LRP: Predictive output activation based on SVD approach for CNNs acceleration

Xinxin Wu[†‡], Zhihua Fan[†‡], Tianyu Liu[†‡], Wenming Li[†], Xiaochun Ye[*†] , Dongrui Fan[†‡]
[†]*State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences*
[‡]*School of Computer and Control Engineering, University of Chinese Academy of Sciences*
{wuxinxin, fanzhihua, liutianyu, liwenming, yexiaochun, fandr}@ict.ac.cn

***Abstract*—Convolutional Neural Networks (CNNs) achieve state-of-the-art performance in a wide range of applications. CNNs contain millions of parameters, and a large number of computations challenge hardware design. In this paper, we take advantage of the output activation sparsity of CNNs to reduce the execution time and energy consumption of the network. We propose Low Rank Prediction (LRP), an effective prediction method that leverages the output activation sparsity. LRP first predicts the output activation polarity of the convolutional layer based on the singular value decomposition (SVD) approach of the convolution kernel. And then it uses the predicted negative value to skip invalid computation in the original convolution. In addition, an effective accelerator, LRPPU, is proposed to take advantage of sparsity to achieve network inference acceleration. Experiments show that our LRPPU achieves 1.48× speedup and 2.02× energy reduction compared with dense networks with slight loss of accuracy. Also, it achieves on average 2.57× speedup over Eyeriss and has similar performance and less accuracy loss compared with SnaPEA.**

***Index Terms*—Convolutional Neural Networks, Output activation, SVD approach, Prediction, Sparsity.**

I. Introduction

CNNs have proven human-level accuracy in many tasks and are increasingly used in various applications, including speech recognition, natural language processing, image processing. Parameters of CNNs tend to be designed to be more and more complicated in order to obtain better performance. Huge computing and storage put higher demands on hardware resources, which challenges the limited hardware resources. It is necessary to reduce the high computational cost to deploy CNNs in a wider range of applications, especially in embedded and mobile environments, without sacrificing inference accuracy significantly.

Exploring sparsity in CNNs is a key technique to reduce the high computational cost. In convolutional layers of many modern CNNs, each convolution operation is commonly followed by an activation function called a Rectifying Linear Unit (ReLU) that returns zero for negative inputs and yields the input itself for the positive ones, which leads to a large amount of zeros in the activations. The activation sparsity can be classified into input activation sparsity and output activation sparsity. Compared with the sparsity of input activation, the sparsity of output activation can only be known after the calculation of the current layer, which makes its sparsity difficult to use.

[*] Xiaochun Ye is the corresponding author.

In this paper, we propose Low Rank Prediction (LRP), an effective method to predict the sparsity of output activation in CNNs. The main idea of LRP is running a convolution operation based on SVD approach to predict the binary sparse mask of output activation (OA), and using this binary sparse mask to manage the original convolution operation. We define ineffectual output activation (iOA) as those zero value output activations that have no influence on the convolution operation of the next layer. The main contributions of this paper are:

- We predict the OA sparsity of the convolutional neural networks based on the SVD approach of the convolution kernel. And we fine-tune offline the network based on the SVD to maintain the accuracy of the prediction.
- We propose the LRPPU accelerator, adding predictors and detection units to take advantage of OA sparsity, thereby eliminating some invalid computations.
- We evaluate our system using various popular CNN models over the ImageNet ILSVRC-2012 datasets. Experimental results show that, compared with Eyeriss accelerator, the proposed approach can significantly improve performance with a marginal accuracy loss, and enables a trade-off between performance and accuracy.

II. Related work

Various prediction methods and sparse optimization techniques have been proposed to improve the performance and energy efficiency of the neural network. PredictiveNet [1], and Prediction Based [2] first execute the most significant bits to predict zeros. However, less information is present in significant bits if batch normalization is used. At the same time, serial multiplication unit is specifically needed to support different data bit widths of the convolutional layer. SnaPEA [3] uses a combination of static re-ordering weights and sign check to terminate the computation in advance. The predictive activation unit detects that the partial sum is negative and terminates the remaining calculation. However, it disrupts the original execution order and requires a weight index buffer to fetch the corresponding input value for each weight to ensure the accuracy of the result, which is not applicable to statically mapped dataflow. ComPEND [4] also proposes a computation pruning technique that detects whether the result of a sum of products is negative by adopting an inverted two's complement expression for weights and a bit-serial sum of products. It can skip a large amount of computations for negative results. But it needs to modify the computing unit to support the bit-serial approach. SparseNN [5] and LRADNN [6] dynamically predict OA sparsity of small-scale network before actually calculating the current layer by using end to end training and SVD truncation approach respectively, and then use output sparsity to bypass zero OA in order to improve energy efficiency. However, they only apply to small-scale networks (several simple networks containing hidden layers), and do not conduct medium to large convolutional neural network experiments. In addition, the microstructures they designed are only suitable for the matrix multiplication operation, and cannot perform the convolution operation.

