Journal of
*Low Power Electronics and Applications*

MDPI

*Article*

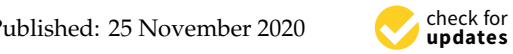
# An Improved K-Spare Decomposing Algorithm for Mapping Neural Networks onto Crossbar-Based Neuromorphic Computing Systems

**Thanh D. Dao and Jaeyong Chung ***

System on Chip Laboratory, Department of Electronics Engineering, Incheon National University, Incheon 22012, Korea; thanhdd@inu.ac.kr

* Correspondence: jychung@inu.ac.kr

Received: 28 September 2020; Accepted: 23 November 2020; Published: 25 November 2020




**Abstract:** Mapping deep neural network (DNN) models onto crossbar-based neuromorphic computing system (NCS) has recently become more popular since it allows us to realize the advantages of DNNs on small computing systems. However, due to the physical limitations of NCS, such as limited programmability, or a fixed and small number of neurons and synapses of memristor crossbars (the most important component of NCS), we have to quantize and decompose a DNN model into many partitions before the mapping. However, each weight parameter in the original network has its own scaling factor, while crossbar cell hardware has only one scaling factor. This will cause a significant error and will reduce the performance of the system. To mitigate this issue, the K-spare neuron approach has been proposed, which uses additional K spare neurons to capture more scaling factors. Unfortunately, this approach typically uses a large number of neurons overhead. To mitigate this issue, this paper proposes an improved version of the K-spare neuron method that uses a decomposition algorithm to minimize the neuron number overhead while maintaining the accuracy of the DNN model. We achieve this goal by using a mean squared quantization error (MSQE) to evaluate which crossbar units are more important and use more scaling factor than others, instead of using the same k-spare neurons for all crossbar cells as previous work does. Our experimental results are demonstrated on the ImageNet dataset (ILSVRC2012) and three typical and popular deep convolution neural networks: VGG16, Resnet152, and MobileNet v2. Our proposed method only uses 0.1%, 3.12%, and 2.4% neurons overhead for VGG16, Resnet152, and MobileNet v2 to keep their accuracy loss at 0.44%, 0.63%, and 1.24%, respectively, while other methods use about 10–20% of neurons overhead for the same accuracy loss.

**Keywords:** decomposing; deep neural networks; memristor crossbar; neuromorphic; mapping

## 1. Introduction

Deep learning has achieved impressive results in recent years due to the high prediction performance for recognition problems such as voice recognition, image, and video classification [1–4]. However, it uses expensive computing resources such as powerful server computers, large graphic processing cards, and requires high energy consumption. On the other hand, in many applied systems such as autonomous vehicles, mobile phones, or any embedded systems, we cannot afford to have such energy/power-hungry resource components. In many circumstances, these systems need to process large amounts of information very quickly to make real-time decisions without delay. Given this, we argue that it should process information locally instead of sending data to a central server through the Internet, which may introduce a significant amount of delay. While deep neural network (DNN) models can help us process a large amount of data, we cannot assume that they will be successfully

deployed without any difficulty onto these systems, as they would require a large amount of computing operations and power. Finding a solution for this kind of problem was inspired by the human brain that performs parallel operations and consumes low power. These demands have led to the development of a new computing architecture that emerges in recent years, called neuromorphic computing systems.

Note that, originally, the term neuromorphic computing system (NCS) was used to mean a very large-scale integration system that imitated the biological neural system of the human brain [5]. Currently, this term is used widely for brain-inspired systems such as spiking neural networks, artificial neural networks (ANN), or any non-von Neumann architecture. These NCS architectures have many advantages that make them very useful, such as parallelizable computation, low power consumption, and collocation of memory and processing. They also have neurons and synapses similar to ANN (both of them were inspired by biological human brains). Due to the popularity of DNNs, many studies are trying to implement them in NCS [6–10]. While the idea of developing an ideal algorithm for DNN applied NCS that learn in real-time, similar to what the biological brains do, it is still facing significant challenges. In practice, researchers currently focus more on the problem of mapping a trained neural network (NN) to hardware devices [11–14].

Numerous recently developed NCSs are based on memristor crossbar array (MCA) components due to their ability to implement matrix multiply-accumulate operation (the most expensive operation in DNN) efficiently [15]. High-resolution software-defined weights of a DNN model will be digitally stored in a Complementary Metal Oxide Semiconductor (CMOS) system. Their values will be converted from digital to analog for storing and calculating. Memristor crossbar arrays (MCA) realizations have also become more popular in the literature in recent years due to their large physical density. They have been demonstrated to perform well in practice [6–8]. However, these crossbars have a fixed and small number of neurons, and the crossbar size can only go up to $1024 \times 1024$. On the other hand, a fully connected layer in VGG [2] has $4096 \times 1024$ neurons. Therefore, implementing a practical crossbar-based NCS requires the integration of multiple MCAs. That is why decomposing the weights into different MCAs is crucial for parallel computing, which is increasingly needed with an increasing network scale.

While previous works [11–14] typically use the vanilla decomposition algorithm (VDA)—a straight forward method and fixed-point quantization, the most recent research has introduced a number of algorithms to decompose NN to $p \times q$ sized arrays [16,17]. It also utilizes a dynamic fixed-point (DFP) quantization method to reduce the model size while keeping the prediction performance the same as the state of the art (as proposed by Reference [18]). This technique uses at least 5-bit quantization to balance model size compression and its accuracy in NCS [19,20]. However, note that each crossbar has only one scaling factor while floating-point numbers before quantization have their own scaling factor. Therefore, this method causes significant errors and the original DNN model loses a significant amount of accuracy. To solve this, Reference [16] has proposed a novel design called a k-spare neuron to reduce space for neurons from a $p \times q$ array to $p \times (q - k)$ and spend k-spare neurons to contain more scaling factors, resulting in using more crossbar arrays than VDA. For the same reason, Reference [17] proposed another algorithm to use less MCA than the previous one but cannot keep the state-of-the-art accuracy of the model. Both of these proposals typically use 10–140% more crossbar arrays when compared to the VDAs. This extra cost makes the system use more hardware resources, consume more energy, and enlarge its size. For the same purpose in reducing computing and storage cost of DNN models, References [21,22] propose several techniques to pre-define fixed non-zero weights and zero-weights subsets, and then map non-zero weight clusters to MCAs. Since large parts of weight parameters are zero-weights, the neuromorphic system will save significant storage memory and energy consumption. The CSrram paper [21] has shown results on small datasets. However, they did both training and inference.

