Dynamic Probabilistic Pruning: A General Framework for Hardware-Constrained Pruning at Different Granularities

Lizeth Gonzalez-Carabarin®, Iris A. M. Huijben®, Graduate Student Member, IEEE, Bastian Veeling, Alexandre Schmid®, Senior Member, IEEE, and Ruud J. G. van Sloun®, Member, IEEE

Abstract— Unstructured neural network pruning algorithms have achieved impressive compression ratios. However, the resulting—typically irregular—sparse matrices hamper efficient hardware implementations, leading to additional memory usage and complex control logic that diminishes the benefits of unstructured pruning. This has spurred structured coarse-grained pruning solutions that prune entire feature maps or even layers, enabling efficient implementation at the expense of reduced flexibility. Here, we propose a flexible new pruning mechanism that facilitates pruning at different granularities (weights, kernels, and feature maps) while retaining efficient memory organization (e.g., pruning exactly \( k \)-out-of-\( n \) weights for every output neuron or pruning exactly \( k \)-out-of-\( n \) kernels for every feature map). We refer to this algorithm as dynamic probabilistic pruning (DPP). DPP leverages the Gumbel-softmax relaxation for differentiable \( k \)-out-of-\( n \) sampling, facilitating end-to-end optimization. We show that DPP achieves competitive compression ratios and classification accuracy when pruning common deep learning models trained on different benchmark datasets for image classification. Relevantly, the dynamic masking of DPP facilitates for joint optimization of pruning and weight quantization in order to even further compress the network, which we show as well. Finally, we propose novel information-theoretic metrics that show the confidence and pruning diversity of pruning masks within a layer.

Index Terms— Deep learning (DL), hardware-oriented pruning, model compression.

I. INTRODUCTION

The evident success of deep learning (DL) models and its multiple applications [1], [2] is accompanied by a steadfast growth in the number of hyperparameters and computational cost. This has become a bottleneck for hardware deployment, which is constrained to certain computational and memory budgets. For instance, the VGG-16 architecture occupies more than 500 MB of storage and performs \( 1.6 \times 10^{10} \) floating-point arithmetic operations [3], [4]. In contrast, field-programmable gate array (FPGA)-based platforms are constrained to a few thousands of computing operations, making them unsuitable for deployment of large DL models. To shrink such big models, different solutions have been proposed such as quantization [5] (and in the extreme case, binarization [6]), knowledge distillation [7], weight sharing [8], and model pruning. Pruning has gained notable attention since the pruned model performance was found to yield on par performance compared with nonpruned counterparts. Remarkably, pruning has been shown to prevent overfitting as well [9]–[11]. Pruning algorithms can result in either structured or unstructured pruning strategies. Structured pruning methods remove model parts at the level of, e.g., layer, channels, or individual feature maps [12]. We refer to this kind of pruning as structured coarse-grained pruning. In addition, we refer to medium-grained pruning as those methods that remove kernels. To avoid ambiguity, across this article, we refer a 2-D slice of a filter as kernel. Unstructured pruning, on the other hand, prunes individual weights (i.e., it sets a fraction of weights to zero). We refer to this type of pruning as unstructured fine-grained pruning. Unstructured fine-grained pruning has achieved impressive compression ratios; nevertheless, implementing fine-grained sparse matrices in dedicated hardware is a challenging task due to the typically irregular distribution of nonzero values. Usually, to avoid storage of a large number of zeros, nonzero values are stored in specific formats [13]. Several other works have proposed more sophisticated compressing coding techniques for sparse weights targeting hardware implementations [14]–[16]. For instance, the COO (Coordinate) format stores the indexes of columns and rows of nonzero values; the ELLPACK format only stores the column indexes, but zero padding is required, wasting a large amount of memory and leading to poor bandwidth usage; the compress sparse raw (CSR) format stores the column indexes and an additional vector for row offset, while compress sparse column (CSC) operates in a similar way with a column offset [see Fig. 1(a)]. Besides, additional control logic is required to compute operations (e.g., matrix multiplication) with such formats, increasing the complexity and power consumption for embedded applications [17]–[19].

Manuscript received 14 July 2021; revised 1 March 2022; accepted 10 May 2022. Date of publication 8 June 2022; date of current version 5 January 2024. This work was supported by the Marie Sklodowska-Curie Actions. (Corresponding author: Lizeth Gonzalez-Carabarin.) Lizeth Gonzalez-Carabarin, Iris A. M. Huijben, and Ruud J. G. van Sloun are with the Electrical Engineering Department, Eindhoven University of Technology, 5612 AP Eindhoven, The Netherlands (e-mail: l.gonzalez.carabarin@tue.nl).
Bastian Veeling with the AMLab, Informatics Institute, University of Amsterdam, 1098 XH Amsterdam, The Netherlands.
Alexandre Schmid is with the School of Engineering, EPFL, CH-1015 Lausanne, Switzerland.
This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TNNLS.2022.3176809.
Digital Object Identifier 10.1109/TNNLS.2022.3176809.
Therefore, we propose a framework that naturally generates structured sparsity for several levels of granularity, by fixing the number of active elements within a candidate set (comprising, e.g., weights, kernels, and feature maps) to $K$. Fig. 1(b) shows the benefits of our approach for unstructured fine-grained versus structured fine-grained pruning (our approach). Notice that both sparse matrices [Fig. 1(a) and (b)] have the same sparsity level, and however, the structure of our approach reduces the memory usage for hardware implementations. In addition, our framework is capable of integrating both pruning and quantization, to benefit from both techniques. The proposed method exploits the notion of deep probabilistic subsampling (DPS) [20] to dynamically generate stochastic pruning masks during training, which are independent of the magnitude of the weights, allowing for direct applications to quantize and even binarize weights in one run without the need of post-training fine-tuning. The main contributions of this work are given as follows.

1) We propose deep probabilistic pruning, which learns to generate hardware-oriented sparse structures for different levels of granularity: fine-grained (weights), medium-grained (kernels), and coarse-grained (feature maps), facilitating memory allocation and access for hardware implementations. We adopt a layerwise sparsity level that can be selected by the user, which is beneficial in case of hardware constraints that dictate maximum memory usage and specific patterns.

2) Due to the fact that DPP is not a magnitude-based pruning algorithm, it allows for joint optimization of pruning masks and quantization (and even binarization) of network parameters, producing ultra-compressed models with low memory and low complexity, suitable for further dedicated hardware.