III. Background and prediction mechanism

A. Convolutional Neural Network

CNN is mainly composed of multiple convolutional layers, which occupy about most of the calculation time of the entire network processing. The convolutional layer applies a filter on the input activation (IA) to generate an output activation (OA). In general, each convolutional layer will be followed by a non-liner activation function in order to enable CNN to learn complex knowledge. ReLU is one of the popular activation functions. Table I shows the parameter description of the convolutional layer operation. The calculation of the convolutional layer is shown in formula 1:

$$\begin{gathered} A^{l+1}[z][u][x][y] = ReLU(Z^{l+1}[z][u][x][y]) = \\ ReLU(\sum_{k=0}^{C-1}\sum_{i=0}^{S-1}\sum_{j=0}^{R-1} A^{l}[z][k][Ux+i][Uy+j] \times W^{l}[u][k][i][j]). \\ 0 \le z \le N, 0 \le u \le M, 0 \le x \le Q, 0 \le y \le P \end{gathered} \quad (1)$$

Where $A^{l+1} \in R^{NMPQ}$, $A^{l} \in R^{NCXY}$,$W^{l} \in R^{MCRS}$ is OA, IA, Filter, respectively. At the same time, A^{l+1} is the activation output of the l+1 layer, A^{l} is the activation input of the l+1 layer, and W^{l} represents the filter/weight/kernel matrix between l layer and l+1 layer.

TABLE I
Convolutional layer parameter description.

Para	Description	Para	Description
N	Batch size of 3D IA	U	Stride size
C	IA/Filter channels	X/Y	IA height/width
R/S	Filter height/width	P/Q	OA height/width
M	Filter numbers/OA channels	-	-

B. Output activation sparsity

The ReLU unit causes the OA and IA of the convolutional layer to have high sparsity. The sparsity of the IA is only known after the calculation of the previous layer is completed, therefore it is known when the current layer is calculated. As for the OA, the sparsity is only known after the calculation of the current layer is completed, which indicates that the sparsity is unknown before the calculation of the current layer is completed. The sparsity of IA is easily detected by non-zero detection units based on the sparsity characteristics, but for OA, it cannot be detected before calculation. We count the sparsity of OA of each convolutional layer of two popular CNN networks, Alexnet and VGG16. The average OA sparsity rates of Alexnet and VGG16 are 72.23% and 60.39%, respectively. If these unnecessary computation related to the iOA can be skipped, it can achieve a high performance. This motivates us to use effective prediction mechanisms and design corresponding structures to make use of the sparsity of the OA of convolutional neural networks.

C. Prediction mechanism

The use of the sparsity of OA requires a prediction mechanism. Inspired by the LRA [7] prediction mechanism, we use the decomposed low rank approximation of the weight matrices as the predictor. Based on the theory and proof of literature [8], we realize an efficient low rank prediction (LRP) way for computing the exact decomposition, which is extremely fast as they have small convolutional kernels and it directly provides the global solution. Specifically, $W = UDQ^{T}$ is the Singular Value Decomposition (SVD) of W. The approximate value of W is expressed as follows:

$$W = UDQ^{T} \approx \tilde{W} = \underbrace{U\sqrt{D_{K,K}}}_{H}\underbrace{\sqrt{D_{K,K}}Q^{T}}_{V^{T}} = HV^{T}. \quad (2)$$

where $H \in R^{M\times1\times S\times K}, V \in R^{K\times R\times1\times C}$ is the horizontal filter and vertical filter, respectively. And K is the rank of filter, giving K singular values of W. Part of the largest singular values can account for most of the matrix feature. We use this method to obtain HV pairs and use them as predictors to eliminate the polarity (the sign of a value) of the OA. Fig. 1 shows the decomposition method (up) and overall flow of the low rank sparsity prediction and sparse convolution for a convolutional layer (down). Convolution execution process with the prediction stage is summarized as follows.

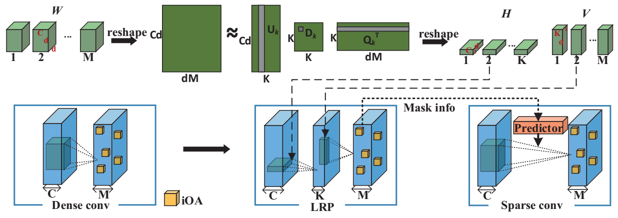


Fig. 1. Decomposition method (up) and overall flow of dense convolution and sparse convolution with LRP (down).

