Efficient Network Construction Through Structural Plasticity

Xiaocong Du, *Student Member, IEEE*, Zheng Li, *Student Member, IEEE*, Yufei Ma, *Member, IEEE*, and Yu Cao, *Fellow, IEEE*

***Abstract*— Deep Neural Networks (DNNs) on hardware is facing excessive computation cost due to the massive number of parameters. A typical training pipeline to mitigate over-parameterization is to pre-define a DNN structure with redundant learning units (filters and neurons) with the goal of high accuracy, then to prune redundant learning units after training with the purpose of efficient inference. We argue that it is sub-optimal to introduce redundancy into training in order to reduce redundancy later in inference. Moreover, the fixed network structure further results in poor adaption to dynamic tasks, such as lifelong learning. In contrast, structural plasticity plays an indispensable role in mammalian brains to achieve compact and accurate learning. Throughout the lifetime, active connections are continuously created while those that are no longer important are degenerated. Inspired by such observation, we propose a training scheme, namely Continuous Growth and Pruning (CGaP), where we start the training from a small network seed, then literally execute continuous growth by adding important learning units and finally prune secondary ones for efficient inference. The inference model generated from CGaP is sparse in the structure, largely decreasing the inference power and latency when deployed on hardware platforms. With popular DNN structures on representative datasets, the efficacy of CGaP is benchmarked by both algorithmic simulation and architectural modeling on Field-programmable Gate Arrays (FPGA). For example, CGaP decreases the FLOPs, model size, DRAM access energy and inference latency by 63.3%, 64.0%, 11.8% and 40.2%, respectively, for ResNet-110 on CIFAR-10.**

***Index Terms*— Deep learning, structural plasticity, model pruning, hardware acceleration, algorithm-hardware co-design.**

I. Introduction

DEEP Neural Networks have various applications including image classification [1], object detection [2], speech recognition [3] and natural language processing [4]. However, the accuracy of DNNs heavily relies on massive amounts of parameters and deep structures, making it hard to deploy DNNs on resource-limited embedded systems. When training or inferring the DNN models on hardware, the model must be stored in the external memory such as dynamic random-access memory (DRAM) and fetched multiple times. These operations are expensive in computation, memory access, and energy consumption. For example, Fig. 1 shows the energy consumption of one inference pass in several modern DNN structures, simulated by the FPGA performance model [5] under the setting of 300 MHz operating frequency and 19.2 GB/s DRAM bandwidth. The input image size is 32×32. A typical DNN model is too large to fit in on-chip memory. For instance, VGG-19 [6] has 20.4M parameters. Running such a model requires frequent external memory access, exacerbating the power consumption of a typical embedded system.

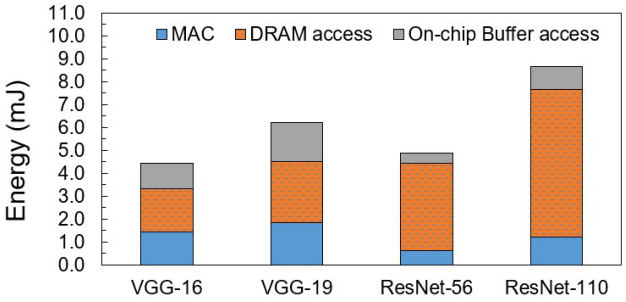


Fig. 1. Energy breakdown for modern DNN structures, results from simulation by the FPGA performance model [5]. Due to the redundancy in parameters, multiply-accumulator (MAC) and external memory (DRAM) access dominate the energy consumption.

Previous researches have designed customized hardware for DNN acceleration [7], [8]. Most of them are limited to relatively small neural networks, such as LeNet-5 [9]. For larger networks such as AlexNet [1] and VGG-16 [6], additional efforts are usually required to improve the hardware efficiency [10], [11]. For example, [10] saves the energy through data gating and zero skipping. Some other works focus on data reuse of convolutional layers and demonstrate the results on specific hardware [7], [12]–[14]. However, their improvements are limited on those networks where fully-connected layer is widely used, such as RNNs and LSTMs.

To support more general models, network pruning is a popular approach by removing secondary weights and neurons. Network pruning executes a three-step procedure, which 1) trains a pre-designed network from scratch, 2) removes less important connections or filters/neurons according to a

Manuscript received April 21, 2019; revised July 1, 2019; accepted July 31, 2019. Date of publication August 5, 2019; date of current version September 17, 2019. This work was supported in part by the C-BRIC, one of six centers in JUMP, in part by the Semiconductor Research Corporation (SRC) Program, and in part by the by the National Science Foundation (NSF) under CCF 1715443. This article was recommended by Guest Editor M. Ziegler. *(Corresponding authors: Xiaocong Du; Yufei Ma.)*

X. Du, Y. Ma, and Y. Cao are with the School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ 85287 USA (e-mail: xiaocong@asu.edu; yufeima@asu.edu; ycao@asu.edu).

Z. Li is with the School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, AZ 85287 USA (e-mail: zhengl11@asu.edu).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JETCAS.2019.2933233

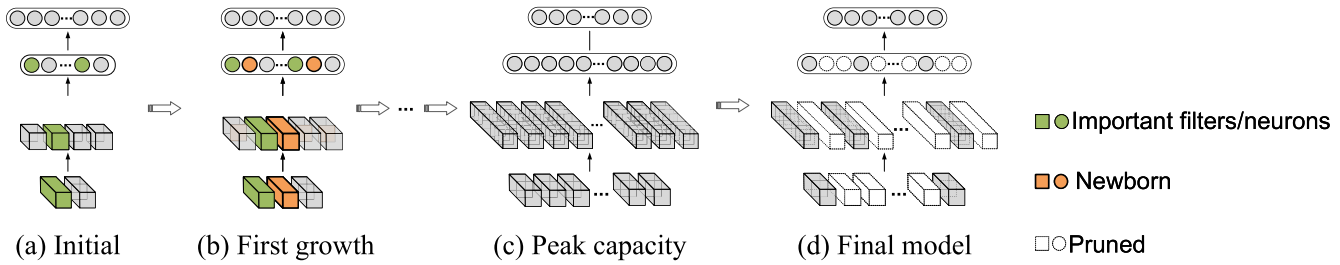


Fig. 2. The proposed CGaP scheme. CGaP starts the training from a seed network instead of an over-parameterized one, gradually grows important learning units during the training and reaches peak capacity at the end of growth, then prunes secondary filters and neurons to generate an inference model with structured sparsity and up-to-date accuracy.

saliency score (a metrics to measure the importance of weights and learning units) [15]–[19], or by adding a regularization term into the loss function [20], [21], and 3) fine-tunes to recover the accuracy.

However, the above pruning techniques suffer from two limitations: (1) training a large and fixed network from scratch could be sub-optimal as it introduces redundancy; (2) in the process of training, pruning only discards less important weights at the end of training but does not strengthen important weights and nodes. These limitations of network pruning confine the learning performance as well as the model pruning efficiency (i.e., how many parameters can be removed and how structured the sparsity is).

In contrast to the static DNN model, the biological nervous system exhibits active growth and pruning through the lifetime. [22]–[24] have observed that the rapid growth of neurons and synapses takes place in an infant's brain and is vital to the maturity of an adult's brain. In brains, some neurons and synapses are used more frequently and are consequently strengthened. Those neurons and synapses that are not used consistently are weakened and removed. The structural plasticity of brain is central to the study of developmental biology.

Inspired by this observation from biology, we propose a training scheme named Continuous Growth and Pruning (CGaP), which leverages structural plasticity to tackle the aforementioned limitations of pruning techniques. Instead of training an over-parameterized network from scratch, CGaP starts the training from a small network seed (Fig. 2(a)), whose size is as low as 0.1%-3% of the full-size reference model. In each iteration of the growth, CGaP locally sorts neurons and filters (also known as output channels in some literature) according to our saliency score (Section III-B). Based on the saliency score, important learning units are selected and the corresponding new units are added (see Fig. 2(b)). The selection and addition of important units help reinforce the learning and increase model capacity. Then a filter-wise and neuron-wise pruning will be executed on the post-growth model (Fig. 2(c)) based on pruning metrics. Finally, CGaP generates a significantly sparse and structured inference model (Fig. 2(d)) with accuracy improved. In the generated inference model, large amounts of filters and neurons have been removed, achieving structured pruning. Compared to non-structured pruning [15], CGaP benefits hardware implementation as it reduces the computation volume and memory access without any additional hardware architecture change.

