN using Algorithm 1, we could perform post-training quantization [38] on a pre-trained floating-point model. Generally, post-training quantization leads to poor performance for uniform quantization with less than 8 bits per weight. However, term quantization, which was originally posed as a post-training quantization approach in [36], leads to improved

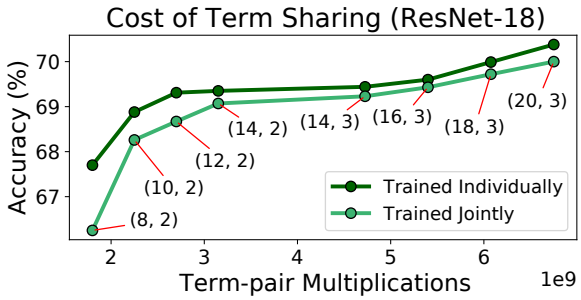

Figure 19: The multi-resolution model (light green) trained with 8 sub-model settings has slightly lower accuracy than settings trained individually (dark green). The (α, β) values for each sub-model are indicated by red lines and texts.

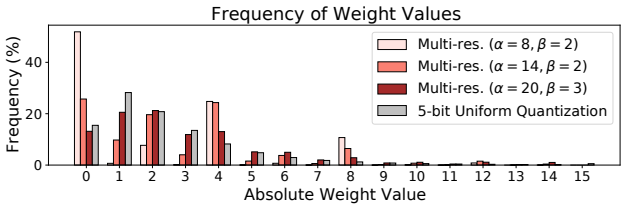

Figure 20: A histogram of the absolute weight values for 3 sub-models of ResNet-18 trained under the multi-resolution paradigm and an individual model trained under 5-bit UQ.

performance compared to UQ even when only a few terms are used per value.

Figure 21 provides a comparison between the proposed multi-resolution training with TQ approach and post-training TQ as in [36] for ResNet-18 and ResNet-50 on ImageNet. For both ResNet-18 and ResNet-50, we see that multi-resolution training outperforms post-training quantization for all settings. Additionally, we see that more aggressive settings lead to a larger degradation in accuracy, demonstrating that multi-resolution training is important to achieve a reasonable trade-off space between performance and number of operations.

6.4 Comparison to UQ Term Sharing

In this section, we compare the number of term-pair multiplications required to process one sample at different sub-model settings across multiple domains (image classification, object detection, and language modeling) using either UQ or TQ under term sharing via a multi-resolution model. The training procedure for the UQ models is the same as in Algorithm 1, except with UQ substituted for TQ. For the UQ models, sub-models are obtained by varying the weight and data bitwidth (*e.g.*, from 5-bit to 2-bit values for the CNNs trained on ImageNet). See the appendix for a complete description of parameters settings used in the evaluation.

6.4.1 ImageNet. Figure 22 (left) compares the performance of our multi-resolution approach under TQ and UQ across ResNet-18,

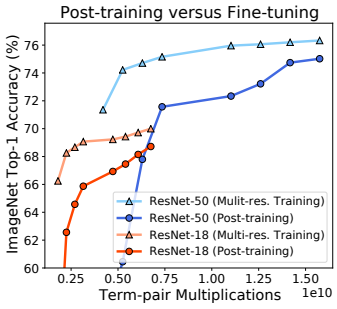

Figure 21: Comparing post-training quantization to the proposed multi-resolution training for ResNet-18 and ResNet-50 on ImageNet. Multi-resolution training significantly improves accuracy.

