BitPruning: Learning Bitlengths for Aggressive and Accurate Quantization

Miloš Nikolić  
*University of Toronto*  
Toronto, Canada  
milos.nikolic@mail.utoronto.ca

Ghoufi Boukli Hacene  
*MILA*  
Montreal, Canada  
bouklihg@mila.quebec

Ciaran Bannon  
*University of Toronto*  
Toronto, Canada  
ciaran.bannon@utoronto.ca

Alberto Delmas Lascorz  
*University of Toronto*  
Toronto, Canada  
a.delmaslascorz@mail.utoronto.ca

Matthieu Courbariaux  
*MILA*  
Montreal, Canada  
mattieu.courbariaux@gmail.com

Omar Mohamed Awad  
*University of Toronto*  
Toronto, Canada  
omar.awad@mail.utoronto.ca

Isak Edo Vivancos  
*MILA*  
Montreal, Canada  
isak.edo@utoronto.ca

Yoshua Bengio  
*MILA*  
Montreal, Canada  
yoshua.bengio@mila.quebec

Vincent Gripon  
*IMT Atlantique, Lab-STICC*  
Brest, France  
vincen.gripon@imt-atlantique.fr

Andreas Moshovos  
*University of Toronto*  
Toronto, Canada  
moshovos@eecg.toronto.edu

Abstract—BitPruning is a training method for minimizing inference bitlengths at any granularity while maintaining accuracy. BitPruning extends the meaning of fixed-point bitlengths into the continuous domain by interpolating between the nearest two integers, enabling gradient descent to learn bitlengths together with other parameters. A novel regularizer penalizes large bitlength representations and can be modified to minimize other quantifiable criteria, such as number of operations or memory footprint. BitPruning learns thrifty representations while maintaining accuracy: With ImageNet, it produces an average per layer bitlength of 3.76 and 4.36 bits on ResNet18 and MobileNet V2 respectively, remaining within 0.5% of the base TOP-1 accuracy. Simple modifications of the BitPruning regularizer can be used to further reduce compute workload by up to 24%, as well as memory footprint in activation or weight-heavy tasks by up to 14% and 8% respectively.

Index Terms—Deep Learning, Quantization

I. INTRODUCTION

Energy efficiency is the primary constraint for computing devices, impacting execution time and operational costs from data centers to edge devices. Reducing Deep Neural Network (DNN) energy usage can speedup inference latency, enable more powerful models, and cut data center operating expenses. A key factor in determining DNN energy needs and execution time during inference is the choice of data type, including the number of bits used per value (e.g., 8-bit vs. 32-bit) and their interpretation (e.g., fixed-point or floating-point). While 32-bit floating-point is definitely sufficient, many DNNs can use more energy-efficient floating- or fixed-point datatypes without sacrificing accuracy. Graphics processors (GPUs), general-purpose processors (CPUs) and accelerator have been introducing energy-efficient datatypes targeting DNNs [1]–[5].

Reducing datatype bitlength improves energy efficiency of memory and computations. **Memory:** Storing and transferring fewer bits uses less energy. This is especially effective for the off-die memory accesses as they are an order of magnitude slower and two orders of magnitude more energy-consuming compared to on-chip accesses or computation. **Computation:** The operations per cycle throughput of many hardware platforms scales inversely proportionally with datatype bit length. Commodity CPUs and GPUs support several bit lengths, e.g., 16b at 1x and 8b at 2x throughput, whereas spatially composable, bit-serial accelerators [3], [4], [6]–[10] and FPGA-based designs [11] can support a full range of bit lengths.

Many prior techniques for selecting datatypes require significant engineering effort to expertly identify value groups, and are limited in their effectiveness versus learning the bitlengths. These approaches include: 1) Forcing the DNN developer to “guess” a good enough datatype (e.g., fixed-point of 8 bits) prior to training a network without help on how to navigate its complex relationship with accuracy, energy, and execution time performance. Once a bitlength choice is made, there are methods to reduce accuracy losses [12]–[15]. 2) Post-training profiling and heuristic-based techniques which partially automate the process of finding better per-layer bitlengths [16]–[18]. 3) Hand-crafted quantization that treat groups of values differently using their expected value distributions [19]–[21].