Algorithmic experiments and hardware simulations validate that CGaP significantly decreases the number of external and on-chip memory accesses, accelerating the inference by bypassing the removed filters and neurons. On the algorithm side, we demonstrate the performance in accuracy and model pruning on several networks and datasets. For instance, CGaP reduces 78.9% parameters of VGG-19 with +0.37% accuracy improvement on CIFAR-100 [25], 85.8% parameters with +0.23% accuracy improvement on SVHN [26]. For ResNet-110 [27], CGaP reduces 64.0% parameters with +0.09% accuracy improvement on CIFAR-10 [25]. These results exceed the state-of-the-art pruning methods [15]–[18], [28], [29]. Furthermore, we validate the efficiency of the inference model generated from CGaP using FPGA simulator [5]. For one inference pass of VGG-19 on CIFAR-100, previous non-structured pruning approach [15] requires energy consumption of 2.7×10^9 pJ in accessing DRAM and 5.6 ms inference latency, while CGaP requires only 2.2×10^9 pJ and 4.4 ms latency.

The contribution of this paper is as follows:

- A brain-inspired training flow (CGaP) with a dynamic structure is proposed. CGaP grows the network from a small seed and effectively reduces over-parameterization without sacrificing accuracy.
- The advantage of structured sparsity of the inference model generated from CGaP is validated using a high-level FPGA performance model including on-chip buffer access energy, external memory access energy and inference latency.
- The discussion and understanding of the reason that the growth improves the learning efficiency are provided.

The rest of the paper is organized as follows. Section II introduces the background of model pruning. Section III demonstrates the saliency score used to select the learning units. Section IV describes the proposed Continuous Growth and Pruning scheme. Section V presents the experimental results from algorithmic simulations. Section VI demonstrates the simulation results from FPGA performance modeling. Section VII discusses the understanding of network plasticity as well as ablation study. Section VIII concludes this work and discusses the insight into future work.

II. Previous Work

There have been broad interests in reducing the redundancy of DNNs in order to deploy them on a resource-limited hardware platform. The structural surgery is a widely used approach and can be categorized into destructive direction and constructive direction. We will discuss these two directions, as well as orthogonal approaches to our methods in this section.

A. Destructive Methods

The destructive methods zero out specific connections or remove filters or neurons in convolutional or fully-connected layers, generating a sparse model. Weight magnitude pruning [15] pruned weights by setting the selected weights to zeros. The selection is based on L1-norm, i.e., the absolute value of the weight. Weight magnitude pruning generates a sparse weight matrix, but not in a structured way. In this case, specific hardware design [30] is needed to take advantage of the optimized inference model, otherwise the non-structured sparsity does not benefit hardware acceleration due to the overhead in model management. The kernel-wise pruning [16] pruned kernels layer by layer based on the saliency metrics of each filter and achieved structured sparsity in the inference model. Compared to [16], CGaP prunes filters, leading to a more structured inference model. Besides the saliency-based pruning, the penalty-based approach has been explored by [21], [31] and structured sparsity was achieved. Our method is different from all the above pruning schemes from two perspectives: (1) We start training from a small seed other than an over-parameterized network; (2) Besides removing secondary filters/neurons, we also reinforce important ones to further improve learning accuracy and model compactness.

B. Constructive Methods

The constructive approaches include techniques that add new connections or filters to enlarge the model capacity. [32], [33] increased network size by adding random neurons with fresh initialization (i.e., weights are randomly initialized, without pre-trained information). They evaluated their approach on basic XOR problems. Different from their approach, CGaP selectively adds neurons and filters that are initialized with the information learned from the previous training. Meanwhile, CGaP is validated on modern DNNs and datasets under more realistic scenarios. [34] grew the smallest Neural Tree Networks (NTN) to minimize the number of classification errors on Boolean function learning tasks, and used pruning to enhance the generalization of NTN. [35] improved the accuracy of radial basis function (RBF) networks on function approximation tasks by adding and removing hidden neurons. To enhance the accuracy of spike-based classifiers, [36] progressively added dendrites to the network, and then optimized the topology of the dendritic tree. Different from them, CGaP aims at improving the efficiency of the inference model of modern Deep Neural Networks on image classification tasks. [37] constructed the DNN by activating connections and choosing a set of convolutional filters among a bunch of randomly generated filters according to their influence on the training performance. However, this approach highly depends on trial and error to find the optimal set of filters that could reduce loss the most. This approach is sensitive to power and timing budgets, limiting its extension on large datasets. Unlike their work, CGaP directly grows the network from a seed, minimizing the effort on trail and error.

C. Orthogonal Methods

The orthogonal methods, such as low-precision quantization and low-rank decomposition, compress the DNN models by quantizing the parameters to fewer bits [38], [39], or by finding a low-rank approximation [40], [41]. Note that our CGaP approach can be combined with these orthogonal methods to further improve inference efficiency.

III. Saliency Score

In this section, we describe the detailed methodology of CGaP, starting from the saliency score, which is used to sample the importance of a learning unit. Section III-A defines the terminology we use in this paper. Section III-B provides the mathematical proof of the saliency score we adopt.

A. Terminology

A DNN can be treated as a feedforward multi-layer architecture that maps the input images to certain output vectors. Each layer is a certain function, such as convolution, ReLU, pooling and inner product, whose input is $\mathcal{X}$, output is $\mathcal{Y}$ and parameter is $\mathcal{W}$ in case of convolutional and fully-connected layers. Hereby the convolutional layer (conv-layer) is formulated as: $\mathcal{Y}_l = \mathcal{X}_l * \mathcal{W}_l$, wherein $\mathcal{X}_l \in \mathbb{R}^{I_l \times Wi_l \times Hi_l}$, $\mathcal{Y}_l \in \mathbb{R}^{O_l \times Wo_l \times Ho_l} \Leftrightarrow \mathcal{X}_{l+1} \in \mathbb{R}^{I_{l+1} \times Wi_{l+1} \times Hi_{l+1}}$, $\mathcal{W}_l \in \mathbb{R}^{O_l \times I_l \times K \times K}$, where subscript $_l$ denotes the index of the layer. And the fully-connected layer is represented by: $\mathcal{Y}_l = \mathcal{X}_l \cdot \mathcal{W}_l$, where the input $\mathcal{X}_l \in \mathbb{R}^{I_l}$, the output $\mathcal{Y}_l \in \mathbb{R}^{O_l} \Leftrightarrow \mathcal{X}_{l+1} \in \mathbb{R}^{I_{l+1}}$, and the parameter matrix is $\mathcal{W}_l \in \mathbb{R}^{O_l \times I_l}$.

1) Convolutional Layer (Conv-Layer) l: the 4 dimensions of its weight matrix are: the number of output channels O_l, the number of input channels I_l, and the kernel width and height K, respectively. We denote the o-th **3D filter**, which generates the o-th output channel in the feature map, as $W_l^o \in \mathbb{R}^{I_l \times K \times K}$. The i-th **2D kernel** in the o-th filter is denoted as $W_l^{o,i} \in \mathbb{R}^{K \times K}$. On the other hand, a **4D weight tensor** $\mathbf{W}_l^i \in \mathbb{R}^{O_l \times 1 \times K \times K}$, which operates on the i-th input feature map, is a package of O_l kernels across all output channels. For example, in Fig. 3, $W_{l,picked}^j$ is a 3D filter consisting of I_l kernels, and $\mathbf{W}_{l+1,projected}^j$ as well as $\mathbf{W}_{l+1,mapped}^j$ are both 4D tensors with dimension of $O_l \times 1 \times K \times K$, which include all the output channels but have only one input channel located at j. The $W_l^{o,i,m,n} \in \mathbb{R}^{1 \times 1}$ refers to one weight at the m-th row and the n-th column in the o-th filter of the i-th input channel.

2) Fully-Connected Layer (fc-Layer) l: input $\mathcal{X}_l$ propagate from one hidden activation i to the next layer. We refer the whole set of $W_{l,fan-out}^i$ as a neuron N_l^i. This neuron receives information from previous layer $l-1$ through its **fan-in** weights $W_{l,fan-in}^i \in \mathbb{R}^{1 \times I_{l-1}}$ (as shown in Fig. 4)

Algorithm 1 Entire Flow

Input: Model seed $M_{initial}$

1: Initialize a small network model $M_{current} \leftarrow M_{initial}$.
2: **for** epoch = 1 to E **do**
3: Train current model $M_{current}$ and fetch $Accuracy$.
4: **if** $epoch\%\frac{1}{f_{growth}} = 0$ and $M_{current} < \tau_{capa.}$ **then**
5: Grow the network according to Algorithm 2
6: $M_{current} \leftarrow M_{grown}$.
7: **end if**
8: $M_{peak} \leftarrow M_{current}$.
9: **if** epoch$\%\frac{1}{f_{pruning}} = 0$ and $Accuracy > \tau_{accu.}$ **then**
10: Prune the network following Algorithm 3
11: $M_{current} \leftarrow M_{pruned}$.
12: **end if**
13: **end for**
14: $M_{final} \leftarrow M_{current}$ and test M_{final}.

