Efficient Majority Voting in Digital Hardware

Stefan Baumgartner, *Graduate Student Member, IEEE*, Mario Huemer, *Senior Member, IEEE*, and Michael Lunglmayr, *Member, IEEE*

***Abstract*—In recent years, machine learning methods became increasingly important for a manifold number of applications. However, they often suffer from high computational requirements impairing their efficient use in real-time systems, even when employing dedicated hardware accelerators. Ensemble learning methods are especially suitable for hardware acceleration since they can be constructed from individual learners of low complexity and thus offer large parallelization potential. For classification, the outputs of these learners are typically combined by majority voting, which often represents the bottleneck of a hardware accelerator for ensemble inference. In this brief, we present a novel architecture that allows obtaining a majority decision in a number of clock cycles that is logarithmic in the number of inputs. We show, that for the example application of handwritten digit recognition a random forest processing engine employing this majority decision architecture implemented on an FPGA allows the classification of more than seven million images per second, resulting in a speed-up factor of more than 29 compared to the fastest state-of-the-art implementation considered.**

***Index Terms*—Random forests, majority decision, classification, field programmable gate array (FPGA), hardware acceleration.**

I. Introduction

HARDWARE accelerators for machine learning algorithms have become an important research topic in recent years. For example, a large amount of research has been spent on accelerators for neural networks, e.g., [1]–[8]. Alternatives to neural networks seem to have been investigated to a lesser extent. However, as it has been thoroughly demonstrated, e.g., in [9], alternative methods such as ensemble learning methods (e.g., random forests) can have inference performance comparable to neural networks.

Ensemble learning uses multiple base learners (the ensemble) whose outputs are combined to make a final decision [10]. For classification algorithms, the learners' outputs are typically combined by a majority decision. Famous techniques for ensemble learning methods are bootstrap aggregation (also called bagging), which is used for so-called random forests [11], or boosting [10]. From a hardware implementation perspective, using ensemble learning methods has a variety of advantages. Members of this group often rely on fundamental operations other than the scalar product (as neural networks do), that can be implemented with less complexity in digital hardware. Since multiple learners, typically operating independently of each other on the data, are employed, such methods have a high parallelization potential. The computationally more expensive inference operations of the learners can be calculated in parallel if the available hardware resources allow. The final bottleneck of the whole inference task is then only the combination of the learners' outputs. In this brief, we describe and analyze an efficient architecture combining the learners' outputs by a majority vote. Majority voting is an approach used for many applications, from safety-critical/fault-tolerant systems [12] over histogram analysis in image processing [13] to feature selection for music information retrieval [14]. Therefore, the presented work can also be utilized in hardware architectures for such use cases.

This brief is structured as follows. In Section II, we discuss ensemble learning and one of its subgroups, namely random forests, that are used in this brief. We advocate a simple architecture suitable for parallel processing of multiple trees in digital hardware. We then present a novel architecture for majority voting in Section III. Combining these building blocks enables an efficient random forest acceleration, by exploiting parallelism within its structure. We derive equations for the required number of clock cycles to make a majority decision as well as to fully process a random forest. In Section IV, we show FPGA synthesis results demonstrating the low area requirements and high clock speeds that are achievable with this architecture. Finally, we present synthesis and accuracy results for a full random forest engine incorporating the majority decision block for the example of handwritten digit recognition on the MNIST database [15]. These results demonstrate the high inference speed that is achievable with this architecture as, for this example, we were able to obtain a 96% recognition rate with a processing speed that allows classifying over 7 million 28×28 pixel images per second.

II. Decision Trees and Random Forests

Random forests are a special case of ensemble learning methods, where decision trees are employed as base learners. A decision tree consists of levels of comparison nodes and a final (leaf) level where the inference result is output. We will consider only axis parallel binary trees since they are mainly utilized for random forests. For inference, a tree is applied to an input vector $\mathbf{x}$. Starting from its root node, a decision tree

Manuscript received October 29, 2021; revised December 2, 2021; accepted January 10, 2022. Date of publication January 18, 2022; date of current version March 28, 2022. This work was supported by the "University SAL Labs" initiative of Silicon Austria Labs (SAL) and its Austrian partner universities for applied fundamental research for electronic based systems. This brief was recommended by Associate Editor H.-G. Yang. *(Corresponding author: Stefan Baumgartner.)*

Stefan Baumgartner and Mario Huemer are with the JKU LIT SAL eSPML Lab and the Institute of Signal Processing, Johannes Kepler University Linz, 4040 Linz, Austria (e-mail: stefan.baumgartner@jku.at; mario.huemer@jku.at).