Our work is similar to CSrram. However, we have chosen a somewhat larger area for the models to achieve better resolution. In this paper, we address the issue by proposing a novel k-spare optimization method to minimize the number of crossbar array usage. In particular, we show that our algorithm requires a number of crossbar arrays close to the VDA's solution without any significant loss on the accuracy of pre-trained models for inference on a large dataset (ImageNet).

## 2. Background

### 2.1. Dynamic Fixed-Point Format

The parameters of NN models are often stored in the 32-bit floating-point format, which consumes more energy and requires large storage and a memory footprint—the main challenges for embedded systems. The fixed-point representation helps reduce the number of bits per parameter, which, in turn, reduces the model size. The fixed-point uses a single scaling factor to store the radix point for all numbers and causes a significant quantization error. In the dynamic fixed-point representation, there are multiple scaling factors with each being shared by a group of numbers. Table 1 shows an example of a 5-bit dynamic fixed point. The top three values have no differences between a fixed-point and dynamic fixed-point, but the next four lost their values in a fixed-point and, if changing the scaling factor from $2^{-4}$ to $2^{-7}$ to retain these values, the second row's value will be lost. A dynamic fixed point retains all of them by adjusting the scaling factor. In this way, a dynamic fixed point reduces the quantization error than the fixed-point that was demonstrated in Reference [18].

**Table 1.** This table shows examples of a 5-bit dynamic fixed-point format.

| Floating Point | Fixed Point | Dynamic Fixed Point |
|---|---|---|
| w11 = 0.6027 | $01010_{(2)} \times 2^{-4}$ | $00101_{(2)} \times 2^{-3}$ |
| w12 = 0.5012 | $01000_{(2)} \times 2^{-4}$ | $00001_{(2)} \times 2^{-1}$ |
| w21 = 0.3528 | $00110_{(2)} \times 2^{-4}$ | $00011_{(2)} \times 2^{-3}$ |
| w22 = 0.0062 | $00000_{(2)} \times 2^{-4}$ | $00001_{(2)} \times 2^{-7}$ |
| w31 = 0.0352 | $00000_{(2)} \times 2^{-4}$ | $00001_{(2)} \times 2^{-5}$ |
| w32 = 0.0251 | $00000_{(2)} \times 2^{-4}$ | $00011_{(2)} \times 2^{-7}$ |
| w41 = 0.0178 | $00000_{(2)} \times 2^{-4}$ | $00001_{(2)} \times 2^{-6}$ |
| w42 = 1.1098 | $10010_{(2)} \times 2^{-4}$ | $01001_{(2)} \times 2^{-3}$ |

### 2.2. Crossbar-Based Neuromorphic System

Our work is based on ReRAM, PRIME architecture because of its advantages, which provide a framework for both software developers and hardware designers to work on. We believe our proposed algorithm will be suitable for many kinds of crossbar-based NCS because the most important component of these systems is memristor crossbar arrays (MCA) that implement matrix-vector multiplication. The input goes through digital analog converter (DAC) as voltage multiplies with conductance (an inverse of resistance) used to represent weights that will generate current as output. The summation of current at the end of each crossbar column by Kirchhoff's current law is the result of matrix-vector multiplication. MCAs can usually accommodate a scaling factor per column. For example, consider the design given in Figure 1 that uses a shifter at the end of each crossbar column as a scaling factor.

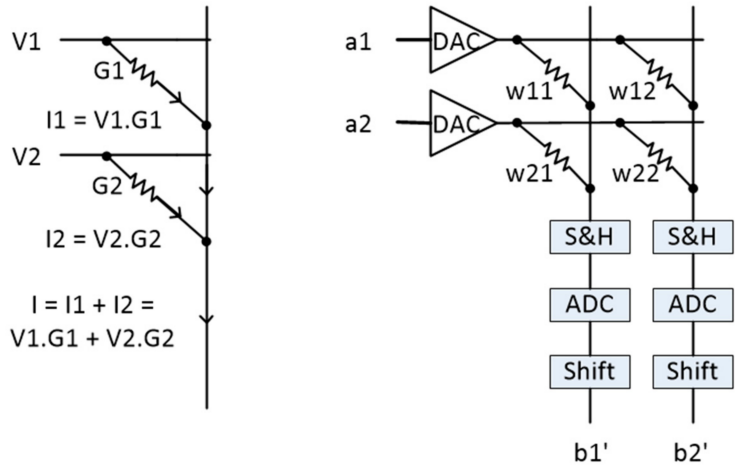



**Figure 1.** This is an example of matrix multiplication and crossbar 2 × 2.

## 3. Varied-K Decomposition Algorithm

### 3.1. Problem Definition

Since the size of the crossbar is fixed while the size of a neural network can vary and often become larger than the crossbar size, we have to decompose a neural network to smaller ones that can map onto crossbars. The most popular partitioning method used in literature is VDA, which is mentioned for the first time in Reference [16], but it was used in many previous works [6–8,10–14]. It is a simple and straightforward method, as an example in Figure 2. In this example, a NN 4 × 2 is decomposed into two 2 × 2 partitions. Their weight parameters are quantized in dynamic fixed-point format and mapped to two corresponding crossbar 2 × 2. Then outputs of these crossbars will be merged by an adder in PRIME [11] as a combined cost to restore the original outputs. However, before quantization, each number has its own scaling factor while, in VDA, all weight parameters in the same crossbar column share a scaling factor (implemented by a shifter). This reduces the accuracy substantially.

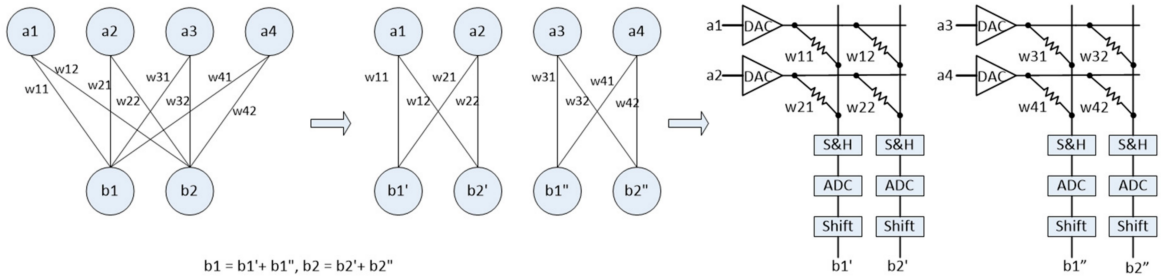



**Figure 2.** This is an example of the vanilla decomposition algorithm (VDA). A neural network 4 × 2 is divided into two partitions 2 × 2 and mapped to two 2 × 2 crossbars.

