FLightNNs: Lightweight Quantized Deep Neural Networks for Fast and Accurate Inference

Ruizhou Ding, Zeye Liu, Ting-Wu Chin, Diana Marculescu, and R. D. (Shawn) Blanton
{rding,zeyel,tingwuc,dianam,rblanton}@andrew.cmu.edu
Carnegie Mellon University, Pittsburgh, U.S.A.

ABSTRACT

To improve the throughput and energy efficiency of Deep Neural Networks (DNNs) on customized hardware, lightweight neural networks constrain the weights of DNNs to be a limited combination (denoted as $k \in \{1, 2\}$) of powers of 2. In such networks, the multiply-accumulate operation can be replaced with a single shift operation, or two shifts and an add operation. To provide even more design flexibility, the k for each convolutional filter can be optimally chosen instead of being fixed for every filter. In this paper, we formulate the selection of k to be differentiable, and describe model training for determining k-based weights on a per-filter basis. Over 46 FPGA-design experiments involving eight configurations and four data sets reveal that lightweight neural networks with a flexible k value (dubbed FLightNNs) fully utilize the hardware resources on Field Programmable Gate Arrays (FPGAs), our experimental results show that FLightNNs can achieve 2× speedup when compared to lightweight NNs with $k = 2$, with only 0.1% accuracy degradation. Compared to a 4-bit fixed-point quantization, FLightNNs achieve higher accuracy and up to 2× inference speedup, due to their lightweight shift operations. In addition, our experiments also demonstrate that FLightNNs can achieve higher computational energy efficiency for ASIC implementation.

1 INTRODUCTION

Emerging vision, speech and natural language applications have widely adopted deep learning models and, as a result, have achieved state-of-the-art accuracy. Furthermore, recent industrial efforts have focused on implementing the models on mobile devices [1]. However, real-time applications based on these deep models may incur unacceptably large latencies and can easily drain the battery on energy-limited devices. For example, smartphones can only run the AlexNet-based object detection for one hour [28]. Therefore, prior research has proposed model compression techniques including pruning and quantization to satisfy the stringent energy and speed requirements [16].

One of the recently proposed quantization approaches, LightNN, constrains the weights of DNNs to be a sum of k powers of 2, and therefore can use shift and add operations to replace the multiplications between activations and weights [9]. For LightNN-1[1], all the multiplications of the DNNs will be replaced by a shift operation, while for LightNN-2, two shifts and an add replace the multiplication. Since shift operations are much more lightweight on customized hardware (*e.g.*, FPGA or ASIC), LightNNs can achieve faster speed and lower energy consumption, and generally maintain accuracy for over-parameterized models [8, 9]. Although LightNNs provide better energy-efficiency, they lack the flexibility to provide fine-grained trade-offs between energy and accuracy. As shown in Fig. 1, the energy efficiency for these models also exhibits gaps, making the Pareto front of accuracy and energy discrete. However, a continuous accuracy and energy/latency trade-off is an important feature for designers to target different market segments (*e.g.*, IoT devices, edge devices, and mobile devices).

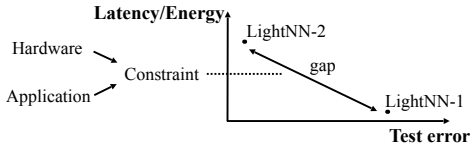

Figure 1: A discrete Pareto-optimal curve for LightNN models $w.r.t.$ test error and latency/energy. More continuous Pareto-optimal points are needed to adapt to the latency/energy constraints determined by the hardware and application.

To provide a more flexible Pareto front for the LightNN framework, we propose to equip each convolutional filter with the freedom to use a different number of shift-and-add operations to approximate multiplications. Specifically, we introduce a set of free variables $\mathbf{k} = \{\mathbf{k}_1, \ldots, \mathbf{k}_F\}$ where each element represents the number of shift-and-add for the corresponding convolutional filter. As a result, a more contiguous Pareto front can be achieved. For example, if we constrain $\mathbf{k} \in \{1, 2\}^F$, then the throughput and energy consumption of the new model will sit between LightNN-1 ($\mathbf{k} = \{1\}^F$) and LightNN-2 ($\mathbf{k} = \{2\}^F$). Formally, we are solving $\min_{\mathbf{w},\mathbf{k}} \mathcal{L}(\mathbf{w}, \mathbf{k})$, where $\mathcal{L}$ is the loss function and $\mathbf{w}$ is the weights vector. However, the commonly adopted stochastic gradient descent (SGD) algorithm does not apply in this case since $\mathcal{L}$ is non-differentiable $w.r.t.$ $\mathbf{k}$. In this paper, we propose a *differentiable training algorithm* which enables end-to-end optimization with standard SGD. The resulting network is dubbed ***F**LightNN* for its flexible $\mathbf{k}$ values.

2 RELATED WORK

Prior work has extensively explored approaches to reduce latency and energy consumption of DNNs on hardware, through both algorithmic [14, 28] and hardware [4, 30] efforts. Since the latency and energy consumption of DNNs generally stem from computational cost and memory accesses, prior work in the algorithmic domain mainly focuses on the reduction of FLOPs and model size. Some work reduces the number of parameters through weight pruning [13], while some other work introduces structural sparsity via filter pruning for Convolutional Neural Networks (CNNs) [26] to enable speedup on general hardware platforms incorporating CPUs and GPUs. To reduce the model size, previous work has also conducted neural architecture search with energy constraint [21, 23, 24, 28]. In addition to algorithmic advances, prior art has also proposed methodologies to achieve fast and energy-efficient DNNs. Some previous work proposes the co-design of the hardware platform

[1] LightNN-k quantizes weights to be the sum of k powers of 2.

and the architecture of the neural network running on it [3]. Some work proposes more lightweight DNN units for faster inference on general-purpose hardware [22], while others propose hardware-friendly DNN computation units to enable energy-efficient implementation on customized hardware [25].