First, the values of the HV pair for each layer are obtained as in equation 2. The HV pair needs to be fine-tuned offline to reduce the loss of accuracy. Based on H and V, we transform the original convolution process in one layer into a two-step simple calculation process to generate the sparsity mask information as a prediction. This process is called low rank prediction. Second, the mask information which predict the value of OA is sent to the predictor. With the predicted information, some of the multiplications involved in the sparse convolution operation over W and IA are disabled. If the polarity of OA is negative, the related computations is bypassed. In short, the prediction method can reduce unnecessary computation, which provides the possibility to speedup the network.

In theory, the direct convolution by definition of equation 1 requires $O(RSMCPQ)$ operations. Based on the above scheme, the computational cost associated with the vertical filters is $O(RKCPQ)$ and with horizontal filters $O(SMKPQ)$, giving a total computational cost of $O((RC+SM)KPQ + RSMCPQ \times (1\text{-}\alpha))$, supposing the sparsity of OA is α. So acceleration can be achieved when we choose $K < \alpha RMC/(M+C)$(where $R=S$). The choice of K is a trade-off between speedup and accuracy.

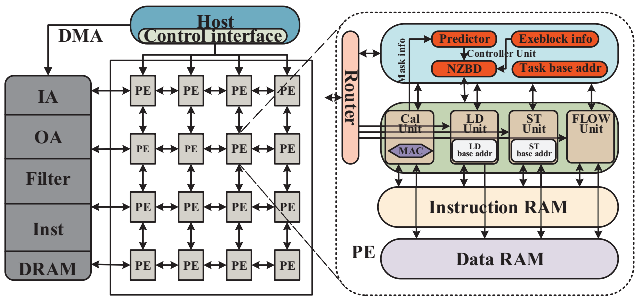


Fig. 2. LRPPU architecture overview.

IV. Accelerator Overview

In this paper, LRPPU is proposed to accelerate the inference phase of the CNNs by making use of the prediction mechanism which is mentioned above. LRPPU adopts Risc-NN [9] hardware infrastructure as the basic microarchitecture. It has the following distinct features apart from Risc-NN: (1) A sparsity predictor is added to support the prediction function, and the non-zero *exeblocks* (execution block) detection unit (NZBD) is used to skip invalid *exeblocks*. (2) The iterative pipeline execution method of *exeblocks* is implemented to ensure the busyness and load balancing of the PE array.

A. PE architecture with output sparsity bypass

The overall architecture of the LRPPU accelerator, as shown in Fig. 2, includes a host with control interface, DRAM with memory control, PE array and NoC. The PE array is connected through a 2D mesh network, and each PE can exchange data with its neighbors. Based on the execution model of Risc-NN, an application consists of a sequence of consecutive *Task*s, and each *Task* consists of multiple execution blocks (*exeblocks*) which are organized in the dataflow manner before being mapped into the PE array. Each *exeblock* consists of up to four consecutive execution stages, namely *load (ld)*, *calculation (cal)*, *flow*, *store (st)* stage. In the beginning of an application, the host uploads the *exeblocks* and data onto the DRAM through DMA, and then sends the control information to initialize PEs through a control interface. Before running each PE, PE needs to load all required *exeblocks* in the DRAM into the instruction RAM by the router. After that, the PE array starts to execute until all *exeblocks* are implemented.

Inside each PE, there are the controller, execution unit, data and instruction RAM module. The four execution units execute the instructions of the different stages of *exeblocks* respectively. The data and instruction RAM store the data and instructions of *exeblocks*. The controller schedules the execution of *exeblocks* according to the *exeblocks* information. Each *exeblock* can be executed multiple times, and each execution is called an iteration. After one iteration is executed, the exeblock info is updated with the information of the next iteration. Based on this, we add a predictor and a non-zero *exeblocks* detection unit (NZBD) to realize the prediction function. The predictor is used to generate OA's flag information in the prediction phase. And the non-zero *exeblocks* detection unit is used to filter *exeblocks* related to iOA in the execution stage. The workflow of the predictor and NZBD unit is shown in Fig. 3. The predictor saves the flag information for OA according to the mask value of calculation. The NZBD unit takes the flag information of the OA and the *exeblock* selected by scheduler as input, and then selects a valid *exeblock*.