Approaches for learning the datatypes include: 1) Neural Architecture Search (NAS) [22] over a super-net with all candidate datatypes. 2) Reinforcement learning (RL) [23], [24]. The search space grows rapidly with the set of candidate datatypes severely limiting it in practice. 3) Per bit learning during training introduces high overhead of the per bit parameters, restricts bitlengths to powers of two [25] [26],
and sometimes, is limited to only weights [26]. We present BitPruning to jointly optimize bitlength and accuracy targeting both memory and compute benefits. Our goal is to squeeze out every possible benefit from reducing the bitlength used at any desired granularity and with any priority. BitPruning maps bitlengths from the discrete into the continuous domain and uses gradient descent to learn them. The most closely related method, Mixed Precision DNN (MPDNN) [27], adjusts bitlengths so that a model fits within a user-provided overall memory size constraint (e.g., the memory of a target device). MPDNN never tries to reduce bitlengths beyond this constraint, leaving improvement potential untapped. MPDNN cannot be used during hardware design or configuration time to determine the memory needed for an application or to target other criteria, e.g., operation count.

II. Method

We begin by defining a linear quantization scheme with integer bitlengths in the forward pass. We then expand it to use non-linear bitlengths and we describe how this interpretation allows bitlengths to be learned using gradient descent. Subsequently, we introduce a parameterizable loss function, which enables BitPruning to penalize larger bitlengths. Ultimately, we describe the final selection of integer bitlengths.

1) Quantization: The greatest challenge for learning bitlengths is that they represent discrete values over which there is no obvious differentiation. Let’s consider a uniform fixed-point quantization of a float value \( v \) to \( n \) bits:
\[
\text{Int}(V,n) = \text{Round}((V - L_{\text{min}})/\text{Scale}(n))
\]
where \( \text{Int}(V,n) \) is the integer value with bitlength \( n \), \( L_{\text{min}} \) the minimum value in the layer (across the whole batch), and \( \text{Scale} \) is the smallest representable difference:
\[
\text{Scale}(n) = (L_{\text{max}} - L_{\text{min}})/(2^n - 1)
\]
where \( L_{\text{max}} \) is the maximum value in the layer (batch). This scheme quantizes an input float value \( V \) to:
\[
Q_i(V,n) = L_{\text{min}} + \text{Int}(V,n) \cdot \text{Scale}(n)
\]
BitPruning’s goal is to learn \( n \). Unfortunately, the above quantization does not allow the learning of \( n \) with gradient descent due to its discontinuity and non-differentiability. To overcome this challenge, we expand the definition to real-valued \( n \), by interpolating between the values represented by the nearest two integers:
\[
Q_i(V, b + \alpha) = (1 - \alpha) \cdot Q_i(V, b) + \alpha \cdot Q_i(V, b + 1)
\]
where \( n = b + \alpha \), with \( 0 < \alpha < 1 \), and \( Q_i(V, b) \) is the integer bitlength quantization with \( b \) bits.

The scheme is applied to activations and weights separately. Since the minimum bitlength per value is 1, \( n \) is clipped at 1.0. This presents a reasonable extension of the meaning of bitlength in continuous space and allows for the loss to be differentiable with respect to bitlength. The final bitlength of each group for inference is then selected as the smallest integer greater or equal to the bitlength parameter learned during training.

During the forward pass, the formulae are applied to activations and weights. The values are converted to a floating point value that can be represented by integer quantization. For non-integer bitlengths their two nearest integer representations are interpolated. During the backward pass, we use the straight-through estimator [28] to prevent propagating zero gradients that result from the discontinuity of the rounding operation.