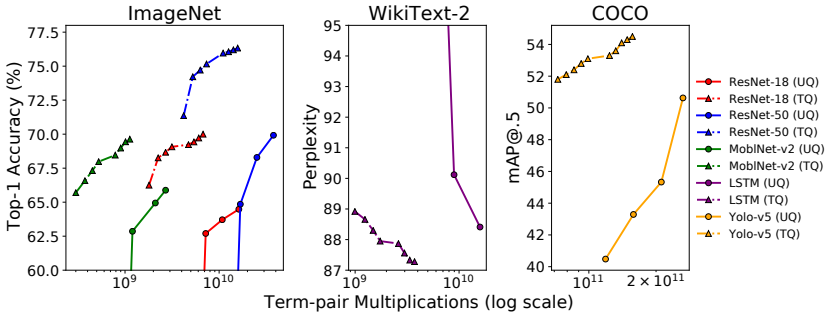

Figure 22: Comparing multi-resolution models trained under uniform quantization (UQ) and term quantization (TQ) for CNNs on ImageNet (left), an LSTM on Wikitext-2 (middle), and YOLO-v5 on COCO (right). The UQ sub-models vary the weight and data bitwidth (*e.g.,* from 2-bit to 5-bit for ImageNet), while the TQ sub-models vary in α (number of terms per weight group) and β (number of terms per data value).

Table 1: Multi-resolution DNN training complexity.

Model	Time per epoch (Multi-resolution)	Batch size	Number of sub-models	Time per epoch (single model)
ResNet-18	47 mins	1024	8	24 mins
ResNet-50	92 mins	512	8	44 mins
MobileNet-v2	51 mins	512	8	27 mins
LSTM	8 mins	20	8	4 mins
YOLO-v5	68 mins	64	10	35 mins

Table 2: FPGA resource consumption of MAC designs.

	pMAC	bMAC	mMAC
LUT	57	12	21
FF	44	14	25

ResNet-50, and MobileNet-v2 on ImageNet. For all three models, we observe that the multi-resolution approach using TQ greatly reduced the number of term multiplications compared to UQ while also achieving significantly better performance of roughly 5%. Enforcing term sharing across the UQ settings leads to a significant degradation in model performance, as all of the sub-models must share a common scale factor for quantization. Deriving a common scale factor is difficult for the 5-bit and 2-bit settings. Additionally, the trade-off between performance and operations is more graceful under TQ compared to UQ due to the more fine-grained nature of TQ as each point varies by two additional nonzero terms instead of a reduced bitwidth.

6.4.2 LSTM. Figure 22 (middle) compares the performance of the two approaches on a 2-layer LSTM with 650 hidden units (*i.e.,* neurons), a word embedding of length 650, and a dropout rate of 0.5 trained on WikiText-2, following the PyTorch word language model example. We see that our multi-resolution approach with TQ outperforms UQ by a wide margin, with even the most aggressive sub-model setting still achieving a reasonable perplexity.

6.4.3 COCO. Finally, Figure 22 (right) provides a comparison on YOLO-v5 (small) trained on COCO. We find that object detection requires significantly more precision compared to image classification to achieve good performance. Due to this, the UQ settings span from 8-bit to 5-bit representations for weights and data. By comparison, the sub-models in our multi-resolution approach span from ($\alpha = 22, \beta = 4$) to ($\alpha = 38, \beta = 5$). Since TQ only specifies the number of nonzero terms and not the bitwidth, we are able to achieve better performance by using a large bitwidth (8-bit) for all settings while varying the term budget in each sub-model.

6.5 Multi-resolution DNN Training Cost

Instead of training all the sub-models each iteration, our proposed meta multi-resolution DNN training leverages knowledge distillation and selects two sub-models to optimize for every iteration as described in Algorithm 1. Selecting only two models per iteration, instead of all sub-models which can be as large as 10, leads to significant saving on training time and memory usage. As two sub-models are selected per iteration, the total time for multi-resolution training is around twice of the time for training a single-resolution DNN model. In comparison, joint training of all the sub-models will cause the total training time and memory consumption to grow linearly with the number of sub-models.

Table 1 summarizes the training complexity of the proposed multi-resolution training scheme for each DNN model in Figure 22. Specifically, we evaluate the training complexity in terms of: 1. time for one epoch of multi-resolution training, 2. batch size of multi-resolution training, 3. number of sub-models for the multi-resolution training, 4. time required to perform one epoch of DNN training over a single term-quantized model. To train the single term-quantized model, we use the same batch size as the multi-resolution training for each DNN model, so the total number of iterations per epoch will be the same for both scenarios. We use 8 Nvidia GeForce GTX 1080 GPUs to perform multi-resolution training, with each GPU having a memory size of 11GB. We notice that multi-resolution training takes 1.92× longer on average than training a single model. This is because at every iteration two sub-models are selected to train jointly.