3) Leveraging the probabilistic nature of DPP, we propose novel information-theoretic metrics that capture the confidence and diversity of the pruning masks leveraged within a network layer. We show how these metrics differ during training and between fully connected (FC) and convolutional layers.

II. RELATED WORK

In this section, we give an overview of different pruning strategies. Network pruning approaches can be coarsely divided into single-stage and multistage strategies, strategies that prune at different granularities, i.e., fine-grained (weights), medium-grained (kernels), and coarse-grained (feature maps), and approaches that do or do not result in hardware-friendly pruned models.

Hardware-friendly pruning is achieved when the structure is present in the pruning pattern since an unstructured selection results in memory storage using matrices that need (memory-inefficient) zero filling. When pruning entire feature maps (or even layers), such structure is present, and therefore, conventionally, the term coarse-grained pruning is often used analogously with the term structured pruning. Similarly, fine-grained pruning is often referred to as unstructured pruning. Following the terminology introduced in [21], we slightly redefine terms and use structured pruning for methods that result in hardware-friendly pruned models at any granularity.

Our approach generalizes advantages of different methods into one general framework that—opposed to other approaches—facilitates single-stage, hardware-friendly pruning at any granularity; fine-, medium-, and coarse-grained. In addition, it allows for joint optimization of pruning and quantization (and even binarization).

A. Unstructured Fine-Grained Pruning

The early work of [22] proposed a three-stage pipeline for unstructured fine-grained pruning, which was later extended to deep compression [8]. These works were followed by single-stage pruning approaches that jointly/dynamically optimize both the model parameters and pruning process. Zhu and Gupta [23] and Narang et al. [24] pruned weights based on the gradual increment of sparsity during training, while a magnitude-based pruning framework in which a minimal sparsity value is set is proposed in [25]. Furthermore, the work [26] proposes dynamic sparse reparameterization (DSR) based on an adaptive threshold for pruning and an automatic reallocation of pruning parameters across layers. The work [27] proposes to select the weights with the highest momentum, which significantly improved accuracy. All these works yield unstructured fine-grained pruning matrices, which leads to inefficiencies in terms of memory access and allocation considering current hardware platforms [28], [29].
B. Structured Coarse-Grained Pruning

To avoid unstructured pruning masks, previous work has explored pruning at the architectural level (rather than the weight level) such as pruning feature maps or layers. In fact, it has been experimentally shown that rather than eliminating weight connections, pruning at the architectural level may offer more benefits to reduce memory while retaining the state-of-the-art accuracy [12]. Most works adopt a multistage approach, where pruning and model training occur disjointly [30]–[35]. Such multistage methods typically suffer from performance drops after pruning, requiring fine-tuning steps. The magnitudes of the weights are often used as an indicator of the necessity or not to prune an entire structure; Li et al. [31] pruned feature maps based on the sum of the absolute magnitudes of the corresponding weights, and Wen et al. [36] proposed the structured sparsity learning (SSL) method to prune feature maps, channels, and depth structures. Different from the magnitude-based approaches, Hu et al. [37] relied on the output of activation layers and calculated an average percentage of zeros as weighting of the feature map relevance. Kang and Han [38] and Wang et al. [39] leveraged, opposed to the other methods, a single-stage approach. The work [39] most resembles our general framework when applied on the coarse level. However, we are guaranteed to exactly select k-out-of-n feature maps, while this number k is only approximated using a proxy for the l0 penalty during training in the work [39]. This is important as this enables the designer/user to select this number according to given hardware constraints. Second, unlike [39], our method is not exclusively applied for pruning feature maps but allows for pruning at fine- and medium-grained levels as well within the same framework. To conclude, all methods reviewed in this paragraph prune at coarse-grained level (layers or feature maps) and have not been transferred to prune models at fine-grained levels (i.e., weights), making them more restricted than our generalized framework that is suitable for pruning any of the granularities.

C. Structured Fine-Grained Pruning

To the best of our knowledge, only a few works have addressed the issue of structured pruning of fine-grained elements (weights). Ma et al. [40] introduced convolutional sparse patterns for hardware-oriented pruning; however, this approach limits the possible pruning patterns to only a few options, reducing its flexibility. The work [41] also introduces a hardware-friendly sparse approach for kernel pruning; however, it requires several stages to achieve competitive results, which contrasts with our single-stage approach. The generation of structured fine-grained pruning by generating sparse patterns compatible with their GPU A100 was proposed in [42]; the approach also considers a multistage approach, and it is focused on fine-grained pruning, without exploring other levels of granularity. Recently, Zhou et al. [21] extended the work [42] to a single-stage approach, by jointly training the model to generate structured fine-grained pruning from scratch. Their method in its current form, however, is only applicable to fine-grained pruning since the pruning mask is directly created from the magnitudes of the weights. Moreover, the authors adopt the straight-through estimator (STE) to circumvent the nondifferentiable pruning procedure, while we adopt a more principled gradient estimator, dedicated to differentiable subset sampling [43]. Finally, Kundu et al. [44] proposed a hardware-friendly predefined structure; however, the lack of joint optimization of pruning and weights leads to a large degradation of accuracy of the pruned models.

D. Pruning and Quantization

Works that optimize pruning and quantization offer ultra-compressed models for higher memory savings for unstructured pruning [45], [46] and structured pruning [8], [45]–[51]. Nevertheless, many of them require several stages during training to achieve this integration; none of these works has achieved joint optimization of quantization and structured fine-grained pruning.

III. METHODOLOGY

A. Notation

We introduce a neural network with L layers, indexed with i. Each layer is parameterized by a matrix $W^{(i)} \in \mathbb{R}^{N^{(i-1)} \times a \times N^{(i)}}$ and a bias vector $b^{(i)} \in \mathbb{R}^{N^{(i)}}$ (which we ignore in the rest of the notations), where $N^{(i-1)}$ and $N^{(i)}$ are the number of feature maps (or channels) of layer $i-1$ and $i$, respectively, and $a = (\sqrt{a} \times \sqrt{a})$ denotes the size of a 2-D convolutional kernel.

The output of the $i$th layer can be defined as $x^{(i)} = g_{W^{(i)}}(x^{(i-1)})$, where the functionality of $g_{W^{(i)}}$ depends on the layer being, e.g., fully connected (FC) or convolutional, and linearly or nonlinearly activated. Note that the FC layer is a special case of a convolutional layer with $a = 1$.