Output: Final compact model M_{final}

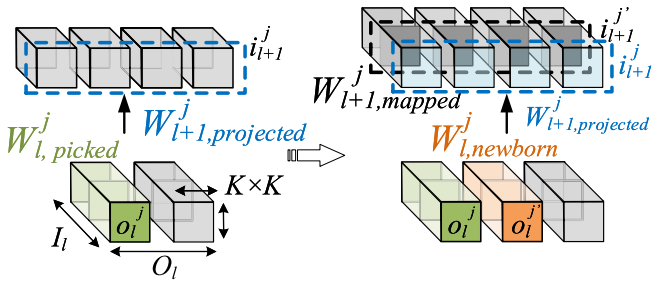


Fig. 3. Illustration of two-step growth in conv-layers. The growth phase follows a two-step (growing and mapping) procedure. After the filter $W_{l,picked}^{j}$ (green) is picked and split aside, giving birth to $W_{l,newborn}^{j}$ (orange), the projected input-wise filter, $\mathbf{W}_{l+1,projected}^{j}$ (blue) in layer $l+1$, is as well split aside, generating $\mathbf{W}_{l+1,mapped}^{j}$ (black).

and propagates to the next layer through **fan-out** weights $W_{l,fan-out}^{i} \in \mathbb{R}^{O_l \times 1}$. Also note that the output dimension of layer $l-1$ equals to the input dimension of layer l, i.e., $O_{l-1} = I_l$. The weight pixel in layer l at the cross-point of row o and column i is denoted as $W_{l,fan-out}^{o,i} \in \mathbb{R}^{1\times 1}$. Moreover, the 'depth' of a DNN model indicates the number of layers, and the 'width' of a DNN model refers to the number of filters or neurons of each layer.

3) Learning Units: Growing or pruning a **filter** W_l^o indicates adding or removing $W_l^o \in \mathbb{R}^{I_l \times K \times K}$ and its corresponding output feature map. Growing or pruning a **neuron** N_l^i means adding or removing both $W_{l,fan-out}^{i} \in \mathbb{R}^{O_l \times 1}$ and $W_{l,fan-in}^{i} \in \mathbb{R}^{1 \times I_{l-1}}$.

B. Saliency Score

We adopt a saliency score to measure the effect of a single filter/neuron on the loss function, i.e., the importance of each learning unit. The saliency score is developed from Taylor Expansion of the loss function. Previously, [42] applied it on pruning. In this paper, we adopt this saliency score and apply it on the growth and pruning scheme. In this section, we provide a mathematical formulation of the saliency score.

The saliency score represents the difference between the loss with and without each unit. In other words, if the removal of a filter/neuron leads to relatively small accuracy degradation, this unit is recognized as an unimportant unit, and vice versa. Thus, the objective function to get the filter with the highest saliency score is formulated as:

$$\begin{aligned} &\underset{W_l^o}{argmin} |\Delta\mathcal{L}(W_l^o)| \\ &\quad \Leftrightarrow \underset{W_l^o}{argmin} |\mathcal{L}(\mathcal{Y};\mathcal{X},\mathcal{W}) - \mathcal{L}(\mathcal{Y};\mathcal{X},W_l^o = \mathbf{0})|. \end{aligned} \tag{1}$$

Using the first-order of the Taylor Expansion:

$$|\mathcal{L}(\mathcal{Y};\mathcal{X},\mathcal{W}) - \mathcal{L}(\mathcal{Y};\mathcal{X},W_l^o = \mathbf{0})| \ at \ W_l^o = \mathbf{0}. \tag{2}$$

we get:

$$\begin{aligned} |\Delta\mathcal{L}(W_l^o)| &\simeq |\frac{\partial\mathcal{L}(\mathcal{Y};\mathcal{X},\mathcal{W})}{\partial W_l^o} W_l^o| \\ &= \sum_{i=0}^{I_l}\sum_{m=0}^{K}\sum_{n=0}^{K} |\frac{\partial\mathcal{L}(\mathcal{Y};\mathcal{X},\mathcal{W})}{\partial W_l^{o,i,m,n}} W_l^{o,i,m,n}|. \end{aligned} \tag{3}$$

Similarly, the saliency score of a neuron is derived as:

$$\begin{aligned} |\Delta\mathcal{L}(N_l^i)| &\simeq |\frac{\partial\mathcal{L}(\mathcal{Y};\mathcal{X},\mathcal{W})}{\partial W_{l,fan-out}^{i}} W_{l,fan-out}^{i}| \\ &= \sum_{o=0}^{O_l} |\frac{\partial\mathcal{L}(\mathcal{Y};\mathcal{X},\mathcal{W})}{\partial {W_{l,fan-out}}^{o,i}} W_{l,fan-out}^{o,i}|. \end{aligned} \tag{4}$$

IV. CGaP Methodology

With the saliency score as the foundation, we develop the entire CGaP flow atop. This section explains the overall flow and the detailed implementation of each step in CGaP.

The CGaP scheme is described in Algorithm 1. Starting from a small network seed, the growth takes place periodically at a frequency of f_{growth} (see Algorithm 1 line 4, where '%' denotes the operation to obtain the remainder of division). During each growth, important learning units are chosen and grown at growth ratio β layer by layer from the bottom (input) to top (output), based on the local ranking of the saliency score. The growth phase stops when reaching a capacity threshold $\tau_{capa.}$, followed by several epochs of training on the peak model M_{peak}. When the training accuracy reaches a threshold $\tau_{accu.}$, the pruning phase starts. Pruning is performed layer by layer, from the bottom layer to the top layer, at the frequency of $f_{pruning}$. The details in the growth phase and the pruning phase is demonstrated as follows.

A. Growth Phase

Algorithm 2 presents the methodology in the growth phase. Each iteration of growth in a layer consists of two steps: growth in layer l and mapping in the adjacent layer. There are two conditions need to be discussed separately: convolutional layers (Fig. 3) and fully-connected layers (Fig. 4). Due to the difference between these two kinds of operation as discussed previously, after the growth of layer l, the mapping in conv-layer takes place at the adjacent layer $l+1$. In fc-layers, the mapping is in layer $l-1$.

Algorithm 2 Growth Phase

Input: Current network $M_{current}$

for each layer l = 1 to L **do**
  for each filter W_l^o in conv-layer l, or each neuron N_l^i in fc-layer l **do**
    Calculate growth score $GS_{W_l^o}$ according to Eq. 3 and $GS_{N_l^i}$ according to Eq. 4.
  end for
  Sort all units and select βO_l filters or βI_l neurons with the highest $GS_{W_l^o}$ or $GS_{N_l^i}$.
  for each filter j = 1 to βO_l (for fc-layer, βI_l) **do**
    Add one filter/neuron on the side of the each picked filter/neuron in layer l.
    Initialize picked and new-born filters (neurons) according to Eq. 5 and Eq. 6.
    Map corresponding input-wise weight in layer $l+1$ (fan-in weights in layer $l-1$).
    Initialize projected and mapped filters according to Eq. 7 and Eq. 8 (neurons according to Eq. 9 and Eq. 10).
  end for
end for

Output: M_{grown}

Algorithm 3 Pruning Phase

Input: Current network $M_{current}$

for each weight $W_l^{o,i,m,n} \in \mathbb{R}^{1\times 1}$ in conv-layer l or each $W_l^{o,i} \in \mathbb{R}^{1\times 1}$ in fc-layer l **do**
  Calculate weight pruning score PS_W according to Eq. 11 for conv-layers and Eq. 12 for fc-layers.
end for
Sort weights by PS_W.
Zero-out
the lowest $\gamma_W \prod(O_l, I_l, K, K)$ weights in conv-layer and $\gamma_W \prod(I_l, O_l)$ weights in fc-layer.
for each filter W_l^o (neuron N_l^i) in all layers **do**
Zero-out
  entire filter W_l^o (neuron N_l^i) if the weight sparsity is larger than pruning rate γ_F (γ_N).
end for

Output: M_{pruned}

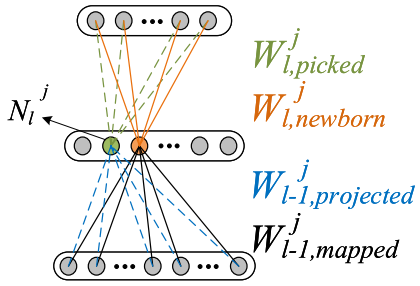


Fig. 4. Illustration of two-step growth in fc-layers. First, fan-out weights $W^j_{l,newborn}$ (orange) is added, then fan-in weights $\mathbf{W}^j_{l-1,mapped}$ (black) form the connections from the newborn neuron to all neurons in layer $l-1$.

