Accelerating Deep Convolutional Neural Network Inference Based on OpenCL

Yong Wu, Huming Zhu, Lingyun Zhang, Biao Hou, Licheng Jiao

Accelerating Deep Convolutional Neural Network Inference Based on OpenCL

Yong Wu[1], Huming Zhu[2*], Lingyun Zhang[2], Biao Hou[2], and Licheng Jiao[2]

1 Xidian-Wuhu Research Institute

2 Key Laboratory of Intelligent Perception and Image Understanding, Ministry of Education, Xidian University, Xi'an 710071

zhuhum@mail.xidian.edu.cn

Abstract. In recent years, in order to facilitate the efficient application of deep convolutional neural networks, it has become increasingly important to accelerate the inference stage of deep convolutional neural networks. But with the development of numerous heterogeneous computing devices, today's popular deep learning inference tools only support specific devices, so they cannot effectively utilize different GPU devices to accelerate DNN inference. To address this issue, we propose an OpenCL-based parallel deep convolutional neural network inference algorithms. Firstly, we design and implement parallel kernel code using OpenCL to accelerate depthwise separable convolution, and implement parallel matrix multiplication combined with clBLAS to accelerate traditional convolution. Meanwhile, we design OpenCL parallel kernel codes for other operations in the inference stage of deep convolutional neural networks. Secondly, we further improve the inference performance by means of kernel fusion and increasing the workload per core. Finally, MobileNet v1 network and the 21-layer residual network based on OpenCL are run on AMD Radeon Vega Frontier GPU and Nvidia GeForce GTX 1070 GPU. Compared to the Caffe implementation, 40.16x, 1.67x speedups are achieved on the AMD GPU and 14.95x, 1.11x speedups are achieved on the Nvidia GPU.

Keywords: OpenCL, Deep Convolutional Neural Network, Inference, GPU.

1 Introduction

In recent years, deep neural networks have been widely used in image analysis [1], speech recognition, object detection [2], semantic segmentation, face recognition, and autonomous driving because of their excellent performance.

GPUs have been used to accelerate the training and inference of various neural network models due to their powerful parallel computing capabilities [3-4]. However, with the development of numerous heterogeneous computing devices, the manufacturers and models of GPUs have become increasingly complex and diverse. The programming environment for different GPUs also tends to be different. Therefore, it is of great value to study parallel algorithms with portability to adapt to different GPUs.

OpenCL is a cross-platform parallel programming standard, and it provides APIs

with parallel programming. It not only can be applied on FPGA, but also can on CPU, GPU and DSP (Digital Signal Processors). By using OpenCL implement the inference of deep convolutional networks[5-6], we can deploy the networks on different devices, which extend the scope of application.

Accelerating deep neural networks usually involves two stages. The first stage is training a model on a large dataset. In this stage, the commonly used parallel methods are data parallelism and model parallelism. For single-node data parallelism, by using parallel programming technology, independent computation is distributed to multiple computing cores of a single hardware device. Model parallelism decomposes the network model, distributes the convolution operation located in the same layer to different computing devices for calculation, and the output results are synchronized and transmitted to the next layer through communication between devices. The second stage is deep neural network inference stage. In this stage, we need to deploy the trained model on a device for image classification or object detection [7]. And there are many tools have been proposed to accelerate different deep neural network models on parallel computation device. Such as Intel OpenVINO[8] and NVIDIA TensorRT[9]. These inference tools both only support one manufacturer's GPU, which limited the scope of application.

Our main contributions of this work are summarized as follows:

Compared with related works, we firstly design OpenCL kernel code to accelerate depthwise separable convolution. We test the performance of MobileNet v1 network and the 21-layer residual net-work based on OpenCL on AMD Radeon Vega Frontier and Nvidia Ge-Force GTX 1070 GPU. By using kernel fusion and batch image processing, we further improve the performance of parallel acceleration without decreasing the accuracy.