2) Loss Function: Our modified loss function \( L \) penalizes bitlength by adding a weighted average (with weights \( \lambda_i \)) of the bits \( n_i \) required for weights and activations of all layers:
\[
L = L_i + \gamma \sum (\lambda_i \times n_i)
\]
where \( L_i \) is the original loss function, \( \gamma \) is the regularization coefficient used for selecting how aggressive the quantization should be, \( \lambda_i \) is the relative importance of the \( i \)th group of values, and \( n_i \) is the bitlength of group’s values. \( L \) can be adjusted to target any quantifiable criteria. For most of this paper, we assume that the benefit of reducing bitlengths is equal across all groups. Section III-A2 considers other assumptions.

3) Final Bitlength Selection: Our method will produce non-integer bitlengths which are meaningless for practical hardware. Hence, the learned non-integer bitlengths are rounded up at the end. While this initially affects accuracy, Tables I and III show that continuing training recovers this drop. The final phase keeps bitlengths constant and only updates the weights.

III. Evaluation: Bitlengths and Accuracy

Without loss of generality, we report experimental results for per-layer granularity, and for weights and activations separately. As this requires many repeated training runs, these proof of concept experiments are performed with CIFAR10 [29]. Section III-B reports experiments with ImageNet [30]. Finally, Section III-C considers tasks other than image classification.

A. Proof of Concept

1) Bitlengths and Accuracy: We initially focus on AlexNet [31] and ResNet18 [32] to study the effect of the various training options. Training with BitPruning proceeds into two phases: 1) treat the bitlengths as continuous space parameters, then 2) map them to discrete values and continues adjusting the weight values to improve accuracy.

Phase 1: Learning the Continuous Bitlengths: The networks were trained over 300 epochs with the default fast.ai parameters and one cycle policy in PyTorch [33]. Table I shows top-1 validation accuracy for a 32b float (FP32) baseline, as well as BitPruning-quantized models. The bitlength weights \( \lambda_i \) are set to normalize all bitlengths to 8 bits and to emphasize all layers equally. If all layers use bitlength 8, loss will be \( \gamma \). We change \( \gamma \) to produce regularizers of progressively higher strength. Table I reports the resulting average bitlengths over all layers. Accuracies comparable to the baseline can be achieved with less than 3 bits on average for AlexNet, and 2 bits for ResNet. Stronger regularizers achieve smaller bitlengths with a slight degradation in accuracy. Weights consistently achieve smaller bitlengths than activations, while activations tend to benefit from more aggressive regularizers.
TABLE I: Activation/weight bitlengths and achieved accuracy of aggressive quantization and different strength regularizers on CIFAR10 for non-integer and integer bitlengths.

| Network     | γ    | TOP1 of bits | Wgs # of bits | Acts # of bits | TOP1 of bits | Wgs # of bits | Acts # of bits |
|-------------|------|--------------|---------------|----------------|--------------|---------------|----------------|
| AlexNet Base | 0.5  | 78.8         | 32 Float      | 32 Float       | 78.5         | 32 Float      | 32 Float       |
|             | 1.0  | 78.5         | 3.03          | 3.89           | 77.9         | 3.50          | 4.33           |
|             | 2.5  | 78.3         | 2.45          | 3.18           | 75.0         | 3.00          | 3.67           |
|             | 5.0  | 78.2         | 2.06          | 2.72           | 75.4         | 2.50          | 3.17           |
|             | 10.0 | Does not converge | Does not converge |                |              |                |                |
| ResNet18 Base | 0.5  | 94.9         | 32 Float      | 32 Float       | 94.9         | 32 Float      | 32 Float       |
|             | 1.0  | 93.5         | 1.30          | 2.26           | 94.1         | 1.43          | 2.90           |
|             | 2.5  | 93.1         | 1.15          | 2.01           | 93.4         | 1.24          | 2.43           |
|             | 5.0  | 92.8         | 1.14          | 1.99           | 93.3         | 1.24          | 2.48           |
|             | 10.0 | 94.1         | 1.61          | 2.35           | 94.2         | 1.90          | 2.90           |

Fig. 1: CIFAR10 accuracy (solid) and bitlength (dotted) during training for different regularizers.

