BoolHash: A New Convolutional Algorithm for Boolean Activations

Grigor K. Gatchev and Valentin S. Mollov
Department of Computer Systems, Faculty of Computer Systems and Technologies
Technical University of Sofia
8 Kliment Ohridski blvd., 1000 Sofia, Bulgaria
grigor@gatchev.info, mollov@tu-sofia.bg

Abstract – **Integer algorithms, where applicable, can both decrease the memory requirements and improve the speed of the convolutional neural networks (CNN). Boolean activations can further increase the speed gain. Here, we propose a convolutional algorithm called BoolHash. It is based on pre-calculated inference lookup tables (PCILTs). In addition, it uses activation merging to additionally increase the inference speed. We used a CNN with INT16 input weights, INT8 filter weights and boolean activations to compare the speed of BoolHash to that of a classic weight-adder (WA) convolutional algorithm.**

Keywords – b**oolean activations, convolutional neural network, inference speed, integer weights. PCILTs.**

This paper describes an extended version of the base BoolHash algorithm, which was originally reported in [1].

I. Introduction

The convolutional neural networks (CNNs) achieve some of the best artificial neural networks (ANNs) precisions. At the same time, they have lower computing requirements (CRs) than most other ANNs. Their algorithms however involve a lot of arithmetical operations. To achieve better productivity, they are usually run on hardware architectures that feature high-speed ALUs.

A different road to faster work is the usage of algorithms that rely on faster and/or fewer operations. Yet another is using operations that require simpler hardware – it is not only faster, but also uses less on-chip size, thus allowing to fit more processing units on a single die. These approaches can often be combined to allow for faster and cheaper solutions.

II. Overview and Recent Research Reports

Many attempts to reduce the CRs of the CNNs exist. V. Sze et al. [2] categorize them into three classes, based on their design levels: hardware platforms, memory technologies and software algorithms. We focused our work on the second and the third area, while taking into account how they reflect on the first area.

Most research in memory technologies is focused on decreasing the bit width of the processed values (input weights, filter weights, activations, etc), and/or on using integer instead of floating-point arithmetic. Examples are:

- Ilin et al. [3] use 8-bit integer arithmetic (INT8) in image recognition to approximate calculations.
- Truong et al. [4] compares the memory footprint of Integer-Net with a 32-bit floating-point (FP32) implementation and achieves 7x reduction, at the cost of only 2% loss of performance.
- Wu et al. [5] create a NN model with integer-only data: 2-bit (INT2) weights, INT8 activations, INT8 gradients and INT8 errors.
- Das et al. [6] implement AlexNet and other ANNs, using 16-bit (INT16) and 32-bit (INT32) dynamic fixed point values. With these, they achieve improved throughput on Xeon CPUs, while preserving the accuracy of the originals.
- de Bruin et al. [7] implement ANNs on low-end hardware (embedded ARM CPUs) by achieving sufficient quantization on 16-bit CPU accumulators.
- F. Zhu et al. [8] train unified INT8 ANNs and research their gradients. They propose universal techniques for managing these that avoid the direction deviation and the illegal gradient updates.
- Rastegari et al. [9] describe Binary Weight Networks, where filter weights are boolean, and XNOR-Networks, where both weights and activations are boolean. With these, they achieve results equal to or better than AlexNet and a BinaryNet implementation of ImageNet.
- While trying to improve the precision of Binary Weight Networks, Li et al. [10] propose Ternary Weight Networks. There weights can have three possible values instead of two. C. Zhu, et al. [11] add a quantization technique to these.
- X. Lin et al [12] work on the performance of CNNs with binary weights and activations. They improve it by approximating full-precision weights with linear combination of multiple binary weight bases. They also use multiple binary activations to alleviate information loss.
- In [13], Jacob et al. describe an algorithm that quantizes weights and activations down to INT8, and bias vectors down to INT32.
- Gysel et al. [14] create a framework for approximating ANNs while reducing the bit width of their values. They state that it is often able to reduce a network to using INT8 values with a loss of precision smaller than 1%.
- Yu-Chen Lin et al. [15] replace the floating point multiplier in IA-Net with an integer adder. They also target memory reduction through model compression. They also achieve 20% reduction of the inference time.