By reducing the weight and activation precision, DNN quantization has proved to be an effective technique to improve the speed and energy efficiency of DNNs on customized hardware, due to its lower computational cost and fewer memory accesses [11]. Gupta *et al.* show that a DNN with 16-bit fixed-point representation can achieve competitive accuracy compared to the full-precision network [11]. In the same vein, Zhou *et al.* explored the DNN accuracy *w.r.t.* a wide range of bit widths [31]. These uniform quantization approaches enable fixed-point hardware implementation for DNNs. Courbariaux *et al.* propose BinaryConnect, which uses only 1 bit for the DNN parameters, turning multiplications into XNOR operations on customized hardware [6]. However, these models require an over-parameterized model size to maintain a high accuracy [9].

LightNNs constrain the model weights to be a power of 2, or the sum of a limited number of powers of 2 [9], while the activations use fixed-point quantization. Therefore, the multiplication between weights and activations can be implemented in hardware by shift operations and fixed-point additions. Compared to DNNs with fixed-point quantization, LightNNs replace the fixed-point multipliers by more lightweight shift operators, or shift and additions. Since the shift operators can be implemented using Look-Up Table (LUT) on FPGA while fixed-point multipliers require Digital Signal Processing (DSP) units, LightNNs can have higher inference speed than fixed-point DNNs when run on DSP-bounded FPGAs. In addition, in an ASIC implementation, shift operations are more lightweight than multiplications, making LightNNs more energy and area efficient than fixed-point DNNs.

However, LightNNs use a single k value (*i.e.*, the number of shifts per multiplication) across the whole network, and therefore lack flexibility to provide a fine-grained energy/latency and accuracy trade-off for hardware designers. Therefore, we propose FLightNNs which use customized k values for each convolutional filter to enable a more continuous Pareto front. Recent work has explored the idea of differentiable training for architecture search [17] and neural network pruning [18]. In this paper, we propose an end-to-end differentiable training algorithm for FLightNNs via approximate gradient computation for non-differentiable operations and regularization to encourage sparsity. Moreover, the proposed differentiable training approach uses gradual quantization, which can achieve higher accuracy than LightNN-1 without increasing latency. In summary, this paper has the following key contributions:

(i) We propose a differentiable training algorithm for FLightNNs, which provides a continuous Pareto front for hardware designers to search for a highly accurate model under the hardware resource constraints.

(ii) The differentiable training for FLightNNs enables gradual quantization, and further pushes forward the Pareto-optimal curve.

3 LIGHTNN OVERVIEW

As a quantized DNN model, LightNNs constrain the weights of a network to be the sum of k powers of 2, denoted as LightNN-k.

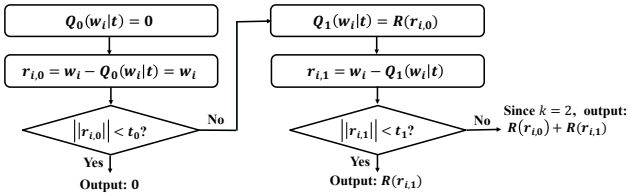

Figure 2: Quantization flow for $k = 2$.

Thus, the multiplications between weights and activations can be implemented with k shift operations and $k-1$ additions. Specifically, LightNN-1 constrains the weights to be a power of 2, and only uses a shift for a multiplication. The approximation function used by LightNN-k to quantize a full-precision weight w can be formulated in a recursive way: $Q_k(w) = Q_{k-1}(w) + Q_1(w - Q_{k-1}(w))$ for $k > 1$, where $Q_1(w) = sign(w) \times 2^{[log(|w|)]}$ which rounds the weight w to a nearest power of 2.

LightNNs are trained with a modified backpropagation algorithm. In the forward phase of each training iteration, the parameters are first approximated using the Q_k function. Then, in the backward phase, the gradients of loss *w.r.t.* quantized weights are computed, and applied to the full-precision weights in the weight update phase. LightNNs have been proved to be accurate and energy-efficient on customized hardware [9]. LightNN-2 can generally have an accuracy close to full-precision DNNs, while LightNN-1 can achieve higher energy efficiency than LightNN-2. Due to the nature of the discrete k values, there exists a gap between LightNN-1 and LightNN-2 *w.r.t.* accuracy and energy. We propose to customize the k values for each convolutional filter, and thus, achieve a smoother energy-accuracy trade-off to provide hardware designers with more design options.

4 DIFFERENTIABLE TRAINING FOR FLIGHTNNS

In this section, we first define the quantization function, and then introduce the end-to-end training algorithm for FLightNNs, equipped with a regularization loss to penalize large $\mathbf{k}$ values.