Figure 1 shows the validation accuracy and average bitlengths of activations and weights over the 300 epochs. The bitlengths converge quickly and concurrently. While not shown in detail, within 30-40 epochs the bitlengths across all groups approach their final values. Further, the more aggressive γ's increase accuracy loss and decrease the bitlengths, as expected. However, using too high of a γ can adversely affect accuracy or bitlengths.

While we do not show these results, we note the following: 1) Bitlengths vary per layer, creating more opportunities for specialized hardware; Uniform per-network bitlengths would leave a lot of potential untapped. 2) Bitlengths show a slight descending trend towards the latter layers.

Phase 2: Final Discrete Bitlengths: After initial training, bitlengths are set to the ceiling of their learned value, resulting in an accuracy drop. There are two reasons for this: our interpolation is imperfect (larger bitlengths often, but not always give better accuracy) and the network is trained for the smaller bitlength. Crucially, fine-tuning the networks regains this lost accuracy within 60 epochs. Table I shows the drop in accuracy of integer bitlength networks as well as the effects of fine-tuning. In all cases, selecting the integer bitlengths increases the bitlengths by about 0.5 bits. Finally, Table I shows the validation accuracy and average bitlengths of these fine-tuned integer bitlength versions in comparison with the baseline and non-integer bitlength versions. Generally, integer and non-integer cases produce similar accuracies.

2) Targeted Optimization: Next, we train AlexNet and ResNet18 with a targeted loss function to minimize memory footprint and operation count. The bit loss is weighted according to the number of operations or elements in each layer, for activations and weights separately. Other hyper-parameters are kept the same. We consider memory footprint for inference with batch sizes 1 and 128, representing a weight and activation-heavy case, operation count and the original loss function.

TABLE II: Influence of loss function weighting on bitlengths: BS - Batch Size, MAC - Multiply Accumulate Operations

| Network | target | TOP1 of bits | Wgs # of bits | Acts # of bits | TOP1 of bits | Wgs # of bits | Acts # of bits |
|---------|--------|--------------|---------------|----------------|--------------|---------------|----------------|
| AlexNet | regular | 78.3         | 3.27          | 3.32           | 78.0         | 3.32          | 3.32           |
|         | BS 1   | 80.0         | 3.35          | 3.35           | 78.3         | 3.35          | 3.35           |
|         | BS 128 | 78.7         | 2.42          | 2.90           | 78.0         | 2.42          | 2.90           |
|         | MAC    | 79.1         | 2.89          | 3.39           | 78.4         | 2.89          | 3.39           |

ResNet18 | regular | 97.5         | 1.74          | 2.85           | 97.2         | 1.74          | 2.85           |
| BS 1     | 94.7    | 1.62         | 2.65          | 1.78           | 94.4         | 1.62          | 2.65           |
| BS 128   | 93.6    | 1.67         | 2.83          | 2.35           | 93.3         | 1.67          | 2.83           |
| MAC      | 94.1    | 1.53         | 2.35          | 1.74           | 93.8         | 1.53          | 2.35           |

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
granularities. While the figure is specifically for ResNet18, similar trends are exhibited by all networks. Generally, the first and last layers require more bits. Similarly, activations typically require larger bitlengths than weights.

3) Comparison with Other Techniques: Table IV clearly shows the advantage of our approach against uniform 4-bit (PACT) quantization and a profiled per-layer quantization in both validation accuracy and bitlength.

C. Other Tasks

Table V reports that BitPruning works well across a wide variety of models and tasks. All experiments used the default training parameters for the baseline and BitPruning.

IV. EVALUATION: HARDWARE BENEFITS

Table VI shows the benefits of BitPruning on several hardware designs (some are already used in practice and some are proposed). All measurements are normalized to the corresponding 8b baseline. A performance ratio higher than 1.0 means that BitPruning improves performance. A ratio below 1.0 for memory and energy means that BitPruning improves memory footprint and energy consumption respectively.