1) Growth in Conv-Layer l: According to the local ranking of the saliency score (Eq. 3), we sort all the 3D filters in this layer. With a growth ratio β, $\beta O_{l,t}$ filters are selected in the l-th layer at the t-th growth. On the side of each selected filter $\mathbf{W}^j_{l,picked} \in \mathbb{R}^{I_l\times K\times K}$, as shown in Fig. 3, we create a new filter that has the same size, named $\mathbf{W}^j_{l,newborn} \in \mathbb{R}^{I_l\times K\times K}$.

In the ideal case, the new filter $\mathbf{W}^j_{l,newborn}$ and existing filter $\mathbf{W}^j_{l,picked}$ are expected to collaborate with each other and optimize the learning. The existing filter $\mathbf{W}^j_{l,picked}$ has already learned on the current task. To keep the same learning pace between the existing filter and the new filter, we initialize $\mathbf{W}^j_{l,newborn}$ as follows:

$$\mathbf{W}^j_{l,newborn} = \sigma \mathbf{W}^j_{l,picked} + X \sim U([-\mu, \mu]), \quad (5)$$

$$\mathbf{W}^j_{l,picked} = \sigma \mathbf{W}^j_{l,picked} + X \sim U([-\mu, \mu]), \quad (6)$$

where $\sigma \in (0, 1]$ is a scaling factor and X is a constant following uniform distribution in $[-\mu, \mu]$, where $\mu \in (0, 1]$. Instead of random initialization, the above initialization helps reconcile the learning status of the newborn filters with the old filters. Meanwhile, the scaling factor prevents output from an exponential explosion caused by the feedforward propagation $\mathcal{Y}_l = \mathcal{X}_l * \mathcal{W}_l$. The noise X prevents the learning from sticking at a local minimum that leads to sub-optimal solutions. No matter which distribution the noise X follows, X in a reasonable range is able to provide similar performance. However, other distributions usually introduce more hyper-parameters and thus, require more efforts in parameter tuning. For example, Gaussian noise introduces more hyper-parameter, e.g., the standard deviation, than uniform noise. For simplicity, we use uniformly distributed noise.

2) Mapping in Conv-Layer l+1: After the number of filters in layer l grows from $O_{l,t}$ to $(1+\beta)O_{l,t}$, the number of output feature maps also increases from $O_{l,t}$ to $(1+\beta)O_{l,t}$. Therefore, the input-wise dimension of layer $l+1$ should

increase correspondingly in order to be consistent in data propagation. To match the dimension, we first locate the 4D tensor $\mathbf{W}^j_{l+1,projected}$ in layer $l+1$, which processes the feature maps generated by $W^j_{l,picked}$. Then we add a new 4D tensor $\mathbf{W}^j_{l+1,mapped}$ adjacent to $\mathbf{W}^j_{l+1,projected}$. The $\mathbf{W}^j_{l+1,mapped}$ and $\mathbf{W}^j_{l+1,projected}$ are initialized as follows:

$$\mathbf{W}^j_{l+1,mapped} = \sigma \mathbf{W}^j_{l+1,projected} + X \sim U([-\mu, \mu]), \quad (7)$$

$$\mathbf{W}^j_{l+1,projected} = \sigma \mathbf{W}^j_{l+1,projected} + X \sim U([-\mu, \mu]). \quad (8)$$

To summarize, as illustrated in Fig. 3, the filter $W^j_{l,picked}$ (green) is selected according to the saliency score and a new tensor $W^j_{l,newborn}$ (orange) is added. Then the input-wise tensor $\mathbf{W}^j_{l+1,projected}$ (in blue dashed rectangular) in layer $l+1$ is projected, and $\mathbf{W}^j_{l+1,mapped}$ (in black dashed rectangular) is generated.

After layer l grows and layer $l+1$ is mapped, layer $l+1$ grows and layer $l+2$ is mapped, so on and so forth till the last convolutional layer. It is worth mentioning that for the 'projection shortcuts' [27] with 1×1 convolutions in ResNet [27], the dimension mapping is between the two layers that the shortcut connects, not necessarily to be the adjacent layers.

3) Growth and Mapping in fc-Layers: As illustrated in Fig. 4, the neuron growth in fc-layers l occurs at fan-out weights, and its initialization follows Eq. 5 and 6.

The mapping in fc-layers take place in the fan-in weights as follows:

$$\mathbf{W}^j_{l-1,mapped} = \sigma \mathbf{W}^j_{l-1,projected} + X \sim U([-\mu, \mu]), \quad (9)$$

$$\mathbf{W}^j_{l-1,projected} = \sigma \mathbf{W}^j_{l-1,projected} + X \sim U([-\mu, \mu]). \quad (10)$$

After growing the last conv-layer, We flatten the output feature map of this conv-layer, treat it as the input from layer $l-1$ and map in the same manner.

B. Pruning Phase

Pruning in each layer consists of two steps: weight pruning and unit pruning. First, we sort weight pixels locally in each conv-layer according to Eq.11:

$$PS_{W^{o,i,m,n}_l} = |\frac{\partial \mathcal{L}(\mathcal{Y}; \mathcal{X}, \mathcal{W})}{\partial W^{o,i,m,n}_l} W^{o,i,m,n}_l| \quad (11)$$

and in each fc-layer according to Eq.12:

$$PS_{W^{o,i}_l} = |\frac{\partial \mathcal{L}(\mathcal{Y}; \mathcal{X}, \mathcal{W})}{\partial W_{l,fan-out}{}^{o,i}} W^{o,i}_{l,fan-out}| \quad (12)$$

In each layer, $100\gamma_W\%$ weight pixels with the lowest PS_W are set as zero, where $\gamma_W \in (0, 1)$ is the weight pruning rate. Then the entire filter/neuron whose sparsity is larger than the filter/neuron pruning rate γ_F or $\gamma_N \in (0, 1)$ is set to zero. In this way, a large amount of entire filters/neurons are pruned, leading to a compact inference model.

V. Algorithmic Experiments

To evaluate the proposed approach, we present experimental results in this section. We perform experiments on several modern DNN structures (LeNet [9], VGG-Net [6], ResNet [27]) and representative datasets (MNIST [9], CIFAR-10, CIFAR-100 [25], SVHN [26]).

A. Training Setup

1) Network Structures: The LeNet-5 architecture consists of two sets of convolutional, ReLU [44] and max pooling layers, followed by two fully-connected layers and finally a softmax classifier. The VGG-16 and VGG-19 structures we use have the same convolutional structure as [6] but are redesigned with only two fully-connected to be fairly compared with the pruning-only method [16]. Therefore, the VGG-16 (VGG-19) has 13 (16) convolutional layers, each is followed by a batch normalization layer [45] and a ReLU activation. The structures of ResNet-56 and ResNet-110 follow [16]. Each convolutional layer is followed by a batch normalization layer and ReLU activation. During the training, the depth of the networks remains constant since CGaP does not touch the depth of the network, but the width of each layer changes.

Note that in the following text, we denote the full-size models trained from scratch without sparsity regularization as 'baseline' models. The three-step pruning schemes that remove weights or filters but do not execute network growth are denoted as 'pruning-only' models.

2) Datasets: MNIST is a handwritten digit dataset in grey-scale (i.e., one color channel) with 10 classes from digit 0 to digit 9. It consists of 60,000 training images and 10,000 testing images. The CIFAR-10 dataset consists of 60,000 32×32 color images in 10 classes, with 5000 training images and 1000 testing images per class. The CIFAR-100 dataset has 100 classes, including 500 training images and 100 testing images per class. The Street View House Number (SVHN) is a real-world color image dataset that is resized to a fixed resolution of 32×32 pixels. It contains 73,257 training images and 26,032 testing images.