To mitigate this issue, in Reference [16], the authors propose a method named k-spare neuron decomposing algorithm (k-SDA) to reduce the space for neurons in a crossbar from $p \times q$ to $p \times (q - k)$ and spend $k$ spare neurons for more $k$ scaling factors. K is fixed as an input of the algorithm and is the same for all crossbars. Suppose we have an ANN that we would like to map onto a collection of crossbar arrays of size $p \times q$ (this is assumed to be fixed and determined by our hardware). In the '$p \times (q - k)$' approach, the partition no longer has $p$ unique inputs and $q$ unique outputs, but, rather, we reduce the

output width to $q - k$ unique outputs. The remaining $k$ lines are then used to provide additional scaling factors to the selected $q$ outputs. Using the same example as in Figure 2, we put two $2 \times 2$ partitions into two new crossbars size $p \times q = 2 \times 4$ shown in Figure 3, $k = 2$. We also map their weight values in Table 1 to these crossbars. Without using spare neurons, the weight w22 = $00001_{(2)} \times 2^{-7}$ is changed to $00001_{(2)} \times 2^{-1}$ in order to share the same scaling factor $2^{-1}$ with w12 and this causes errors. It happens similarly to w32 and w41. After using spare neurons, w22, w32, and w41 are assigned their own scaling factors and retained their value, so the quantization error is reduced.

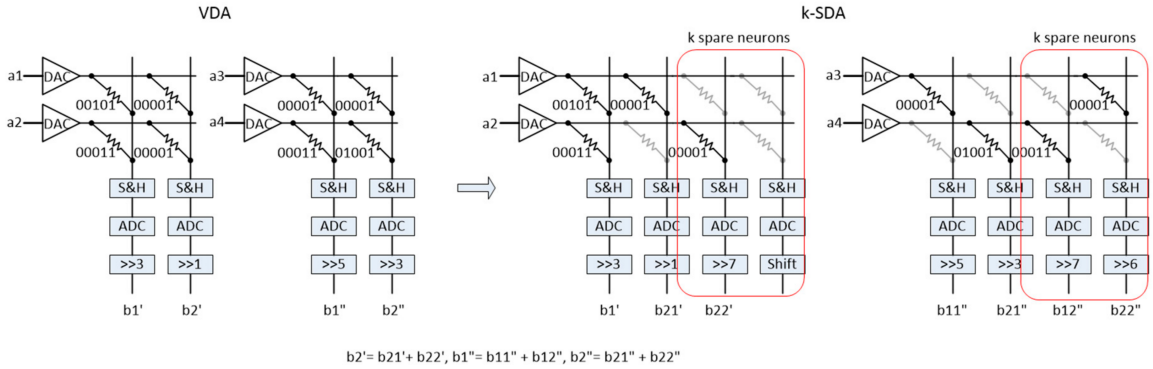



**Figure 3.** This is an example for the k-SDA method with 2 MCAs $2 \times 4$, each MCA has k = 2 spare neurons. Two partitions $2 \times 2$ in Figure 2 with values in Table 1 are mapped onto these MCAs.

However, two $2 \times 4$ crossbars in k-SDA equivalent to four $2 \times 2$ crossbars in VDA, it means doubling computing resources, energy consumption, and memory footprint to VDA. To reduce this overhead, the author also proposed two algorithms KS2 [17] and iterative split-in-two (IST) [16] to choose which weights parameter will be assigned their own scaling factors. Both algorithms use a mean squared quantization error (MSQE) to score the neurons. In the KS2, $k$ salient neurons are selected based on their scores. Each neuron will have one more scaling factor. Then, the weights of each neuron are partitioned into two groups and assigned a scaling factor to each group instead of sharing only one scaling factor. In the IST, for $q$–$k$ columns, each contains the weights of a distinct neuron. The algorithm will compute the saliency of each neuron, and the most salient neuron is split into two or more groups. This process is applied in $k$ iterations. Fixed k-SDA with IST keeps the accuracy of models but causes large overhead while fixed k-SDA with KS2 uses fewer neurons overhead but loses significant accuracy. In this paper, we propose a novel algorithm to minimize the neuron overhead while keeping the state-of-the-art accuracy of the NN model by varying $k$ for each crossbar.

### 3.2. Proposed Method

In the example in Figure 3, each MCA has two spare neurons, but the first one only uses one spare neuron. Therefore, we can optimize k-SDA by assigning varied $k$ spare neurons for each MCA. We name our method varied k-SDA (VSDA). To assign different k to each MCA, we need to evaluate MCAs in which MCAs are more important and use more k-spare neurons than others. Since our purpose is to reduce the error between the original model and the quantized model, we can use mean squared quantization error (MSQE) to evaluate MCAs. We denote the weight parameters matrix of a NN is $\boldsymbol{W}$, size $M \times N$. It will be decomposed to sub-matrices $\boldsymbol{W}_{i,j}$, size $m_i \times n_j$, $k_{i,j}$ is the number of spare neurons corresponding with $\boldsymbol{W}_{i,j}$, $m_i \times n_j = p \times (q - k_{i,j})$ or smaller (for sub-matrices that contain the last row or last column). ${\boldsymbol{W}^{DFP}}_{i,j}$ is the quantized dynamic fixed-point (DFP) matrix of $\boldsymbol{W}_{i,j}$ and ${\boldsymbol{W}^{VDA}}_{i,j}$ is a result of ${\boldsymbol{W}^{DFP}}_{i,j}$ after reducing the number of scaling factors to map onto MCAs in VDA. We need to find $k_{i,j}$ to minimize $\sum_{i,j} k_{i,j}$ and error between $\boldsymbol{W}_{i,j}$ and ${\boldsymbol{W}^{VDA}}_{i,j}$ We denote ${\boldsymbol{S}^{\boldsymbol{W}}}_{i,j}$ is MSQE of $\boldsymbol{W}_{i,j}$ and ${\boldsymbol{W}^{VDA}}_{i,j}$:

$$ {\boldsymbol{S}^{\boldsymbol{W}}}_{i,j} = \sum_g (\boldsymbol{W}_{g,j} - {\boldsymbol{W}^{VDA}}_{g,j})^2 / m_i, g = 1, 2, \ldots, m_i \quad (1) $$

The overall algorithm was shown in Algorithm 1. We use a pre-trained model as an input and DFP quantization with parameters: $m$, $t$, and $e$ are total bit, fractional bit, and exponent bit, respectively. Our algorithm is applied for CNN and fully-connected (FC) neural networks. It is easy to decompose a 2D neural network size $M \times N$ into $p \times q$ sized partitions, but it is more difficult for CNN because its weight matrix is four dimensions and has convolutional operations. To address this problem, we use the Toeplitz matrix. The 4D weight parameters matrix will be flattened to 2D matrix (lines 2, 3). The 4D input will also be converted to the 2D Toeplitz matrix and the convolutional operation will become matrix multiplication. The output result will be the same as the convolutional operation.