All accelerators were modeled using the same methodology for consistency. Since we do not have access to the actual hardware, we model their behavior via a custom cycle-accurate simulator. For energy estimation, models of the designs (compute units only) were synthesized for 1GHz operation with the Synopsys Design Compiler and laid out with Cadence Innovus for a TSMC 65nm library. To model the various memory components we use CACTI [35] for internal weight and activation buffers and model external DRAM as LPDDR3.

### TABLE III: ImageNet bitlength and validation accuracy.

| Network   | TOP1 | Wgts # of bits | Acts # of bits | TOP1 | Wgts # of bits | Acts # of bits |
|-----------|------|----------------|---------------|------|----------------|---------------|
| AlexNet   | Base | 57.12          | 32            | 57.12| 32 Float       | 32 Float      |
| ResNet18  | Base | 69.54          | 32            | 69.54| 32 Float       | 32 Float      |
| MobileNet V2 | 1.0  | 70.99          | 32            | 70.99| 32 Float       | 32 Float      |

![Fig. 2: Number of bits for ResNet18 activations/weights for each layer listed in a breadth first manner.](image]

### TABLE IV: Comparison with other quantization techniques

| Method   | TOP1 | Wgts | Acts | TOP1 | Wgts | Acts |
|----------|------|------|------|------|------|------|
| AlexNet  | Prof | 55.7 | 7.63 | 69.2 | 4.38 | 4.38 |
| ResNet18 | Prof | 55.8 | 7.63 | 65.6 | 6.41 | 6.34 |
| MobileNet V2 | Prof | 55.1 | 3.88 | 69.2 | 3.38 | 4.14 |

### TABLE V: Demonstration of BitPruning on various tasks

| Accelerator | Baseline | Prof | Perf | Mem | Pwr | Perf | Mem | Pwr |
|-------------|----------|------|------|-----|-----|------|-----|-----|
| Stripes [3]  | 1.69*    | 0.95*| 0.69*| 1.26*| 0.98*| 0.84*|     |     |
| Dpred [6]    | 5.97*    | 0.91*| 0.43*| 3.35*| 0.91*| 0.46*|     |     |
| BitFusion [4]| 1.63*    | 0.60*| 0.61*| 1.00*| 1.00*| 1.00*|     |     |
| Loom [7]     | 3.74*    | 0.27*| 0.27*| 2.81*| 0.33*| 0.54*|     |     |
| Protues [8]  | —       | 0.47*| 0.85*| —    | 0.98*| 0.99*|     |     |

| Accel. | Baseline | Prof | Perf | Mem | Pwr |
|--------|----------|------|------|-----|-----|
| Stripes [3]  | 1.72*    | 0.94*| 0.71*| 1.23*| 0.98*| 0.87*|     |     |
| Dpred [6]    | 3.98*    | 0.89*| 0.48*| 3.88*| 0.90*| 0.49*|     |     |
| BitFusion [4]| 2.47*    | 0.53*| 0.45*| 1.00*| 1.00*| 1.00*|     |     |
| Loom [7]     | 4.11*    | 0.29*| 0.26*| 3.44*| 0.35*| 0.31*|     |     |
| Protues [8]  | —       | 0.50*| 0.81*| —    | 0.81*| 0.93*|     |     |

### TABLE VI: Trained vs profiled quantization on select accelerators.

| Method   | TOP1 | Wgts | Acts | TOP1 | Wgts | Acts | TOP1 | Wgts | Acts |
|----------|------|------|------|------|------|------|------|------|------|
| AlexNet  | Prof | 55.7 | 7.63 | 69.2 | 4.38 | 4.38 | 69.9 | 7.33 | 7.02 |
| ResNet18 | Prof | 55.8 | 7.63 | 65.6 | 6.41 | 6.34 | 69.9 | 7.33 | 7.02 |
| MobileNet V2 | Prof | 55.1 | 3.88 | 69.2 | 3.38 | 4.14 | 70.1 | 4.15 | 4.57 |

Download :: [link] for full dataset.
