The Key Technologies of Embedded AI

Subjects: Engineering, Electrical & Electronic

Contributor: Zhaoyun Zhang, Jingpeng Li

Advancements in artificial intelligence algorithms and models, along with embedded device support, have resulted in the issue of high energy consumption and poor compatibility when deploying artificial intelligence models and networks on embedded devices becoming solvable.

Keywords: artificial intelligence ; embedded intelligence ; model compression

1. Introduction

Over the years, the development of artificial intelligence and its applications has greatly reduced the complexity of many machine learning models, making it easier to deploy them on resource-constrained devices. Furthermore, corresponding support for models and algorithms on these devices has emerged. These developments have facilitated a new research direction: embedded artificial intelligence [1][2][3]. The concept of embedded AI was first introduced in reference [3], which proposed that the IoT could evolve into the Wisdom Web of Things (W2T) and emphasized that embedded intelligence about individuals, the environment, and society could increase the number of users of existing IoT systems, promote environmental sustainability, and enhance social awareness. Recent developments in embedded AI are described in references [1][2], both of which combine embedded AI with IoT technology. The current mainstream research direction of embedded AI is to integrate it with IoT, which includes edge computing with convolutional accelerator [4] and load distribution [5]. Reference [6] mentions that the combination of embedded intelligence and IoT is the future direction of development. In addition, edge computing can be combined with artificial intelligence, resulting in what is known as edge intelligence [7].

The current development of embedded AI is two-way: the optimization of AI models and algorithms reduces the difficulty of deploying them on embedded devices, while hardware accelerators in embedded devices increase support for AI models and algorithms. Furthermore, hardware resources are being developed, and AI is rapidly advancing in mobile devices. For example, reference [8] describes the deployment of neural networks on cell phones, and there are also neural networks specifically designed for mobile devices, such as MobileNet [9]. MobileNet will be discussed in detail in Section 3 on lightweight networks.

2. Model Compression of Neural Network

2.1. Network Structure Redesign

The network structure design is a method of improving existing neural networks by designing new network structures. Many researchers have undertaken significant work in this area.

In 2017, Landola et al. [10] proposed SqueezeNet, a lightweight network that maintains accuracy using fewer parameters. SqueezeNet consists of two parts: a convolutional neural network architecture designed by the authors, and the Fire module. To maintain accuracy, three strategies were used in designing the convolutional neural network architecture: 1. using 1 × 1 filters instead of partial 3 × 3 filters, 2. using a squeeze layer to reduce the input channels of 3 × 3 filters, and 3. delaying downsampling (postponing the downsampling process to the end of the network).

In 2017, Howard et al. [9] proposed MobileNet, a lightweight network for mobile and embedded vision applications. This network introduced two global hyperparameters, α (Width Multiplier) and ρ (Resolution Multiplier), that can be balanced in terms of latency and accuracy. The core components of MobileNet include depthwise separable convolution, width multiplier, and resolution multiplier. The standard convolution is decomposed into a pointwise convolution and a depthwise convolution, where the depth convolution is connected after the input channel, and the pointwise convolution is connected between the depth convolution and the output.

Depthwise separable convolution separates the filtering and merging functions in standard convolution, using one layer for filtering and another layer for merging. This decomposition method can significantly reduce computational effort and model size. In MobileNet, when targeting specific applications that require smaller models and lower computational costs, the depth and point convolutions are computed as separate network layers, resulting in a total of 28 layers. In 2018, Zhang et al. [11] proposed ShuffleNet, a lightweight network for devices with computational constraints. The network architecture employs pointwise group convolution and channel shuffle techniques to significantly reduce computational costs while maintaining accuracy. Based on the efficient depth-separable convolution or group convolution of Xception [12] and ResNeXt [13], ShuffleNet considers 1 × 1 group convolution. The feature map generated for each group in the previous layer is divided into separate channels for several subgroups in each group to provide different subgroups for each group in the next layer.