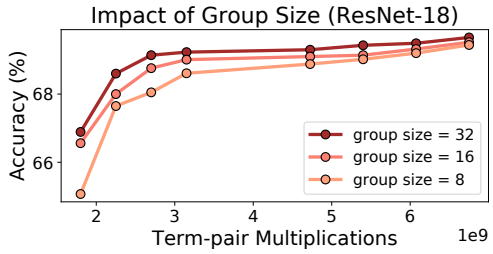

Figure 23: Comparing the impact of group size on classification accuracy. A larger group size improves the classification accuracy of a multi-resolution model compared to a smaller group size for the same number of term-pair multiplications.

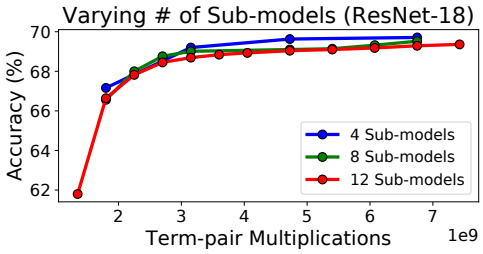

Figure 24: Increasing the number of sub-models leads to a slight degradation in classification while enabling a more fine-grained trade-off between classification accuracy and term-pair multiplications.

6.6 Sensitivity Analysis on Group Size

In this section, we evaluate the impact of the group size g on the classification accuracy of the multi-resolution DNN model. Specifically, we train three multi-resolution DNNs with different group sizes ($g = 8$, $g = 16$, and $g = 32$), while keeping the average term budget per weight value the same across all three multi-resolution models. For example, for the $g = 16$ model, the weight term budgets α for the sub-models are $20, 18, 16, 14, 12, 10, 8$ in contrast to $10, 9, 8, 7, 6, 5, 4$ for the group size $g = 8$ model.

Figure 23 compares the number of term-pair multiplications and classification accuracies of ResNet-18 for the multi-resolution models with varying group sizes. The model with the largest group size $g = 32$ achieves the highest classification accuracy for the same number of term-pair multiplications across all sub-model settings. However, the $g = 16$ setting has similar performance while also being significantly more efficient to implement in hardware due to a reduced multiplexer size. Therefore, in Section 7, we use $g = 16$ throughout the evaluation.

6.7 Scalability of Multi-resolution Training

In this section, we evaluate how well the multi-resolution training approach can support an increasing number of sub-models. Note that only two sub-models are selected for each iteration of training and one is always the largest sub-model. Since the total number of training rounds are fixed, as the number of sub-models increases, each sub-model will be selected for fewer training iterations.

Figure 24 shows the classification accuracies for three multi-resolution ResNet-18 models with 4, 8 and 12 sub-models trained using Algorithm 1 for 60 epochs. We observe that increasing the number of sub-models (e.g., from 4 to 12 sub-models) allows for a more fine-grained trade-off between accuracy and number of term-pair multiplications while incurring a slight degradation in accuracy. The multi-resolution model with 12 sub-models is within 1% classification accuracy of the model with 4 sub-models across the range of sub-model settings.

7 HARDWARE EVALUATION

In this section, we evaluate the performance of the mMAC system described in Section 5. We have synthesized our mMAC system using Xilinx VC707 FPGA evaluation board. We first illustrate the advantage of mMAC design by comparing it against conventional bit-serial and bit-parallel MAC (Section 7.1). Next, in Section 7.2, we compare mMAC against the Laconic Processing Element [51], a recent MAC design which also performs term-level operations. Then, in Section 7.3, we evaluate the hardware performance under different resolutions. Finally, in Section 7.4, we compare our system against the other FPGA accelerators.

7.1 Comparison to Conventional MACs

We evaluate the efficiency of our mMAC design by comparing it against bit-serial and bit-parallel implementations of a conventional MAC. We evaluate all three designs on the following computation: $y_{out} = \sum_{i=1}^{g} x_i w_i + y_{in}$, where y_{in}, y_{out}, x_i and w_i are 16-bit, 16-bit, 5-bit and 5-bit, respectively, and g is the number of accumulating operations (*i.e.,* group size in TQ).