B. Dynamic Masking Based on Probabilistic Subsampling

We aim to jointly optimize the parameters of the model while simultaneously learning to prune them. In this section, we explain how our framework, which we refer to as dynamic probabilistic pruning (DPP), achieves this joint learning.

For all layers $s^{(i)}$, with $i \in \{1, \ldots, L\}$, we introduce a binary mask $M^{(i)} \in \{0, 1\}$ parameterized by $\Phi^{(i)}$. By means of elementwise multiplication, it activates a subset of $s^{(i)}$ elements from $W^{(i)}$. The generation of these masks $M^{(i)}$ follows the DPS-topK framework [20], on which we will elaborate here.

Huijben et al. [20] proposed an end-to-end framework for joint learning of a discrete sampling mask with a downstream task model by introducing DPS, a parameterized generative sampling model

$$P(M^{(i)}|\Phi^{(i)}).$$

Trainable parameters $\Phi^{(i)} \in \mathbb{R}^{N^{(i-1)} \times a \times N^{(i)}}$ denote unnormalized log probabilities (logits). In order to generate a binary sampling mask realization $\tilde{M}^{(i)}$ from $\Phi^{(i)}$, we select/sample exactly $K$ unique values over one axis of $\Phi^{(i)}$. This implies that pruning takes place over this same axis in $W^{(i)}$ (e.g., pruning the $N^{(i)}$ channels of layer $i$ implies sampling...
over the axis of length $N^{(i)}$. As this pruning axis differs per case in Section IV, we denote it with $p$-axis (pruning axis) temporally. Each element in $\Phi^{(i)}$ is thus the (unnormalized) log probability of activating the corresponding element in $W^{(i)}$. Note that due to the fact that the binary mask $\tilde{M}_{\Phi^{(i)}}$ is parameterized on $\Phi^{(i)}$ rather than $W^{(i)}$, we can combine DPP with quantization of the values in $W^{(i)}$, which can jointly be learned as well.

Similarly, as in [20] and [52], we adopt Gumbel top-$K$ sampling [53], [54] to draw $K$ unique elements from the elements in the pruning axis. We denote this operation by $\text{top}_{K_{p-axis}}(\cdot)$, where the subscript $p$-axis indicates that sampling is only performed over the pruning axis. In order to create a binary sampling mask of equivalent size as $W^{(i)}$, we transform the samples to a $K$-hot vector, which contains $K$ ones at the selected indices and zeros at the remaining/nonselected positions. Formally, we define the binary mask realization $\tilde{M}_{\Phi^{(i)}}$ as

$$\tilde{M}_{\Phi^{(i)}} = \text{Khot}_{p-axis}\left(\text{top}_{K_{p-axis}}(\Phi^{(i)} + \beta E^{(i)})\right)$$

(2)

where $E \in \mathbb{R}^{N^{(i)} \times 1}$ are independent and identically distributed (i.i.d.) Gumbel noise samples from $\text{Gumbel}(0, 1)$, scaled with a scalar $0 < \beta \leq 1$. Note that the pruning axis relates only one of the three axes of the 3-D matrix $\tilde{M}_{\Phi^{(i)}}$. Therefore, the total number of active elements $S_p$ in $\tilde{M}_{\Phi^{(i)}}$ does not equal $K$, but $K$ times the size of the two remaining axes.

If we allowed $S_p$ ones to be distributed randomly within $\tilde{M}_{\Phi^{(i)}}$, it would result in unstructured pruning of the model when applying this mask elementwise on $W^{(i)}$. However, by demanding exactly $K$ elements to be active over the pruning axis, we enforce structure in the binary mask $\tilde{M}_{\Phi^{(i)}}$.

To define structure at different granularities (e.g., pruning weights, kernels, or feature maps), we “tie” together certain elements in $\Phi^{(i)}$ such that they all update equivalently during training. It effectively reduces the number of trainable logits within $\Phi^{(i)}$. As a result, if, for example, all logits over the kernel axis with size $\sqrt{a} \times \sqrt{a}$ are tied, a kernel will be either activated or deactivated as a whole, rather than on weight level. Section III-C elaborates on structured pruning at different granularities.

During backpropagation, the $\text{Khot}_{p-axis} \circ \text{top}_{K_{p-axis}}$ operation must be relaxed as it is nondifferentiable. The works [20], [52] proposed to adopt the Gumbel-softmax relaxation [55], [56] to this aim. It relaxes the nondifferentiable argmax operation using a temperature $(\tau)$-parameterized $\text{softmax},(\cdot)$ function. Kool et al. [54] showed that sampling $K$ times without replacement from the same distribution is equivalent to top-$K$ sampling, and Xie and Ermon [43] showed that iterative sampling without replacement from its relaxed counterpart (using the softmax) is a valid top-$K$ relaxation. As such, we can directly leverage Gumbel-softmax sampling without replacement during backpropagation in order to flow gradients to $\Phi^{(i)}$ of $\forall i \in \{1, \ldots, L\}$. During training, we use a penalization term $L_c$ that penalizes high-entropy distributions, therewith promoting (close-to) one-hot categorical distributions toward the end of training. This penalization term is defined as follows:

$$L_c(\Phi) = -\sum_{i=1}^{D} \sum_{j=1}^{C} \pi_{i,j} \log \pi_{i,j}$$

(3)

where $\pi_{i,j} = (\exp{\phi_{i,j}} / \sum_j \exp{\phi_{i,j}})$ is the normalized probability of class $j$ in the $i$th distribution, $D$ is the number of independent categorical distributions, and $C$ is the number of classes of each of these distributions.

Fig. 2 and Algorithm 1 provide a schematic overview and pseudocode, respectively, of the full training procedure.

C. DPP for Pruning Different Levels of Granularity

As explained in Section III-B, DPP learns a binary mask $\tilde{M}_{\Phi^{(i)}}$ that selects exactly $S_a$ elements from $W^{(i)}$. By connecting trainable log probabilities in $\Phi^{(i)}$ during training, we enforce pruning at different granularities. We define three different pruning scenarios: 1) fine-grained pruning (DPP-F) during training, we enforce pruning at different granularities. We define three different pruning scenarios: 1) fine-grained pruning (DPP-F); 2) medium-grained pruning (DPP-M); and 3) coarse-grained pruning (DPP-C).