The efforts on decreasing the CRs of CNNs appear to be focused mostly on researching faster matrix multiplication algorithms. Examples are:

- Mathieu at al. [16] compute convolutions as Fourier pointwise products. They achieve speedup of over a magnitude by reusing the transformed feature map.
- Abtahi et al. [17] increase convolution speed several times by using FFT variants.

978-1-6654-4518-4/21/$31.00 ©2021 IEEE

- Chitsaz et al. [18] point that splitting solves some problems in FFT computation with small kernels, like the ones in a typical CNN.
- Lavin et al. [19] propose a family of algorithms, based on Winograd's minimal filtering, that use fast matrix multiplication. They reduce the CNN multiplications up to 2.25 times.
- In [20], Ju et al. analyze many fast convolution algorithms, presenting them as formal bilinear ones. They show that the overlap-add and Winograd family algorithms rival the accuracy of FFT while avoiding complex arithmetic. They present a corollary for the minimum rank of a bilinear algorithm for linear convolution, and present algorithms that achieve it.
- Kim et al. [21] test AlexNet versions on GPUs and find that both FFT and Winograd / Toom-Cook methods are up to 4x faster than the direct multiplication (DM) method. (We believe that this might not be true on custom ASICs, due to the bigger and more complex circuitry required by these methods.)
- In [22], Sifre introduces separable convolution. Lebedev et al. use in [23] spatially separable convolution, and increase speed with only a small precision loss, through not using some filters.
- Chollet [24] and Ghosh [25] describe depthwise convolution and show that it avoids some limitations of the spatially separable convolution.
- Lebedev et al. and He et al. [26] apply decomposition (CP- and depth-wise, respectively) to speed up separable convolution.

III. Considerations

Comparing the hardware energy consumption and on-chip area of INT8 and FP32 operations, Daily [27] finds that the difference in speed is 30x for addition and 18.5x for multiplication, and the difference in on-chip area is 116x for addition and 27x for multiplication, in favor of INT8. Assuming an addition-based DM algorithm variant and integer-only ALU, we deduce on this base that an ASIC implementing an INT8-based CNN might be over 300x faster than an ASIC that implements an FP32-based CNN.

Jacob et al. [13] observe that on modern hardware with pipelined instructions addition instructions are not faster than multiply/add instructions. However, in a custom ASIC an addition-only circuitry will inevitably be faster and smaller on-chip than a multiply/add circuitry.

Rastegari et al. [9] show that boolean filter values permit using addition instead of multiplication as a convolutional operation. Further on, they show that having also boolean activations permits using bitwise operations in convolution. This results in a very fast algorithm with acceptably precise results in some tests. Li et al. [10] also use addition instead of multiplication in ternary weight networks.

In [28], Ko et al. conclude that substantial bit width decreases can severely degrade performance. However, Zhou et al. [29] design an incremental network quantization algorithm that achieves significant bit width decrease without performance loss, at the cost of being significantly more complex. The biological neural networks (BNNs), whose functionality the ANNs imitate, achieve in many neurons a fine-grainedness of the input weighting that is equivalent to bit width of 8 or more. Some of them also achieve an equivalent of an activation bit width of 4 bits or more, mostly through varying the frequency of their spikes.

Due to this, we believe that boolean input and/or filter weights might be insufficient for most CNN tasks, especially when combined with boolean activations. However, we also note that activation bit width over 1 can often be replaced with higher bit width of the filter and/or the input weights, and possibly with a higher connectivity (eg. number of neurons in a layer), without degrading the results precision.

Many biological neurons and even some "layers" in BNNs have effectively boolean activations. This makes us conclude that for some tasks, sizable parts of ANNs or even entire ANNs can rely on boolean activations without compromising the task they perform.