3) Hyper-Parameters: We set the learning rate to be 0.1 and divide by 10 for every 30% of the training epochs. We train our model using Stochastic Gradient Descent (SGD) with a batch size of 128 examples, a momentum of 0.9, and a weight decay of 0.0005. The loss function is the cross-entropy loss with softmax function. We train 60, 200, 220 and 100 epochs on MNIST, CIFAR-10, CIFAR-100 and SVHN datasets, respectively. In the growth phase, we have hyper-parameters set as follows: the growth stopping condition $\tau_{capa.} = O_{1,baseline}$, i.e., the growth stops at the t-th growth if the number of filters in the $(t+1)$-th growth is larger than the baseline model. The growth ratio β is set as 0.6. The growth frequency f_{growth} is set as 1/3. The scaling factor σ in Eq. 5 to Eq. 10 is set to 0.5 and μ is 0.1. The pruning frequency $f_{pruning}$ is set to be 1. The setting of the weight pruning rate γ_W follows [15], [16] and [18] for LeNet-5, VGG-Net and ResNet, respectively. γ_F and γ_N is set to be same as γ_W.

4) Framework and Platform: The experiments are performed with PyTorch [46] framework on one NVIDIA

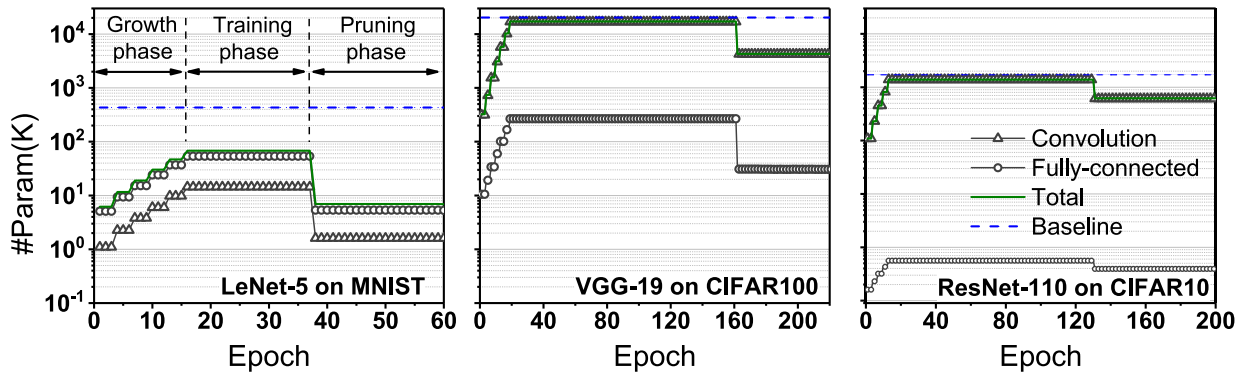


Fig. 5. Number of parameters during training, plotted at the end of each epoch. In the beginning, the model size increases gradually due to the growth. After the growth ends and several epochs of training on the peak model, one drop can be observed after the first pruning. There are several iterations of pruning at a frequency of 1.

TABLE I
EVALUATION OF THE PERFORMANCE ON MNIST

Method	Accuracy	FLOPs	Pruned	Param.	Pruned
LeNet5-Baseline	99.29	4.59M	–	431K	–
Pruning [17]	99.26	0.85M	81.5%	112K	74.0%
Pruning [15]	99.23	0.73M	84.0%	36K	92.0%
CGaP	**99.36**	**0.44M**	**90.4%**	**8K**	**98.1%**

TABLE II
EVALUATION OF THE PERFORMANCE ON CIFAR-100. 'NA' MEANS 'NOT AVAILABLE' IN THE ORIGINAL PAPER

Method	Accuracy	FLOPs	Pruned	Param.	Pruned
VGG19-Baseline	72.63	797M	–	20.4M	–
Pruning [28]	71.85	NA	–	10.1M	50.5%
Pruning [18]	72.85	501M	37.1%	5.0M	75.5%
CGaP	**73.00**	**373M**	**53.2%**	**4.3M**	**78.9%**

TABLE III
EVALUATION OF THE PERFORMANCE ON SVHN

Method	Accuracy	FLOPs	Pruned	Param.	Pruned
VGG19-Baseline	96.02	797M	–	20.4M	–
Pruning [18]	96.13	398M	50.1%	3.1M	84.8%
CGaP	**96.25**	**206M**	**74.2%**	**2.9M**	**85.8%**

GeForce GTX 1080 Ti platform. It is worth mentioning that experiments performed with different frameworks may have variation in accuracy and performance. Thus, to have a fair comparison among CGaP, baseline and pruning-only methods, all the results in Table I, II, III and IV are obtained from experiments with PyTorch framework.

B. Performance Evaluation

With training setup as aforementioned, we perform experiments on several datasets with modern DNN architectures. In Table I, Table II, Table III and Table IV, we summarize the performance attained by CGaP on MNIST, CIFAR-100, SVHN, and CIFAR-10 datasets, respectively. To be specific, the second column 'Accuracy' denotes the inference accuracy in percentage achieved by the baseline model, the up-to-date pruning-only approaches and CGaP approach, respectively.

TABLE IV
EVALUATION OF THE PERFORMANCE ON CIFAR-10

Method	Accuracy	FLOPs	Pruned	Param.	Pruned
VGG16-Baseline	93.25	630M	–	15.3M	–
Pruning [16]	93.40	410M	34.9%	5.4M	64.7%
CGaP	**93.59**	**280M**	**56.2%**	**4.5M**	**70.6%**
ResNet-56-Baseline	93.03	268M	–	0.85M	–
Pruning [28]	92.56	182M	32.1%	0.73M	14.1%
Pruning [43]	90.20	**134M**	**50.0%**	NA	-
CGaP	**93.20**	181M	32.5%	**0.53M**	**37.6%**
ResNet-110-Baseline	93.34	523M	–	1.72M	–
Pruning [16]	93.11	310M	40.7%	1.16M	32.6%
Pruning [29]	**93.52**	300M	40.8%	NA	–
CGaP	93.43	**192M**	**63.3%**	**0.62M**	**64.0%**

The column 'FLOPs' represent the calculated number of FLOPs of a single inference pass. The calculation of FLOPs follows the method described in [42]. Fewer FLOPs means lower computation cost in one inference pass. The neighboring column, 'Pruned', represents the reduction of FLOPs in the compressed model as compared to the baseline model. The column 'Param.' stands for the number of parameters of the inference model. Fewer parameters promise a smaller model size. The last column, 'Pruned', denotes the percentage pruned in parameters compared to the baseline. Larger pruned percentage implies fewer computation operations and more compact model. The best result of each column is highlighted in bold.

The results shown in Table I to IV prove that CGaP outperforms the previous pruning-only approaches in accuracy and model size. For instance, as displayed in Table IV, on ResNet-56, our CGaP approach achieves 93.20% accuracy with 32.5% reduction in FLOPs and 37.6% reduction in parameters, while the up-to-date pruning-only method [28] that deals with static structure only reaches 92.56% accuracy with 32.1% reduction in FLOPs and 14.1% reduction in parameters. On ResNet-110, though [29] achieves 0.09% higher accuracy than CGaP, CGaP overwhelms it by trimming 22.5% more FLOPs.

C. Visualization of the Dynamic Structures

Fig. 5 presents the dynamic model size during CGaP training. During the growth phase, the model size continuously

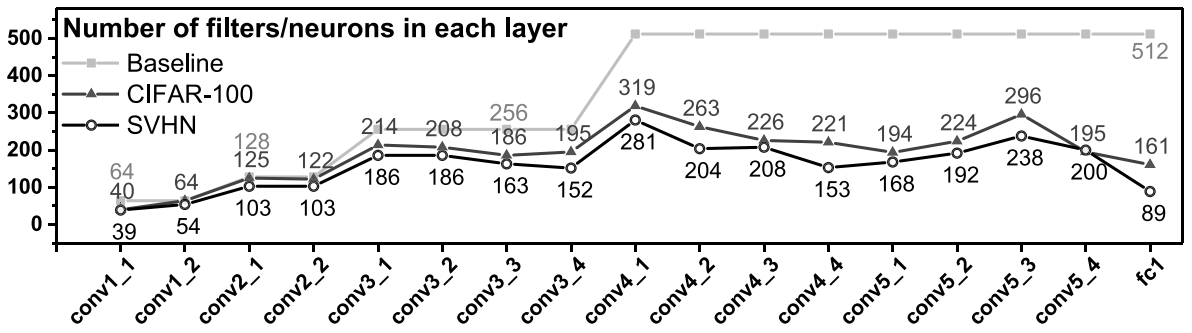


Fig. 6. The VGG-19 structures learned by CGaP on CIFAR-100 and SVHN datasets. The shared Y-axis for three sub-./ is the number of parameters of the model.

increases and reaches a peak capacity. When the pruning phase starts, the model size drops.