Fig. 3 shows the three scenarios. DPP-F activates $K$ out of $a$ kernel weights for each (2-D) kernel within each (3-D) feature map. For this scenario, the 2-D kernels are flattened and $K$ elements are selected from the resulting 1-D vector with size $a$.

DPP-M, on the other hand, activates $K$ out of $N^{(i-1)}$ kernels per $N^{(i)}$ feature maps of layer $i$. Finally, DPP-C activates $K$ entire feature maps per layer. Table I summarizes the three different scenarios and also indicates the number of values to be stored in memory in case of hardware implementation of the pruned model (e.g., on an FPGA). In addition, for clarification, we indicate the effective number of trainable logits within $\Phi^{(i)}$ as a result of connecting logits over certain axes to enforce pruning at different granularities. Taking DPP-M as an example, as we prune $K$ out of $N^{(i-1)}$ entire kernels, the weights axis (of size $a$) has tied logits, as either all or none of the weights in a kernel are activated by the binary mask. We can interpret the resulting 2-D logits matrix $\Phi^{(i)}$ as
containing the log probabilities of $N^{(i)}$ number of categorical distributions, each containing $N^{(i-1)}$ classes.

In the particular case of pruning connections within FC layers, DPP activates $K$ (out of $N^{(i-1)}$) input neurons for each $N^{(i)}$ output neurons. Given the different granularity definitions that we proposed, this pruning situation fits to DPP-M, with $a = 1$. However, as the weights of the input neurons are the smallest possible entity to be pruned in FC layers, this case is in the literature often referred to as structured fine-grained pruning [21], [42]. In the rest of this article, we, therefore, use DPP-F when referring to the special case of pruning connections in FC layers.

D. Training Details

Here, we elaborate on the training details of DPP. We jointly train the model parameters \{\(\Phi, W, b\)\} and the unnormalized logits in \(\Phi\), by means of error backpropagation of the total loss with respect to these parameters. The downstream task model loss is defined as the cross entropy between the targets and the predictions parameterised by \(\Phi\). During training, the unnormalized log probabilities \(\Phi^{(i)}\) (\(i\) \(\in\) \(\mathbb{I}\)) are constantly being updated. Also, the realization of the Gaussian noise matrix \(E^{(i)}\) differs per element within a mini-batch and over training/epochs. As a result, binary mask realizations vary during training and within the mini-batches, allowing the model to efficiently explore different pruned model instantiations. While in the original Gumbel-max trick [53], the Gumbel noise is typically unscaled when sampling from the distribution, heuristically, we found the improved model performance when downscaling the Gumbel noise with a factor $\beta$. In the experiments where we combine DPP with parameter quantization, we follow the quantization procedure proposed in [57]. In all experiments, we prune (and in some cases quantize) only $W$, and not $b$.

Algorithm 1 Dynamic Probabilistic Pruning

Require: Training dataset \(D\), neural network with \(L\) layers, and initialized trainable parameters \{\(\Phi, W, b\)\}. Number of active elements \(K\), Pruning axis \(p\)-axis, Gumbel noise scaling \(\beta\), temperature annealing settings \{\(\tau_{\text{init}}, \tau_{\text{end}}\)\} = \{5.0, 0.5\}, Loss scaling \(\mu\), number of epochs \(n_{\text{iter}}\).

Ensure: Model with trained parameters $\widetilde{W}$ and $\widetilde{b}$, binary mask realizations $\widetilde{M}_{\Phi}$ parameterised by $\Phi$.

- Compute: $\Delta \tau = \frac{n_{\text{iter}} - n_{\text{init}}}{n_{\text{iter}}}$

for $n = 1$ to $n_{\text{iter}}$ do
  // Forward pass
  - Draw random batch $x_n \sim D$
  for $i = 1$ to $L$ do
    - Draw i.i.d. Gumbel noise samples: $E^{(i)}$
    - Sample binary mask: $\widetilde{M}_{\Phi}^{(i)} = \text{Khot}_{p\text{-axis}}\left(\text{top}_K^{p\text{-axis}}(\Phi^{(i)} + \beta E^{(i)})\right)$
    - Apply mask: $\widetilde{W}^{(i)} = W^{(i)} \otimes \widetilde{M}_{\Phi}^{(i)}$
  end for
  - Compute output: $\hat{x}_n = g_{\tilde{L}} \circ g_{\tilde{B}}(x_n)$
  - Compute loss: $L_{CE}(\hat{x}, \hat{x}) + L_{\Phi}(\Phi)$

end for
  // Backward pass
  - Set: $\tau = \tau_{\text{init}} - (i - 1) \cdot \Delta \tau$
  - $\nabla_{\Phi} L_{\Phi} \propto \nabla_{\Phi} E \left[\text{softmax}_{p\text{-axis}}(\Phi + \beta E)\right]$
  - Update: $\{\Phi, W, b\} \propto L_{CE}(\hat{x}, \hat{x}) + \mu L_{\Phi}(\Phi)$
  
Fig. 3. Visualization for activating exactly $K$ elements at three different granularities (weights, kernels, and feature maps). All adopted values are illustrative. Black squares of masks denote selected connections. We also show how the structure in our sparse matrices enables efficient memory implementation for all three cases. (a) Fine-grained pruning (DPP-F). (b) Medium-grained pruning (DPP-M). (c) Coarse-grained pruning (DPP-C).
as the weights in $W$ contribute to the largest part of the parameters in the model.

During inference, one binary mask $\tilde{M}_{\Phi^{(i)}}$ per layer is drawn from the trained log probabilities $\Phi^{(i)}$, which is then used to prune the model and compute the performance on the test set.

**E. Information-Theoretic Metrics on Sparsity Confidence and Diversity**

To get insight into the training dynamics of DPP, we are interested in the change of confidence and diversity of the pruning patterns as training progresses. The probabilistic nature of DPP enables the use of the information-theoretic measures, entropy and mutual information, to evaluate this. We compute these metrics per layer $i$ due to the heterogeneity of the masks between layers.

As defined in Section III-C, within each layer $i$, $D \geq 1$ number of independent categorical distributions, each with $C$ classes is being trained, from which we sample $K$ out of $C$ elements (weights, kernels, or feature maps) without replacement. The average entropy of these $D$ pruning distributions reflects how confident the sparsity patterns on average are within this layer; the lower this average pruning entropy, the sparser the distributions, and thus, the more certain the model is about the binary mask to be applied. Note that this metric can only be computed for DPP-F and DPP-M, as DPP-C implies $D = 1$ (see Table I).