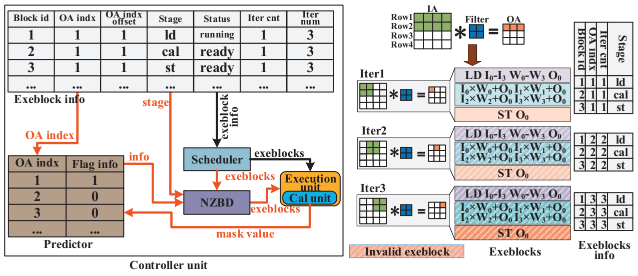
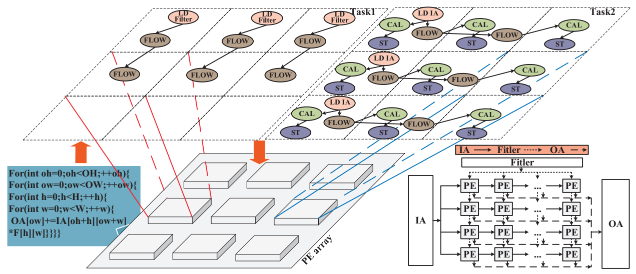


Fig. 3. Workflow of predictor and non-zero execution block detection unit.

According to the execution process with the prediction stage, LRPPU is the first used to calculate the prediction result before the original convolution operation is computed. In the prediction stage, the scheduler selects the ready *exeblocks* according to the *exeblock* information register and sends them to the execution unit directly. The execution unit consumes instructions and generates the prediction result. The predictor uses the prediction result as the mask value to generate the flag information of OA, which uses 1 and 0 to indicate the validity and invalidity of the OA, respectively. In the execution stage, the scheduler selects *exeblocks* and sends them to the NZBD unit. At the same time, according to the index of the OA of *exeblock*, the flag information is obtained from the predictor. Then, the NZBD unit filters out the invalid *exeblocks* and then sends the valid *exeblocks* to the execution unit according to the flag information and the stage information of *exeblock*. In this way, the execution unit only executes valid instructions. For traditional convolution, the scheduler selects ready *exeblocks* and sends them to the execution unit directly.

Fig. 3 shows an example of a prediction-based convolution, which contains three *exeblocks* and the number of iterations of each *exeblock* is three. *Exeblock* 1, 2, and 3 respectively contain three phases (*ld*, *cal*, *st*) related instructions. After the completion of the prediction phase, the predictor saves the OA flag information. Only the OA with index 1 is valid, while those OA with index 2 and 3 are invalid, which indicates several invalid convolutions in the execution stage. Specifically, for the first iterative convolution operation, the OA indices associated with *exeblock* 1, 2, and 3 are all initialized with 1. According to the information of the predictor, OA is valid as the flag is 1. Therefore, all three *exeblocks* are valid and will be sent to the execution unit. However, for the second and third iterations, the OA index is 2, 3, and the corresponding flag is 0, so all *exeblocks* are invalid, and the NZBD will filter out these *exeblocks* to skip invalid computation. (If there are some invalid instructions in an *exeblock* with four stages, NZBD will choose whether to filter the *exeblock* of a certain stage according to the flag information of OA and the stage information of *exeblock*.)



```
For(int oh=0;oh<OH;++oh){
For(int ow=0;ow<OW;++ow){
For(int h=0;h<H;++h){
For(int w=0;w<W;++w){
 OA[ow]+=IA[oh+h][ow+w]
*F[h][w]}}}}
```

Fig. 4. Convolution dataflow mapping and convolution calculation.

B. Dataflow graph mapping and pipeline execution model

Based on the execution model for *exeblocks*, the dataflow diagram of convolution is shown in Fig. 4. To achieve high utilization of computing elements in each PE, PE exploits data reuse across several dimensions in space and time, which avoids multiple accesses to filters and activations in memory. Each activation is shared spatially along each row of PEs, and each filter is spatially shared along each column of PEs. The convolution calculation in the sliding window is unrolled in each PE, and the next output value is calculated by sliding the window. We refer to one convolution calculation in the sliding window as an iteration. In Fig. 4, the convolution process is divided into two *Task*s. *Task* 1 loads the filter from DRAM to the PE in the first row, and other PEs obtain filter data through *flow exeblocks*. After *Task* 2 completes the IA data loading and flowing, PE array starts to perform the convolution operation, and executes *Task* 2 through multiple iterations to complete the entire convolution operation.