Reference [14] proposed the once-for-all network (OFA), which can reduce training costs by selecting dedicated subnetworks without additional training. Additionally, the authors proposed a new progressive shrinkage algorithm for training OFA, which is a generalized pruning method. Experimental results showed that this method outperformed state-of-the-art NAS methods at the time and effectively reduced energy consumption. In another work [15], the network structure of Yolov5 was redesigned by embedding three network modules: CBAM, C3Ghost, and Ghost. The CBAM module was used to enhance the feature extraction capability, while C3Ghost and Ghost modules were used to reduce the number of parameters and floating-point operations. The MS COCO and PASCAL VOC datasets were used for experiments, and the results show that the new network structure had a slightly decreased average detection time, as well as a 50.61% reduction in floating-point operations and a 47.88% reduction in model parameters compared to the original network structure.

MobileNet has two more hyperparameters than SqueezeNet, which makes the application more flexible and able to be adjusted according to the actual application for computational cost and latency. ShuffleNet can generate more feature maps than SqueezeNet, but its deep convolution operation can only be performed on the bottleneck feature map, which leads to some difficulties in deployment on low-power devices. Additionally, most network structures are designed to adjust convolutional operations by replacing complex kernels with simpler ones to reduce computational complexity and the number of model parameters, while maintaining accuracy and reducing energy consumption. However, some network structures may have poor generalization ability and are only applicable in specific scenarios.

2.2. Quantization

Quantization is the compression of floating-point data bits in neural network parameters to reduce model complexity and size by reducing the number of bits used by floating-point numbers, while maintaining model accuracy as much as possible.

In reference [16], BRECQ, a framework for Post-training Quantization (PTQ), was proposed for the first time to limit the bit-width range of the post-training quantization task to INT2. The authors conducted a comprehensive theoretical study of second-order errors and found that the framework was able to balance cross-layer dependence and generalization errors. They also used approximate inter-layer and intra-layer sensitivity, incorporating hybrid precision techniques. Experimental results show that post-training quantization can obtain a model with similar precision to ResNet and MobileNetV2 with only four bits using the Quantization-Aware Training (QAT) method without additional conditions, and can obtain 240 times the production speed of the quantized model. In another work [17], the authors propose data-free quantization methods that do not require data, fine-tuning, or hyperparameter optimization. They suggest a method that uses the scale-equivalence property of the activation function to adjust the range of weights in a network and corrects the errors introduced in the quantization process. Experiments show that the data-free quantization method approaches the original model’s accuracy and is even comparable to more sophisticated training-based methods. The authors of reference [18] propose a mechanism for weight-rounding for post-training quantization, AdaRound, which does not require fine-tuning of the network and can cope with data and task loss by using only a small amount of unlabeled data. Experiments show that this mechanism maintains accuracy loss within 1% by quantizing the weights of ResNet-18 and ResNet-50 to 4 bits. Reference [19] describes the “deep compression” method, a comprehensive approach that is a three-stage pipeline of pruning, training quantization, and Huffman (Huffman) coding running together. This approach can reduce the storage requirements of neural networks by a factor of 35 to 49 without compromising accuracy. The principle of the method is to first learn the significant connections as the basis for pruning the network, then quantize the weights to achieve weight sharing, and finally apply Huffman coding. Before Huffman coding, the authors retrained the network to fine-tune the remaining connections and quantized centroids, and reduced by nine to thirteen times the connections through pruning. Quantization reduced the connection bitwidth from 32 bits to 5 bits. Experimental results show that the deep compression approach reduces the storage space required for AlexNet by a factor of 35 from 240 mb to 6.9 mb on the AlexNet dataset with no loss of accuracy, and achieved 3 to 4 times the layered acceleration and 3 to 7 times the energy efficiency on CPU, GPU, and mobile GPU benchmarks. Finally, the authors of reference [20] propose the Efficient Inference Engine (EIE), which can be deployed to SRAM (Static Random-Access Memory) platforms. This engine utilizes the sparsity of activation functions and weights, and the technique of weight sharing and quantization. The EIE can save 120 times the energy, respectively, 10 times the energy by using sparsity, 8 times the energy by weight assignment, and 3 times the energy by skipping zero activation functions using ReLU. Unlike the large deep neural networks trained by the "deep compression" method, the EIE is suitable for on-chip DRAM (Dynamic Random-Access Memory).