4.1 Quantization function

We first denote the i^{th} filter of the network as $\mathbf{w}_i$ and the quantization function for the filter $\mathbf{w}_i$ as $Q_k(\mathbf{w}_i|\mathbf{t})$, where $k = \max_i \mathbf{k}$ is the maximum number of shifts used for this network, and vector $\mathbf{t}$ is a latent variable that controls the approximation (*e.g.*, some threshold value). Also, we denote the residual resulting from the approximation as $\mathbf{r}_{i,k} = \mathbf{w}_i - Q_k(\mathbf{w}_i|\mathbf{t})$. Then, we formally define the quantization function as follows:

$$Q_k(\mathbf{w}_i|\mathbf{t}) = \begin{cases} 0, \text{ if } k = 0 \\ \sum_{j=0}^{k-1} \mathbb{1}(||\mathbf{r}_{i,j}||_2 > \mathbf{t}_j) R(\mathbf{r}_{i,j}), \text{ if } k \geq 1 \end{cases}$$

where $R(x) = sign(x) \times 2^{[log(|x|)]}$ rounds the input variable to a nearest power of 2, and $[.]$ is a rounding-to-integer function. This quantization flow is shown in Fig. 2. To interpret the thresholds $\mathbf{t}$, $\mathbf{t}_0$ determines whether this filter is pruned out, and $\mathbf{t}_1$ determines whether one shift is enough, *etc.* Then, the number of shifts for the i-th filter is $\mathbf{k}_i = \sum_{j=0}^{k-1} \mathbb{1}(||\mathbf{r}_{i,j}||_2 > \mathbf{t}_j)$. Therefore, choosing $\mathbf{k}_i$ per filter is equivalent to finding optimal thresholds $\mathbf{t}$.

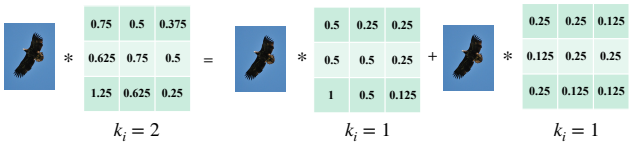

Figure 3: Equivalent conversion from a convolution with a $\mathbf{k}_i > 1$ filter to $\mathbf{k}_i$ convolutions each with a $\mathbf{k}_i = 1$ filter. This transforms the hardware implementation of the FLightNN into LightNN-1.

The FLightNN quantization approach targets efficient hardware implementation. Instead of assigning a customized $\mathbf{k}_i$ for each weight, FLightNNs have customized $\mathbf{k}_i$ values per filter, and therefore preserve the structural sparsity. As shown in Fig. 3, the convolution with a $\mathbf{k}_i = 2$ filter can be equivalently converted to the sum of two convolutions each with a $\mathbf{k}_i = 1$ filter. Thus, FLightNNs can be efficiently implemented as LightNN-1 with an extra summation of feature maps per layer.

4.2 Differentiable training

Instead of picking the thresholds t by hand, we consider them as trainable parameters. Therefore, the loss function $\mathcal{L}(\mathbf{w}^2, \mathbf{t})$ is a function of both weights and thresholds. Similar to prior work on DNN quantization [6, 31], we use the straight-through estimator (STE) [2] to compute $\frac{\partial \mathcal{L}}{\partial \mathbf{w}_i}$. By defining $\frac{\partial \mathbf{w}_i^q}{\partial \mathbf{w}_i} = 1$ where $\mathbf{w}_i^q = Q_k(\mathbf{w}_i|\mathbf{t})$ is the quantized $\mathbf{w}_i$; therefore, we have $\frac{\partial \mathcal{L}}{\partial \mathbf{w}_i} = \frac{\partial \mathcal{L}}{\partial \mathbf{w}_i^q} \cdot \frac{\partial \mathbf{w}_i^q}{\partial \mathbf{w}_i} = \frac{\partial \mathcal{L}}{\partial \mathbf{w}_i^q}$, which becomes a differentiable expression.

To compute the gradient for thresholds, *i.e.*, $\frac{\partial \mathbf{w}_i^q}{\partial \mathbf{t}_j}$, we relax the indicator function $g(x, \mathbf{t}_j) = \mathbb{1}(x > \mathbf{t}_j)$ to a sigmoid function [12], $\sigma(.)$, when computing gradients, *i.e.*, $\hat{g}(x, \mathbf{t}_j) = \sigma(x - \mathbf{t}_j)$. In addition, we use STE to compute the gradient for $R(x)$. Thus, the gradient $\frac{\partial \mathbf{w}_i^q}{\partial \mathbf{t}_j}$ can be computed by:

$$\frac{\partial Q_{\mathbf{k}_i}(\mathbf{w}_i|\mathbf{t})}{\partial \mathbf{t}_j} = \sum_{l=0}^{\mathbf{k}_i - 1} \frac{\partial \sigma(||\mathbf{r}_{i,l}||_2 - \mathbf{t}_l)}{\partial \mathbf{t}_j} R(\mathbf{r}_{i,l}) + \sigma(||\mathbf{r}_{i,l}||_2 - \mathbf{t}_l) \frac{\partial R(\mathbf{r}_{i,l})}{\partial \mathbf{t}_j}$$

$$= \sum_{l=0}^{\mathbf{k}_i - 1} \sigma'(||\mathbf{r}_{i,l}||_2 - \mathbf{t}_l)\left(\frac{\partial ||\mathbf{r}_{i,l}||_2}{\partial \mathbf{t}_j} - \frac{\partial \mathbf{t}_l}{\partial \mathbf{t}_j}\right) R(\mathbf{r}_{i,l}) + \sigma(||\mathbf{r}_{i,l}||_2 - \mathbf{t}_l) \frac{\partial \mathbf{r}_{i,l}}{\partial \mathbf{t}_j}$$

where $\frac{\partial ||\mathbf{r}_{i,l}||_2}{\partial \mathbf{t}_j}$ and $\frac{\partial \mathbf{r}_{i,l}}{\partial \mathbf{t}_j}$ are 0 for $l < j$; otherwise, they can be computed with the result of $\frac{\partial Q_l(\mathbf{w}_i|\mathbf{t})}{\partial \mathbf{t}_j}$. $\frac{\partial \mathbf{t}_l}{\partial \mathbf{t}_j} = \mathbb{1}(l = j)$.