The left side of Figure 25 shows the design of a bit-parallel MAC (pMAC), which performs multiplication between x_i and w_i and sums the result with y_{in} in one cycle and generates y_{out} in g cycles. The bit-serial MAC (bMAC), based on [35], is shown in Figure 25 (right). It consists of a bit-serial multiplier and additional logic elements to negate the multiplier output and perform accumulation. It requires 16 cycles to process one pair of values, for a total of $16 \times g$ cycles to generate y_{out}. In contrast, mMAC takes γ cycles to perform this accumulation operation, where γ is the term-pair budget for the weight and data. We set the group size $g = 16$ for evaluation.

Table 2 depicts the FPGA resource consumption of the three MAC designs in terms of LookUp Tables (LUTs) and Flip-flops (FFs). Compared with pMAC, mMAC requires 2.8× less LUTs and 1.8× less FFs, due to the fact that mMAC performs exponent additions as opposed to multiplication in computing the term products. Although the bMAC achieves an even lower hardware resource consumption than mMAC, it requires a much larger processing latency.

Table 3 shows the evaluation on energy efficiency for all the three MAC designs on FPGA, where all the results are normalized by the performance of mMAC. We evaluate mMAC under different term-pair budget γ used in Figure 19. We observe that the performance of mMAC improves as term-pair budget reduces, with all settings outperforming both bMAC and pMAC. A smaller term-pair budget

Table 3: Comparison on energy efficiency for MAC designs.

γ	16	20	24	28	42	48	54	60
bMAC	0.15×	0.17×	0.22×	0.26×	0.37×	0.44×	0.50×	0.56×
pMAC	0.17×	0.22×	0.27×	0.31×	0.47×	0.53×	0.61×	0.66×
mMAC	1.0×	1.0×	1.0×	1.0×	1.0×	1.0×	1.0×	1.0×

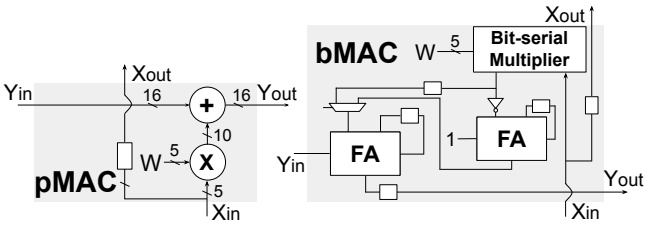

Figure 25: Bit-parallel MAC (left) and bit-serial MAC (right).

allows for a smaller amount of term-pair operations required to generate the result. This reduces the processing time and further improves the energy efficiency of mMAC. In general, compared with pMAC and bMAC, mMAC achieves a 3.1× and 5.6× higher energy efficiency on average across all term-pair budgets.

7.2 Comparison to Laconic Processing Element

We compare the performance of mMAC with the Laconic Processing Element (PE) [51]. Based on the layout shown in Figure 3 of [51], we re-implement a Laconic PE which can perform 16 multiplications between 5-bit weights and 5-bit data in parallel. Each term-pair multiplication results in a single power-of-two term which is delivered to histogram buckets. These buckets tally how many times each term appears (i.e., the coefficient of each power-of-two terms). Additionally, we assume each weight and data can be represented with 3 or fewer power-of-two terms under Booth encoding. With the Booth encoding, the resulting power-of-two terms can have their exponents as large as 6. This means we need to use 3 bits to represent each exponent. In addition, the Booth encoding allows each multiplication to complete within $3 \times 3 = 9$ cycles.

As described in Section 7.1, mMAC takes γ cycles to produce the results, where γ is the term-pair budgets. We use the largest term-pair budget of $\gamma = 60$ in Figure 19, which achieves a prediction accuracy of 69.8% on ImageNet for 5-bit quantized ResNet-18. We implement both MAC designs on an FPGA and compare their performances in terms of the energy efficiency. We observe that mMAC outperforms Laconic PE on energy efficiency by 2.7×.