Despite the various quantization methods mentioned above, there is still a need for better quantization methods that can achieve higher compression rates with lower accuracy degradation. Additionally, it is crucial to consider the energy consumption of the compressed network model to make it suitable for use in resource-constrained devices such as embedded devices.

2.3. Pruning

Pruning is a method used to reduce redundant data in the neural network by determining the importance of each unit and removing unimportant parts.

One pruning method proposed in the literature [21] consists of three steps: first, train the network to learn the important connections; second, prune the unimportant connections; and finally retrain the network to adjust the weights of the remaining connections. Experimental results have shown that reducing the number of parameters by a factor of nine in the AlexNet network structure does not have a significant impact on performance. Another weight pruning method, ProbMask, was proposed in reference [22]. It measures the importance of weights by the probability of global criteria in all network layers and features automatic learning by setting constraints. Experimental results show that the ProbMask method can improve top-1 accuracy by about 10% compared with existing methods. Reference [23] proposes the ManiDp method, which maximizes the dynamic mining and pruning of redundant filters by embedding the manifold information of all instances into the pruned network space. The method achieves the dynamic removal of redundant filters, and experimental results show that it can reduce the number of floating-point operations by 55.3% while decreasing the top-1 accuracy by only 0.57% when applied on ResNet-34. A new channel exploration method, CHEX, is proposed in the literature [24] to solve the problem that traditional pruning methods require fully pre-trained large models and are limited by the pruning process. CHEX repeatedly prunes and regrows the channels during the training process, reducing the risk of prematurely pruning important channels. Experimental results show that a top-1 accuracy of 76% can be obtained using the CHEX compressed ResNet-50 model on the ImageNet dataset, reducing the Flops (floating point operations per second) to only 25% of the original ResNet-50 model. Finally, reference [25] proposes a channel pruning method that uses a random search method to determine the channel model of the pruned network structure. Experimental results show that the performance of the models obtained with different network structures and datasets is close under the random pruning scheme. However, the number of parameters has a great impact on the accuracy of the constructed networks, and the more parameters the lower the error rate of the pruned network after a certain amount of computation.

In conclusion, current pruning methods include weight pruning, channel pruning, and neuron pruning, each with its advantages and disadvantages. The criteria for determining the importance of the unit can impact the accuracy. The use of constraint learning methods can improve it, although the implementation is complex. The random search method of pruning is simple to implement, but the network model's performance is limited compared to other methods that improve it. Therefore, each method has its advantages and needs to be selected in conjunction with the actual application scenario, and there is no single method that can synthesize the complexity and model compression efficiency.

3. Binary Neural Networks and Optimization Techniques

Convolutional neural networks consist of multiple network layers and millions of parameters. Due to their large size, it is challenging to deploy them directly on embedded devices that have high hardware requirements.

To address this issue, a binarization method was proposed to simplify the network parameters [26]. This method quantifies the weights and activation values into one fixed-point parameter, leading to memory savings and reduced inference time. However, the binarization network results in severe information loss. The direction of binarization network research is towards reducing information loss, reducing errors, and preserving model accuracy. The authors of reference [27], Courbariaux et al., proposed the concept of binarized neural networks and used a randomized binarization method during the training forward propagation. During the backward propagation, a clip function was introduced to intercept the full precision weight to update the range, compressing the number of network model parameters to a great extent and preventing the real weights from growing too fast without affecting the binary weights. One year later, after the Binary network was proposed, reference [28] proposed Binary-Weight-Networks and XNOR-Networks. The filters of binary-weighted networks approximate binary values, and XNOR-Networks filters and convolutional layer inputs are binary, which accelerates the speed of convolutional operation and saves 32 times the memory. The experiments using the ImageNet dataset show that the method improved the top-1 accuracy by 16% compared to other network binarization methods used at that time. Reference [29] presents the first hash method training binary method and experiments on CIFAR-10, CIFAR-100 with ImageNet dataset. The main work of the authors was to convert the training binary network into a hash problem, multiply the binary code by a scaling factor to reduce the loss caused by using the hash method, and propose alternate optimization methods to iteratively update the binary code and the scaling factor. Experimental results show a 3.0% improvement in accuracy for the ImageNet classification task compared to the best binarization network at that time. Reference [30] proposes Center-Symmetric Local Binary Convolutional Neural Networks (CS-LBCNN) for handwritten character recognition to address the problem that local binary networks are affected by randomly assigned local binary convolutional weights. The authors also propose an improvement—Threshold Center-Symmetric Local Binary Convolutional Neural Networks (TCS-LBCNN). The experiments were compared in CS-LBCNN and TCS-LBCNN using bilingual, MNIST, and MADBase datasets, and the average accuracies obtained in CS-LBCNN were 99.465%, 99.596%, and 99.502%, respectively. The final average accuracies achieved in TCS-LBCNN were 99.491%, 99.656%, and 99.534%.