2 Related work

At present, there have been some researches on accelerating the inference stage of neural networks, for example, Akshay Dua et al. [10] presents Systolic-CNN, an OpenCL-defined scalable, run-time-flexible FPGA accelerator architecture, optimized for accelerating the inference of various convolutional neural networks (CNNs)in multi-tenancy cloud/edge computing. Dian-Lun Lin et al. [11] introduce SNIG, an efficient inference engine for large sparse DNNs. SNIG develops highly optimized inference kernels and leverages the power of CUDA Graphs to enable efficient decomposition of model and data parallelisms, thereby accelerating large sparse neural network inference in parallel. Shengyu He et al. [12] propose PhoneBit, a GPU-accelerated BNN inference engine for mobile devices that fully exploits the computing power of BNNs on mobile GPUs. Jiale Chen et al. [13] presents a model split framework, namely, splitCNN, in order to run a large CNN on a collection of concurrent IoT sensors. The splitCNN achieves significant reduction in the model size and inference time while maintaining similar accuracy. Although these studies have made significant progress in inference acceleration, there are still some limitations to the current work; most of these studies are based on a particular device, cannot effectively use different GPUs to accelerate inference, and the cross-platform problem remains unresolved.

3 Design,Implementation and Optimization of CNN on OpenCL

3.1 Parallel Strategy for Convolution Layer

Traditional parallel computing of convolution usually uses the method of converting convolution to matrix multiplication. This kind of method is called im2col. MobileNet[14] uses deep separable convolution to replace the traditional convolution operations, which greatly reduces the number of mathematical operations and parameters. It uses depth wise convolution (DWC) and point wise convolution (PWC) to replace traditional convolution. We use OpenCL to design a kernel function to accelerate depth wise convolution of deep separable convolution and use matrix multiplication method to accelerate point wise convolution of deep separable convolution.

In order to achieve cross-platform performance portability, we use OpenCL to implement im2col operation and combine the matrix multiplication API provided by clBLAS library to speed up traditional convolution in parallel.

When using the clBLASCgemm function in clBLAS to speed up batch convolution operations, im2col needs to first convert the four-dimensional input data (NCWH) into a two-dimensional matrix, and then the clBLASCgemm function is called to complete the multiplication of input matrix and convolution core weight matrix. The output matrix is stored in the form of CNWH, so the data need to be rearranged into the storage in the form of NCWH. At the same time, the final output of clBLAS only completes the convolution operation without bias, so it is necessary to start another kernel function to complete the operation of adding bias.

Therefore, in order to realize the clBLAS accelerated convolution operation, it is necessary to use OpenCL to implement three kernels, As shown in the left picture of Fig. 1, to perform im2col, data conversion and adding bias respectively.

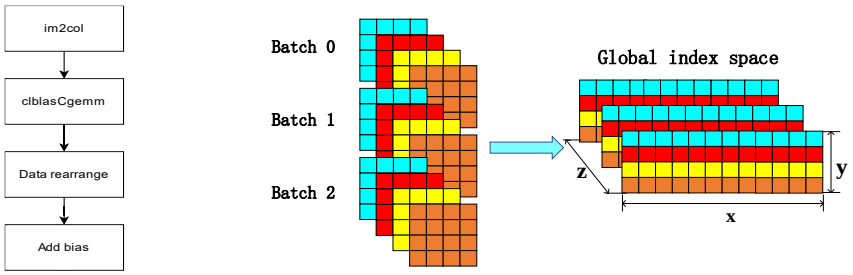

Fig. 1. Convolution process by suing clBLAS(left picture) and mapping relationship of OpenCL global index space for deep convolution(right picture)

The process of depth separable convolution operation on the output feature map is equivalent to a vector point product. The calculation process of different pixels is independent and can be carried out in parallel. In the structure of convolution neural network, the size of convolution filter is generally small, if using parallel reduction, the cost of synchronization may outweigh the benefits of parallelism, so the result of different filter's pixels are added serially. Therefore, the parallel scheme of deep convolution calculation

designed by OpenCL is as follows: the size of the global index space corresponds to the number of pixels in the output feature map, that is, each work item is responsible for calculating a pixel in the output feature map, and a certain number of work items are formed into a working group. The global index space is set to three dimensions: x, y and z. The x dimension corresponds to the pixels of the same channel in the output feature map, the y dimension corresponds to the number of channels in the output feature map, and the z dimension corresponds to the batch size of the output feature map. The pixel points that each thread is responsible for is determined by obtaining the global index value. The mapping relationship is shown in the right figure of Fig. 1.