Using boolean activations also eliminates the need for ReLU layers. This improves the network speed and memory footprint, and simplifies its overall algorithm, making it easier to design an ASIC for. Due to this, boolean activations would be the preferred choice where they can achieve satisfactory results.

IV. Our Model

To improve the inference speed, we designed an experimental CNN model with INT16 input weights, INT8 filter weights and boolean activations. Despite that Krishnamoorthi [30] concludes that input weights are quantizable to 4 bits only, we opted for the largest bit width in them, in order to keep the activations boolean and the size of the filter weights as low as possible (see Part V for the reasons). Our objective was to use as convolutional operation integer-only addition. This would allow us to keep the computational load as low as possible, since this operation makes the most calculations in a CNN network. At the same time, it would preserve sufficient inference base bit width to be able to achieve acceptable precision in many types of tasks. We left open the possibility to add and test other algorithms too.

To preserve the potential ASIC simplicity, we opted also for boolean-only backpropagation / gradient signals, taking the risk of decreasing the backpropagation efficiency and hence the training precision.

An exception from these constraints is that the input layer can process test data with higher bit width. Also, the output layer can produce activations with higher bit width (up to INT64), to allow for better evaluation of the results.

We wanted to test our designs in a strictly integer system, avoiding any possible involvement of floating-point operations and thus affecting the precision. To that goal, we implemented our model in C++ from the ground up, without defining floating-point data types anywhere in the code, or using any external code that could possibly contain floating-point operations.

Seeking for faster convolutional algorithms that benefit specifically from boolean activations, we designed and suggest here an algorithm that we named BoolHash.

V. The BoolHash Algorithm

A. Description

BoolHash combines two algorithmic approaches that so far have been used rarely and separately. One is the usage of look-up tables – in our case, for inference results. The other is treating the activations block as a bitstream that can be separated into processed values in any way that will improve the inference speed.

When using boolean activations, filter weights are added (instead of being multiplied) to the inference function dot result (DR) where the matching receptive field (RF) activation is true (1), and are skipped where the activation is false (0). Thus, the algorithm falls into the group of the weight-adders.

A filter is divided into segments, having N values each. In a receptive field, the activations that match the filter values in a segment are used as bits in an N-bit index. (Figure 1).

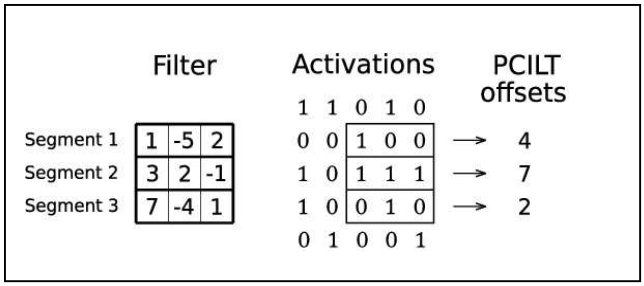

Fig. 1. Filter and RF segments. Calculating PCILT offsets.

This index points to a pre-calculated filter weights sum (FWS), of the segment weights that match the "true" activation values. The FWSes for a segment are held in a pre-calculated inference lookup table (PCILT).

After being located by this index, the FWS for each segment is added to the DR. Thus, the DR is calculated by N times less additions than in the classing weight-adder algorithm. (Figure 2).

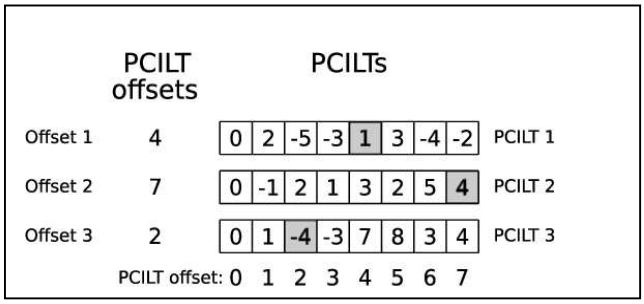

Fig. 2. Finding FWSes in filter PCILTs by PCILT offsets

Every filter segment has its own PCILT, containing the sums for the segment weights matching all possible combinations of activations values (Figure 3). Calculating the PCILTs is done only once in the filter lifetime, so the speed overhead it adds to the data processing is negligible.