In reference [31], the AdaBin method was proposed to adaptively obtain the optimal binary set {b1, b2} for each layer of weights and activations. The method defined a new binary quantization function using the center position and distance of 1-bit values. An equalization method was proposed for the weights to minimize their Kullback–Leibler scatter, and a gradient-based optimization method was introduced to obtain the activation parameters. Experimental results showed that a top-1 precision of 66.4% could be obtained using the ResNet-18 architecture on the ImageNet dataset, and 69.4mAp (mean Average Precision, which is a synthesis of both Precision and Recall metrics) was obtained using SSD300 on the PASCAL VOC.

Binarization techniques are currently available to significantly reduce the size and complexity of models, enabling complex neural networks to be deployed on resource-constrained devices. However, accuracy loss remains a serious issue. To address this, researchers have explored using other values besides the traditional {−1, 1} for weights or activation values, increasing the complexity of the network to improve accuracy. Alternatively, adaptive binarization methods may be the future direction. These methods can dynamically adjust the range of weights and activation values based on the situation, providing a more flexible and accurate approach.

4. CPU/GPU Acceleration Algorithm

In addition to optimizing network performance, hardware-based CPU and GPU algorithms for neural network acceleration can also be optimized in the field of computer science. The current CPU/GPU acceleration algorithms for neural networks are mainly divided into three categories: adjusting the task scheduling strategy, enhancing CPU-GPU parallel computing efficiency, and strengthening GPU utilization.

Reference [32] proposed the RedSync method based on the RGC (Residual Gradient Compression) compression algorithm, which can reduce end-to-end training time in multi-GPU systems. This method significantly accelerated the training speed of large-scale deep neural networks. Reference [33] proposed the RedSync method based on the residual gradient compression (RGC) compression algorithm, which can reduce the end-to-end training time in multi-GPU systems. This method significantly accelerates the training speed of large-scale deep neural networks. In reference [9], the authors propose Troodon, a load-balanced scheduling heuristic suitable for CPUs and GPUs. The main idea of the algorithm is to organize all jobs into job pools according to device suitability, rank the jobs in the job pools according to the predicted speedup rate, and achieve load-balanced scheduling by considering the processing requirements of the jobs and the computational capacity of the devices. This is calculated by computing the compute shares of each device that are related to the available workload to be scheduled. The authors’ experiments were conducted on two CPU-GPU heterogeneous systems, and Troodon’s final processing time was reduced by 78%, 65%, and 41%, respectively, compared to the other three algorithms in the literature (DS, ISG, and AA). In reference [34], the authors propose methods to execute a local respective-field-based Extreme Learning Machine (LU) on a GPU platform. The first method is a new chunked logical unit decomposition algorithm that overcomes the global memory size limitation. The second method is used to accelerate the chunked LU decomposition efficient chunking algorithm Cholesky decomposition algorithm. Finally, the authors propose a heterogeneous CPU-GPU parallel algorithm that can make full use of GPU node resources. The experimental results showed that the chunking Cholesky decomposition algorithm was two times faster compared to the LU decomposition algorithm, while the heterogeneous blocking CPU-GPU acceleration algorithm improved the performance by 5% to 10% compared to the Cholesky algorithm. The authors of reference [35] propose Hybrid Parallel DESPOT (HyP-DESPOT), a massively parallel online planning algorithm based on the DESPOT algorithm that integrates CPU and GPU parallelism in a multilevel scheme for robotic environments. The DESPOT algorithm has three key steps—1. Forward Search, 2. Leaf node initialization, and 3. Backup—with the two steps of forward search and backup being irregular. The experimental results show that the execution speed of the HyP-DESPOT algorithm is significantly faster than that of DESPOT. In addition to CPU and GPU optimization for neural networks, specialized components can also be designed for neural network acceleration, such as the hardware-efficient vector convolutional neural network accelerator proposed in reference [36], which uses a 3 × 3 filter to optimize the shrink array. Here, a one-dimensional broadcast data stream is used to generate partial sums, thus compressing the computational overhead.