For the depth convolution operation, the input data addresses to be accessed by consecutive pixels located in the same channel feature map are consecutive, and the weight addresses are the same. Therefore, in order to combine memory access, consecutive index values in the global index space should correspond to two consecutive output feature maps. pixel. The point-by-point convolution is the same as the traditional convolution method, except that the size of the convolution kernel is changed to 1×1, so we use the matrix multiplication convolution implemented earlier to speed up the point-by-point convolution.

3.2 Parallel Strategy for Other Layers

In order to achieve the parallel acceleration of the deep convolution neural network inference stage in OpenCL, other operations in the inference stage need to be implemented in parallel, such as global average pooling, shortcut operation, batch normalization, activation function and full connection operation.

Global mean pooling directly calculates the global mean of the input feature map. The Shortcut operation is equivalent to a matrix addition operation. Batch normalization is to calculate each pixel in the input feature map according to the calculation process of formula (2) and activation function is to calculate the activation function for each pixel in the input feature map, and the fully connected layer can be regarded as a dot product operation of two vectors. Similar to convolution operations, these operations output a set of feature maps. The computation of pixels in the feature map is independent. Therefore, we use the same parallel approach as convolution to design the kernel function. Finally, the overall inference process of the OpenCL-based parallel deep convolutional neural network is shown in Fig. 2.

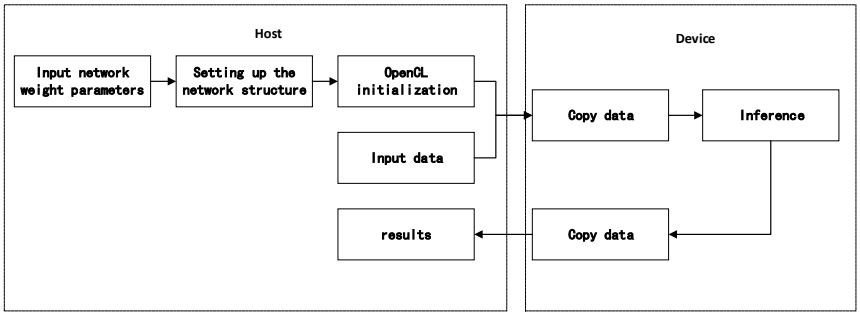

Fig. 2. Inference flow of parallel deep convolution neural network based on OpenCL

3.3 Kernel Fusion and Increasing Global Task

Since the start of the kernel takes extra time, as the number of network layers increases, using kernel frequently would increase more extra consumption. We can use kernel fusion method to reduce the consumption. As Fig. 3 shown, we combine the two operations of BN and Relu, as well as the operations of shortcut and Relu.

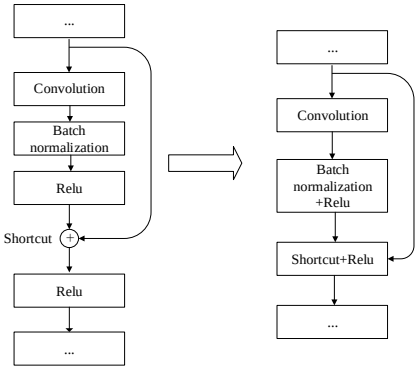

Fig. 3. Schematic diagram of the kernel fusion method

For the kernel fusion of batch normalization and convolution, we can implement it by using the following formula. First, the convolution formula shown as (1).

$$X_{conv} = x \times W + b \tag{1}$$

X_{conv} represents output, x represents input, W represents weights, b represents bias. And the formula of batch normalization shown as (2).

$$X_{bn} = \gamma \times \frac{(x - E[x])}{\sqrt{Var[x] + \varepsilon}} + \beta \tag{2}$$

γ, β are learnable parameters, $E[x]$ represents the mean of x, $Var[x]$ represents the variance of x. ε is a minimal positive number to prevent denominator from becoming zero. We can combine formula (1) and formula (2):

$$X_{bn} = x \times \frac{\gamma W}{\sqrt{Var[x] + \varepsilon}} + \frac{\gamma(b - E[x])}{\sqrt{Var[x] + \varepsilon}} + \beta \tag{3}$$

And then we can use the follows formulas to obtain new weights and bias to replace old ones.

$$W_{new} = \frac{\gamma W}{\sqrt{Var[x] + \varepsilon}} \qquad b_{new} = \frac{\gamma(b - E[x])}{\sqrt{Var[x] + \varepsilon}} + \beta \tag{4}$$

When we obtain new weights and bias and replace old values, we can use them to compute formula (1) directly, and obtain the same results as using old values to compute formula (1) and formula (2). So the batch normalization operation is omitted and has no effect on the results.