Based on the OA's flag information generated by the predictor, the NZBD unit filters out *exeblocks* related to the iOA to skip invalid computations. Risc-NN maximizes the utilization of PE execution units through the simultaneous execution of multiple *exeblocks*. That is, at a certain moment, four execution units can execute four *exeblocks* at different stages without any data dependency. However, previous experiments have shown that there are a large number of invalid *exeblocks*. When NZBD eliminates invalid *exeblocks*, the number of maximum schedulable *exeblock* is reduced accordingly, which may cause idle computing resources. For example, Fig. 5a shows the convolution operation process of an IA and three filters (the number of iterations is four). The PE 0 and PE 1 use IA's row 1-2 and row 2-3 to perform convolution with three filters to calculate OA's row 1 and row 2, respectively. In Fig. 5, each PE contains one *ld exeblock*, three *cal exeblocks* and one *st exeblock*. These *exeblocks* form a dataflow graph according to the dependence of the data, and three OA values are generated by every iteration of execution. One iteration is activated only if all the exeblocks are executed in its previous iteration. In the process of serially executing each iteration, when the

TABLE II
PE DESIGN PARAMETERS AND AREA BREAKDOWN.

Unit	Size	Area	Unit	Size	Area
MAC(16b)	1	1.14%	Flow	1	0.28%
Load	1	0.31%	Store	1	0.44%
Control	1	9.96%	Router	3	9.85%
Data buffer	32KB	33.63%	Predictor	10KB	10.57%
Instructionn buffer	32KB	33.81%	Total	-	0.860mm^2

cal exeblocks in an iteration finishes executing, computing resources will wait in an idle state before it begins to execute in the next iteration, resulting in low utilization rate in the PE array. On the timeline, the computing component is not kept busy all the time as shown in Fig. 5c. On the other hand, it is worth noting that the predicted OA has unstructured sparsity, so the amount of calculating operations of each PE for each iteration is different. For example, in Fig. 5a, there are two and three effective *cal exeblocks* in PE0 and PE1, respectively, in the second iteration. Therefore, it leads to idle cycles and unbalanced *exeblocks* execution in the PE array and hinders performance improvement.

To alleviate the above-mentioned problems, we use *exeblock* iterative pipeline execution to ensure the load balancing of the PE array and the busyness of the PE computing components. The iterative pipeline execution method enables the next iteration to be activated after the execution of the *ld exeblock* of current iteration ends. Fig. 5b shows the activation and response process of host and exeblocks on PE0. At the beginning of the first iteration, the host activates the *ld exeblock*. Once the execution of the *ld exeblock* ends, the PE sends a response signal to the host. After the host receives the response signal, it activates the *ld exeblock* of the next iteration without waiting for the execution of all current *exeblocks* to end. Fig. 5d shows the execution timeline of *exeblocks* after using iterative pipeline execution. It can be seen that the way of iterative pipeline execution ensures the load balancing of the PE array and keeps the computing components busy.

V. EVALUATION

A. Experimental Methodology

Experimental Setup. We implemented the LRPPU microarchitecture including the controller unit with prediction mechanism, execution unit, instruction and data buffer unit in Verilog. We use Synopsys Design Compiler and a TSMC 45nm GP standard VT library to synthesize it and obtain area, delay and energy consumption. Table II lists the configurations and area berakdown of the PE. The LRPPU consists of a 16×16 PE array, and PE nodes are connected by 2D mesh networks.

Benchmarks. To evaluate our strategies, we use representative neural networks AlexNet, VGG-16, Resnet18 and Resnet50 convolution layers as benchmarks which have different sizes and parameter scales. We use Caffe to obtain HV pairs based on SVD approach of model parameters on NVIDIA Titan Xp GPU. Table III lists the K values of different layers (only part of the K of the Resnet layer is listed due to space constraints). Each convolution layer is translated into *exeblocks* through compiler based on LLVM platform.

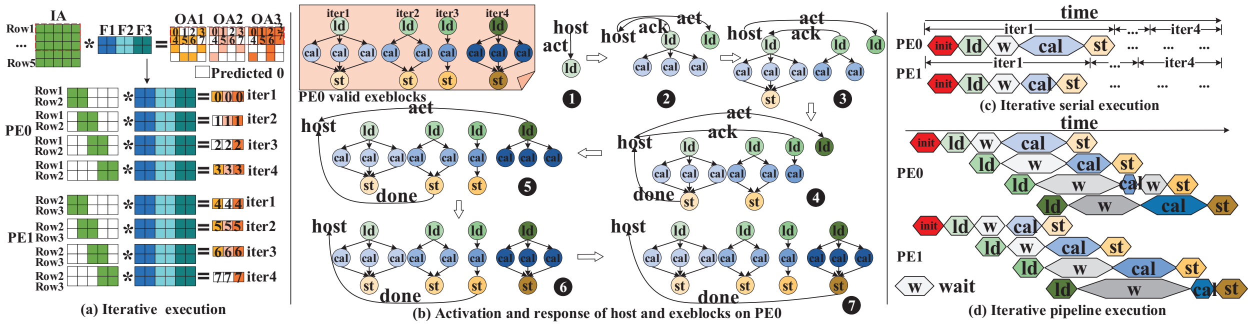

(a) Iterative execution
(b) Activation and response of host and exeblocks on PE0
(c) Iterative serial execution
(d) Iterative pipeline execution
PE0
PE1
host
act
ack
done
time
wait
Fig. 5. Iterative pipeline execution.