The current CPU/GPU algorithms for neural network acceleration have improved support for embedded devices, and research is focused on optimizing their operational efficiency. However, the issue of energy consumption has been largely overlooked. It is important to consider the impact of these algorithms on energy consumption in future research.

5. Summary

This section introduces three key technologies: Model Compression, Binary networks, and CPU/GPU acceleration algorithms, respectively. These three technologies will determine whether embedded AI can deploy more complex and efficient models and algorithms in the future. Furthermore, it is important to consider how the network structure and hardware algorithm support can be better implemented in the future to facilitate the integration with IoT edge computing. **Table 1** provides the characteristics of each work covered in this section.

Table 1. Literature covering key technologies.

Classification	**Reference**	**Proposed Method**	**Advantage**
		Model Compression	
	[10]	**SqueezeNet**	**Fewer parameters**
	[9]	**MobileNet**	**Compatible Resource-scarce embedded devices**
Network Design	[16]	**ShuffleNet**	
	[14]	**Once-for-all network**	**Lower energy consumption**
	[26]	**Improvement of Yolov5**	**Faster detection speed**
	[16]	**BRECQ**	**Faster production**
Quant	[17]	**Data-Free Quantization**	**Less precision loss**
	[18]	**AdaRound**	**Less precision loss**
Quant	[37]	**DeepCompression**	**Fewer storage requirements with no loss of precision**
	[20]	**Efficient Inference Engine**	**Lower energy consumption**
Prune	[10]	**Important Connection Pruning**	**Fewer parameters**
	[22]	**ProbMask**	**Higher precision**
	[23]	**ManiDp**	**Fewer flops**
	[24]	**CHEX**	**Fewer flops and less precision loss**
	[38]	**Channel pruning**	**Less loss of performance**
Binary Neural Network			
	[27]	**Binary Neural Network**	**Significant reduction in the number of parameters**
	[28]	**XNOR-Network**	**Less memory consumption**
	[29]	**From hashing to CNM**	**Improvement of accuracy**
	[30]	**CS-LBCNN and TCS-LBCNN**	**Higher precision**

Classification	Reference	Proposed Method	Advantage
CPU/GPU Acceleration	[31]	AdaBin	Less loss of precision
	[32]	RedSync	Faster training speed
	[9]	Troodon	Less processing time
	[19]	Local respective field-based Extreme Learning Machine	Higher performance and faster decomposition speed
	[35]	HyP-DESPOT	Faster execution Speed
	[36]	Hardware efficient vector-wise accelerator	Less energy consumption and higher hardware utilization
	[39]	GPU-kernel fusion model	Higher F-measure