In addition to kernel fusion, it can further improve performance by increasing batch size to increase the workload per core. and occupy more GPU resources. Compared to inputting only one image, inputting multiple images at a time can reduce the number of OpenCL kernel functions to start, which will reduce the extra time consumption.

4 Experiment and Evaluations

4.1 Experimental Environment

In order to verify the inference performance of the OpenCL accelerating convolution neural network proposed in this paper, we implemented a deeply separable convolutional neural network and a residual neural network using OpenCL as shown in Table 1. We run the proposed method on NVIDIA and AMD GPU to verify portability. The hardware information is shown in Table 2.

Table 1. The networks used in experiments

CNNs	Convolution layers	Dataset	accuracy
MobileNet v1[14]	27	ImageNet	70.81%
ResNet[15]	21	Cifar 10	91.7%

Table 2. The platforms used in experiments

GPU	Single Precision Peak Performance	Cores	Memory
AMD Radeon Vega Frontier	13.1TFLOPS	4096 cores 1600MHz	16GB 483GB/s
NVIDIA GTX 1070	6.5TFLOPS	1920 cores 1506MHz	8GB 256GB/s

As shown in Table 1, the depth separable neural network is MobileNet v1, which is used to classify the Imagenet dataset. There are 27 convolution layers and one full connection layer. The residual neural network we used has 21 convolution layers. And the network is used to classify cifar10 dataset. In addition, the host side of the NVIDIA platform is an AMD FX-8300 processor with a frequency of 3.3GHz, the host side of the AMD platform is an AMD A10-7870K Radeeon R7 processor with a frequency of 3.9GHz, and the host operating system is Ubuntu16.04. The Caffe version is the Hip version, which is an accelerated Caffe GPU version based on ROCm. ROCm is AMD's general-purpose GPU programming framework. This paper adopts ROCm 2.0. The compiler uses g++ 5.4.0. OpenCL uses OpenCL 1.2.

4.2 Performance Comparison of Depthwise Convolution Operations

Firstly, we compare our proposed method that accelerates depth convolution with Caffe and Diagonal method proposed in[16] . We extract depth convolution from MobileNet v1, and divide them into 9 different layers according to the size of input feature map and

convolution parameters. The results for batchsize=1 are shown in Table 3.

Table 3. Time comparison of different depth convolution parallel methods

	OpenCL(ms)	Caffe(ms)	Diagonal(ms)
Conv2_1dw	0.1488	1.7928	0.2915
Conv2_2dw	0.1461	2.8236	0.3024
Conv3_1dw	0.1531	4.2580	0.6749
Conv3_2dw	0.1005	5.1743	0.4954
Conv4_1dw	0.1420	9.3704	1.0614
Conv4_2dw	0.0991	13.2270	0.7529
Conv5dw	0.0985	27.2045	1.6190
Conv5_6dw	0.0895	28.0888	1.7314
Conv6dw	0.0879	51.3935	4.0700

From the result data in the Table 3, we can see that the proposed method is obviously better than the other two. Caffe's depth convolution performance is the worst, and with the increase of the number of input channels, the serial execution between multiple channels results in low parallel efficiency. Especially in several layers of conv4_1dw to conv6d, the input feature maps become relatively small, result in the matrices converted by im2col become small, too. So it can't use more GPU computing resources. The diagonal method avoids serial execution and directly converts the convolution filters and input data into two large matrices for parallel acceleration. Although it avoids the overhead of serial execution and increases the utilization of GPU hardware resources, it also increases the computational complexity and runtime due to its conversion of deep convolution into traditional convolution.

4.3 Comparison of Parallel DCNN Inference Performance

After the implementation of OpenCL inference to accelerate MobileNet V1 and residual neural network for performance comparison with Caffe. First, we run the forward propagation of MobileNet V1 network and residual neural network on the same GPU through Caffe. The inference time of MobileNet V1 network and residual neural network are shown in Table 4.