In pre-trained CNNs the PCILTs can be saved in a permanent memory that allows fast look-ups. Also, the PCILT calculation code can be omitted there, decreasing the memory requirements.

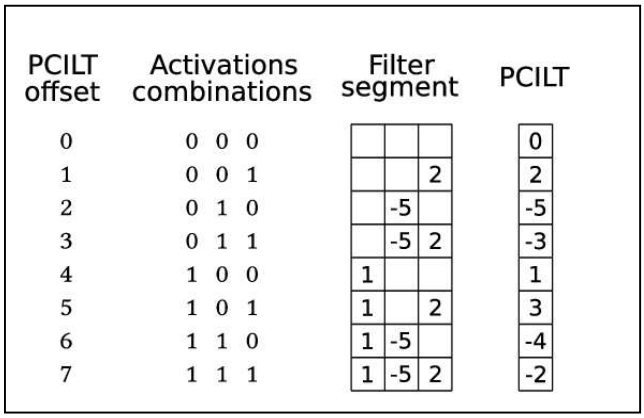

Fig. 3. Calculating a PCILT by a filter segment.

If the filter segments of the different neurons match, the activation values combinations (AVC) for every RF segment can be reused in all neurons that process the same activations block – typically all neurons in a layer. In this case, the AVCs are best calculated when the activations block enters the convolutional layer. The overhead of this is usually much smaller than the economy on calculations of the algorithm. In large CNNs with hundreds or thousands of neurons in a layer, it is negligible.

Calculating PCILTs needs only an adder, and AVCs are calculated through bit masking and shifting. A BoolHash-specific ASIC can use for these operations circuitry that is faster and takes much less on-chip space than a multiplication-able ALU.

B. Disadvantages

BoolHash can require substantially more memory than algorithms like DM, separable convolutions or even FFT. The PCILT size is 2^N: for N=8, a PCILT will have 256 values. Also, a PCILT value might require more memory than a filter weight.

Pre-calculating all AVCs also increases N times the amount of memory needed for the activations. This can be alleviated by the fact that many types of hardware keep for optimal performance every data value in a separate byte, or even a combination of bytes. Thus, even a single data bit might need to be stored in 8, 16, 32 or 64 bits of RAM for optimal speed. Combining the data needs in such cases much less extra memory.

The memory expense importance depends on the specific hardware. BoolHash would be a wrong choice for systems with good arithmetic speed and limited memory. It can also be slow on systems with small RAM cache, if the PCILTs size significantly exceeds it – that would increase the cache swapping frequency. Tuning the algorithm to minimize the cache swapping might alleviate this problem.

BoolHash implementations over suitable ASICs would best integrate logic and PCILT memory as closely as possible. For example, they can calculate an RF inference by having a number of units, equal to the filter segments number, each of them using its AVC as an address in a local fast memory, containing its PCILT.

C. Overcoming the memory expense

PCILT memory can be reduced by decreasing the N factor, seeking a task-specific optimal balance between speed gain and increased memory usage. For example, using N=4 instead of 8 will require PCILT memory for only 32 values instead of 256, and will process 8 activations / filter weights with 2 additions instead of 1 (but still not with 8 additions). This, it will need 8 times less PCILT memory at the cost of 2 times slower inferring process.

For some filters the number of the possible FWSes might be much smaller than the number of the possible weight combinations. (For example, with N=8 there will be 256 possible weights combinations. However, if weight values repeat often, as is the case in many popular filters, there might be only 15-20 different sums of them.) If a FWS has higher bit width than is needed for a PCILT pointer, using an intermediary table with pointers to a PCILT that contains only the unique FWSes might bring a reduction of the memory used, at the cost of an additional indirection.