4.3 Regularization

To encourage smaller $\mathbf{k}_i$ for the filters, we also add a regularization loss: $\mathcal{L}_{reg,k}(\mathbf{w}) = \sum_{j=0}^{k-1} \lambda_j \sum_i ||\mathbf{r}_{i,j}||_2$ where λ_j performs as a handle to balance accuracy and model sparsity. This regularization loss is the sum of several group Lasso losses, since they can introduce structural sparsity [26]. The first item $\lambda_0 \sum_i ||\mathbf{r}_{i,0}||_2 = \lambda_0 \sum_i ||\mathbf{w}_i||_2$ is used to prune the whole filters out, while the other items ($j > 0$) regularize the residuals. Fig. 4 shows the two items of regularization loss and their sum for the case $k = 2$, with λ_0=1e-5 and λ_1=3e-5.

[2]The bias term is omitted for simplicity.

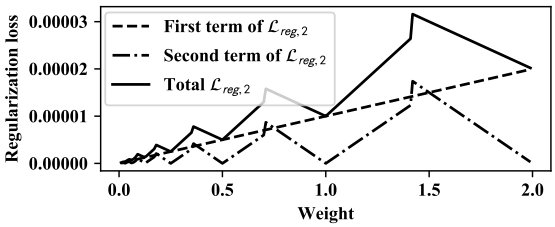

Figure 4: Regularization loss curve $w.r.t.$ weight value.

Algorithm 1: FLightNN Training Epoch

Input: Training dataset ($\mathbf{x}$, $\mathbf{y}$), where $\mathbf{x}$ is input and $\mathbf{y}$ is label; parameters after the $(p-1)$-th iteration: $\mathbf{w}_{p-1}$ (weights), $\mathbf{b}_{p-1}$ (biases), and quantization thresholds $\mathbf{t}_{p-1}$; quantization function $Q_k(\mathbf{w}|\mathbf{t})$; DNN forward computation function $g(x, w, b)$; maximum k value used for all filters; regularization loss coefficients λ; learning rate η.

Output: Updated weights $\mathbf{w}_p$, biases $\mathbf{b}_p$ and thresholds $\mathbf{b}_p$.

for *each mini-batch of* $\boldsymbol{x}$, $\boldsymbol{y}$ **do**

1. **Quantize weights**: $\mathbf{w}^q = Q_k(\mathbf{w}_{p-1}|\mathbf{t}_{p-1})$
2. **Forward**: compute intermediate results and cross entropy loss function $\mathcal{L}_{CE}$ with $g(\cdot)$, $\mathbf{w}^q$, $\mathbf{b}_{p-1}$, and mini-batch of $\mathbf{x}$; compute regularization loss $\mathcal{L}_{reg,k}$ with λ and $\mathbf{w}_{p-1}$; get the total loss $\mathcal{L}_{total} = \mathcal{L}_{CE} + \mathcal{L}_{reg,k}$
3. **Backward**: compute derivatives $\frac{\partial \mathcal{L}_{total}}{\partial \mathbf{w}^q}$, $\frac{\partial \mathcal{L}_{total}}{\partial \mathbf{b}_{p-1}}$, and $\frac{\partial \mathcal{L}_{total}}{\partial \mathbf{t}_{p-1}}$
4. **Update parameters**: $\mathbf{w}_p = \mathbf{w}_{p-1} - \eta \frac{\partial \mathcal{L}_{total}}{\partial \mathbf{w}^q}$; $\mathbf{b}_p = \mathbf{b}_{p-1} - \eta \frac{\partial \mathcal{L}_{total}}{\partial \mathbf{b}_{p-1}}$; $\mathbf{t}_p = \mathbf{t}_{p-1} - \eta \frac{\partial \mathcal{L}_{total}}{\partial \mathbf{t}_{p-1}}$

end

Therefore, the total loss for training a FLightNN is: $\mathcal{L}_{total}(\mathbf{w}, \mathbf{t}) = \mathcal{L}_{CE}(\mathbf{w}, \mathbf{t}) + \mathcal{L}_{reg,k}(\mathbf{w})$.

The new training algorithm is summarized in Algo. 1. This is the same as the conventional backpropagation algorithm for full-precision DNNs, except that in the forward phase, the weights are quantized given the thresholds $\mathbf{t}$. Then, due to the differentiability of the quantization function $w.r.t.$ $\mathbf{w}$ and $\mathbf{t}$, one can compute their gradients and update their values in each training iteration.

5 EXPERIMENTAL RESULTS

In this section, we first introduce the experiment setup. Then, we show the accuracy results of different quantized DNN models by software training, as well as their throughput on the FPGA and energy efficiency on the ASIC, to verify the effectiveness of FLightNNs.