After having all weights matrices in 2D, we decompose them into $p \times q$ sized NN partitions $\boldsymbol{W}_{i,j}$, and then quantize and calculate their MSQE scores (lines 4–10). In the next step, we have to identify each MCA's k-spare neuron based on it MSQE score. The problem becomes a combinatorial optimized problem and it is hard to find an absolutely exact solution for it so we will propose an approximate solution instead. The MCA with the maximum MSQE score of the entire model that causes the largest error will need the largest k-spare neuron. We denote the maximum spare neuron that a MCA can be allocated is *max_k* with $max_k < q$. The MCA that contains the most erroneous neuron will be allocated *max_k* spare neurons. Other MCAs' k spare neurons will be calculated based on the ratio of each MCA's max score to the model's max score and multiplied with *max_k*. We refine the formula by dividing [*min*, *max*]—the MSQE score range of the entire model to *max_k* steps (lines 11, 12) and compare the maximum score of each MCA to these steps and get its k value. However, a maximum score of each MCA depends on its NN partitions size, which is unknown when $k$ is varied for all MCAs. Therefore, the next partition size will depend on the previous ones. To solve this problem, we will find $k$ of each MCA in sequence. We decompose the $M \times N$ sized weight matrix into $a + I$ ($b$) $p \times N$ sized partitions with: $a = M/p$, $b = M\%\ p$, $I(x)$ is indicator function, $I(x) = 1$ if $x > 0$ and $I(x) = 0$ if $x = 0$ (lines 16–20).

**Algorithm 1 Varied k-spare neuron decomposition**

**Input** pre-trained model, $m$, $t$, $e$, $p$, $q$, *max_k*
**Output** *k_hash* contains $k$ of all MCAs of the entire network model

1. **for** each NN layer in a model **do**
2. **if** NN layer is CNN **do**
3. Transform CNN weights to 2d matrix $\boldsymbol{W}$
4. Decompose $M \times N$ sized $\boldsymbol{W}$ to $p \times q$ sized $\boldsymbol{W}_{i,j}$
5. **for** each $\boldsymbol{W}_{i,j}$ **do**
6. $\boldsymbol{W}^{DFP}{}_{i,j} \leftarrow$ Quantize DFP $\boldsymbol{W}_{i,j}$
7. $\boldsymbol{W}^{VDA}{}_{i,j} \leftarrow$ Assign max scaling factor of each column to all scaling factors in the same column in $\boldsymbol{W}^{DFP}{}_{i,j}$
8. $\boldsymbol{S}^{\boldsymbol{W}}{}_{i,j} \leftarrow$ Calculate MSQE score of $\boldsymbol{W}^{VDA}{}_{i,j}$
9. $\boldsymbol{S}^{\boldsymbol{W}} \leftarrow [\boldsymbol{S}^{\boldsymbol{W}}{}_{i,j}]$
10. $\mathbf{S} \leftarrow \{\ \mathbf{S}^{\mathbf{W}}\}$
11. *min*, *max* $\leftarrow$ min($\boldsymbol{S}$), max($\boldsymbol{S}$)
12. *step* $\leftarrow$ *(max – min)/max_k*
13. Initial *k_hash*
14. **for** each NN layer in a model **do**
15. $M, N \leftarrow$ NN layer's weights $\boldsymbol{W}$ size
16. $a \leftarrow M/p$
17. $b \leftarrow M\ \%\ p$

18. *I (b) = 0*
19. **if** *b > 0*
20. *I (b)* ← *1*
21. *k_list* ← []
22. **for** *i* from 1 to *a + I (b)* **do**
23. *from_column* ← 1
24. **while** *from_column < N* **do**:
25. *best_k* ← 0
26. **for** $k_{i,j}$ from 0 to $q - 1$ **do**
27. *to_column* ← min(*from_column* + $q - k_{i,j}$, *N)*
28. $\boldsymbol{S^W}_{i,j}$ ← $\boldsymbol{S^W}$[*i, from_column: to_column*]
29. *temp_k* ← *Integer*((max($\boldsymbol{S^W}_{i,j}$) – *min*)/*step*)
30. **if** *temp_k* equals $k_{i,j}$
31. *best_k* ← *temp_k*
32. **exit for** loop of $k_{i,j}$
33. *k_list* ← *best_k*
34. *from_column* ← *from_column* + *best_k*
35. **end while**
36. **end for**
37. *k_hash*[layer] ← *k_list*
38. **end for**
39. **return** *k_hash*

We start with the first $p \times N$ sized partition to find the first MCA's NN partition. We use two variables *from_column*, *to_column* to track the position of the current weight partition in $N$ columns. Let $k$ run from 0 to $q - 1$ so its NN partition size will be changed from $p \times q$ to $p \times 1$ for example. With each NN partition that corresponds with each size, we calculate its temporary k spare neurons—*temp_k* by comparing its maximum MSQE score to *max_k* levels (lines 26–29). But *temp_k* may differ from $k$, so we only get the right $k$ if $k$ equals to *temp_k* to ensure the NN partition contains the corresponding maximum MSQE score (line 30). The loop of $k_{i,j}$ of the current MCA is also stopped in this case. If we cannot find any $k$ equals to *temp_k* after the for loop of $k_{i,j}$, the best $k$ will be zero. This may be not correct because the NN partition may contain the max scored neuron that has $k$ greater than zero, but our algorithm cannot find a better $k$ for it. This is an error of our approximate solution. However, it helps reduce the $\sum_{i\,j} k_{i,j}$ or the neuron overhead. We do the same for the next MCA and so on. We get all $k$ values of all MCAs.

We show an example of VSDA in Figure 4. Because our main concern is scaling factors for convenience, we only represent their exponent values of all neurons in MCA. MCA is also represented in a simple form with shifters represented by colored plus signs. The scaling factor's exponent values are represented by color dots with the value range in the color bar. A $5 \times 10$ NN is decomposed into two $5 \times 5$ NN partitions and mapped to two $5 \times 5$ MCAs in VDA. In k-SDA, it is decomposed to five $5 \times 2$ partitions and mapped to $5 \times 5$ MCAs for using 3-spare neurons for more three scaling factors. VSDA decomposed the original NN to three partitions of $5 \times 5$, $5 \times 2$, and $5 \times 3$ that correspond with their k values k = 0, 2, 3 of their spare neurons. Both weight parameters in k-SDA and VSDA are moved to their spare neuron by the IST algorithm. The neuron overhead of k-SDA is $(5 - 2)/2 \times 100\% = 150\%$. VSDA's neuron overhead is $(3 - 2)/2 \times 100\% = 50\%$. We easily realize the improvement of VSDA compared to k-SDA.