References

1. Ang, K.L.-M.; Seng, J.K.P. Embedded Intelligence: Platform Technologies, Device Analytics, and Smart City Application s. IEEE Internet Things J. 2021, 8, 13165–13182.
2. Dick, R.P.; Shang, L.; Wolf, M.; Yang, S.-W. Embedded Intelligence in the Internet-of-Things. IEEE Des. Test 2019, 37, 7–27.
3. Guo, B.; Zhang, D.; Yu, Z.; Liang, Y.; Wang, Z.; Zhou, X. From the internet of things to embedded intelligence. World Wi de Web 2012, 16, 399–420.
4. Ardakani, A.; Condo, C.; Gross, W.J. Fast and Efficient Convolutional Accelerator for Edge Computing. IEEE Trans. Co mput. 2019, 69, 138–152.
5. Li, H.; Ota, K.; Dong, M. Learning IoT in Edge: Deep Learning for the Internet of Things with Edge Computing. IEEE Ne tw. 2018, 32, 96–101.
6. Manavalan, E.; Jayakrishna, K. A review of Internet of Things (IoT) embedded sustainable supply chain for industry 4.0 requirements. Comput. Ind. Eng. 2018, 127, 925–953.
7. Xu, D.; Li, T.; Li, Y.; Su, X.; Tarkoma, S.; Jiang, T.; Crowcroft, J.; Hui, P. Edge Intelligence: Empowering Intelligence to th e Edge of Network. Proc. IEEE 2021, 109, 1778–1837.
8. Poniszewska-Maranda, A.; Kaczmarek, D.; Kryvinska, N.; Xhafa, F. Studying usability of AI in the IoT systems/paradig m through embedding NN techniques into mobile smart service system. Computing 2018, 101, 1661–1685.
9. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H.J. Mobilenets: Effici ent convolutional neural networks for mobile vision applications. arXiv 2017, arXiv:1704.04861.
10. Iandola, F.N.; Han, S.; Moskewicz, M.W.; Ashraf, K.; Dally, W.J.; Keutzer, K.J. SqueezeNet: AlexNet-level accuracy with 50× fewer parameters and<0.5 MB model size. arXiv 2016, arXiv:1602.07360.
11. Zhang, X.; Zhou, X.; Lin, M.; Sun, J. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Device s. arXiv 2017, arXiv:1707.01083v2.
12. Chollet, F. Xception: Deep learning with depthwise separable convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition 2017, Honolulu, HI, USA, 21–26 July 2017; pp. 1251–1258.
13. Xie, S.; Girshick, R.; Dollár, P.; Tu, Z.; He, K. In Aggregated residual transformations for deep neural networks. In Proce edings of the IEEE Conference on Computer Vision and Pattern Recognition 2017, Honolulu, HI, USA, 21–26 July 201 7; pp. 1492–1500.
14. Cai, H.; Gan, C.; Wang, T.; Zhang, Z.; Han, S. Once-for-all: Train one network and specialize it for efficient deployment. arXiv 2019, arXiv:1908.09791.
15. Dong, X.; Yan, S.; Duan, C. A lightweight vehicles detection network model based on YOLOv5. Eng. Appl. Artif. Intell. 2 022, 113, 104914.
16. Li, Y.; Gong, R.; Tan, X.; Yang, Y.; Hu, P.; Zhang, Q.; Yu, F.; Wang, W.; Gu, S. Brecq: Pushing the limit of post-training q uantization by block reconstruction. arXiv 2021, arXiv:2102.05426.

17. Nagel, M.; Van Baalen, M.; Blankevoort, T.; Welling, M. Data-Free Quantization Through Weight Equalization and Bias Correction. In Proceedings of the IEEE/CVF International Conference on Computer Vision 2019, Seoul, Republic of Kor ea, 27 October–2 November 2019.

18. Nagel, M.; Amjad, R.A.; Van Baalen, M.; Louizos, C.; Blankevoort, T. Up or down? adaptive rounding for post-training q uantization. In Proceedings of the International Conference on Machine Learning 2020, Virtual, 3–18 July 2020; pp. 71 97–7206.

19. Han, S.; Mao, H.; Dally, W.J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv 2015, arXiv:1510.00149.

20. Han, S.; Liu, X.; Mao, H.; Pu, J.; Pedram, A.; Horowitz, M.A.; Dally, W.J. EIE: Efficient inference engine on compressed deep neural network. ACM SIGARCH Comput. Archit. News 2016, 44, 243–254.

21. Han, S.; Pool, J.; Tran, J.; Dally, W. Learning both weights and connections for efficient neural network. Adv. Neural Inf. Process. Syst. 2015, 28.

22. Zhou, X.; Zhang, W.; Xu, H.; Zhang, T. Effective sparsification of neural networks with global sparsity constraint. In Proc eedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, Virtual, 19–25 June 2021; p p. 3599–3608.