We can measure the average entropy from these $D$ pruning distributions in the $i$th-layer mask marginal probabilities $\{\pi^{(i)} : 0 \leq \pi^{(i)}_d \leq 1, \sum \pi^{(i)}_d = K\}$. No tractable function exists to compute marginal probabilities $\pi$ from the unnormalized log-probabilities in $\Phi$. Instead, we can easily obtain a Monte Carlo estimate by computing the average of $T$ realizations of $M_{\Phi^{(i)}}$.

$$\pi^{(i)} \approx \frac{1}{T} \sum_{t=1}^{T} \tilde{M}_{\Phi^{(i)}}, \quad \tilde{M}_{\Phi^{(i)}} \sim P(M_{\Phi^{(i)}}|\Phi^{(i)})$$

which can effectively be estimated in parallel after every epoch for $T = 100$. It is trivial to show that the entropy of any Gumbel-top-$K$ distributed variable $x$ can be computed using the typical Shannon entropy $H(x) = -\sum_{c \in x} P(c \in x) \log P(c \in x)$ and is upper bounded by $-K \log(K/C)$. As such, we can compute the average pruning entropy for layer $i$.

$$H(M_{\Phi^{(i)}}|d) = \frac{1}{D} \sum_{d=1}^{D} \left[ -\sum_{c=1}^{C} \pi^{(i)}_{d,c} \log \pi^{(i)}_{d,c} \right].$$

Furthermore, we can measure the diversity of the different sparsity patterns (within layer $i$) that result from sampling from the $D$ independent categorical distributions. This diversity can be measured as the mutual information between the different masks in layer $i$.

This pruning diversity metric can be formalized as

$$I(M_{\Phi^{(i)}}, d) = H(M_{\Phi^{(i)}}) - H(M_{\Phi^{(i)}}|d)$$

where $H(M_{\Phi^{(i)}})$ denotes the entropy of the average mask in layer $i$.

$$H(M_{\Phi^{(i)}}) = -\sum_{c=1}^{C} \bar{\pi}^{(i)}_c \log \bar{\pi}^{(i)}_c , \quad \text{with} \quad \bar{\pi}^{(i)} = \frac{1}{D} \sum_{d=1}^{D} \pi^{(i)}_d \in \mathbb{R}^{1 \times C}.$$  

### IV. EXPERIMENTS

Table II shows the terminology used for each experiment, as well as its description. Notice that for pruning FC layers, only DPP-F can be used, while for pruning convolutional layers, DPP-F, DPP-M, and DPP-C can be used. We first assess DPP for pruning weights (DPP-F-F) on small architectures (LeNet) for the MNIST dataset. In addition, we will demonstrate the performance in LeNet architectures in combination with quantized and binary weights based on [57]. Across these experiments, we set a $K$ value per layer, which determines the exact number of nonpruned weights assigned to each output neuron and kernel for FC and convolutional layers, respectively. Second, we test DPP for structured medium- and coarse-grained pruning (DPP-M-F and DPP-C-F) in medium-size networks (VGG-16, MobileNet v1, and ResNet-18). For DPP-C, we use $K$ as a selector of the number of feature maps that must remain active in each layer, while for DPP-M, $K$ is the number of nonpruned kernels per output feature map. In addition, we integrate quantization for medium-size

| Granularity level | Pruning axis | Effective nr of trainable logits per layer $i$ | D: nr of independent categorical distr. | $S_a$: nr of active weights in $W^{(i)}$ | $S$: Stored values per layer $i$ |
|-------------------|--------------|-----------------------------------------------|----------------------------------------|---------------------------------|------------------|
| Fine (DPP-F)      | Kernel weights (K out of a) | $N^{(i-1)} \times a \times N^{(i)}$ | $N^{(i-1)} \times N^{(i)}$ | $N^{(i-1)} \times K \times N^{(i)}$ | $2S$ |
| Medium (DPP-M)    | Kernels (K out of $N^{(i-1)}$) | $N^{(i-1)} \times 1 \times N^{(i)}$ | $1 \times N^{(i)}$ | $K \times a \times N^{(i)}$ | $S + KN^{(i)}$ |
| Coarse (DPP-C)    | Feature maps (K out of $N^{(i)}$) | $1 \times 1 \times N^{(i)}$ | $1 \times 1$ | $N^{(i-1)} \times a \times K$ | $S$ |
architectures (VGG-16). We train all our models from scratch, without using any pretrained model. Quantized models are jointly trained with pruning, without any fine-tuning stage.

### A. Metrics

In this section, we explain the used metrics. Nonpruned accuracy refers to the baseline accuracy for nonpruned networks, while pruned accuracy refers to the accuracy obtained after the network is pruned. $\Delta_{\text{acc}}$ is obtained after the network is pruned, and it is the difference between the pruned accuracy and the nonpruned accuracy. We also include the remaining parameters after pruning, as well as the sparsity $S_p$. Since in some experiments, DPP is jointly integrated with quantization [57], the bit representation is considered, as well as the compression ratio ($cr$) after pruning and quantization. To calculate the compression ratio, we use the following formula:

$$cr = \frac{1}{1 - \frac{S_p}{100}} \times \frac{B_{\text{ns}}}{B_q}$$ (8)

where $B_{\text{ns}}$ is the bit representation for the nonsparse and nonquantized network and $B_q$ is the bit representation if quantization is applied. Notice that if quantized values are not used, then $(B_{\text{ns}}/B_q)$ remains as one. Finally, the epoch time during training is included for each experiment.

### B. Experimental Setup

We have performed experiments on four datasets: MNIST, CIFAR-10, CIFAR-100, and Tiny ImageNet, for which the experimental setup (hyperparameters) is shown in Table III. All values of $K$ were manually set. For all experiments, we used a NVIDIA GPU GeForce GTX1080.

1) **MNIST**: We evaluate DPP first on the MNIST benchmark dataset consisting of a total of 70000 grayscale images of handwritten digits having a size of $28 \times 28$ pixels. We evaluate the performance on the architectures: LeNet 300-100 and LeNet-5 [58].