5.1 Setup

We conduct experiments on both small and large CNNs for CIFAR-10, SVHN, CIFAR-100 and ImageNet datasets. The eight adopted network configurations are shown in Table 1. To explore the FLightNN performance on different types of network structures, we use a VGG structure with a series of stacked convolutional layers for Network 1, 3, 4 and 5, and adopt the ResNet structure with skip connections across layers for network 2, 6, 7 and 8. Networks 1,

Table 1: Network settings. “Depth" is the number of convolutional layers in the network. “Width" is the number of convolutional filters of the largest layer.

Network ID	Parameters	Structure	Depth	Width
1	0.08M	VGG	7	64
2	0.7M	ResNet	18	128
3	4.6M	VGG	7	512
4	0.03M	VGG	4	64
5	0.1M	VGG	4	128
6	0.7M	ResNet	18	128
7	2.8M	ResNet	18	256
8	1.8M	ResNet	10	256

2 and 3 are used for experiments on CIFAR-10; networks 4 and 5 are used for SVHN; networks 6 and 7 are used for CIFAR-100; the last one, network 8, is used for ImageNet. For all networks, each convolutional layer is followed by a batch normalization layer and a Leaky ReLU activation function [19], and optionally followed by a max-pooling layer. We use the Adam optimizer [15] to train the network. For each of the networks, we train different quantized models including full-precision DNNs, fixed-point DNNs with 4-bit weights and 8-bit activations, LightNN-2 with 8-bit weights and 8-bit activations, LightNN-1 with 4-bit weights and 8-bit activations, and FLightNNs with 8-bit activations. Due to large training times and limitations in computing resources, we train the ImageNet dataset on a ResNet-10 with reduced width (*i.e.*, network 8), for LightNN-1, LightNN-2 and FLightNNs. For all FLightNNs, we initialize the threshods $\mathbf{t}$ to $\mathbf{0}$, and set the largest shifts k as 2. For all, except the 32-bit full-precision model, we use 8-bit fixed-point quantization for the activations. By varying λ, we can have different accuracy-throughput or accuracy-energy trade-offs for FLightNNs. All these networks are trained in software through PyTorch.

5.2 Accuracy-throughput trade-off on FPGA

To show the accuracy-throughput trade-off of the models, we implement the inference of each network's largest convolutional layer for each of the quantized DNN models on FPGA since prior work has shown that convolution operations typically take over 90% of the computation time of a CNN [29]. Our implementation is built on the Xilinx Zynq ZC706 evaluation board. Its working frequency is 100 MHz. Pre-synthesis is executed on an Intel i7-4790 CPU (3.6GHz) with 16GB RAM. We use Vivado HLS [27] for FPGA implementation. The C code of DNN designs are parallelized by adding HLS-defined pragma and the parallel version is validated with the Vivado HLS timing analysis tool. To make a fair comparison, the same pragma and directives are used for full-precision, fixed-point DNNs, LightNNs and FLightNNs, and we follow the same scheduling settings as prior work [8]. Batched inference is adopted, and the maximum batch size without running out of FPGA resources is set to obtain the highest throughtput.

Tables 2, 3, 4 and 5 show the accuracy and throughput comparison for full-precision DNNs, fixedpoint DNNs, LightNNs and FLightNNs. For all the experimented datasets, LightNNs show the advantage of flexible accuracy-speed trade-offs. In most of the networks (*e.g.*, networks 1, 3, 6 and 7), FLightNNs can achieve an accuracy close to LightNN-2, but have much higher speedup than LightNN-2. Thus, FLightNNs provide continuous trade-offs for accuracy and speed. Compared to the fixed-point quantization, FLightNNs can achieve higher accuracy, and up to 2.0×, 1.8× and 1.8× speedup for CIFAR-10, SVHN, CIFAR-100 datasets, respectively. This is because the multiplication is replaced by shift operators, which require only LUT resources on FPGA while the multipliers require DSP units which are generally more scarce than LUT. Therefore, the computation for FLightNNs allows larger batch sizes than that of fixed-point DNNs, increasing data parallelism, and thus, improving the throughput.

It is also interesting to note that by comparing some FLightNNs (*e.g.*, FL_{1a}, FL_{2a}, FL_{3a}, FL_{6a} and FL_{7a}) with LightNN-1, we find that FLightNNs can achieve higher accuracy with the same or even lower storage as LightNN-1. This is because initially FLightNNs quantize all the filters with two shifts (since $\mathbf{t}$ is initialized as $\mathbf{0}$), and gradually add constraints to the filters. This gradual quantization may be better than training a network with only one shift from scratch, as LightNN-1 does. The benefit of gradual quantization has also been observed by prior work [10] which shows that gradually imposing quantization constraints can achieve better accuracy than directly quantizing with a strict constraint.

Table 6 shows the FPGA resources utilization for networks 7 and 8. Since full-precision and fixed-point DNNs require DSP for both multiplication and addition, while LightNNs and FLightNNs only need DSP for addition, full-precision and fixed-point DNNs have larger DSP resource utilization. Compared to full-precision DNNs which use 32-bit floating point operations, fixed-point DNNs only use 4-bit weights and 8-bit activations, and therefore consume fewer DSP units. LightNNs and FLightNNs use LUT to implement the multipliers, and have a higher utilization of LUT than full-precision and fixed-point DNNs. However, the performance of (F)LightNNs is not bounded by LUT resources since the maximum usage of LUT by LightNN-2 is only 42% and 17% for networks 7 and 8, respectively. Instead, the memory resource (BRAM) bounds the performance for (F)LightNNs, while for full-precision and fixed-point DNNs, the performance is bounded by both BRAM and DSP.