While Laconic PE also focuses on term-level operation, it does not use the concept of group-based quantization. This means that the Laconic PE must assume that each weight or data value requires the maximum number of terms (i.e., 3 terms) to ensure correctness. Therefore, it processes a dot product of length 16 by assuming there are $3 \times 3 \times 16 = 144$ term-pairs. By comparison, our mMAC requires only $\gamma = 60$ term-pairs to process a dot product of the same length. Through multi-resolution training, our mMAC enforces a much tighter processing bound for each group via the term-pair budget γ, thereby mitigating this straggler issue.

Table 4: Comparison of our FPGA implementation of ResNet-18 to other FPGA-based accelerators on ImageNet.

	[37]	[52]	[54]	[36]	Ours
FPGA Chip	VC709	Virtex-7	ZC706	VC707	VC707
Frequency (MHz)	150	100	200	170	150
FF	262k(30%)	348k(40%)	51k(12%)	316k(51%)	409k(66%)
LUT	273k(63%)	236k(55%)	86k(39%)	201k(65%)	275k(91%)
DSP	2144(59%)	3177(88%)	808(90%)	756(27%)	996(36%)
BRAM	1913(65%)	1436(49%)	303(56%)	606(59%)	524(51%)
Latency (ms)	2.56	11.7	5.84	7.21	3.98
Energy eff. (frames/J)	12.93	8.39	40.7	25.22	71.48

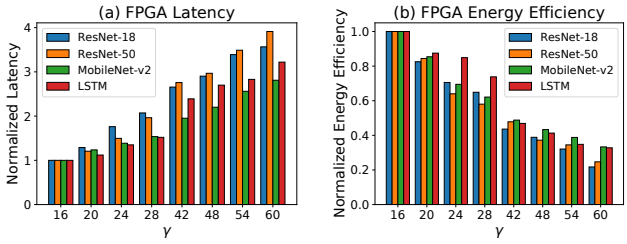

Figure 26: The mMAC system supports a range of designs with varying energy efficiency and latency under different γ settings across multiple networks (normalized to $\gamma = 16$).

Additionally, Laconic PE adopts histogram buckets to record the coefficients of all the output power-of-two terms. Each bucket is represented as a 6-bit number to record the power-of-two coefficient. For groups of size 16, many of these buckets will be underutilized or empty, as it is more likely for small terms (e.g., 2^2) to be produced than large terms (e.g., 2^6). During the reduction stage, this will result in many additions of power-of-two terms with zero coefficients. In comparison, mMAC directly sums the output terms with accumulation using cheaper half adders. Finally, mMAC supports multi-resolution inference by adaptively adjusting the amount of term operations to perform. In comparison, Laconic MAC only supports single resolution inference.

7.3 FPGA System Evaluation

In this section, we evaluate our mMAC system performance under different resolutions. The mMAC system contains a 128×128 systolic array. We adopt the corresponding term-pair budgets used in Figure 22 for evaluation.

Figure 26 illustrates the trend in average processing latency (*i.e.,* number of cycles to finish one input sample) and energy efficiency (*i.e.,* number of input samples processed for one Joule of energy) of the mMAC system under different γ values (*i.e.,* term-pair budget) across multiple models. We notice that the processing latency decreases (3.1× on average) and the energy efficiency grows (3.25× on average), as γ reduces from 60 to 16. This is because a lower term-pair budget leads to a smaller mMAC processing time for a group of weight and data values. Additionally, a smaller term-pair budget also reduces the amount of memory accesses and on-chip traffic between the memory and computation engine, since only the terms in the low-resolution terms need to be loaded and transferred. Figure 26 demonstrates that our mMAC system can

Table 5: Hyperparameters for ResNet-18 and ResNet-50.

Hyperparameter	Values
Total training epochs	60
Momentum	0.9
Batch size (ResNet-18)	1024
Batch size (ResNet-50)	512
Learning rates	0.1, 0.01, 10^{-3}, 10^{-4}, 10^{-5}
Weight decay	10^{-4}

Table 6: Hyperparameters for MobileNet-V2.

Hyperparameter	Values
Total training epochs	60
Momentum	0.9
Batch size	512
Learning rates	0.1,0.01,10^{-3},10^{-4},10^{-5}
Weight decay	10^{-4}

provide a range of designs with varying energy efficiency and latency by dynamically adjusting its computational cost based on the term-pair budget γ.