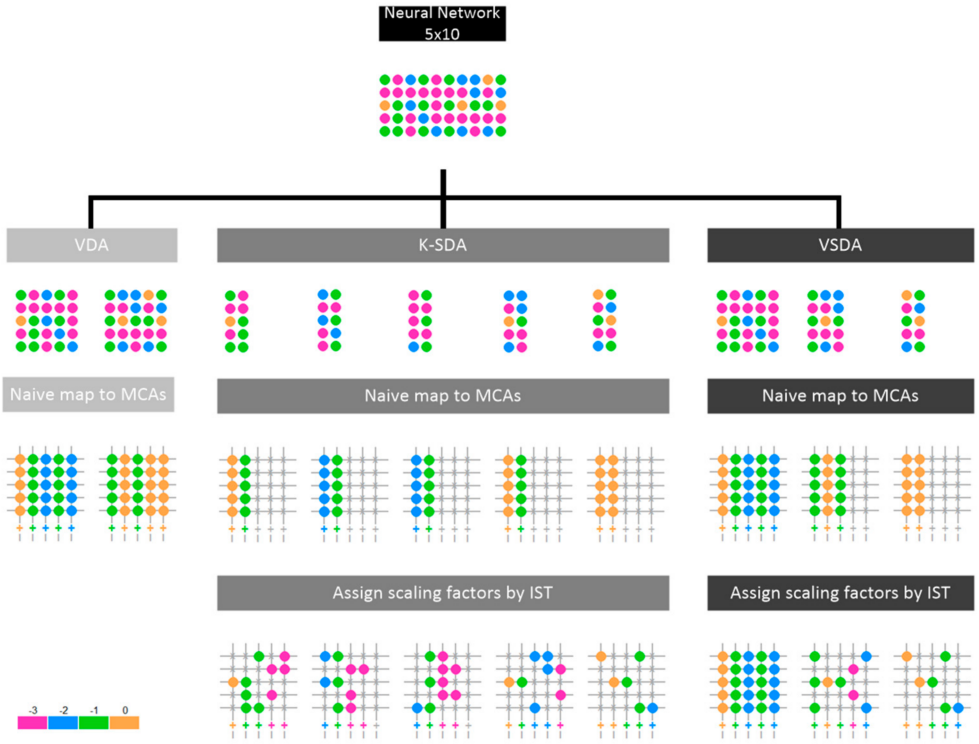



**Figure 4.** This is an example to compare VDA, k-SDA, and VSDA in decomposing a $5 \times 10$ NN.

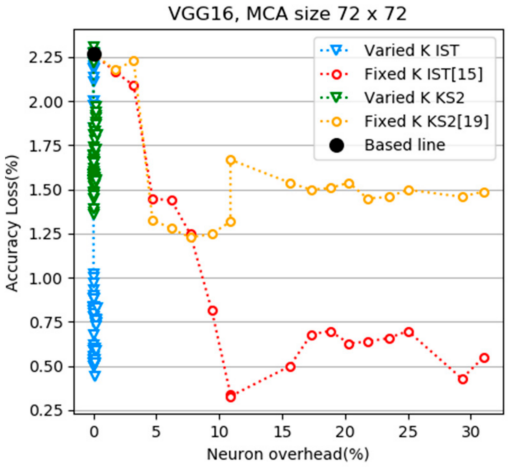
## 4. Experimental Results

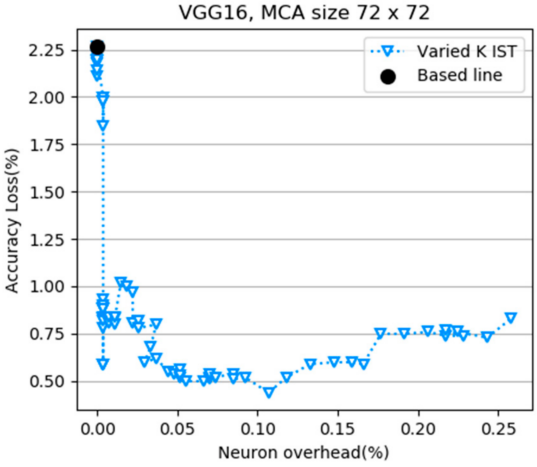
We do our experiments on three popular and typical DNNs: VGG16 [2], Resnet152 [23], and MobileNet v2 [24] with their original predict performances of 71.63%, 78.25%, and 71.85%, respectively, ImageNet dataset (ILSVRC2012) [25], dynamic fixed point $m = 5$, $t = 3$, and $e = 3$ for VGG16 and Resnet152, which is demonstrated in Reference [18] for keeping the state-of-the-art accuracy of the model.

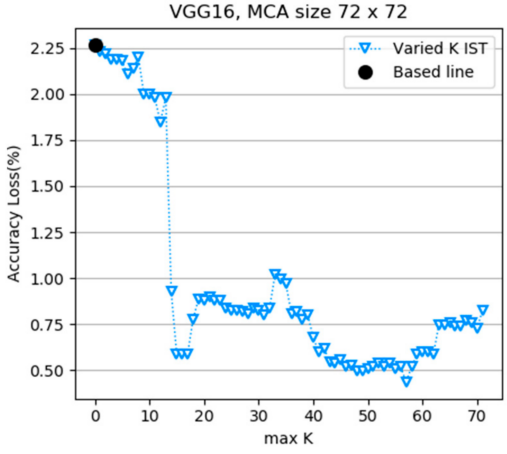
However, MobileNet is a highly optimized network and sensitive to error so we have to use $m = 7$ bit instead of a 5 bit to keep the accuracy of the model. 7-bit is also the maximum bits of PRIME. In addition, since our experiments are only for evaluating the efficacy of the proposed algorithm for the CNN and FC layer, we do experiments on simulation and the results do not include the error of implementation other layers or any realization errors. For all experiments, we use Pytorch [26], three pre-trained models of VGG16, Resnet152, and MobileNet v2 from Pytorch model zoo [27], and a *max_k* run in a range $[0, q - 1]$ to observe its affection on accuracy loss and neuron overhead and find the best *max_k*, $max_k = 0$ corresponding with VDA. We also compare VSDA in both KS2 and IST, and the baseline is VDA.