Table 4. Parallel acceleration time of MobileNet V1 and ResNet neural network inference

	Caffe(ms)	OpenCL(ms)	Speedup
MobileNet v1	45.9460	7.4873	7.2275
ResNet	8.1670	4.4939	1.8174

According to Table 4, the parallel inference time of MobileNet V1 is 7.4873 ms, which is 7.2275 times faster than that of Caffe GPU, thanks to the acceleration of deep convolution. The parallel inference time of residual neural network is 4.4939 ms, which is 1.8174 times faster than that of Caffe. In order to further optimize the acceleration, we use the method of kernel fusion to eliminate the time proportion of batch normalization

and activation function.

We using kernel fusion combine the kernels of convolution and batch normalization, after this, we continue to combine the kernels of convolution and activation. Finally, the result shown in Table 5 We can see that, after kernel fusion. The performance of MobileNet v1 improves 22.26%, and the performance of ResNet improves 30.95%.

Table 5. Process time before and after kernel fusion

	Before (ms)	**After (ms)**	**Reduced time percentage (%)**
MobileNet v1	7.48729	5.8145	22.26
ResNet	4.4939	3.1031	30.95

After kernel fusion, we can increase the batch size to improve global task load, and to occupy more hardware resources. Compared to classifying only one image at a time, increasing batch size can reduce the overhead of OpenCL kernels when classify the same number images.

Table 6. Inference time of MobileNet V1 and ResNet in different batch size

Bitch	OpenCL GPU		Caffe GPU		Diagonal GPU
Size	MobileNet V1	ResNet	MobileNet V1	ResNet	MobileNet V1
1	5.8145ms	3.1031ms	45.9460ms	8.1670ms	16.5178ms
5	2.1951ms	0.7953ms	40.0650ms	1.6890ms	6.1303ms
10	1.7568ms	0.4330ms	39.6705ms	0.9090ms	5.3144ms
20	1.7178ms	0.2543ms	40.7603ms	0.4940ms	6.0869ms
30	1.6190ms	0.1928ms	40.1693ms	0.3510ms	5.8368ms
40	1.5480ms	0.1652ms	47.8613ms	0.2820ms	5.4615ms
50	1.4891ms	0.1434ms	46.6712ms	0.2500ms	5.1117ms
60	1.4108ms	0.1385ms	45.4217ms	0.2230ms	4.8595ms
70	1.2815ms	0.1273ms	50.3016ms	0.2130ms	3.8701ms
80	1.2642ms	0.1597ms	48.8303ms	0.1960ms	3.8503ms
90	1.2024ms	0.1373ms	48.2878ms	0.1900ms	3.8516ms
100	1.2370ms	0.1494ms	52.8464ms	0.1840ms	3.7961ms

As shown in Table 6, first, with the increase of batch size, the time of inference single image decreases gradually. But when the batch size is above 90, even if the batch size increases, the time for a single image will not decrease any more. Therefore, the MobileNet V1 network performs best when the batch size is 90, It shows that when the batch size is 90, the number of work items can make full use of the hardware computing resources of GPU. At this time, the inference time of an image is 1.2024 ms. Residual network has the best performance when the batch size is 70. The inference time of an image is 0.1273 ms, which improves the performance by 79.32% and 95.90% respectively compared with that before increasing the global task load.

4.4 Performance Comparison of Different Hardware Environments

To verify OpenCL code's portability, we test the performance of parallel deep convolutional neural networks on NVIDIA GeForce GTX 1070. The results are shown in Table 7.

Table 7. Inference time of MobileNet V1 and ResNet on NVIDIA GeForce GTX 1070

Bitch Size	OpenCL implementation		Caffe implementation	
	MobileNet V1	ResNet	MobileNet V1	ResNet
1	3.9697ms	3.0494ms	31.3674ms	7.8862ms
5	2.7944ms	0.9113ms	25.2806ms	1.7380ms
10	2.4275ms	0.6165ms	26.2420ms	0.8253ms
20	1.9580ms	0.3997ms	24.7225ms	0.4764ms
30	1.9634ms	0.2786ms	25.3207ms	0.3343ms
40	1.7546ms	0.2670ms	25.4780ms	0.2991ms
50	1.7213ms	0.2498ms	25.6426ms	0.2583ms
60	1.6903ms	0.2188ms	25.2632ms	0.2440ms