Michael Lunglmayr is with the Institute of Signal Processing, Johannes Kepler University Linz, 4040 Linz, Austria (e-mail: michael.lunglmayr@jku.at).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSII.2022.3144047.

Digital Object Identifier 10.1109/TCSII.2022.3144047

is processed level by level, moving along its edges depending on the results obtained from the decision nodes [16]. Each comparison node compares one coordinate of a $p \times 1$ input vector $\mathbf{x}$ with the comparison value of the current node. As this results in splitting an area of $\mathbb{R}^p$ into two parts, we will call the coordinate of a node *splitting coordinate* and the comparison value *splitting value*. In this brief, the splitting coordinates and the splitting values are learned from training data by the CART [11] algorithm for random forests.

A. State-of-the-Art on Tree Inference Hardware Accelerators

In [17], [18], implementations of tree inference structures on FPGAs have been discussed. In these works, the main focus was on building pipelined versions of tree accelerators relying on parallelized node implementations in logic. There, the described versions range from accelerators using single node hardware up to implementing all nodes in parallel. In the latter, the parallelizing node comparisons and combining comparison results require a large amount of distributed logic and distributed memory resources.

For random forest implementations, one has to process multiple trees to obtain a classification result. This opens the possibility of processing multiple trees in parallel, which, according to the authors' opinion, is preferable to parallelizing the processing of a single tree. For this, we advocate the use of a lightweight tree inference architecture, as discussed in the next section. Although the structure is similar to the "Universal Node Architecture" of [17], it is even more simplified and it utilizes the Block RAMs of an FPGA for implementation. To enable the simple structure shown in Fig. 1, we learned random forests only comprised of balanced trees. This allows using a tree node numbering scheme (shown on the left in Fig. 1) where a child node has an address of either two times the address number of the parent node (left child) or two times the address number of the parent node plus one (right child; the root address is one). Therefore, a simple transition from one level of the tree to the next can be used, as shown in Fig. 1. In contrast, the "Universal Node Architecture" of [17] uses a more complex addressing scheme (with the advantage that it can implement unbalanced trees that we did not use in this brief).

For random forests, the classifications of the individual trees have to be combined, typically using a (unweighted) majority decision. As this represents a bottleneck, efficient hardware acceleration of this task is crucial. An interesting approach to obtain a majority vote in hardware is described in [19], [20]. There, multiple sorting networks are utilized to accomplish the majority decision. However, the number of clock cycles needed for sorting with the proposed sorting networks scales linearly with the number of trees T in a random forest as well as with the number of classes K: the complexity in terms of the number of clock cycles is of $\mathcal{O}(T + K)$. Even with more time-optimized sorting networks (that have a more complicated structure), the number of clock cycles would still be of $\mathcal{O}((\log(T))^2)$ [21]. Another majority voting hardware architecture has been discussed in [22], reporting a number of required clock cycles that grows linearly with the number of inputs. Our majority decision block, which is detailed in the following, has a clock cycle complexity of $\mathcal{O}(\log(T))$, which is clearly beneficial for a large number of decision trees.

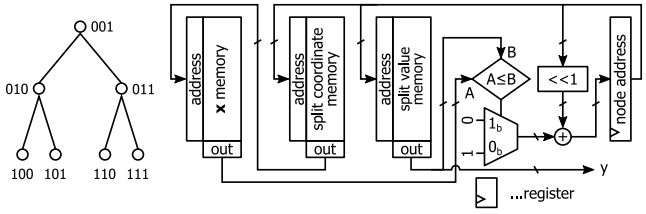

Fig. 1. Example tree with node numbers and tree inference architecture.