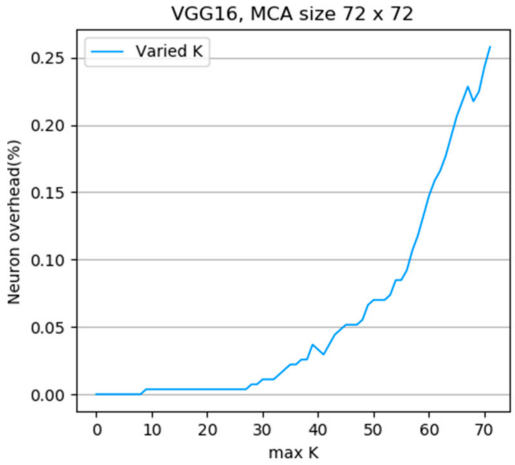
We use three metrics to evaluate the algorithms. The first one is the accuracy loss, which is defined as the original accuracy minus the accuracy of the mapped network. The second one is the neuron overhead, which is the ratio of the extra neurons used to the minimum required neurons in VDA. The last one is *max_k*.

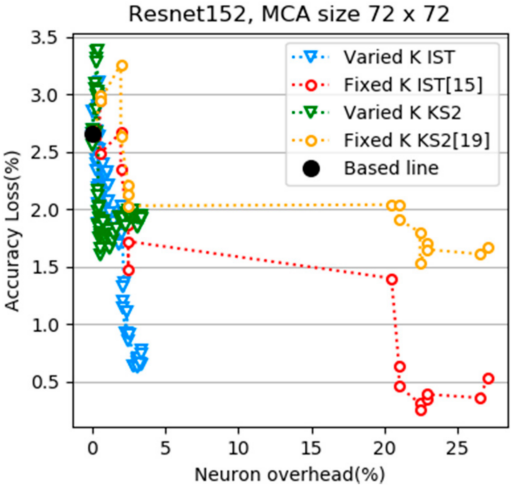
For the MCA size, we do experiments on crossbar size $p = q = 72$ firstly to compare our method with References [16,17]. The results are shown in Figure 5. Our algorithm gets better results in all cases including both varied k KS2 and IST. It keeps the state-of-the-art accuracy of models by using only 0.1%, 3.12%, and 2.4% neurons overhead for 0.44%, 0.63%, and 1.24% accuracy lost in VGG16, Resnet152, and MobileNet v2, respectively, which is compared to 10–20% neuronal overhead in other methods. The result also shows the changing of the neuron overhead and accuracy lost by *max_k*. In Reference [17], the authors concluded that KS2 is better than IST based on only a small range of k experiments. Our experiments with a larger range of *max_k* demonstrate IST is better than KS2.