Furthermore, the sparsity achieved by CGaP is structured. In other words, large amounts of filters and neurons are entirely pruned. For instance, the baseline LeNet-5 without sparsity regularization has 20, 50 filters in conv-layer 1 and conv-layer 2, 500 and 10 neurons in fc-layer 1 and fc-layer 2, denoted as [20-50-500-10] (number of filters/neurons in [conv1-conv2-fc1-fc2]). The model achieved by CGaP contains only 8, 17 filters and 23, 10 neurons, denoted as [8-17-23-10]. Compared to baseline results, CGaP significantly decreases 60%, 66%, 95.4% units for each layer (the output layer should remain the same as the number of classes all the time). In this case, the pruned filters and neurons are skipped in the inference pass and thus accelerating the computation pipeline on hardware.

Another example is provided in Fig. 6, which visualizes the VGG-19 structures from CGaP as well as the baseline structure on two different tasks. In the baseline model, the width (number of filters/neurons) of each layer is abundant, from 64 filters (the bottom conv-layers) to 512 filters (the top conv-layers). The baseline VGG-19 structure is designed to have a large enough size in order to guarantee the learning capacity. However, it turns out to be redundant, as proved by the structure that CGaP generated: 37.7% to 82.6% filters are pruned out in each layer. Meanwhile, in the baseline model, the top conv-layers are designed to have more filters than the bottom layers, but CGaP shows that it is not always necessary for top layers to have a relatively large size.

D. Validating the Saliency-Based Growth

Fig. 7 validates the efficacy of our saliency-based growth policy. Selective growth, which emphasizes the important units according to the saliency score, has lower cross-entropy loss than randomly growing some units. The spiking in Fig. 7 is caused by the first iteration of pruning and this loss is recovered by the following iterative fine-tuning. In selective growth, this loss is 1.4× lower than that in random growth. This phenomenon supports our argument that selective growth assists the pruning phase. The detailed understanding of growth will be further discussed in Section VII.

To summarize the results from the algorithm simulations, the proposed CGaP approach:

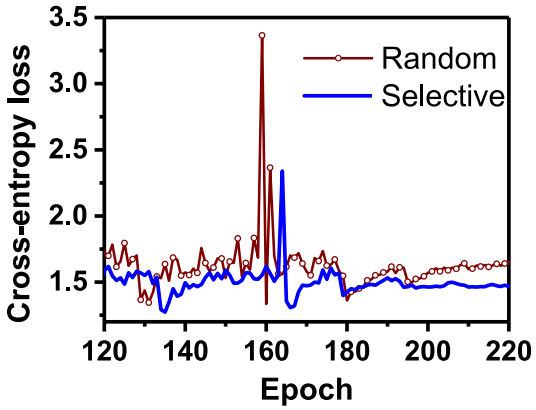


Fig. 7. Saliency-based growth outperforms random growth. The loss is monotonically decreasing from epoch 0 to 220 with small glitches. Here we zoomed in from epoch 120 to 220 to show the loss at the end of the training.

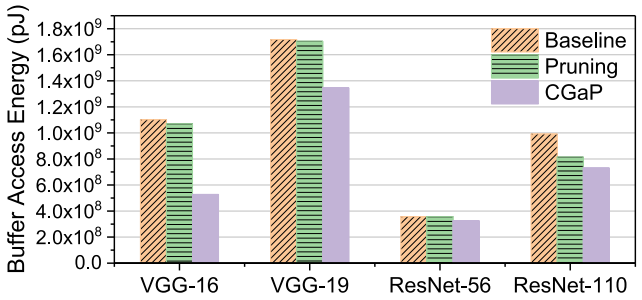
- Largely compresses the model size by 37.6% (ResNet-56) to 98.1% (LeNet-5) for representative DNN structures.
- Decreases the inference cost, to be specific, number of FLOPs, by 32.5% (ResNet-56) to 90.4% (LeNet-5) on various datasets.
- Does not sacrifice accuracy and even improves accuracy.
- Outperforms the state-of-the-art pruning-only methods that deal with fixed structures.

VI. Experiments on FPGA Simulator

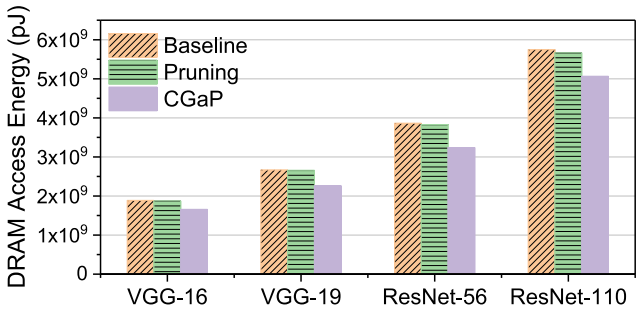
The results above demonstrate that CGaP generates an accurate and small inference model. In this section, we further evaluate the on-chip inference cost of the generated models and compare CGaP with previous non-structured pruning [15]. As CGaP achieves structured sparsity, CGaP outperforms the previous work on non-structured pruning in hardware acceleration and power efficiency. We validate this by performing the estimation of buffer access energy, DRAM access energy and latency using the performance model for FPGA [5].

A. Overview of the FPGA Simulator

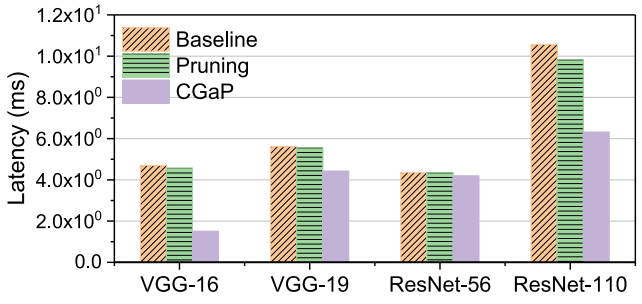
[5] is a high-level performance model designed to estimate the number of external and on-chip memory access, as well as the latency. The resource costs are formulated by the acceleration strategy as well as the design variables that control the loop tiling and unrolling. The performance model has



(a) Comparison of three schemes in buffer access energy (pJ) for VGG-16 on CIFAR-10, VGG-19 on CIFAR-100, ResNet-56 and ResNet-110 on CIFAR-10.



(b) Comparison of three schemes in DRAM access energy (pJ) for VGG-16 on CIFAR-10, VGG-19 on CIFAR-100, ResNet-56 and ResNet-110 on CIFAR-10.



(c) Comparison of three schemes in on-chip inference latency (ms) for VGG-16 on CIFAR-10, VGG-19 on CIFAR-100, ResNet-56 and ResNet-110 on CIFAR-10.

Fig. 8. Estimation on FPGA performance model.

been validated across several modern DNN algorithms in comparison to on-board testings on two FPGAs, with the differences within 3% [5].

In the following experiments, the setup follows: the pixels and weights are both 16-bit fixed point, the data width of DRAM controller is 512 bits, the accelerator operating frequency is 300 MHz, and the DRAM bandwidth is 19.2 GB/second. The parameters related to loop tiling and unrolling follow the setting in [5].

B. Results From FPGA Performance Model

The on-chip and external memory access energy across VGG-16, VGG-19, ResNet-56 and ResNet-110 is displayed in Fig. 8(a) and Fig. 8(b), respectively. The inference latency is shown in Fig. 8(c). Though the models generated from weight magnitude pruning and CGaP have the same sparsity, CGaP outperforms non-structured magnitude weight pruning in hardware efficiency and acceleration. For example, with the same setup of the pruning ratio during training, magnitude weight pruning decreases 1.0% on-chip access energy, 1.0% DRAM access energy and 0.8% latency for VGG-19 on CIFAR-100, while the CGaP achieves 21.6%, 15%, and 21.1% reduction. The non-structured weight pruning [15] is able to improve the power and latency efficiency in comparison to baseline. However, the improvement is limited. In contrast, CGaP achieves significant acceleration and energy reduction. The reason is that the non-structured sparsity, i.e., scattered weight distribution, leads to irregular memory access that weakens the acceleration on hardware in a real scenario.