B. Accelerating Inference for Random Forests

When using an appropriate numbering of the nodes (as depicted on the left in Fig. 1), starting at the root node with number 1, the transition from a node of one level of a tree to a node of a lower level can be implemented very efficiently by a left shift of the node number followed by an increment of one if the node comparison was true, or by no increment otherwise. This leads to the architecture for tree inference that is schematically shown on the right in Fig. 1. It consists of two memories representing the tree (implemented as Block RAMs on the FPGA), the split coordinate memory, and the split value memory. The latter also contains the values of the leaf nodes that are output as classification result y if the leaf level is reached. The addresses of the memory entries correspond to the node numbers of the trees. The simple node address calculation hardware is depicted on the right of the memories in Fig. 1. Due to the constant left shift when moving down the levels of the tree, the addition of one can be implemented by simply setting the least significant bit of the node address. This structure allows processing one level of a tree within three clock cycles (one clock cycle to get the outputs of the tree memories, one clock cycle to access the corresponding coordinate of $\mathbf{x}$, and one clock cycle to perform the comparison and update the node address). Although the architecture shown in Fig. 1 is not directly pipelineable, the simplicity of this structure easily allows parallel processing of multiple trees in hardware (the number of trees that can be implemented in parallel is mainly defined by the number of available memories), which is beneficial for a random forest scenario. This is also supported by the fact that in a random forest typically the trees are different from each other. Thus, for a pipelined implementation one might have to exchange the tree parameters (the comparison values and the coordinates) during inference, complicating the overall structure and potentially emptying the pipeline. The proposed structure, however, easily allows implementing 40 trees of height 14 in parallel plus the majority decision architecture on state-of-the-art FPGAs, as we demonstrate in Section IV-B.

III. Majority Decision

The majority decision part of a random forest maps a vector of T integer numbers (the class outputs of the individual trees) to the number occurring most frequently. In Fig. 2 we schematically show the proposed architecture for performing a majority vote in hardware. As described below, the complexity scales only logarithmically with T. However, as the drawing in Fig. 2 shows, the area requirements for the proposed architecture scale linearly with K. We will first describe the iterative version of the architecture (as it is drawn) and then detail how the architecture can be pipelined.

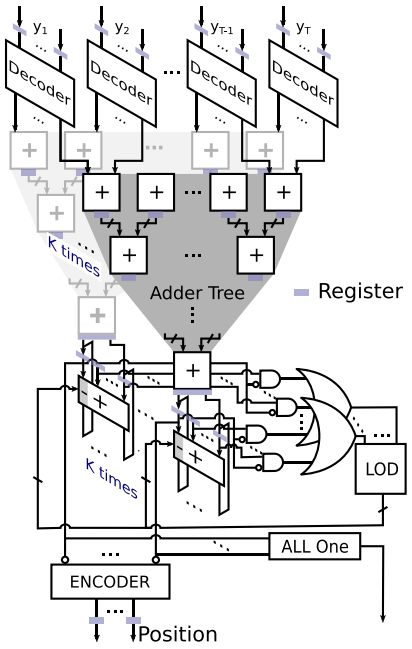

Fig. 2. Architecture of the majority decision block.

A. Architecture

1) Iterative Architecture: In a first step, the T integer input class values $y_i \in 0, \ldots, K-1, i = 0, \ldots, T-1$ are decoded into their one-hot representation, i.e., into vectors of length K containing all '0's but a '1' at the y_i-th position. Then, the count of the members of each class in the set of input values should be determined. Hence, all bits at the j-th position, $j = 0, \ldots, K-1$, of the one-hot vectors have to be summed up. For this purpose, K parallel adder trees are employed. That is, the first stage of every adder tree consists of $\lceil \frac{T}{2} \rceil$ one-bit adders with an output bit width of two, while at its last stage one $\lceil \log_2(T) \rceil$-bit adder is used. To maintain a high clock frequency, registers (drawn in violet in Fig. 2) are introduced after every stage of an adder tree. Every output of the K adder trees is the count of the corresponding class in the set of input class values. For further processing, the class counts are represented as signed numbers and are input to the subtractors (the left inputs of the subtractors are the subtrahends).

These class counts serve as the initial values of the registers feeding the K parallel subtractors, shown below the adder trees in Fig. 2. The residual part of the circuit shown in Fig. 2 is used to iteratively subtract the most significant '1' bit of all class counts (determined by the OR gates and the leading one detector - LOD) from these counts individually. For this, only those counts that are still positive will be considered (AND gates with negated inputs from the sign bits). This will be done until all class counts are negative. The index of the last non-negative class count[1] identifies the class that was output by the learners with maximum frequency. This number is output by the encoder at the bottom in Fig. 2. As one is interested in the class number of the cycle before all class counts are negative, the encoder's output is delayed by a register stage.