2) **CIFAR-10 and CIFAR-100**: We evaluate DPP on VGG-16 [59] and MobileNet-v1 for both CIFAR-10 and CIFAR-100. In addition, we assess DPP on ResNet18 trained on CIFAR10. Data augmentation was used for these experiments.

3) **Tiny ImageNet**: In addition, we evaluate DPP on ResNet18 trained on Tiny ImageNet, which consists of 200 classes and 100000 samples of size $64 \times 64$. For this case, the FC layer remains unpruned. Data augmentation was used for these experiments.

### C. Results

1) **MNIST**: For this experiment, we compare DPP with one of the latest fine-grained pruning algorithms [60] and show the results in Table IV. Notice that, joint quantization and pruning lead to higher compression ratios with little degradation on accuracy. Remarkably, DPP allows to jointly prune and binarize both networks, which leads to ultrahigh compression ratio with little accuracy degradation.

2) **CIFAR-10 and CIFAR-100**: In these experiments, both convolutional and FC layers are structurally pruned, in comparison with most conventional structured pruning approaches (which only prune convolutional layers). Results are shown in Table V. Relevantly, the accuracy degradation is minimal for all cases, and in some scenarios, $\Delta_{\text{acc}}$ is positive, meaning that accuracy increased after pruning. Moreover, quantization was jointly optimized with pruning for CIFAR-10, which leads to higher compression rates with an acceptable accuracy degradation. In addition, Figs. 4 and 5 shows the test accuracy...
for different compression ratios, and it also shows that DPP has a better performance than other methods for structured pruning. In the case of MobileNet v1 trained on CIFAR-100, the work [61] shows a higher test accuracy, and however, this is due to its higher nonpruned accuracy; in this case, notice that the number of our remaining parameters is less than the ones obtained in [61], but our $\Delta_{\text{acc}}$ shows that DPP reaches higher sparsity levels with a higher improvement of the test accuracy in comparison with its own nonpruned baseline.

3) Tiny ImageNet: Results are shown in Table V, and they confirm that our pruned model has a better test accuracy than similar works [44]. Relevantly, our model has a better $\Delta_{\text{acc}}$, suggesting that our pruning approach leads to less performance degradation.

Important to remark is that during inference, we take the top-$K$ values of the distribution (without adding Gumbel noise), to create the final pruning mask and test the resulting pruned model. In order to expand the results’ analysis, we have performed additional simulations for which, during inference, we have sampled ten realizations of the pruning mask (thus including Gumbel noise again). This experiment was performed on two datasets: MNIST and CIFAR-10 trained on LeNet300-100 and VGG-16, respectively. The obtained variance is 0.0044 and 0.00067 for LeNet300-100 and VGG-16, respectively, which leads to variations in accuracy within an acceptable range.

D. Analysis of Sparsity Belief Over Time
To analyze the pruning behavior during training, we visualize the proposed metrics in Figs. 6 and 7, where we normalized by the upper bound of the entropy to facilitate straightforward comparison between layers. Recall that a low average pruning entropy $H(M_{\phi^i}/d)$ denotes a high confidence in pruning masks, whereas a high pruning diversity $I(M_{\phi^i}, d)$ indicates high mask diversity/specialization. Fig. 6 shows the aforementioned metrics for the first two layers of LeNet300-100. Both layers show similar behavior, a quick improvement both in confidence ($H(M_{\phi^i}/d)$) and diversity ($I(M_{\phi^i}, d)$) at the start of training, after which both metrics stabilize. The difference in behavior between Figs. 6 and 7 can be explained as follows: the smaller the network, the faster a solution is achieved (and the faster the stochastic masking can commit to a single pseudo-fixed mask). Therefore, the entropy metric $H$ is expected to evolve quicker in small networks (LeNet300-100) compared to larger networks (VGG-16). Moreover, we find interesting dynamics in Fig. 7(a) and (b) (VGG16–CIFAR10), where convolutional layers seem to more quickly learn a diverse set of sparse patterns ($I(M_{\phi^i}, d)$ grows faster) with high confidence ($H(M_{\phi^i}/d)$ drops quicker) than FC layers.

E. FLOPs Versus Compression Ratio
In order to determine the efficiency in terms of floating-point operations (FLOPs) reduction, a comparative analysis was performed between DPP-C-F and DPP-M-F trained on VGG-16. Fig. 8 shows that the coarse approach performs better to reduce the number of FLOPs as the compression ratio increases. Important to notice from Fig. 4, DPP-M-F shows a better performance in terms of test accuracy for the same compression ratios in comparison with DPP-C-F. Therefore, a tradeoff exits between test accuracy, compression ratio, and FLOP’s reduction; thus, it is necessary to choose the proper model that adapts to particular needs.

F. Analysis of Memory Usage
In this section, we analyze the additional memory to store the auxiliary indexes that indicate the position of the nonzero values. Table VI shows the required additional memory to store nonpruned values, where $S_p$ is the number of nonpruned
TABLE IV
EXPERIMENTAL RESULTS OF DPP FOR MNIST DATASET USING LeNet Architectures

| Network   | Model           | Non-pruned acc. (%) | Pruned acc. (%) | \(\Delta_{acc} \) | Remain. params. (%) | \(S_p\) (%) | Bits | \(cr\) | Time per epoch (sec.) |
|-----------|-----------------|---------------------|----------------|-----------------|---------------------|-------------|-------|--------|----------------------|
| LeNet300-100 | [60] DPP-F-F (This work) | 98.16               | 98.03           | -0.13           | 2.48                | 97.52       | 32    | 35.21x | -                    |
| DPP-F-F (This work) | 98.19               | 97.90           | -0.21           | 1.95             | 98.05       | 32    | 51.28x | 7                    |
| DPP-F-F (This work) | 98.19               | 97.82           | -0.37           | 6.71             | 93.29       | 8     | 59.61x | 27                   |
| DPP-F-F (This work) | 98.19               | 96.81           | -1.38           | 21               | 79          | 2     | 76.19x | 29                   |

| LeNet5-Caffe | [60] DPP-F-F (This work) | 99.18               | 99.11           | -0.07           | 1.64                | 98.36       | 32    | 20.32x | -                    |
| DPP-F-F (This work) | 99.23               | 99.23           | 0.0             | 2.5              | 97.5       | 32    | 40x    | 7                    |
| DPP-F-F (This work) | 99.23               | 99.00           | -0.23           | 4.1              | 95.9       | 8     | 97.65x | 50                   |
| DPP-F-F (This work) | 99.23               | 98.36           | -0.87           | 4.1              | 95.9       | 2     | 390.24x | 43                   |