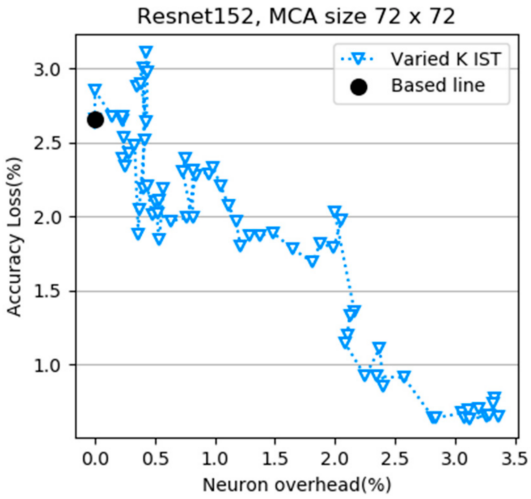




**Figure 5.** *Cont.*

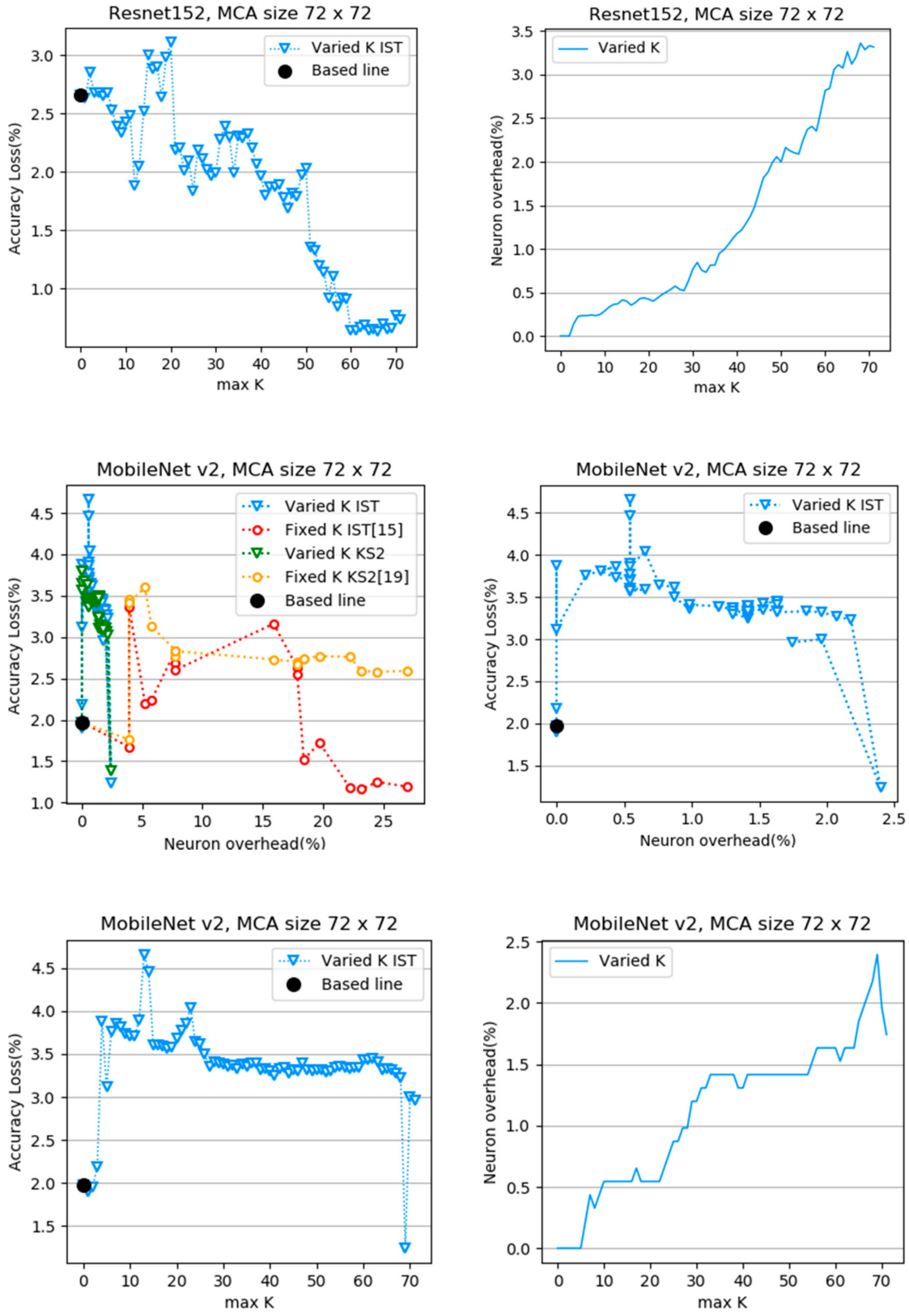



**Figure 5.** This figure is a graph that shows the results of experiments of VGG16.

However, the crossbar size $p = q = 72$ is not the standard and popular size. We also extend our experiments on a wide range of popular and standard size of MCA with $p = q = 16, 32, 64, 128, 256,$

and 512 to observe the effect of MCA size, the potential of the algorithm in reducing the quantization error, and may help us find the optimized crossbar size for each model.

The results are shown in Table 2. It shows the best case of *max_k* of our experiments in a wide range of *max_k*. In this table, we compare the results of varied k IST and VDA in accuracy lost and neuron overhead (L/OH) and also observe the affection of the crossbar size. The result shows that:

- Small crossbars generally get a better result than big crossbars because the small group of dynamic fixed-point numbers causes a smaller error than the big group
- Optimized network like Resnet and MobileNet need more neuronal overhead to restore the accuracy of models than the sparse network as VGG
- Small-size crossbars cause a small quantization error but it uses plenty of crossbar amount. Therefore, it is more expensive for combination after decomposing.
- Big size crossbar cause a larger quantization error but use a small resource for combination. In addition, the varied k algorithm still works well to restore the accuracy of models in big size crossbars.
- We obtained a better result for MobileNet v2 than the previous size with a larger-sized MCA $p = q = 128$. We achieve accuracy lost at 0.91% while using 6.11% neuronal overhead.

**Table 2.** This table shows results of the VSDA and VDA in a wide range of crossbar size in VGG16, Resnet152, and MobileNet v2.

| **Crossbar Size** | | **16 × 16** | | **32 × 32** | | **64 × 64** | |
|---|---|---|---|---|---|---|---|
| **DNN** | **Method** | **Acc (%)** | **MCAs** | **Acc (%)** | **MCAs** | **Acc (%)** | **MCAs** |
| Vgg16 | Origin | | | 71.63 | | | |
| | VDA | 70.35 | 540,536 | 69.56 | 135,198 | 69.46 | 33,800 |
| | VSDA | 71.43 | 540,543 | 71.23 | 135,273 | 71.21 | 34,100 |
| | L/OH | **0.2** | **0.001%** | **0.4** | **0.06%** | **0.42** | **0.89%** |
| | max_k | 3 | | 11 | | 47 | |
| Resnet152 | Origin | | | 78.25 | | | |
| | VDA | 77.20 | 234,600 | 76.13 | 58,682 | 75.27 | 14,671 |
| | VSDA | 77.82 | 236,480 | 77.60 | 60,092 | 77.65 | 17,026 |
| | L/OH | **0.43** | **0.80%** | **0.65** | **2.40%** | **0.6** | **16.05%** |
| | max_k | 12 | | 30 | | 63 | |
| MobileNet v2 | Origin | | | 71.85 | | | |
| | VDA | 70.14 | 13,806 | 69.74 | 3588 | 69.82 | 1041 |
| | VSDA | 70.20 | 13,808 | 69.76 | 3589 | 69.84 | 1041 |
| | L/OH | **1.65** | **0.01%** | **2.09** | **0.03%** | **2.01** | **0 %** |
| | max_k | 2 | | 1 | | 1 | |
| **Crossbar Size** | | **128 × 128** | | **256 × 256** | | **512 × 512** | |
| **DNN** | **Method** | **Acc (%)** | **MCAs** | **Acc (%)** | MCAs | **Acc (%)** | **MCAs** |
| Vgg16 | Origin | | | 71.63 | | | |
| | VDA | 69.50 | 8454 | 69.13 | 2121 | 69.04 | 543 |
| | VSDA | 71.29 | 8647 | 71.11 | 2259 | 71.10 | 595 |
| | L/OH | **0.34s** | **2.28%** | **0.52** | **6.51%** | **0.53** | **9.58%** |
| | max_k | 92 | | 251 | | 426 | |
| Resnet152 | Origin | | | 78.25 | | | |
| | VDA | 74.23 | 3684 | 74.98 | 968 | 74.97 | 445 |
| | VSDA | 77.64 | 4959 | 77.62 | 1803 | 77.66 | 585 |
| | L/OH | **0.61** | **34.61%** | **0.63** | **86.26%** | **0.59** | **31.46%** |
| | max_k | 116 | | 232 | | 380 | |
| MobileNet v2 | Origin | | | 71.85 | | | |
| | VDA | 69.90 | 360 | 69.91 | 142 | 69.86 | 78 |
| | VSDA | 70.94 | 382 | 71.16 | 164 | 71.12 | 93 |
| | L/OH | **0.91** | **6.11%** | **0.69** | **15.49%** | **0.73** | **19.23%** |
| | max_k | 125 | | 253 | | 509 | |

L/OH: Accuracy lost and neuron overhead of VSDA compared to VDA. Acc (%): The accuracy of the mapped system by using VDA and VSDA. MCAs: The number of MCAs are used to map the whole network.

## 5. Conclusions

We have proposed a varied k-spare neuron algorithm to minimize neuronal overhead to restore accuracy loss of DFP quantized DNN models after decomposing and mapping onto crossbar-based NCS. The proposed method optimizes neuronal overhead by allocating varied and dynamic k-spare neurons for each MCA instead of fixed k as previous work does. We also observe the affection of the DNN model type and crossbar size on the novel algorithm. The proposed method shows better results on all experiments compared to previous works. It also gives an insight for choosing the crossbar size to balance between accuracy lost, combination cost, and neuronal overhead. With these promising results, we believe the proposed algorithm will give more motivation for realizing dynamic fixed point crossbar-based NCS in future work.

**Author Contributions:** Conceptualization, J.C.; methodology, T.D.D. and J.C.; software, T.D.D.; validation, T.D.D. and J.C.; formal analysis, J.C.; investigation, T.D.D. and J.C.; data curation, T.D.D. and J.C.; writing—original draft preparation, T.D.D. and J.C.; writing—review and editing, J.C.; visualization, T.D.D.; supervision, J.C.; project administration, J.C.; funding acquisition, J.C. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was supported by the Institute for Information and Communications Technology Promotion funded by the Korea Government under Grant 1711073912.