In addition, in large CNNs a neuron often has many filters. Usually the range and the diversity of the weight values in most of these are similar – their unique FWSes will overlap significantly. Moreover, some filters often repeat across many of the neurons in a layer. This makes it possible to use a common unique PCILT for all filters in a neuron, or even in a layer, decreasing further the memory it needs.

In many cases, filters can be divided into segments in such a way that many of their PCILTs will repeat – within the same filters, and / or between different filters in a neuron, layer or even the entire CNN. Segments that are identical by values and their order will have identical PCILTs. These can

be replaced with pointers to an unique one. This can reduce the number of actual PCILTs in a CNN, leading to a decrease of the memory needed. It is best expressed with filters of a low actual cardinality. For example, an arbitrarily big CNN, having INT16 filter weights with actual cardinality of 32 and N=4, will need only about 64 MB PCILT memory. This is comparable with the RAM cache size for the best CPUs, as of 2021, and would be easy to provide in a CNN ASIC. (It however might require filter-specific AVCs, the generation of which might require more resources.)

Another way to decrease the memory needs is to consider the combined bit width of all filter weights in a segment. If the weights are INT7 and N is 8 (3 bits), a FWS value will need 10 bits, which would be rounded in most systems up to two bytes. However, if the filter weights are INT5 and N is 8, the memory needed by a FWS value will be 8 bits – one byte only. (That is why we used in our model larger bit width in the input weights and smaller one in the filter weights.)

D. Additional improvements and variants

With sufficient PCILT memory and small filters, one segment can cover an entire filter (Figure 4). This makes possible to obtain DR by a single memory addressing, without using – or having at all – an ALU.

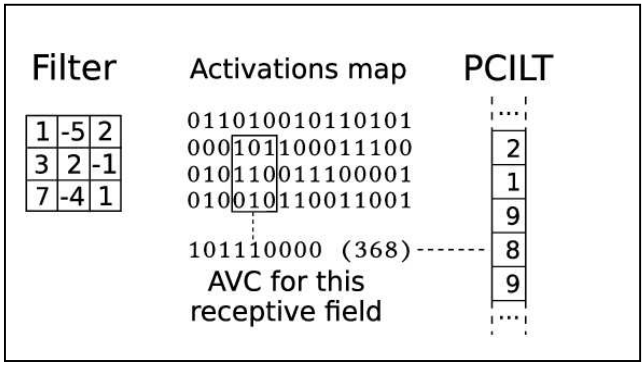

Fig. 4. A filter covered entirely by one segment, obtaining the inference result for an entire RF by just one PCILT access.

On different hardware, different filter segment layouts might ensure better speed than others. For example, on some hardware the inference might be faster if FWSes span columns, and on another if they span rows.

The value positions in a filter segment do not have to be consecutive. This allows implementing sparse filters that skip from processing any irrelevant RF positions or regions, increasing further the inference speed. AVCs are shared between same-input filters, and different filters might emphasize on different data regions, which limits the irrelevant percentage, require calculating different AVCs for different filters or sets of filters, etc. However, most types of data usually have a substantial irrelevant percentage, and most CNN layers have many filters that repeat between neurons. Thus, the processing speed might benefit from this ability of BoolHash.

Some value positions can be included in more than one filter segment, or even more than one time in the same segment. This allows weighting some RF positions or regions beyond what a limited filter weights range can achieve, at the cost of slight processing delay. Thus it allows limiting the weights range, which can decrease the memory needed by BoolHash. It also allows having different weight bit width for different filter positions.

Different segments in a filter may use different weight ranges and/or count of values per segment. This can be used to increase the speed and decrease the memory usage, esp. with non-consecutive value positions, or RF regions needing different weighting range.

The PCILT values can incorporate the input weight too, removing the need to multiply the DR by it. During the NN learning phase this will require modifying them during the backpropagation adjustment. Such a variant will be slower and more complex than modifying one input weight only. A suitable algorithm for modifying only relevant values in PCILTs might limit the slowing and potentially can make the weight adjustment more selective, where that is needed or acceptable.