TABLE V
EXPERIMENTAL RESULTS OF DPP FOR CIFAR-10 AND CIFAR-100 DATASETS USING VGG-16 AND MobileNet v1 Architectures

| Dataset | Network | Model           | Non-pruned acc. (%) | Pruned acc. (%) | \(\Delta_{acc} \) | Remain. params. (%) | \(S_p\) (%) | Bits | \(cr\) | Time per epoch (sec.) |
|---------|---------|-----------------|---------------------|----------------|-----------------|---------------------|-------------|-------|--------|----------------------|
| CIFAR -10 | VGG-16 | [41]            | 93.49               | 93.14           | -0.35           | 55.56                | 44.44       | 32    | 1.80x | -                    |
|         |         | [44] DPP-M-F (This work) | 92.8               | 89.5            | -3.3            | 11.11                | 88.89       | 32    | 9x    | 82                   |
|         |         | DPP-M-F (This work) | 93.50               | 93.60           | +0.10           | 10.73                | 89.27       | 32    | 9.32x | 82                   |
|         |         | [62] DPP-C-F (This work) | 93.25               | 93.18           | -0.07           | 26.66                | 73.34       | 32    | 3.75x | -                    |
|         |         | [39] DPP-C-F (This work) | 93.50               | 93.52           | +0.02           | 15.3                 | 84.7        | 8     | 26.14x | 130                  |
|         |         | DPP-C-F (This work) | 93.50               | 93.52           | +0.02           | 19.49                | 80.51       | 32    | 5.13x | 43                   |
| MobileNet v1 | [61] DPP-C-F (This work) | 92.3               | 91.77           | -0.53           | 25                | 75         | 32    | 4x    | -                    |
|         |         | DPP-C-F (This work) | 93.6               | 93.14           | -0.46           | 36.95                | 63.06       | 32    | 2.70x | 178                  |
| ResNet18 | [44]            | 92.9               | 92.5            | -0.4            | 55.35                | 44.65       | 32    | 1.80x | -                    |
|         |         | [44] DPP-M-F (This work) | 92.9               | 91.1            | -1.8            | 23.4                 | 76.6        | 32    | 4.27x | -                    |
|         |         | DPP-M-F (This work) | 93.78               | 93.56           | -0.22           | 37.3                 | 62.7        | 32    | 2.68x | 35                   |
|         |         | [62] DPP-C-F (This work) | 93.78               | 93.96           | +0.18           | 37.3                 | 62.7        | 32    | 2.68x | 36                   |
| CIFAR -100 | VGG-16 | [62] DPP-C-F (This work) | 73.26               | 73.33           | +0.07           | 68                | 32         | 32    | 1.47x | -                    |
|         |         | DPP-C-F (This work) | 70.52               | 70.40           | +0.08           | 18.8                | 81.2       | 32    | 5.31x | 102                  |
| MobileNet v1 | [61] DPP-C-F (This work) | 69.1               | 68.52           | -0.58           | 35                | 65         | 32    | 2.85x | -                    |
|         |         | DPP-C-F (This work) | 72.35               | 72.50           | -0.15           | 40                | 60         | 32    | 2.50x | 174                  |
| Tiny ImageNet | ResNet18 | [44] DPP-C-F (This work) | 62.4               | 61.7            | -0.54           | 55.65                | 44.35       | 32    | 1.79x | -                    |
|         |         | DPP-C-D (This work) | 61.96               | 61.62           | -0.32           | 54.46                | 45.54       | 32    | 1.83x | 38                   |
|         |         | DPP-M-D (This work) | 61.96               | 61.72           | -0.24           | 60.32                | 39.68       | 32    | 1.65x | 72                   |

TABLE VI
ADDITIONAL MEMORY USAGE FOR UNSTRUCTURED PRUNING [60] VERSUS STRUCTURED FINE-GRAINED (DPP-F) AND MEDIUM-GRAINED (DPP-M) PRUNING

| Model       | COO \(S_{b\text{col}} + S_{b\text{row}}\) | ELLPACK \(ZP + S_{b\text{col}}\) | CSR \(S_{b\text{col}} + n_{\text{rows}}(b_{\text{row}})\) |
|-------------|------------------------------------------|----------------------------------|--------------------------------------------------|
| DPP-F       | \(S_{b\text{col}} + S_{b\text{row}}\)  | \(S_{b\text{col}} + S_{b\text{kernel}}\) | \(S_{b\text{col}} + n_{\text{rows}}(b_{\text{row}})\) |
| DPP-M       | \(S_{b\text{col}} + S_{b\text{row}}\)  | \(S_{b\text{col}} + S_{b\text{kernel}}\) | \(S_{b\text{col}} + n_{\text{rows}}(b_{\text{row}})\) |

elements and \(b_{\text{col}}\) is the number of bit representation. For the unstructured case, COO, ELLPACK, and CSR formats are shown. Since the COO format requires the rows and column coordinates of the nonpruned values, then the additional memory is twice the number of nonpruned elements, where \(b_{\text{col}}\) and \(b_{\text{row}}\) are the maximum numbers of bits that represent the number of columns and rows, respectively. For the ELLPACK format, \(ZP\) refers to the amount of zero padding, which depends on the sparse structure (the less structured the nonpruned elements are, the less efficient the ELLPACK format is). Finally, the CSR format requires to store the column coordinates only, while an additional index \(n_{\text{rows}}\) stores the row offset.
Fig. 7. (a) $H(M_{\theta_d}|d)$ and (b) $I(M_{\theta_d}, d)$ for VGG16 (CIFAR10). The plots correspond to the 5th, 8th, and 11th convolutional layers, and the 1st FC layer.

Fig. 8. Percentage of remaining FLOPs versus compression ratio for VGG-16 trained on CIFAR10 for both DPP-M-F and DPP-C-F.

Fig. 9. Top: weight distribution per output neurons of the FC layer of VGG-16 trained on CIFAR-10 for DPP-F (red) and unstructured pruning [60] (blue). Bottom: visualization of weight distribution for the first feature map with 64 input channels for DPP-M (red) and unstructured pruning [60] (blue).