2) Pipelined Architecture: The architecture described in the last section works in an iterative manner, since the adder tree output values are decreased iteratively until all values are negative, which might take up to $\lceil \log_2(T) \rceil + 1$ iterations. Thus, a majority decision cannot be made at every clock cycle. In many application scenarios, this is not a problem, since not at every clock cycle classification results from e.g., decision tree classifiers of a random forest are available, which is also the case for our random forest implementation. However, in some scenarios, a pipelined majority decision architecture might be needed and thus we make slight adaptions such that the proposed architecture can provide a majority decision at every clock cycle. That is, the iterative subtraction of the adder tree output values by the LOD output values is "unfolded" to $\lceil \log_2(T) \rceil$ stages. In each stage, one LOD decision and K parallel subtractions take place. If all sign bits in a stage are '1', the outputs of the previous stage are passed to all following stages, discarding their outputs.

[1]In case of a draw the class with the highest class number is the output class.

B. Analysis

In the following, we consider the number of clock cycles that are needed to make a majority decision with the architectures described in Section III-A. Since a decoder is a low-complexity block, no registers between decoders and adder trees are introduced. The K parallel adder trees consist of $\lceil \log_2(T) \rceil$ adder stages, and thus $\lceil \log_2(T) \rceil$ clock cycles (one might reduce this number by combining multiple stages of low-complex adders in one clock cycle) after the input valid strobe the class counts are available at the outputs of the adder trees. For the iterative architecture described in Section III-A1, the required number of clock cycles can be determined as follows. The adder tree results are stored in registers before the iterative subtraction of the class counts starts, which adds another clock cycle. Given the stored class counts in the registers, the residual number of clock cycles is equal to the number of '1' bits in the bit pattern of the maximum class count's value plus one cycle to make all count values negative. The best case is, when the maximum count is a power of two, giving a lower limit of the overall required number of clock cycles for the majority decision

$$N_{iter,min} = \lceil \log_2(T) \rceil + 3. \tag{1}$$

The worst case occurs, when the binary maximum class count contains only '1' bits at bit positions behind the most significant '1' bit (i.e., the maximum class count is a power of two minus one). As the maximum number of '1' bits is $\lfloor \log_2(T) \rfloor$, the upper bound of the overall required number of clock cycles for the majority decision is

$$N_{iter,max} = \lceil \log_2(T) \rceil + \lfloor \log_2(T) \rfloor + 2. \tag{2}$$

Since only a part of the whole architecture works iteratively, there is no need to wait until the majority decision has finished feeding the next input values into the architecture. This means, every $\lceil \log_2(T) \rceil + 1$ clock cycles a new majority decision can be started, which guarantees that the proposed iterative architecture works properly for all constellations of input values.

For the pipelined version of the majority decision architecture detailed in Section III-A2, the number of needed clock cycles is constant:

$$N_{pipe} = \lceil \log_2(T) \rceil + \lfloor \log_2(T) \rfloor + 1. \tag{3}$$

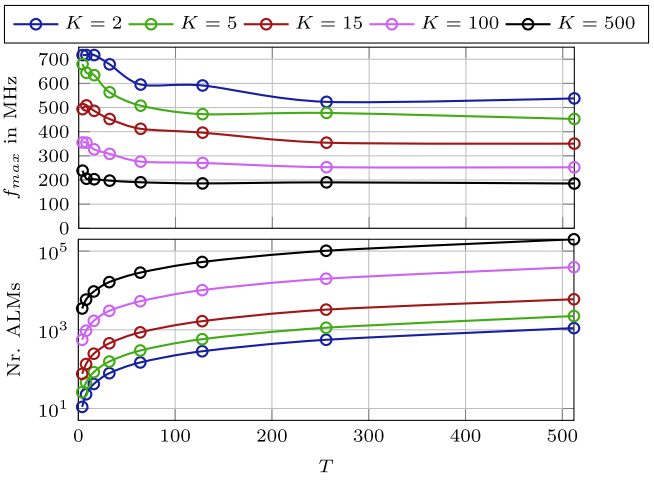

Fig. 3. Synthesis results for the iterative architecture.

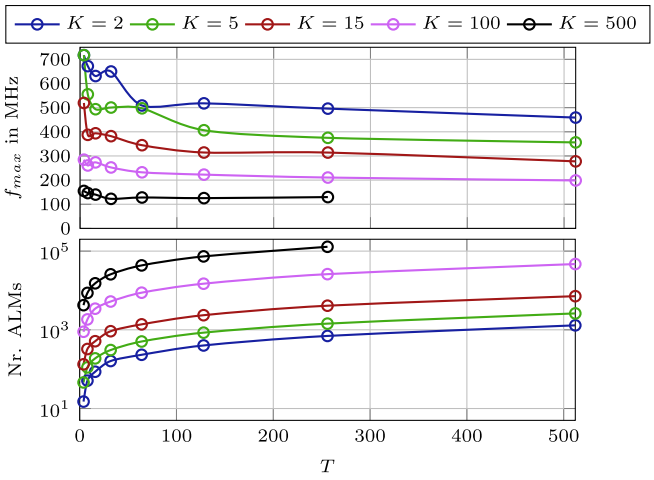

Fig. 4. Synthesis results for the pipelined architecture.