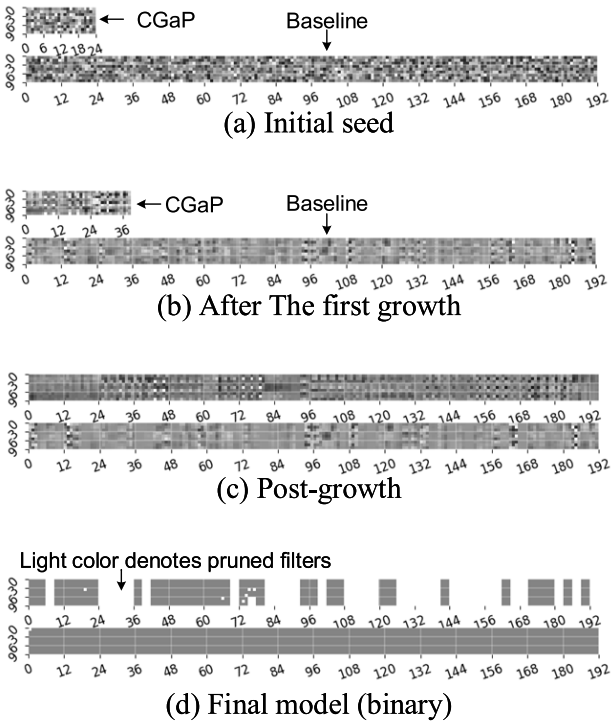


Fig. 9. Visualization of the filters in conv1_1 in VGG-19 on CIFAR-100 at four specific moments (a-d). Inside each figure, the top bar is CGaP model and the bottom bar is baseline model. X-axis is the index of output-wise weights and Y-axis is the index of input-wise weights.

VII. Discussion

In Section V and VI, the performance of CGaP has been comprehensively evaluated on algorithm platforms and hardware platforms. In this section, we provide a more in-depth understanding of the growth to explain why selective growth is able to improve the performance from the traditional pipelines. Furthermore, we provide a thorough ablation study to validate the robustness of the proposed CGaP method.

A. Understanding the Growth

Fig. 9 illustrates a visualization of the weights in the bottom conv-layer (conv1_1) in VGG-19, at the moment of initialization, after the first growth, after the last growth and when training ends. Inside each figure, the upper bar is the CGaP model, whose size varies at different training moments. The lower bar is from the baseline model, whose size is static during training. At the initialization moment (Fig. 9(a)), CGaP

TABLE V
THE IMPACT OF VARIOUS STRUCTURES AND SIZES OF THE INITIAL SEED OF VGG-19

Initial seeds		**'2'**	**'4'**	**'6'**	**'8'**	**'10'**	**'12'**
#filters	conv1_n	2	4	6	8	10	12
	conv2_n	4	8	12	16	20	24
	conv3_n	8	16	24	32	40	48
	conv4_n	16	32	48	64	80	96
	conv5_n	16	32	48	64	80	96
#param	Initial (M)	0.01	0.06	0.13	0.23	0.36	0.53
Testing accuracy*		-0.69%	-0.2%	-0.16%	+0.37%	+0.04%	0.29%
*Relative accuracy of the final VGG-19 model on CIFAR-100 as compared to the baseline.							

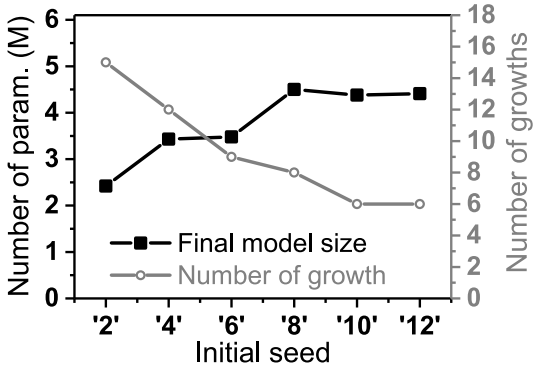


Fig. 10. A larger seed leads to a larger final model but fewer iterations in the growth phase.

model only has 8 filters in this layer while the baseline model has 64 filters. Then the number of filters grows to 13 after one iteration growth (Fig. 9(b)), meaning the most important 5 filters are selected and added. It is clear that the pattern in Fig. 9(b) is more active than that in (a), indicating the filters have already fetched effective features from the input images. More important, along with the growing, the pattern in CGaP model becomes more structured than that in the baseline model, as shown in Fig. 9(c). Benefiting from this well-structured pattern, our CGaP model has higher learning accuracy than the baseline model. From Fig. 9(c) to Fig. 9(d), relatively unimportant filters are removed, and important ones are kept. We observe that most of the filters that are favored by the growth, such as filters at index 36, 48, 72, 96 in Fig. 9(c), are still labeled as important filters in Fig. 9(d) even after a long training process between the growth phase and the pruning phase. Leveraging the growth policy, the model is able to recover quickly from the loss caused by pruning (the spiking in Fig. 7).

B. Robustness of the Seed

The performance of CGaP is stable under the variation of the initial seeds. To prove this, we scan several seeds in different size and present the variation in accuracy and inference model size. The structure of 6 scanned seeds is listed in Table V. Each seed has a different number of filters in each layer, e.g., seed '2' has 2 filters in block conv1. The size of the seeds varies from 0.01M to 0.53M. Fig. 10 presents the final model size and the number of growth of each seed. A larger seed leads to a larger final model but requires fewer iterations of growth to reach the intended model size. Generally speaking, there is a trade-off between the inference accuracy and the model size. Though the seed varies a lot from each other, the final accuracy is quite robust, as listed in the 'Accuracy' row in Table V. It is worth mentioning that, even though the seed '2' degrades the accuracy of 0.69% from baseline, the inference model size is only 2.4M, significantly smaller than the baseline size (20.4M).

C. Robustness of the Hyper-Parameters

CGaP is conditioned on a set of hyper-parameters to achieve optimal performance, while it is stable under the variation of these hyper-parameters. Empirically, we leverage the following experience to perform parameter optimization: a smaller growth rate β for a larger seed and vice versa; threshold τ_{capa} is set based on the user's intended model size; a smaller f_{growth} for a complicated dataset and vice versa; a relatively greedy growth (larger β and f_{growth}) prefers a larger noise μ but smaller σ to push the model away from sticking at a local minimum. Tuning of the pruning ratio of each layer is in a similar manner to that of the other pruning works [15] [16].

In particular, we scan 121 combinations of the scaling factor σ and noise μ in the range [0.0, 1.0] with the step=0.1 and provide the following discussion. For VGG16 on CIFAR-10, the accuracy of several corner cases are 90% (μ=1, σ=0, which is a case of random initialization), 89% (μ=1, σ=1), 84% (μ=0, σ=1, which is another case of mimicking its neighbor without scaling) and 10% (μ=0, σ=0, the training is invalid in this case), 93% (μ=0, σ=0, which is another case of mimicking its neighbor with scaling), 88% (μ=0.5, σ=0, which is another case of random initialization). The best accuracy of 93.6% is under μ=0.1, σ=0.5. The combinations in the zone that $\mu \in [0, 0.5]$ and $\sigma \in (0, 0.5]$ always provide >92% accuracy. To summarize, σ impacts more than μ as μ is relatively small; σ should not be too large and 0.5 is safe for future tasks and networks; adding a noise improves the accuracy (like from 93% to 93.6%) as it prevents local minimum; inheriting from the neighbor is more efficient than randomly initializing since the network is able to resume the learning right after the growth.

VIII. CONCLUSION AND FUTURE WORK

Modern DNNs typically start training from a fixed and over-parameterized network, which leads to redundancy and

is lack of structural plasticity. We propose a novel dynamic training algorithm, Continuous Growth and Pruning, that initializes training from a small network, expands the network width continuously to learn important learning units and structures and finally prunes secondary ones. The effectiveness of CGaP depends on where to start and stop the growth, which learning unit (filter and neuron) should be added, and how to initialize the newborn units to ensure model convergence. Our experiments on benchmark datasets and architectures demonstrate the advantage of CGaP on learning efficiency (accurate and compact). We further validate the energy and latency efficiency of the inference model generated by CGaP on FPGA performance simulator. Our approach and analysis will help shed light on the development of adaptive neural networks for dynamic tasks such as continual and lifelong learning.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[2] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 91–99.

[3] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2013, pp. 6645–6649.

[4] P. Zhang, Y. Goyal, D. Summers-Stay, D. Batra, and D. Parikh, "Yin and Yang: Balancing and answering binary visual questions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 5014–5022.

[5] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, "Performance modeling for CNN inference accelerators on FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, to be published.

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: https://arxiv.org/abs/1409.1556

[7] Y. Chen *et al.*, "DaDianNao: A machine-learning supercomputer," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchit.*, Dec. 2014, pp. 609–622.