**TABLE VIII**

| Unstructured (Mb) | $N_{index}$ | DPP-M (Mb) | $N_{index}$ |
|-------------------|-------------|------------|-------------|
| COO               | 1.51        | 0.13       | 1           |
| ELLPACK           | 2.35        | 1          | 2           |
| CSR               | 1.32        | 2          |             |

In order to visualize an example of an unstructured sparse matrix, we show the last FC layer of VGG-16 trained on CIFAR-10 from [60]. Fig. 9 shows how the number of nonzero weights (y-axis) is distributed among output neurons (x-axis). Visually, this distribution (blue) is highly unstructured, and thus, formats such as ELLPACK are not suitable. For the same model with the same sparsity, DPP-F offers a compact and structured nonzero elements distribution (red). In addition, for the convolutional layers, we show the unstructured case from [60] versus DPP-M. As it is observed, Liu et al. [60] unstructurally pruned weights of kernels among channels, while DPP-M prunes complete kernels reducing the need of additional indexes. Tables VII and VIII show the actual additional memory usage for VGG-16 trained on CIFAR10 for the unstructured case [60] and DPP. Importantly, for the CSR format, the difference is not significant, and however, the CSR format requires two auxiliary vectors, which increases the complexity of the control flow to manage both the column index and the row offset. For the convolutional case, we chose one sample of the feature maps to compare the additional memory usage of FPP-M and the unstructured case. For both cases, DPP shows superior performance in terms of the amount of additional memory.

V. CONCLUSION

In this article, we propose dynamic probabilistic pruning, an algorithm that enables training sparse networks based on stochastic and dynamic masking. DPP is a general framework that enables structured pruning for FC and convolutional layers, suitable for hardware implementations. We demonstrate that DPP enables large hardware memory saving by leveraging structured pruning at different levels of granularities (fine, medium, and coarse). Leveraging its probabilistic nature, we showed how one can assess the confidence and diversity of pruning masks among neurons by monitoring the proposed information-theoretic metrics. These metrics must be further investigated since they could provide information regarding the quality of pruning of specialized structures (e.g., kernels or feature maps).

Since DPP does not rely on magnitudes for determining the relevance of weights, it can be straightforwardly...
integrated with weight quantization (including binarization). This allows for a larger model compression as observed in the results. We test its performance in four benchmark datasets and obtain competitive accuracies for different architectures. In conclusion, our method generates ultra-compressed models, enabling a more efficient implementation on existing hardware platforms. Furthermore, the potential of DPP should be explored to generate even more efficient sparsity patterns for hardware such as tiling at different levels of granularities.

REFERENCES

[1] M. Al-Sharman et al., “A sensorless state estimation for a safety-oriented cyber-physical system in urban driving: Deep learning approach,” IEEE/CAC 8 J. Autom. Sinica, vol. 8, no. 1, pp. 169–178, Jan. 2021.

[2] S. Liu, Y. Xia, Z. Shi, H. Yu, Z. Li, and J. Lin, “Deep learning in shielding metal benders with a novel theory-guided deep neural network,” IEEE/CAC J. Autom. Sinica, vol. 8, no. 3, pp. 565–581, Mar. 2021.

[3] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, “A survey of model compression and acceleration for deep neural networks,” Sep. 2019, arXiv:1710.09282.

[4] V. Sze, Y.-H. Chen, J. Emer, A. Suleiman, and Z. Zhang, “Hardware for machine learning: Challenges and opportunities,” in Proc. IEEE Custom Integr. Circuits Conf. (CICC), Apr. 2018, pp. 1–8.

[5] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Quantized neural networks: Training neural networks with low precision weights and activations,” J. Mach. Learn. Res., vol. 18, pp. 6869–6898, Sep. 2016.

[6] M. Courbariaux and Y. Bengio, “BinaryNet: Training deep neural networks with weights and activations constrained to +1 or -1,” 2016, arXiv:1602.02830.

[7] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” in Proc. NIPS Deep Learn. Represent. Learn. Workshop, 2014.

[8] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding,” 2015, arXiv:1510.00149.

[9] Y. LeCun, J. S. Denker, and S. A. Solla, “Optimal brain damage,” in Advances in Neural Information Processing Systems, D. S. Touretzky, Ed. San Mateo, CA, USA: Morgan Kaufmann, 1990, pp. 598–605.

[10] S. J. Hanson and L. Y. Pratt, “181.01907/ing biases for minimal network construction with back-propagation,” in Advances in Neural Information Processing Systems, D. S. Touretzky, Ed. San Mateo, CA, USA: Morgan Kaufmann, 1989, pp. 177–185.

[11] G. Wang, J. Qiao, J. Bi, Q. Jia, and M. Zhou, “An adaptive deep belief network with sparse restricted Boltzmann machines,” IEEE Trans. Neural Netw. Learn. Syst., vol. 31, no. 10, pp. 4217–4228, Oct. 2020.

[12] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, “Rethinking the value of network pruning,” in Proc. Int. Conf. Learn. Represent., 2019.

[13] R. Dorrance, F. Ren, and D. Marković, “A scalable sparse matrix-vector multiplication kernel for efficient-energry-sparse-blaz on FPGAs,” in Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA). New York, NY, USA: Association for Computing Machinery, Feb. 2014, pp. 161–170.

[14] J. Fowers, K. Otvichar, K. Strauss, E. S. Chung, and G. Sitt, “A high memory bandwidth FPGA accelerator for sparse matrix-vector multiplication,” in Proc. IEEE 22nd Annu. Int. Symp. Field-Program. Custom Comput. Mach., May 2014, pp. 36–43.

[15] D. Fujiki, N. Chatterjee, D. Lee, and M. O’Connor, “Near-memory data transformation for efficient sparse matrix multi-vector multiplication,” in Proc. SC. New York, NY, USA: Association for Computing Machinery, 2017, pp. 1–10.

[16] M. Kreutzer, G. Hager, G. Wellein, H. Fehske, A. Basermann, and A. R. Bishop, “Sparse matrix-vector multiplication on GPGPU clusters: A new storage format and a scalable implementation,” in Proc. IEEE 26th Int. Parallel Distrib. Process. Symp. Workshops PhD Forum, May 2012, pp. 1696–1702.

[17] L. Lu, J. Xie, R. Huang, J. Zhang, W. Lin, and Y. Liang, “An efficient hardware accelerator for sparse convolutional neural networks on FPGAs,” in Proc. IEEE 27th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM), Apr. 2019, pp. 17–25.