23. Tang, Y.; Wang, Y.; Xu, Y.; Deng, Y.; Xu, C.; Tao, D.; Xu, C. Manifold regularized dynamic network pruning. In Proceedin gs of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, Virtual, 19–25 June 2021; pp. 501 8–5028.

24. Hou, Z.; Qin, M.; Sun, F.; Ma, X.; Yuan, K.; Xu, Y.; Chen, Y.-K.; Jin, R.; Xie, Y.; Kung, S.-Y. Chex: Channel exploration fo r CNN model compression. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, New Orleans, LA, USA, 18–24 June 2022; pp. 12287–12298.

25. Li, Y.; Adamczewski, K.; Li, W.; Gu, S.; Timofte, R.; Van Gool, L. Revisiting random channel pruning for neural network compression. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, New Or leans, LA, USA, 18–24 June 2022; pp. 191–201.

26. Courbariaux, M.; Hubara, I.; Soudry, D.; El-Yaniv, R.; Bengio, Y. Binarized neural networks: Training deep neural networ ks with weights and activations constrained to+ 1 or-1. arXiv 2016, arXiv:1602.02830.

27. Courbariaux, M.; Bengio, Y.; David, J.P. Binaryconnect: Training deep neural networks with binary weights during propa gations. Adv. Neural Inf. Process. Syst. 2015, 28, 777–780.

28. Rastegari, M.; Ordonez, V.; Redmon, J.; Farhadi, A. In Xnor-net: Imagenet classification using binary convolutional neu ral networks. In Proceedings of the Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Nether lands, 11–14 October 2016; Springer: Berlin/Heidelberg, Germany, 2016; pp. 525–542.

29. Hu, Q.; Wang, P.; Cheng, J. From Hashing to CNNs: Training Binary Weight Networks via Hashing. Proc. Conf. AAAI Ar tif. Intell. 2018, 32.

30. Al-Wajih, E.; Ghazali, R. Threshold center-symmetric local binary convolutional neural networks for bilingual handwritte n digit recognition. Knowledge-Based Syst. 2023, 259.

31. Tu, Z.; Chen, X.; Ren, P.; Wang, Y. Adabin: Improving Binary Neural Networks with Adaptive Binary Sets, Proceedings of the Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, 23–27 October 2022; Springer: Berli n/Heidelberg, Germany, 2022; pp. 379–395.

32. Fang, J.; Fu, H.; Yang, G.; Hsieh, C.-J. RedSync: Reducing synchronization bandwidth for distributed deep learning trai ning system. J. Parallel Distrib. Comput. 2019, 133, 30–39.

33. Khalid, Y.N.; Aleem, M.; Ahmed, U.; Islam, M.A.; Iqbal, M.A. Troodon: A machine-learning based load-balancing applica tion scheduler for CPU–GPU system. J. Parallel Distrib. Comput. 2019, 132, 79–94.

34. Li, S.; Niu, X.; Dou, Y.; Lv, Q.; Wang, Y. Heterogeneous blocked CPU-GPU accelerate scheme for large scale extreme l earning machine. Neurocomputing 2017, 261, 153–163.

35. Cai, P.; Luo, Y.; Hsu, D.; Lee, W.S. HyP-DESPOT: A hybrid parallel algorithm for online planning under uncertainty. Int. J. Robot. Res. 2021, 40, 558–573.

36. Chang, K.-W.; Chang, T.-S. VWA: Hardware Efficient Vectorwise Accelerator for Convolutional Neural Network. IEEE Tr ans. Circuits Syst. I Regul. Pap. 2019, 67, 145–154.

37. Krishnamoorthi, R.J. Quantizing deep convolutional networks for efficient inference: A whitepaper. arXiv 2018, arXiv:18 06.08342.

38. Venieris, S.I.; Bouganis, C.S. fpgaConvNet: A toolflow for mapping diverse convolutional neural networks on embedded FPGAs. arXiv 2017, arXiv:1711.08740.

39. Ahmed, U.; Lin, J.C.-W.; Srivastava, G. A ML-based resource utilization OpenCL GPU-kernel fusion model. Sustain. Comput. Inform. Syst. 2022, 35, 100683.