TABLE III
THE K VALUE OF THE FOUR NETWORKS CONVOLUTIONAL LAYERS.

Alexnet	**a**	**b**	**c**	**d**
c1-c2-c3	8-40-60	8-30-40	8-20-30	8-10-15
c4-c5	100-100	70-70	50-50	30-30
VGG16	**a**	**b**	**c**	**d**
c1_1-c1_2	5-24	5-16	5-12	5-6
c2_1-c2_2	48-48	32-32	24-24	12-12
c3_1-c3_2-c3_3	64-128-160	45-95-106	32-64-80	16-32-40
c4_1-c4_2-c4_3	192-192-256	128-128-170	96-96-128	48-48-64
c5_1-c5_2-c5_3	320-320-320	210-210-210	160-160-160	80-80-80
Resnet18	**a**	**b**	**c**	**d**
c2a_2b-c2b_2a-c2b_2b	32-32-32	20-20-20	16-16-16	8-8-8
c3a_2a-c3a_2b-c3b_2a	32-64-64	20-40-40	16-32-32	8-16-16
c3b_2b-c4a_2a-c4a_2b	64-64-128	40-40-90	32-32-64	16-16-32
c4b_2a-c4b_2b-c5a_2a	128-128-128	90-90-90	64-64-64	32-32-32
c5a_2b-c5b_2a-c5b_2b	200-200-200	130-130-130	100-100-100	50-50-50
Resnet50	**a**	**b**	**c**	**d**
c2b_2b-c2c_2b-c3a_2b	32-32-64	20-20-40	16-16-32	8-8-16
c3b_2b-c3c_2b-c3d_2b	64-64-64	40-40-40	32-32-32	16-16-16
c4b_2b-c4c_2b-c4d_2b	120-120-120	80-80-80	60-60-60	30-30-30
c4e_2b-c4f_2b-c5a_2a	120-120-120	80-80-80	60-60-60	30-30-30
c5a_2b-c5b_2b-c5c_2b	200-200-200	130-130-130	100-100-100	50-50-50

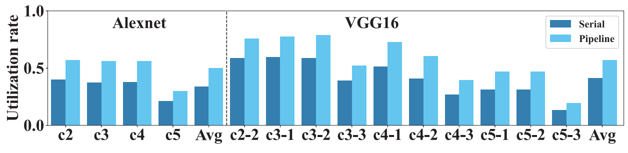


Fig. 6. The computing resource utilization of iteration serial and iteration pipeline execution methods.

B. Experimental Results

Performance and Energy Consumption. The network with the added prediction phase can effectively avoid a lot of invalid computing overhead during execution, thereby bringing benefits to performance and energy consumption. Also, we apply the iterative pipeline execution method to keep the computing resource of the PE as busy as possible. Fig. 6 shows the usage of computing resources in iteration serial and iteration pipeline execution methods, it can be seen that the utilization rate of computing resources varies with OA sparsity and the utilization rate of the Alexnet and VGG16 increases by 46.45% and 38.88% respectively, which demonstrates the effectiveness of the iterative pipeline method.

Fig. 7 shows the performance improvement of the convolutional layer of the four networks compared with the dense network (without SVD) after using different K values in Table III to predict the OA sparsity. When the K value becomes smaller, the prediction time becomes shorter and the performance increases accordingly. And when the filter size is the same, the higher the OA sparsity of the convolutional layer, the higher the performance, such as c3_1, c3_2, c3_3 and c4_1, c4_2, c4_3 of VGG16. The performance improvement of Alexnet is greater than that of other networks. One reason is that the K value of Alexnet is lower than that of VGG16 and Resnet as shown in Table III. Another reason is that the OA sparsity of Alexnet is higher than other networks (their average OA sparsity is 77.7%, 68.6%, 62.9%, and 61.9% respectively). Therefore, the total time consumption is relatively small.