[8] Z. Du *et al.*, "ShiDiaNnao: Shifting vision processing closer to the sensor," *ACM Sigarch Comput. Archit. News*, vol. 43, no. 3, pp. 92–104, 2015.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[10] Y. H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[11] K. Guo *et al.*, "Angel-Eye: A complete design flow for mapping CNN onto embedded FPGA," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 37, no. 1, pp. 35–47, Jan. 2018.

[12] A. Shafiee *et al.*, "ISAAC: A convolutional neural network accelerator with *in-situ* analog arithmetic in crossbars," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 14–26, 2016.

[13] J. Qiu *et al.*, "Going deeper with embedded FPGA platform for convolutional neural network," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, 2016, pp. 26–35.

[14] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun, "CNP: An FPGA-based processor for convolutional networks," in *Proc. Int. Conf. Field Program. Logic Appl.*, Aug./Sep. 2009, pp. 32–37.

[15] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.

[16] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," 2016, *arXiv:1608.08710*. [Online]. Available: https://arxiv.org/abs/1608.08710

[17] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," 2016, *arXiv:1607.03250*. [Online]. Available: https://arxiv.org/abs/1607.03250

[18] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proc. IEEE Int. Conf. Comput. Vis.*, Oct. 2017, pp. 2755–2763.

[19] J.-H. Luo, J. Wu, and W. Lin, "ThiNet: A filter level pruning method for deep neural network compression," in *Proc. IEEE Int. Conf. Comput. Vis.*, Oct. 2017, pp. 5058–5066.

[20] V. Lebedev and V. Lempitsky, "Fast convnets using group-wise brain damage," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 2554–2564.

[21] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2074–2082.

[22] J. H. Gilmore *et al.*, "Regional gray matter growth, sexual dimorphism, and cerebral asymmetry in the neonatal brain," *J. Neurosci.*, vol. 27, no. 6, pp. 1255–1260, 2007.

[23] S. J. Lipina and J. A. Colombo, *Poverty and Brain Development During Childhood: An Approach From Cognitive Psychology and Neuroscience*. Washington, DC, USA: American Psychological Association, 2009.

[24] M. Butz and A. van Ooyen, "A simple rule for dendritic spine and axonal bouton formation can account for cortical reorganization after focal retinal lesions," *PLoS Comput. Biol.*, vol. 9, no. 10, 2013, Art. no. e1003259.

[25] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009, vol. 1, no. 4, p. 7.

[26] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proc. NIPS Workshop Deep Learn. Unsupervised Feature Learn.*, 2011.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.

[28] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," 2018, *arXiv:1810.05270*. [Online]. Available: https://arxiv.org/abs/1810.05270

[29] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," 2018, *arXiv:1808.06866*. [Online]. Available: https://arxiv.org/abs/1808.06866

[30] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Architecture (ISCA)*, Jun. 2016, pp. 243–254.

[31] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky, "Sparse convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 806–814.

[32] T. Ash, "Dynamic node creation in backpropagation networks," *Connection Sci.*, vol. 1, no. 4, pp. 365–375, Jan. 1989.

[33] B. J. Briedis and T. D. Gedeon, "Using the grow-and-prune network to solve problems of large dimensionality," in *Proc. Austral. Conf. Neural Netw.*, Brisbane, QLD, Australia, 1998.

[34] A. Sakar and R. J. Mammone, "Growing and pruning neural tree networks," *IEEE Trans. Comput.*, vol. 42, no. 3, pp. 291–299, Mar. 1993.

[35] G.-B. Huang, P. Saratchandran, and N. Sundararajan, "A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation," *IEEE Trans. Neural Netw.*, vol. 16, no. 1, pp. 57–67, Jan. 2005.

[36] S. Hussain and A. Basu, "Multiclass classification by adaptive network of dendritic neurons with binary synapses using structural plasticity," *Frontiers Neurosci.*, vol. 10, p. 113, Mar. 2016.

[37] X. Dai, H. Yin, and N. K. Jha, "NeST: A neural network synthesis tool based on a grow-and-prune paradigm," 2017, *arXiv:1711.02017*. [Online]. Available: https://arxiv.org/abs/1711.02017

[38] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," 2014, *arXiv:1412.6115*. [Online]. Available: https://arxiv.org/abs/1412.6115

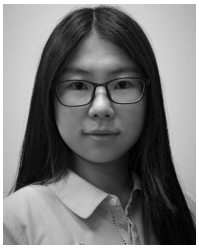
[39] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6869–6898, 2017.

[40] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 1269–1277.

[41] C. Leng, Z. Dou, H. Li, S. Zhu, and R. Jin, "Extremely low bit neuralnetwork: Squeeze the last bit out with ADMM," in *Proc. AAAI Conf. Artif. Intell.*, Aug. 2019.

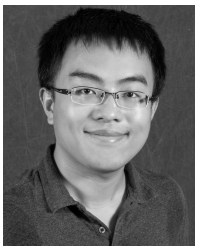
[42] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," 2016, *arXiv:1611.06440*. [Online]. Available: https://arxiv.org/abs/1611.06440

[43] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "AMC: AutoML for model compression and acceleration on mobile devices," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 784–800.
[44] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. 27th Int. Conf. Mach. Learn. (ICML)*, 2010, pp. 807–814.
[45] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015, *arXiv:1502.03167*. [Online]. Available: https://arxiv.org/abs/1502.03167
[46] A. Paszke *et al.*, "Automatic differentiation in pytorch," in *Proc. NIPS Workshop Autodiff*, 2017.



Xiaocong Du (S'19) received the B.S. degree in control engineering from Shandong University, Jinan, China, in 2014, and the M.S. degree in electrical and computer engineering from the University of Pittsburgh, Pittsburgh, PA, USA, in 2016. She is currently pursuing the Ph.D. degree in electrical engineering with Arizona State University, Tempe, AZ, USA. Her research interests include efficient algorithm and hardware co-design for deep learning, neural architecture search, continual learning, and neuromorphic computing.

Zheng Li (S'19) received the B.S. degree in electronics and information engineering from Beihang University, Beijing, China, in 2014, and the M.S. degree in electrical and computer engineering from the University of Pittsburgh, Pittsburgh, PA, USA, in 2017. He is currently pursuing the Ph.D. degree in computer engineering with Arizona State University, Tempe, AZ, USA. He was a Summer Intern in machine learning with MobaiTech, Inc., Tempe, AZ, USA, in 2018. His current research interests include algorithm design and optimization for computer vision tasks such as object detection and autonomous driving.



Yufei Ma (S'16–M'19) received the B.S. degree in information engineering from the Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2011, the M.S.E. degree in electrical engineering from the University of Pennsylvania, Philadelphia, PA, USA, in 2013, and the Ph.D. degree from Arizona State University, Tempe, AZ, USA, in 2018.

His current research interests include the high-performance hardware acceleration of deep learning algorithms on digital application-specified integrated circuit and field-programmable gate array.

Yu Cao (S'99–M'02–SM'09–F'17) received the B.S. degree in physics from Peking University in 1996, and the M.A. degree in biophysics and the Ph.D. degree in electrical engineering from the University of California, Berkeley, in 1999 and 2002, respectively.

He was a Summer Intern with Hewlett-Packard Labs, Palo Alto, CA, USA, in 2000, and the IBM Microelectronics Division, East Fishkill, NY, USA, in 2001. After working as a Post-Doctoral Researcher with the Berkeley Wireless Research Center (BWRC), Arizona State University, Tempe, AZ, USA, where he is currently a Professor of electrical engineering. He has published numerous articles and two books on nano-CMOS modeling and physical design. His research interests include physical modeling of nanoscale technologies, design solutions for variability and reliability, reliable integration of post-silicon technologies, and hardware design for on-chip learning.

Dr. Cao has served on the technical program committee of many conferences. He was a recipient of the 2000 Beatrice Winner Award from the International Solid-State Circuits Conference, the 2004 Best Paper Award from the International Symposium on Quality Electronic Design, the 2006 and 2007 IBM Faculty Award, the 2006 NSF CAREER Award, the 2007 Best Paper Award from the International Symposium on Low Power Electronics and Design, the 2008 Chunhui Award for outstanding oversea Chinese scholars, the 2009 ACM SIGDA Outstanding New Faculty Award, the 2009 Promotion and Tenure Faculty Exemplar, Arizona State University, the 2009 Distinguished Lecturer of the IEEE Circuits and Systems Society, the 2010, 2012, 2013, 2015, and 2016 Top 5% Teaching Award from the School of Engineering, Arizona State University, and the 2012 Best Paper Award from the IEEE Computer Society Annual Symposium on VLSI. He has served as Associate Editor for the IEEE TRANSACTIONS on Computer-Aided Design.