5.3 Accuracy-energy trade-off on ASIC

For all quantized DNNs, we designed pipelined implementations with one stage per neuron, where the computation unit is reused for each neuron. A 65nm commercial standard library is adopted. The Synopsys Design Compiler [20] is used to generate the gate-level netlist of the computation units. The power consumption of all computation operations within one layer is calculated using Synopsys Primetime. We keep all the DNN architectures implemented in an unoptimized fashion because our main objective is to compare how different quantized DNNs impact computational energy.

The accuracy and computational energy trade-offs for the quantized DNN models are shown in Fig. 5. The energy shown in Fig. 5 only includes the computational energy consumption for the largest layer of each network. We can clearly observe that FLightNNs provide a more continuous Pareto front for LightNN-2 and LightNN-1, regardless of the network type (*i.e.*, VGG or ResNet), size and the datasets. Similar to the observation in Sec. 5.2, in some networks FLightNNs can achieve higher accuracy than LightNN-1 with lower computational energy cost.

Table 2: Accuracy and FPGA throughput for CIFAR-10. In the "Model" column, "Full", "L-2", "L-1", "FP", "FL" indicate full-precision DNN, LightNN-2, LightNN-1, Fixed-point DNN, and FLightNN, respectively. The subscript "xWyA" indicates x bits for weights and y bits for activations. The FLightNN results are shown in bold face. We use subscript a and b to denote the two trained FLightNNs for each network. These notations also apply for Table 3, 4 and 5.

ID	Model	Accuracy (%)	Storage (MB)	Throughput (images/s)	Speedup
1	Full	86.36	0.31	3.2e2	1×
	L-2_{8W8A}	86.17	0.08	2.2e3	7.0×
	L-1_{4W8A}	84.82	0.04	4.5e3	14.4×
	FP_{4W8A}	85.09	0.04	3.3e3	10.5×
	FL_{1a}	**85.70**	**0.04**	**4.8e3**	**15.0×**
	FL_{1b}	**85.91**	**0.06**	**4.0e3**	**12.6×**
2	Full	91.70	2.8	1.4e2	1×
	L-2_{8W8A}	91.64	0.7	1.6e3	11.5×
	L-1_{4W8A}	91.15	0.4	2.7e3	19.0×
	FP_{4W8A}	91.17	0.4	1.5e3	10.7×
	FL_{2a}	**91.36**	**0.4**	**2.8e3**	**18.9×**
	FL_{2b}	**91.48**	**0.7**	**1.9e3**	**13.0×**
3	Full	92.85	18.5	1.3	1×
	L-2_{8W8A}	92.72	4.6	10.2	7.8×
	L-1_{4W8A}	91.93	2.3	39.2	30.2×
	FP_{4W8A}	92.23	2.3	19.8	15.2×
	FL_{3a}	**92.59**	**2.3**	**39.2**	**30.2×**
	FL_{3b}	**92.62**	**3.3**	**27.2**	**21.0×**

Table 3: Accuracy and FPGA throughput for SVHN.

ID	Model	Accuracy (%)	Storage (MB)	Throughput (images/s)	Speedup
4	Full	94.96	0.12	2.2e3	1×
	L-2_{8W8A}	94.90	0.03	4.5e3	2.09×
	L-1_{4W8A}	94.16	0.02	8.3e3	3.63×
	FP_{4W8A}	93.70	0.02	3.7e3	1.70×
	FL_{4a}	**94.67**	**0.02**	**6.7e3**	**3.11×**
	FL_{4b}	**94.88**	**0.03**	**5.3e3**	**2.37×**
5	Full	96.44	0.4	1.1e3	1×
	L-2_{8W8A}	96.38	0.1	2.1e3	2.00×
	L-1_{4W8A}	95.93	0.05	3.7e3	3.53×
	FP_{4W8A}	96.02	0.05	1.8e3	1.71×
	FL_{5a}	**96.21**	**0.06**	**3.2e3**	**3.06×**
	FL_{5b}	**96.24**	**0.08**	**3.0e3**	**2.84×**

6 DISCUSSION

Since FLightNNs customize the $\mathbf{k}_i$ for each filter, LightNN-1 and LightNN-2 can be considered as two special cases for FLightNNs. Therefore, the Pareto front created by the searched FLightNN solutions should be the upper bound for the front of LightNN-1 and LightNN-2 with varied parameter numbers. We test this hypothesis on CIFAR-100 dataset using networks with varied number of convolutional filters. As shown in Fig. 6, the accuracy-storage

Table 4: Accuracy and FPGA throughput for CIFAR-100.