This variant has much more modifiable weight values space than the DM algorithm. The difference with the separate convolution algorithms is even greater. This creates a potential for achieving better precision. (The redundancy there would probably be high. However, it can be removed post-learning, or even during learning through suitable methods for network compression.) Importantly, the weights space enlargement does not carry a proportional increase of the calculation load, unlike the other convolution algorithms – to the contrary, it decreases the computational load.

Another benefit is the ability to have segments that include weight values from different input filters. While misusing this might degrade the learning, using it correctly can allow for memory economy and increased speed.

By combining sparse, overlayed and/or multi-filter segments, BoolHash can perform through a single fast filter tasks that in some cases need more complex and / or slower processing (Figure 5).

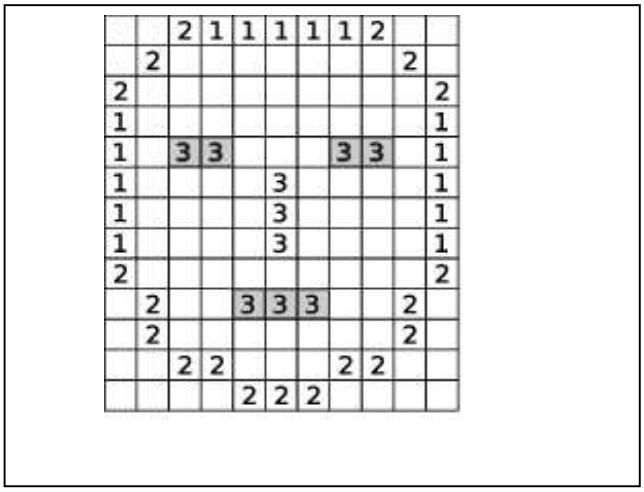

Fig. 5. A filter with INT2 weights,
having both sparse and overlayed segments.
Zero weights (not shown) are not processed, increasing speed.
Weights with gray background are included in FWSes more than once, increasing their weight beyond the INT2 range.

The algorithmic base of BoolHash – using PCILTs with combined activations – is not limited to CNNs with boolean activation. It can work with integer activations and any filter weights. With non-boolean activations however it will need multiplication as a convolutional operation. Its only limitation is the amount of memory that might require. Activations with too big cardinality might require prohibitively big PCILTs, and/or might slow processing by increasing the RAM cache swapping.

BoolHash is also not limited to integer-only CNNs: it works equally well with floating-point filter weights. With them, it should bring even bigger speed improvements, but also will increase the memory requirements.

BoolHash can be used at the first stage of both spatially separable convolutions and depthwise convolutions. A BoolHash-specific CNN ASIC, will likely implement that as two successive operations in the separable convolution inference process, each using its own circuitry. This can help parallelizing the inference operations, and additionally speeding them up.

VI. Results

We compared the speed and the memory requirements of BoolHash and the classic DM algorithm in our implementation on 5 different general-purpose CPUs (Intel Core i7-3770K, Intel Xeon E5-2690, Intel Xeon E5-2450L, Intel Xeon E31220 and AMD A8-5600K). For tests fitting in their Level 1 RAM caches, all of them deviated from the average results by less than 5%. We attribute the deviations to differences between the CPUs in the size and the policies of the Level 2 and 3 RAM cache, and of the ALU and RAM access. Bigger deviations were observed where RAM usage exceeded the Level 1 cache size.

We used as data sources the MNIST Handwritten Digits database (60,000 images, 28 x 28 pixels each) [31]. One segment per filter row was used, no sparsity or overlaying. Three layers were alternated between the two algorithms. The PCILT and AVC creation overhead was found to be relatively small – between 10% and 25% of the speed gain.