**Conflicts of Interest:** The authors declare no conflict of interest.

## Abbreviations

| Abbreviation | Description |
|---|---|
| NN | Neural Network |
| DNN | Deep Neural Network |
| VDA | Vanilla Decomposition Algorithm |
| VSDA | Varied K spare neuron Decomposing Algorithm |
| VGG, VGG16 | Names of very deep convolutional neural networks proposed by Visual Geometry Group [2] |
| CMOS | Complementary Metal Oxide Semiconductor |
| NCS | Neuromorphic Computing System |
| ANN | Artificial Neural Network |
| MCA | Memristor Crossbar Array |
| DFP | Dynamic Fixed-Point |
| DAC | Digital Analog Converter |
| KS2 | K Split-in-Two |
| IST | Iterative Split-in-Two |
| K-SDA | K-Spare neuron Decomposing Algorithm |
| MSQE | Mean Squared Quantization Error |

## References

1. Lecun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef] [PubMed]
2. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.
3. Wu, C.; Karanasou, P.; Gales, M.J.; Sim, K.C. Stimulated Deep Neural Network for Speech Recognition. *Proc. Interspeech* **2016**, 400–404.
4. Karpathy, A.; Toderici, G.; Shetty, S.; Leung, T.; Sukthankar, R.; Fei-Fei, L. Large-Scale Video Classification with Convolutional Neural Networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 23–28 June 2014; pp. 1725–1732.
5. Mead, C. Neuromorphic eletronic systems. *Proc. IEEE* **1990**, *78*, 1629–1636. [CrossRef]
6. Hu, M.; Li, H.; Chen, Y.; Wu, Q.; Rose, G.S.; Linderman, R.W. Memristor crossbar-based neuromorphic computing system: A case study. *IEEE Trans. Netw. Learn. Syst.* **2014**, *25*, 1864–1878. [CrossRef] [PubMed]

7. Hu, M.; Strachan, J.P.; Li, Z.; Grafals, E.M.; Davila, N.; Graves, C.; Lam, S.; Ge, N.; Yang, J.J.; Williams, R.S. Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication. In Proceedings of the 2016 53rd ACM/EDAC/IEEE Design Automation Conference (DAC), Austin, TX, USA, 5–9 June 2016; pp. 1–6.
8. Boybat, I.; Le Gallo, M.; Nandakumar, S.R. Neuromorphic computing with multi-memristive synapses. *Nat. Commun.* **2018**, *9*, 2514. [CrossRef] [PubMed]
9. Chung, J.; Shin, T. Simplifying deep neural networks for neuromorphic architectures. In Proceedings of the 53rd Design Automation Conference (DAC), Austin, TX, USA, 5–9 June 2016; pp. 1–6.
10. Chung, J.; Shin, T.; Kang, Y. INsight: A neuromorphic computing system for evaluation of large neural networks. *arXiv* **2015**, arXiv:1508.01008.
11. Chi, P.; Li, S.; Xu, C.; Zhang, T.; Zhao, J.; Liu, Y.; Wang, Y.; Xie, Y. PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory. *Acmsigarch Comput. Arch. News* **2016**, *44*, 27–39. [CrossRef]
12. Shafiee, A.; Nag, A.; Muralimanohar, N.; Balasubramonian, R.; Strachan, J.P.; Hu, M.; Williams, R.S.; Srijumar, V. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM Sigarch Comput. Arch. News* **2016**, *44*, 14–26. [CrossRef]
13. Yao, P.; Wu, H.; Gao, B.; Tang, J.; Zhang, Q.; Zhang, W.; Yang, J.J.; Qian, H. Fully hardware-implemented memristor convolutional neural network. *Nature* **2020**, *577*, 641–646. [CrossRef] [PubMed]
14. Cai, Y.; Tang, T.; Xia, L.; Li, B.; Wang, Y.; Yang, H. Low Bit-Width Convolutional Neural Network on RRAM. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2020**, *39*, 1414–1427. [CrossRef]
15. Fatahalian, K.; Sugerman, J.; Hanrahan, P. Understanding the efficiency of gpu algorithms for matrix-matrix multiplication. In Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, Grenoble, France, 29–30 August 2004; pp. 133–137.
16. Kim, C. A Neural Network Decomposition Algorithm for Crossbar-based Neuromorphic System. Master's Thesis, Incheon National University, Incheon, Korea, February 2020.
17. Kim, C.; Abraham, A.J.; Kang, W.; Chung, J. A Neural Network Decomposition Algorithm for Mapping on Crossbar-Based Computing Systems. *Electronics* **2020**, *9*, 1526. [CrossRef]
18. Gysel, P.; Motamedi, M.; Ghiasi, S. Hardware-oriented approximation of convolutional neural networks. *arXiv* **2016**, arXiv:1605.06402.
19. Kang, Y.; Yang, J.; Chung, J. Weight partitioning for dynamic fixed-point neuromorphic computing systems. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2019**, *38*, 2167–2171. [CrossRef]
20. Kang, Y.; Chung, J. A dynamic fixed-point representation for neuromorphic computing systems. In Proceedings of the International SoC Design Conference (ISOCC), Seoul, South Korea, 5–8 November 2017; pp. 44–45.
21. Fayyazi, A.; Kundu, S.; Nazarian, S.; Beerel, P.; Pedram, M. CSrram: Area-Efficient Low-Power Ex-Situ Training Framework for Memristive Neuromorphic Circuits Based on Clustered Sparsity. In Proceedings of the 2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Miami, FL, USA, 15–17 July 2019; pp. 465–470.
22. Kundu, S.; Nazemi, M.; Pedram, M.; Chugg, K.; Beerel, P. Pre-Defined Sparsity for Low-Complexity Convolutional Neural Networks. *IEEE Trans. Comput.* **2020**, *69*, 1045–1058. [CrossRef]
23. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
24. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 4510–4520.
25. Deng, J.; Dong, W.; Socher, R.; Li, L.; Li, K.; Li, F. ImageNet: A large-scale hierarchical image database. In Proceedings of the 2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), Miami, FL, USA, 20–25 June 2009; pp. 248–255.

26. Pytorch. Available online: https://pytorch.org (accessed on 1 February 2019).
27. Pytorch Model Zoo. Available online: https://pytorch.org/docs/stable/torchvision/models.html (accessed on 1 February 2019).

**Publisher's Note:** MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).