ID	Model	Accuracy (%)	Storage (MB)	Throughput (images/s)	Speedup
6	Full	69.16	2.8	2.5e2	1×
	L-2_{8W8A}	68.84	0.7	1.6e3	6.4×
	L-1_{4W8A}	67.32	0.4	2.7e3	10.6×
	FP_{4W8A}	67.67	0.4	1.5e3	5.98×
	FL_{6a}	**68.59**	**0.4**	**2.7e3**	**10.6×**
	FL_{6b}	**68.76**	**0.6**	**1.8e3**	**6.88×**
7	Full	71.22	11.2	7.4e1	1×
	L-2_{8W8A}	70.96	2.8	6.0e2	8.11×
	L-1_{4W8A}	69.71	1.4	1.1e3	15.2×
	FP_{4W8A}	69.34	1.4	6.9e2	9.26×
	FL_{7a}	**70.85**	**1.4**	**1.1e3**	**15.2×**
	FL_{7b}	**70.87**	**2.4**	**7.4e2**	**9.98×**

Table 5: Top-5 Accuracy and FPGA throughput for ImageNet.

ID	Model	Accuracy (%)	Storage (MB)	Throughput (images/s)	Speedup
8	L-2_{8W8A}	75.04	1.8	2.7e2	1×
	L-1_{4W8A}	72.94	0.9	5.2e2	1.95×
	FL_{8a}	**74.80**	**1.5**	**3.1e2**	**1.16×**
	FL_{8b}	**75.00**	**1.7**	**2.8e2**	**1.06×**

Table 6: FPGA resource utilization for different quantized DNN models.

ID	Model	Maximum resource utilization				Speedup
		BRAM	DSP	FF	LUT	
7	Full	896	642	69,344	128,339	1×
	L-2_{8W8A}	1,024	4	66,491	90,949	8.11×
	L-1_{4W8A}	1,024	4	66,491	90,949	15.2×
	FP_{4W8A}	1,024	514	7,110	90,949	9.26×
	FL_{7a}	1,024	4	84,192	90,949	9.98×
	FL_{7b}	1,024	4	63,648	80,940	15.2×
8	L-2_{8W8A}	832	16	3,156	38,022	1×
	L-1_{8W8A}	800	16	3,084	36,906	1.95×
	FL_{8a}	800	16	5,070	36,098	1.16×
	FL_{8b}	800	16	6,272	37,406	1.06×
Available		1,090	900	437,200	218,600	

Pareto-front created by FLightNNs is consistently higher than the LightNNs. This indicates that instead of only filling in the Pareto front of LightNNs, FLightNNs can *push forward* the Pareto front, due to their larger design space. The proposed differentiable training algorithm optimizes both $\mathbf{k}_i$ and weight values in an end-to-end fashion, and therefore significantly reduces searching effort compared to exhaustive or heuristic methods with multiple rounds of training. Future work will further improve training efficiency by using optimized training loss [7] or proper labels [5].

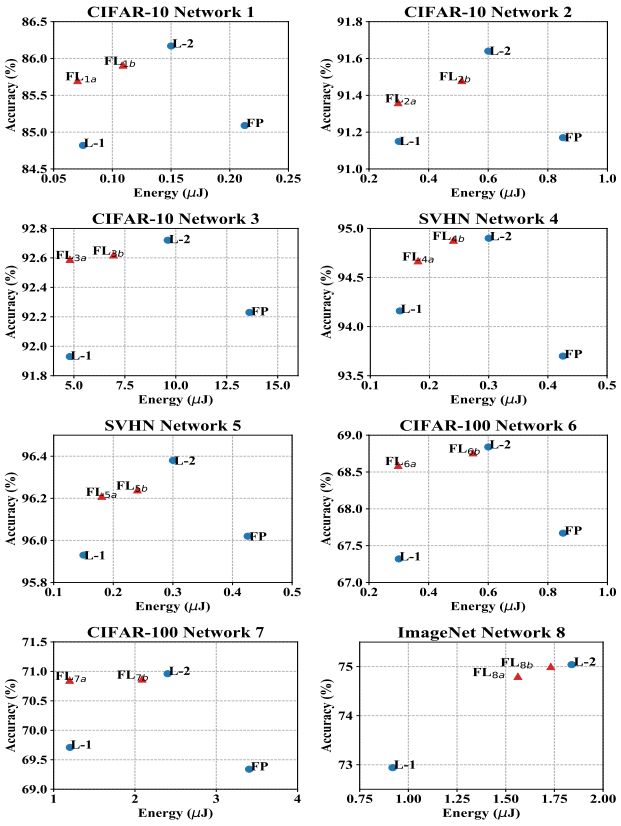

Figure 5: Accuracy and computational energy consumption in ASIC for different quantized models on CIFAR-10, SVHN, CIFAR-100 and ImageNet datasets. FLightNNs are marked as red triangles, while the other models are shown as blue dots.

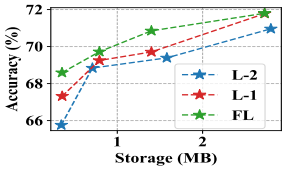

Figure 6: Accuracy-storage front for LightNN-2 LightNN-1 and FLightNN. The Pareto front of FLightNN is the upper bound of LightNNs.

7 CONCLUSION

In this paper, we propose FLightNNs which customize the number of shift operations for each filter of LightNNs. Equipped with the proposed differentiable training algorithm, FLightNNs can achieve a flexible trade-off between accuracy and speed/energy. Our experimental results on FPGA and ASIC simulations show that FLightNNs can provide a more continuous Pareto front for LightNN models and consistently outperform fixed-point DNNs *w.r.t.* both accuracy and speed/energy. Moreover, due to the gradual quantization nature of the differentiable training, FLightNNs can achieve higher accuracy than LightNN-1 without sacrificing speed and energy efficiency, and thus, push forward the Pareto-optimal front. These promising results suggest the potentials for FLightNNs to achieve fast and accurate inference on learning-based customized hardware.