7.4 Comparison against Other FPGA Designs

Lastly, we compare our mMAC system with the other FPGA DNN accelerator designs on ResNet-18. Specifically, we apply term budgets of $(\alpha, \beta) = (20, 3)$ and $g = 16$ for our mMAC system, which can achieve a top-1 prediction accuracy of 69.8% on ImageNet. The results are shown in Table 4. Our system achieves the highest energy efficiency. Although the processing latency of [37] is even lower, it has a much larger hardware resource cost and lower energy efficiency. On average, our system outperforms the other designs by 1.7× and 3.28× on the processing latency and energy efficiency.

Our mMAC system achieves superior performance for several reasons. Most importantly, the multi-resolution DNN with TQ significantly reduces the number of term-pair operations, which further allows the computation engine to achieve a much tighter processing bound (mitigating stragglers) and therefore a lower processing latency. For instance, under a group size $g = 16$ and term-pair budget $\gamma = 60$, the mMAC system achieves a worst case processing time of only 60 cycles to compute the dot product for the 16 values in the group. This is in contrast to the conventional accelerator design, where the computation latency is always impeded by the slowest computation unit in the system. Second, the efficient design of mMAC converts the expensive multiplication operations between values into a series of additions between the term exponents, and the incrementer in the term accumulator avoids the implementation cost of the expensive parallel adder. Finally, the compact memory encoding scheme leads to a lighter traffic between the on-chip buffer and the computation engine.

8 CONCLUSION

We have shown that via term quantization, a single meta model can spawn sub-models of varying resolutions with low system overheads and performance loss. To this end, we train the meta model by jointly optimizing multiple sub-models of different resolutions.

Table 7: Hyperparameters for YOLO (left) and LSTM (right).

Hyperparameter	Values
Total training epochs	40
Optimization algorithm	SGD
Batch size	64
Initial learning rates	0.0015
Weight decay	5×10^{-4}

Hyperparameter	Values
Total training epochs	20
Optimization algorithm	SGD
Batch size	20
Dropout	0.2
Initial learning rates	5

During inference, we implement multiple resolutions by simply adjusting the number of leading nonzero terms on the learned weights of the meta model. To minimize memory footprint of the metal model and streamline its training, we share terms across multiple sub-models. These approaches together lead to a multi-resolution MAC (mMAC) design that can efficiently implement multiple resolutions. Results of this paper demonstrate that training a single model for multi-resolution inference is viable.

ACKNOWLEDGMENTS

This research is supported in part by the Air Force Research Laboratory under award number FA8750-18-1-0112, and the Defense Advanced Research Projects Agency under UCLA award number 0160GXA278 and MIT award number S5181. The authors thank the anonymous reviewers of ASPLOS 2021 for their helpful feedback.

A APPENDIX

In this section, we present the hyperparameter settings for the training procedure described in Algorithm 1.

A.1 Hyperparameters for ResNet

For ResNet-18 and ResNet-50, we use pre-trained full-precision models from the PyTorch official website as the initial models for the proposed multi-resolution paradigm discussed in Section 4 in the paper. The bitwidth of the meta multi-resolution model is set to $b = 5$ for both models. The settings are summarized in Table 5.

A.2 Hyperparameters for MobileNet-V2

We use the pre-trained floating-point model from the github repository as the initial model. The bitwidth of the meta multi-resolution model is set to $b = 5$. The settings are summarized in Table 6.

A.3 Hyperparameters for YOLO-V5

For YOLO-v5, we use the YOLOv5s model provided by the official repository to initialize the full-precision model. The bitwidth of the meta multi-resolution model is set to 8. Cosine learning rate decay is used for adjusting the learning rate per iteration. The hyperparameters are summarized in Table 7 (left). All the other hyperparameters remain the same as the official repository.

A.4 Hyperparameters for LSTM

We adopt the code provided by the official Pytorch repository to initialize the full-precision model, which achieves a perplexity of 86.85. Then we use this full-precision model to initialize the 8-bit meta multi-resolution model in Algorithm 1. Other settings are presented in Table 7 (right). All the other hyperparameters remain the same as the official repository.