As shown in Table 7, the method also shows good acceleration effect on NVIDIA GPU, combining with the performance of this method on AMD GPU, the OpenCL accelerated deep convolution neural network inference algorithm proposed in this paper has achieved good performance on both AMD GPU and NVIDIA GPU hardware platforms. Compared to Caffe implementation, the performance of MobileNet V1 network has been improved by 40.16 and 14.95 times, respectively. And residual neural network has been improved by 1.67 and 1.11 times, respectively. The OpenCL accelerated deep convolutional neural network inference algorithm proposed in this paper is more effective in accelerating MobileNet v1, and some performance portability on NVIDIA and AMD GPUs.

5 Conclusions

In this paper, we propose an OpenCL-based parallel deep convolutional neural network inference algorithm with kernel fusion to further improve the performance and increase the global task load without affecting the accuracy of the algorithm. The problem of not being able to efficiently utilize different GPU devices to accelerate DNN inference is addressed.

In the future, we will consider optimizing our approach for more hardware, and combine it with kernel auto-tuning methods for automatic parameter tuning.

Funding

This work is funded in part by the Key Research and Development Program of Shaanxi(Program No. 2022ZDLGY01-09), funding: GHfund A (No. 202107014474 ） GHfund C (No. 202202036165), Wuhu and Xidian University special fund for industry-university-rsearch cooperation(Project No. XWYCXY-012021013), and Cloud

Computing Key Laboratory of Gansu Province.

References

1. Guo P.: Multi-institutional collaborations for improving deep learning-based magnetic resonance image reconstruction using federated learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2423-2432. IEEE, Piscataway, NJ (2021).
2. Wang J.: End-to-end object detection with fully convolutional network. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recogni-tion, pp. 15849-15858. IEEE, Piscataway, NJ (2021).
3. Das A.: Enabling On-Device Smartphone GPU based Training: Lessons Learned. In: 2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops), pp. 533-538. IEEE, Piscataway, NJ (2022).
4. Kim S.: Performance Evaluation of INT8 Quantized Inference on Mobile GPUs. IEEE Access 9, 164245-164255 (2021).
5. Wai Y J.: Fixed Point Implementation of Tiny-Yolo-v2 using OpenCL on FPGA. International Journal of Advanced Computer Science and Applications 9(10), 506-512 (2018).
6. Mu J.: Optimizing Opencl-Based CNN Design on FPGA with Comprehensive Design Space Exploration and Collaborative Performance Modeling. ACM Transactions on Reconfigurable Technology and Systems (TRETS) 13(3), 1-28 (2020).
7. Koo Y.: OpenCL-Darknet: implementation and optimization of OpenCL-based deep learning object detection framework. World Wide Web 24(4), 1299-1319 (2021).
8. Dagli R.: Deploying a smart queuing system on edge with Intel OpenVINO toolkit. Soft Computing 25(15), 10103-10115 (2021).
9. Marco V S.: Optimizing deep learning inference on embedded systems through adaptive model selection. ACM Transactions on Embedded Computing Systems 19(1), 1-28 (2020).
10. Dua A.: Systolic-CNN: an OpenCL-defined scalable run-time-flexible FPGA accelerator architecture for accelerating convolutional neural network inference in cloud/edge computing. In: Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 231-231. IEEE, Piscataway, NJ (2020).
11. Lin D L.: Accelerating Large Sparse Neural Network Inference Using GPU Task Graph Parallelism. IEEE Transactions on Parallel and Distributed Systems 33(11), 3041-3052 (2021).
12. He S.: An efficient GPU-accelerated inference engine for binary neural network on mobile phones. Journal of Systems Architecture 117, 102156 (2021).
13. Chen J.: Split Convolutional Neural Networks for Distributed Inference on Concurrent IoT Sensors. In: International Conference on Parallel and Distributed Systems (ICPADS), pp. 66-73. IEEE, Piscataway, NJ (2022).
14. Howard A G.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861, 2017.
15. He K.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770-778. IEEE, Piscataway, NJ (2016).
16. Qin Z.: Diagonalwise refactorization: An efficient training method for depthwise convolutions. In: 2018 International Joint Conference on Neural Networks (IJCNN), pp. 770-778. IEEE, Piscataway, NJ (2016).