ACKNOWLEDGMENTS

This research was supported in part by NSF CCF Grant No. 1815899.

REFERENCES

[1] [n. d.]. NNAPI. https://developer.android.com/ndk/guides/neuralnetworks/. Accessed: 2018-10-15.
[2] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432* (2013).
[3] David M Brooks. [n. d.]. Co-designed Systems for Deep Learning Hardware Accelerators. In *IEEE VLSI Design, Automation and Test (VLSI-DAT), International Symposium*. 1–1, 2018.
[4] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. [n. d.]. Eyeriss: An Energy-efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits* 52, 1 ([n. d.]), 127–138, 2017.
[5] Zhuo Chen, Ruizhou Ding, Ting-Wu Chin, and Diana Marculescu. 2018. Understanding the Impact of Label Granularity on CNN-based Image Classification. In *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, 895–904.
[6] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. [n. d.]. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*. 3123–3131, 2015.
[7] Ruizhou Ding, Ting-Wu Chin, Diana Marculescu, and Zeye Liu. 2019. Regularizing Activation Distribution for Training Binarized Deep Networks. In *IEEE CVPR*. IEEE.
[8] Ruizhou Ding, Zeye Liu, RD Blanton, and Diana Marculescu. 2018. Lightening the Load with Highly Accurate Storage- and Energy-Efficient LightNNs. *ACM transactions on Reconfigurable Technology and Systems* 11, 3 (2018), 20–44.
[9] Ruizhou Ding, Zeye Liu, Rongye Shi, Diana Marculescu, and RD Blanton. 2017. LightNN: Filling the Gap between Conventional Deep Neural Networks and Binarized Networks. In *Proceedings of the on Great Lakes Symposium on VLSI*. ACM, 35–40, 2017.
[10] Yinpeng Dong, Renkun Ni, Jianguo Li, Yurong Chen, Jun Zhu, and Hang Su. 2017. Learning Accurate Low-Bit Deep Neural Networks with Stochastic Quantization. *BMVC* (2017).
[11] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. [n. d.]. Deep learning with limited numerical precision. In *Proceedings of the 32nd International Conference on Machine Learning*. 1737–1746, 2015.
[12] Jun Han and Claudio Moraga. [n. d.]. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *International Workshop on Artificial Neural Networks*. 195–201, 1995.
[13] Song Han, Jeff Pool, John Tran, and William Dally. [n. d.]. Learning Both Weights and Connections for Efficient Neural Network. In *Advances in Neural Information Processing Systems*. 1135–1143, 2015.
[14] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized Neural Networks. In *Advances in Neural Information Processing Systems*. 4107–4115, 2016.
[15] Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *ICLR* (2015).
[16] Darryl Lin, Sachin Talathi, Sachin Talathi, and Sreekanth Annapureddy. 2016. Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning*. 2849–2858, 2016.
[17] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).
[18] Christos Louizos, Max Welling, and Diederik P. Kingma. 2018. Learning Sparse Neural Networks through L_0 Regularization. In *International Conference on Learning Representations*.
[19] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. [n. d.]. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, Vol. 30. 3, 2013.
[20] Synopsys MEDICI User's Manual. 2010. Synopsys inc. *Mountain View, CA* (2010).
[21] Diana Marculescu, Dimitrios Stamoulis, and Ermao Cai. 2018. Hardware-aware machine learning: modeling and optimization. In *Proceedings of the International Conference on Computer-Aided Design*. ACM, 137.
[22] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. [n. d.]. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4510–4520, 2018.
[23] Dimitrios Stamoulis, Ermao Cai, Da-Cheng Juan, and Diana Marculescu. 2018. HyperPower: Power-and memory-constrained hyper-parameter optimization for neural networks. In *2018 Design, Automation & Test in Europe Conference &*

Exhibition (DATE). IEEE, 19–24.

[24] Dimitrios Stamoulis, Ting-Wu Rudy Chin, Anand Krishnan Prakash, Haocheng Fang, Sribhuvan Sajja, Mitchell Bognar, and Diana Marculescu. 2018. Designing adaptive neural networks for energy-constrained image classification. In *Proceedings of the International Conference on Computer-Aided Design*. ACM, 23.

[25] Hokchhay Tann, Soheil Hashemi, R Iris Bahar, and Sherief Reda. [n. d.]. Hardware-software Codesign of Accurate, Multiplier-free Deep Neural Networks. In *54th Design Automation Conference (DAC)*. 1–6, 2017.

[26] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. [n. d.]. Learning Structured Sparsity in Deep Neural Networks. In *Advances in Neural Information Processing Systems*. 2074–2082, 2016.

[27] Xilinx. 2017. Vivado High-Level Synthesis. *https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html* (2017).

[28] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. [n. d.]. Designing Energy-Efficient Convolutional Neural Networks Using Energy-Aware Pruning. In *IEEE CVPR*. 6071–6079, 2017.

[29] Chen Zhang, Peng Li, Guanyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. [n. d.]. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*. 161–170, 2015.

[30] Hengyu Zhao and Jiawen Liu. 2018. Processing-in-Memory for Energy-efficient Neural Network Training: A Heterogeneous Approach. In *IEEE/ACM International Symposium on Microarchitecture*.

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