Table 1. BoolHash speed and memory requirements, compared to a classic weight-adder algorithm

Filter size		8x8	7x7	6x6	5x5	4x4	3x3
BoolHash N		8	7	6	5	4	3
Speed gain, times	4x6x8 neurons	6.59	5.71	4.78	3.87	2.92	2.15
	20x30x 8 neurons	5.47	5.96	5.24	3.64	3.15	2.17
Filter memory usage, times		64.0	36.6	21.3	12.8	8.0	5.3

Table 1 shows how BoolHash compares to the classic algorithm in the processing speed and the memory requirements, only for the convolutional operations. In the N=8 filter column and the row for 20 x 30 x 8 neurons can be seen a speed dip, caused by the PCILTs size exceeding the RAM cache.

Specialized hardware might bring different results. We expect that a BoolHash-specific CNN ASIC, when compared to a DM/WA-specific CNN ASIC, will achieve at least the same speed advantage as our tests do, and a greater advantage when using floating-point weights.

VII. Conclusions

Based on our theoretical analysis and the results from the practical tests, we conclude that BoolHash shows significantly better inference speed than the classic (DM/WA) style algorithm in CNNs with boolean or integer activations, and can also be used to speed up separable convolutions algorithms. We also conclude that it can easily be implemented in custom ASICs, requiring less circuitry than the classic DM/WA algorithm. The speed gain grows with the N value, at the cost of higher memory requirements, allowing for choosing a task-specific optimal balance.

CNNs that use data and weights of limited cardinality usually have a small memory footprint, so the required memory will often be within acceptable limits. We also describe methods to decrease the memory requirements of BoolHash. We note also that adding some extra memory into a hardware platform is often justified by the speed gain over the system lifetime.

Finally, BoolHash allows in some cases implementing through a single fast-working filter some functionalities that are usually achieved in more complex and slow ways.

VIII. Acknowledgements

The authors would like to thank all reviewers for their constructive comments and valuable suggestions and the Science Research Sector of the Technical University – Sofia for the financial support to report this paper.

IX. References

[1] G. Gatchev, V. Mollov, “Integer convolutional neural networks with boolean activations: the BoolHash algorithm”, 2020 European Conference on Circuit Theory and Design, doi:10.1109/ECCTD49232.2020.9218306.

[2] V. Sze, Y. Chen, T. Yang and J. S. Emer, “Efficient processing of deep neural networks: A tutorial and survey”, Proceedings of the IEEE, vol. 105, no. 12, pp. 2295–2329, Dec 2017, doi:10.1109/JPROC.2017.2761740.

[3] D. Ilin, E. Limonova, V. V. Arlazarov and D. Nikolaev, “Fast integer approximations in convolutional neural networks using layer-by-layer training”, Research Gate, 103410Q. 10.1117/12.2268722, doi:10.1117/12.2268722.

[4] N. D. Truong et al, “Integer convolutional neural network for seizure detection”, IEEE Journal of Emerging and Selected Topics in Circuits and Systems, vol. 8, no. 4, pp. 849-857, doi:10.1109/JETCAS.2018.2842761.

[5] S. Wu, G. Li, F. Chen and L. Shi, “Training and inference with integers in deep neural networks”, arXiv:1802.04680.

[6] D. Das et al, “Mixed precision training of convolutional neural networks using integer operations”, arXiv:1802.00930v2.

[7] B. de Bruin, Z. Zivkovic and H. Corporaal, “Quantization of constrained processor data paths applied to convolutional neural networks”, in: 2018 21st Euromicro Conference on Digital System Design (DSD), 2018, pp. 357–364, doi:10.1109/DSD.2018.00069.

[8] F. Zhu et al, “Towards unified INT8 training for convolutional neural network”. ArXiv:1912.12607.

[9] M. Rastegari, V. Ordonez, J. Redmon and A. Farhadi, “XNOR-Net: ImageNet classification using binary convolutional neural networks”, EECV 2016 Proceedings, Part IV, pp. 525-542, doi:10.1007/978-3-319-46493-0_32.

[10] F. Li, B. Zhang, B. Liu, “Ternary weight networks”, arXiv:1605.04711.

[11] C. Zhu, S. Han, H. Mao and W. Daily, “Trained ternary quantization”, arXiv:1612.01064.