Fig. 8 shows the energy consumption of the convolutional layers of four networks with different K values normalized to the dense network (without SVD). Similarly, when the K value becomes smaller, the amount of calculating operations required for prediction becomes less, and the energy consumed decreases accordingly. Since Alexnet's K value is smaller than other networks, and its sparsity is higher than VGG16 and Resnet, the magnitude of energy reduction is more than VGG16 and Resnet.

Compared with Other Accelerators. In addition to analyzing performance compared to dense networks, we also compare our LRPPU with other accelerators as shown in Table IV. In Table V, we report the speedup comparison of LRPPU and SnaPEA over Eyeriss baseline across all the benchmarks with the same peak performance (256GOPS), frequency (500MHz) and technology (45nm). It also shows the energy reduction based on the dense network and the Top5 accuracy drop of the four networks achieved by LRPPU and SnaPEA. When the K value of the selected filter is smaller, the filter information it contains also becomes smaller, so this will have an impact on the accuracy. We fine-tune offline the network based on SVD approach to reduce the loss of accuracy, thereby reducing the error prediction of OA. In all cases, the accuracy loss of Alexnet is larger, because the value of K is smaller than other networks, which is more aggressive to accuracy. In the case b, the accuracy of LRPPU is similar to that of SnaPEA (except for the Alexnet), and the average performance of the network is slightly higher than that of SnaPEA. In the case c, the accuracy of LRPPU is better than that of SnaPEA (except for the Alexnet), although the average performance of the network is slightly lower than that of SnaPEA. In the case of d, this is

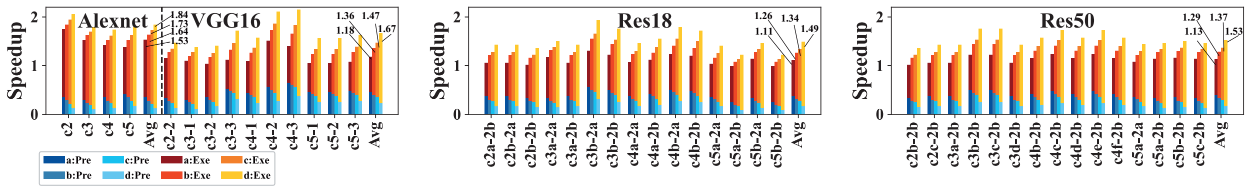


Fig. 7. Improved performance of convolutional layers over dense networks in the LRPPU.

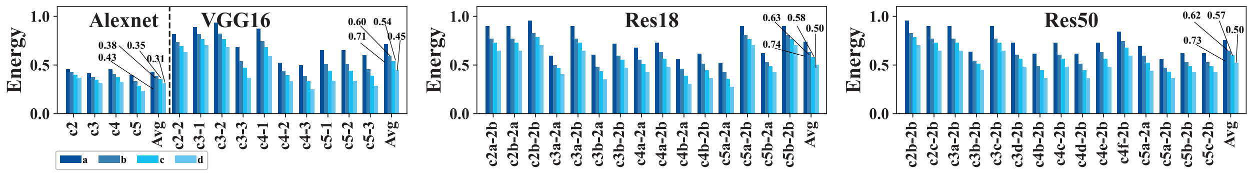


Fig. 8. The energy consumption of convolutional layer in the LRPPU (normalized to dense network).

TABLE IV
HARDWARE COMPARISON BETWEEN LRPPU AND OTHER ACCELERATORS.

Acce	Tech (nm)	Width (bits)	Clock (MHz)	MAC	Peak Perf (GOPS)	Area (mm^2)	Operation	Prediction Method	Power (mW)	Efficiency (GOPS/W)
SparseNN	65	16	500	64	64	78.41	Matrix	End to End training	452-705	91-143
LRPPU	45	16	500	256	256	220.13	Convolution	Conv SVD + fine-tuning	154.95-842.62	304-1652
SnaPEA	45	16	500	256	256	18.6	Convolution	Weight re-ordering	-	-
Eyeriss	45	16	500	256	256	17.8	Convolution	No	-	-

TABLE V
LRPPU AND SNAPEA PERFORMANCE IMPROVEMENT (BASED ON EYERISS) AND ENEYGY REDUCE (BASED ON DENSE NETWORK) AND TOP5 ACCURACY RATE DROP. (FOR SNAPEA, A, B, C, AND D INDICATE A DECREASE IN ACCURACY OF 0%, 1%, 2%, AND 3% RESPECTIVELY.)