REFERENCES

[1] Jorge Albericio, Alberto Delmás, Patrick Judd, Sayeh Sharify, Gerard O'Leary, Roman Genov, and Andreas Moshovos. Bit-pragmatic deep neural network computing. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 382–394. ACM, 2017.

[2] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. Cnvlutin: Ineffectual-neuron-free deep neural network computing. In *ACM SIGARCH Computer Architecture News*, volume 44, pages 1–13. IEEE Press, 2016.

[3] Algirdas Avizienis. Signed-digit number representations for fast parallel arithmetic. *IRE Transactions on Electronic Computers*, EC-10:389–400, 1961.

[4] Andrew D Booth. A signed binary multiplication technique. *The Quarterly Journal of Mechanics and Applied Mathematics*, 4(2):236–240, 1951.

[5] Han Cai, Chuang Gan, and Song Han. Once for all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019.

[6] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *ACM SIGARCH Computer Architecture News*, 42(1):269–284, 2014.

[7] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *ACM SIGARCH Computer Architecture News*, volume 44, pages 367–379. IEEE Press, 2016.

[8] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. Dadiannao: A machine-learning supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622. IEEE Computer Society, 2014.

[9] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

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

[11] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*, 2014.

[12] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[13] Alberto Delmas, Patrick Judd, Dylan Malone Stuart, Zissis Poulos, Mostafa Mahmoud, Sayeh Sharify, Milos Nikolic, and Andreas Moshovos. Bit-tactical: Exploiting ineffectual computations in convolutional neural networks: Which, why, and how. *24th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019.

[14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[15] Tim Dettmers. 8-bit approximations for parallelism in deep learning. *arXiv preprint arXiv:1511.04561*, 2015.

[16] Barry L Drake, Richard P Bocker, Mark E Lasher, Richard H Patterson, and William J Miceli. Photonic computing using the modified signed-digit number representation. *Optical Engineering*, 25(1):250138, 1986.

[17] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. Tetris: Scalable and efficient neural network acceleration with 3d memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 751–764, 2017.

[18] Scott Gray, Alec Radford, and Diederik Kingma. Gpu kernels for block-sparse weights. https://s3-us-west-2.amazonaws.com/openai-assets/blocksparse/blocksparsepaper.pdf, 2017. [Online; accessed 12-January-2018].

[19] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *International Conference on Machine Learning*, pages 1737–1746, 2015.

[20] Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, Huazhong Yang, and William J. Dally. Ese: Efficient speech recognition engine with sparse lstm on fpga. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 75–84. ACM, 2017.

[21] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pages 243–254. IEEE, 2016.

[22] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[23] 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.

[24] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *International Conference on Computer Vision (ICCV)*, volume 2, 2017.

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

[26] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[27] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q Weinberger. Multi-scale dense networks for resource efficient image classification. *arXiv preprint arXiv:1703.09844*, 2017.

[28] Gao Huang, Shichen Liu, Laurens van der Maaten, and Kilian Q Weinberger. Condensenet: An efficient densenet using learned group convolutions. *arXiv preprint arXiv:1711.09224*, 2017.

[29] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[30] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018.

[31] Jonathan Jedwab and Chris J Mitchell. Minimum weight modified signed-digit representations and fast exponentiation. *Electronics Letters*, 25(17):1171–1172, 1989.

[32] Glenn Jocher, Alex Stoken, Jirka Borovec, NanoCode012, ChristopherSTAN, Liu Changyu, Laughing, Adam Hogan, lorenzomammana, tkianai, yxNONG, AlexWang1900, Laurentiu Diaconu, Marc, wanghaoyang0106, ml5ah, Doug, Hatovix, Jake Poznanski, Lijun Yu, changyu98, Prashant Rai, Russ Ferriday, Trevor Sullivan, Wang Xinyu, YuriRibeiro, Eduard Reñé Claramunt, hopesala, pritul dave, and yzchen. ultralytics/yolov5: v3.0, August 2020.