[12] X. Lin, C. Zhao and W. Pan. “Towards accurate binary convolutional neural network”, Advances in Neural Information Processing Systems 30 (NIPS 2017)..

[13] B. Jacob et al, “Quantization and training of neural networks for efficient integer-arithmetic-only inference”, 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition”, Salt Lake City, UT, 2018, pp. 2704-2713, doi:10.1109/CVPR.2018.00286.

[14] P. Gysel, J. Pimentel, M. Mohammad and S. Ghiasi, “Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks”. IEEE Transactions on Neural Networks and Learning Systems, vol. 29, no. 11, pp. 5784-5789, doi:10.1109/TNNLS.2018. 2808319.

[15] Yu-Chen Lin, Yi-Te Hsu, Szu-Wei Fu, Yu Tsao and Tei-Wei Kuo, “IA-NET: Acceleration and compression of speech enhancement using integer-adder deep neural network”, Research Gate, 10.21437/Interspeech.2019-1207, doi:10.21437/Interspeech.2019-1207.

[16] M. Mathieu, M. Henaff, Y. LeCun: “Fast training of convolutional networks through FFTS”, International Conference on Learning Representations (ICLR), 2014.

[17] T. Abtahi, C. Shea, A. Kulkarni, and T. Mohsenin: “Accelerating convolutional neural network with FFT on embedded hardware”, IEEE Trans. Very Large Scale Integr. Syst., vol. 26(9), pp.1737-1749, 2018, doi:10.1109/TVLSI. 2018.2825145.

[18] K. Chitsaz, M. Hajabdollahi, N. Karimi, S. Samavi, S. Shirani: “Acceleration of convolutional neural network using FFT-based split convolutions”, 2020, arXiv:2003.12621..

[19] A. Lavin, S. Gray: “Fast algorithms for convolutional neural networks”, arXiv:1509.09308, doi:10.1109/CVPR.2016.435.

[20] C. Ju, E. Solomonik: “Derivation and analysis of fast bilinear algorithms for convolution”, 2020, arXiv:1910.13367, doi:10.1137/19M1301059.

[21] H. Kim, H. Nam, W. Jung, J. Lee: “Performance analysis of CNN frameworks for GPUs”, in 2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), IEEE, 2017, pp. 55–64, doi:10.1109/ISPASS.2017.7975270.

[22] L. Sifre: “Rigid-motion scattering for image classification, PhD, 2014, https://www.di.ens.fr/data/publications/papers/phd_sifre.pdf, arXiv:1403.1687.

[23] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, V. Lempitsky: “Speeding-up convolutional neural networks using fine-tuned CP-decomposition”, 24 Apr 2015, arXiv:1412.6553.

[24] F. Chollet: “Xception: Deep learning with depthwise separable convolutions”, 4 Apr 2017, arXiv:1610.02357, doi:10.1109/CVPR.2017.195.

[25] T. Ghosh: “Towards a new interpretation of separable convolutions”, 16 Jan 2017, arXiv:1701.04489.

[26] Y. He, J. Qian, J. Wang: “Depth-wise decomposition for accelerating separable convolutions in efficient convolutional neural networks”, 21 Oct 2019, arXiv:1910.09455.

[27] W. Daily, “High-performance hardware for machine learning”, 2015, Tutorial, NIPS..

[28] J. H. Ko, J. Fromm, M. Philipose, I. Tashev, and S. Zarar, “Precision scaling of neural networks for efficient audio processing”, 2017, arXiv preprint arXiv:1712.01340..

[29] A. Zhou, A. Yao, Y. Guo, L. Xu and Y. Chen, “Incremental network quantization: Towards lossless CNNs with low-precision weights”, in Proc. ICLR, 2017..

[30] R. Krishnamoorthi, “Quantizing deep convolutional networks for efficient inference: A whitepaper”, arXiv:1806.08342.

[31] Y. LeCun, C. Cortes and C.J. C. Burges, The MNIST Database of Handwritten Digits, http://yann.lecun.com/exdb/mnist/ .