Here, the worst case for the maximum class count has to be assumed, but the iterative subtraction of the class counts can in this case already be abandoned when all values but one are negative. However, this is only an initial delay and afterward a new majority decision is available at every clock cycle.

IV. Results

A. Synthesis Results for the Majority Decision Block

In the following, we present the synthesis results of the proposed majority decision block for an Altera Stratix V 5SGXEA7 FPGA. To gain insight into how the number of inputs T and classes K influence the restricted maximum clock frequency f_{max} and the number of required adaptive logic modules (ALMs) required, syntheses are conducted for different T and K, varying from 4 to 512 and 2 to 500, respectively. The obtained results for the iterative architecture are shown in Fig. 3. Especially for more than two classes, it can be observed that f_{max} is only slightly affected by the number of inputs T above around 60 inputs. Obviously, the number of classes K has the larger influence on f_{max}. The number of ALMs needed scales linearly with T and K and even a majority decision block for 500 classes and 512 inputs fits into the specified FPGA (199795 ALMs are required for this setup, which corresponds to a logic utilization of 85 %).

The synthesis results for the pipelined architecture are plotted in Fig. 4. It can be observed that the behavior of the obtained curves is similar to that of the iterative architecture. However, the maximum clock frequency of the pipelined architecture for a specific setup (fixed T and K) is significantly lower than the corresponding one of the iterative architecture. Furthermore, the number of ALMs required for the pipelined architecture is higher and for $T = 512$ inputs and $K = 500$ classes, the design no longer fits into the FPGA.

B. Application Example: Handwritten Digit Recognition

In this section, we describe results for a random forest processing engine comprised of multiple instances of the tree processing units as shown in Fig. 1 and an instance of the majority decision unit. Processing of a tree with l levels of decision nodes followed by one level of terminal nodes requires $3l + 1$ clock cycles (three clock cycles per level for the processing of a decision node and one clock cycle to output the value of the terminal node as y). Assuming that T trees can be processed in parallel in digital hardware, and combined with the iterative majority decision architecture, this leads to a worst-case number of clock cycles of

$$3l + \lceil \log_2(T) \rceil + \lfloor \log_2(T) \rfloor + 3 \tag{4}$$

to finish the classification of an input vector $\mathbf{x}$ using a random forest of T trees.

We synthesized the described architecture for a random forest trained on the MNIST handwritten digit database [15]. For this, we used 40 trees and 14 levels of decision nodes per tree. Each tree of the random forest has been learned with a random selection of 75% of the 60000 training images. To obtain the splitting coordinates, only $\sqrt{784} = 28$ of the 784 coordinates (again selected randomly) of each image (28×28 pixels) have been allowed to be selected to learn the best splitting coordinates, as it is typical for random forests [16]. This allowed obtaining a classification performance on the MNIST test set (10000 images) of 96% correctly classified digits. For reproducibility, we uploaded the contents of the tree memories to [23]. Although this performance is below the best classification performances described in the literature [15], it is comparable to the state-of-the-art of hardware architectures for this problem in terms of inference accuracy. When calculating the required number of clock cycles using (4) for this use case one obtains $3 \cdot 14 + \lceil \log_2(40) \rceil + \lfloor \log_2(40) \rfloor + 3 = 56$ clock cycles for the classification of a single image. However, assuming that processing a tree requires more clock cycles than the majority decision, one can input new data vectors $\mathbf{x}$ already after tree processing is finished. That is, after the 56 clock cycles for the first classification one can obtain new classifications every $3 \cdot 14 + 1 = 43$ clock cycles.[2] Tab. I shows results after placement and routing. As one can see from Tab. I, besides the number of block RAM bits, the hardware requirements of the random forest architecture using the proposed majority decision block are very low. The block RAM bits are used to store the vectors $\mathbf{x}$ as well as the split coordinates and the split values for each tree. The memory is typically also the limiting factor for this architecture for

[2]The specified timing is valid only for the described architecture not including the time required for the data handling.