[33] Supriya Kapur, Asit Mishra, and Debbie Marr. Low precision rnns: Quantizing rnns without losing accuracy. *arXiv preprint arXiv:1710.07706*, 2017.

[34] H. T. Kung. Why systolic architectures? *IEEE Computer*, 15:37–46, 1982.

[35] H. T. Kung, Bradley McDanel, and Sai Qian Zhang. Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization. *24th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019.

[36] H. T. Kung, Bradley McDanel, and Sai Qian Zhang. Term revealing: Furthering quantization at run time on quantized dnns. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020.

[37] Huimin Li, Xitian Fan, Li Jiao, Wei Cao, Xuegong Zhou, and Lingli Wang. A high performance fpga-based accelerator for large-scale convolutional neural networks. In *Field Programmable Logic and Applications (FPL), 2016 26th International Conference on*, pages 1–9. IEEE, 2016.

[38] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning*, pages 2849–2858, 2016.

[39] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[40] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. *arXiv preprint arXiv:1707.06342*, 2017.

[41] Mostafa Mahmoud, Dylan Malone Stuart, Zissis Poulos, Alberto Delmas Lascorz, Patrick Judd, Sayeh Sharify, Milos Nikolic, Kevin Siu, Isak Edo Vivancos, and Andreas Moshovos. Accelerating image sensor based deep learning applications. *IEEE Micro*, 2019.

[42] Bradley McDanel, Sai Qian Zhang, H. T. Kung, and Xin Dong. Full-stack optimization for accelerating cnns using powers-of-two weights with fpga validation. *International Conference on Supercomputing*, 2019.

[43] Mason McGill and Pietro Perona. Deciding how to decide: Dynamic routing in artificial neural networks. *arXiv preprint arXiv:1703.06217*, 2017.

[44] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

[45] Daisuke Miyashita, Edward H Lee, and Boris Murmann. Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025*, 2016.

[46] Sharan Narang, Eric Undersander, and Gregory F. Diamos. Block-sparse recurrent neural networks. *CoRR*, abs/1711.02782, 2017.

[47] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally. Scnn: An accelerator for compressed-sparse convolutional neural networks. In *ACM SIGARCH Computer Architecture News*, volume 45, pages 27–40. ACM, 2017.

[48] Eunhyeok Park, Junwhan Ahn, and Sungjoo Yoo. Weighted-entropy-based quantization for deep neural networks. In *Proceedings of the IEEE Conference on*

Computer Vision and Pattern Recognition, pages 5456–5464, 2017.

[49] Ao Ren, Tianyun Zhang, Shaokai Ye, Jiayu Li, Wenyao Xu, Xuehai Qian, Xue Lin, and Yanzhi Wang. Admm-nn: An algorithm-hardware co-design framework of dnns using alternating direction methods of multipliers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 925–938. ACM, 2019.

[50] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[51] Sayeh Sharify, Alberto Delmas Lascorz, Mostafa Mahmoud, Milos Nikolic, Kevin Siu, Dylan Malone Stuart, Zissis Poulos, and Andreas Moshovos. Laconic deep learning inference acceleration. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 304–317. ACM, 2019.

[52] Yongming Shen, Michael Ferdman, and Peter Milder. Maximizing cnn accelerator efficiency through resource partitioning. In *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*, pages 535–547. IEEE, 2017.

[53] 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.

[54] Junsong Wang, Qiuwen Lou, Xiaofan Zhang, Chao Zhu, Yonghua Lin, and Deming Chen. Design flow of accelerating hybrid extremely low bit-width neural network in embedded fpga. *arXiv preprint arXiv:1808.04311*, 2018.

[55] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 409–424, 2018.

[56] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082, 2016.

[57] 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.

[58] Jiahui Yu and Thomas S Huang. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1803–1811, 2019.

[59] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. In *7th International Conference on Learning Representations, ICLR 2019*, 2019.

[60] Shijin Zhang, Zidong Du, Lei Zhang, Huiying Lan, Shaoli Liu, Ling Li, Qi Guo, Tianshi Chen, and Yunji Chen. Cambricon-x: An accelerator for sparse neural networks. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, page 20. IEEE Press, 2016.

[61] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044*, 2017.

[62] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.