Para	Net	Alexnet		VGG16		Resnet18		Resnet50		Geomean	
	Acc	80.27%		90.6%		90.85%		93.29%		-	
		LRP	Sna	LRP	Sna	LRP	Sna	LRP	Sna	LRP	Sna
Speedup	a	1.27	**1.28**	**1.36**	1.25	1.04	**1.19**	1.21	**1.30**	1.21	**1.24**
	b	**1.34**	**1.34**	**1.50**	1.26	**1.20**	**1.20**	**1.36**	1.35	**1.36**	1.29
	c	1.39	**1.55**	**1.60**	1.50	1.31	**1.43**	1.46	**1.55**	1.46	**1.50**
	d	**1.95**	1.90	**1.76**	1.60	1.50	**1.52**	1.62	**1.65**	1.62	**1.67**
Energy Reduce	a	**2.32**	1.20	**1.41**	1.11	**1.35**	1.18	**1.32**	1.16	**2.74**	1.38
	b	**2.60**	1.30	**1.68**	1.15	**1.59**	1.30	**1.55**	1.27	**1.79**	1.19
	c	**2.84**	1.42	**1.86**	1.20	**1.72**	1.60	**1.67**	1.35	**1.66**	1.45
	d	**3.20**	1.60	**2.21**	1.30	**1.98**	1.70	**1.92**	1.50	**1.62**	1.32
Top5 acc drop(%)	a	0.77	0	0.04	0	0.42	0	0.45	0	-	-
	b	1.87	1	0.6	1	0.9	1	0.93	1	-	-
	c	2.77	2	1.10	2	1.40	2	1.50	2	-	-
	d	5.27	3	4.30	3	4.50	3	4.60	3	-	-

impractical. Because their K value is very small, the accuracy loss of these networks is very large (Alexnet loss is the most serious). It is worth noting that the energy reduction of LRPPU is more than that of SnaPEA in all cases. In summary, the choice of the convolutional layer K value becomes a knob for controlling precision and performance.

VI. CONCLUSION

The OA of the convolutional neural network has high sparsity, causing many invalid computations. In this paper, we use the SVD approach to predict the sparsity of OA in convolutional neural networks, and design the accelerator LRPPU to take advantage of the sparsity of OA. LRPPU has a predictor and invalid *exeblocks* detection unit to leverage these sparsity, thereby improving the performance of the network. This paper is the first attempt to predict the sparsity of OA with SVD approach to speed up CNN networks. Experimental results show that the proposed approach can significantly speedup state-of-the-art networks with a marginal accuracy loss, and the choice of filter rank makes a better trade-off between performance and accuracy.

VII. ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China (Grant No. 61732018, 61872335, and 61802367), Austrian-Chinese Cooperative R&D Project (FFG and CAS) Grant No. 171111KYSB20200002, the Strategic Priority Research Program of Chinese Academy of Sciences (Grant No. XDC05000000).

REFERENCES

[1] Y. Lin, C. Sakr, Y. Kim, and N. Shanbhag, "Predictivenet: An energy-efficient convolutional neural network via zero prediction," in *2017 IEEE international symposium on circuits and systems (ISCAS)*. IEEE, 2017, pp. 1–4.

[2] M. Song, J. Zhao, Y. Hu, J. Zhang, and T. Li, "Prediction based execution on deep neural networks," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 752–763.

[3] V. Akhlaghi, A. Yazdanbakhsh, K. Samadi, R. K. Gupta, and H. Esmaeilzadeh, "Snapea: Predictive early activation for reducing computation in deep convolutional neural networks," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 662–673.

[4] D. Lee, S. Kang, and K. Choi, "Compend: Computation pruning through early negative detection for relu in a deep neural network accelerator," in *Proceedings of the 2018 International Conference on Supercomputing*, 2018, pp. 139–148.

[5] J. Zhu, J. Jiang, X. Chen, and C.-Y. Tsui, "Sparsenn: An energy-efficient neural network accelerator exploiting input and output sparsity," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 241–244.

[6] J. Zhu, Z. Qian, and C.-Y. Tsui, "Lradnn: High-throughput and energy-efficient deep neural network accelerator using low rank approximation," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2016, pp. 581–586.

[7] A. Davis and I. Arel, "Low-rank approximations for conditional feedforward computation in deep neural networks," *arXiv preprint arXiv:1312.4461*, 2013.

[8] C. Tai, T. Xiao, Y. Zhang, X. Wang *et al.*, "Convolutional neural networks with low-rank regularization," *arXiv preprint arXiv:1511.06067*, 2015.

[9] T. Xiang, L. Zhang, S. An, X. Ye, M. Zhang, Y. Liu, M. Yan, D. Wang, H. Zhang, W. Li *et al.*, "Risc-nn: Use risc, not cisc as neural network hardware infrastructure," *arXiv preprint arXiv:2103.12393*, 2021.