TABLE I
LITERATURE REPORTS ON MACHINE LEARNING HARDWARE FOR MNIST

Work	[1]	[2]	[3]	[4]	this work
Learning Algorithm	SVM	CNN	SNN	FCNN	RF
FPGA/Board	Stratix III EP3SE260	Artix 7	Virtex 7 VC707	Virtex 7 VC707	Stratix V 5SGXEA7
Max. clock frequ. (MHz)	≤ 250	300	≈ 100	490.9	303.4
Clock cycles	n.r.[3]	17400	n.r.	2034	56
Correctly classified	98.96 %	98.80 %	92.92 %	98.60 %	96.00 %
Slices/ALMs	n.r.	7986	485000	34848	2709
Registers	n.r.	n.r.	n.r.	36057	2495
DSP blocks	n.r.	n.r.	2800	602	0
Block RAM (MBits)	n.r.	n.r.	37	21	28
Class. images/second	$\approx$ 10 000	$\approx$ 17 250	$\approx$ 320	240 905	7 055 813

larger problem sizes. The authors would like to point out that the main aim was not to achieve the best classification performance, but to demonstrate the inference speed that is achievable with this architecture. For the described problem, the architecture is able to classify more than seven million images per second while still maintaining a, as we think, decent classification performance. Furthermore, we want to point out that the used random forest was applied directly on the image vectors without using any pre-processing at all. This demonstrates the potential of the proposed random forest architecture. A comparison with state-of-the-art implementations of hardware accelerators for the MNIST application is shown in Tab. I. Although some of the compared methods show a better classification performance for the problem at hand, the presented architecture is about 30 times faster than the fastest state-of-the-art implementation used for comparison. Among the compared works, only [3] reported on power consumption. Based on the reported figures, one can calculate an average power consumption of approximately 1.6 W for the implementation described in [3]. We used Altera/IntelFPGA's "PowerPlay Early Power Estimator" to estimate a power consumption of 1.5 W for our implementation.

V. CONCLUSION

We presented a novel hardware implementation for finding a majority decision in digital hardware. For this architecture, the number of required clock cycles for a majority vote depends logarithmically on the number of its inputs. We analyzed the number of clock cycles for iterative and pipelined variants of the architecture. Furthermore, we showed synthesis results demonstrating the resource requirements as well as the obtainable clock frequencies for different problem sizes. Finally, we demonstrated the capabilities of our approach for majority voting in hardware in combination with a low complexity tree inference architecture. We applied the resulting random forest hardware implementation to the MNIST dataset and demonstrated that more than seven million classifications per second are possible on the utilized FPGA.

[3]Not reported.

REFERENCES

[1] M. Papadonikolakis and C.-S. Bouganis, "A novel FPGA-based SVM classifier," in *Proc. Int. Conf. Field-Program. Technol.*, Beijing, China, 2010, pp. 283–286.

[2] S. Mujawar, D. Kiran, and H. Ramasangu, "An efficient CNN architecture for image classification on FPGA accelerator," in *Proc. 2nd Int. Conf. Adv. Electron. Comput. Commun. (ICAECC)*, Bangalore, India, 2018, pp. 1–4.

[3] S. Li, Z. Zhang, R. Mao, J. Xiao, L. Chang, and J. Zhou, "A fast and energy-efficient SNN processor with adaptive clock/event-driven computation scheme and online learning," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 4, pp. 1543–1552, Apr. 2021.

[4] L. D. Medus, T. Iakymchuk, J. V. Frances-Villora, M. Bataller-Mompeán, and A. Rosado-Muñoz, "A novel systolic parallel hardware architecture for the FPGA acceleration of feedforward neural networks," *IEEE Access*, vol. 7, pp. 76084–76103, 2019.

[5] X. Chang, H. Pan, W. Lin, and H. Gao, "A mixed-pruning based framework for embedded convolutional neural network acceleration," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 4, pp. 1706–1715, Apr. 2021.

[6] T. Yuan, W. Liu, J. Han, and F. Lombardi, "High performance CNN accelerators based on hardware and algorithm co-optimization," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 1, pp. 250–263, Jan. 2021.

[7] J. Wang, J. Lin, and Z. Wang, "Efficient hardware architectures for deep convolutional neural network," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 6, pp. 1941–1953, Jun. 2018.

[8] A. A. Gilan, M. Emad, and B. Alizadeh, "FPGA-based implementation of a real-time object recognition system using convolutional neural network," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 4, pp. 755–759, Apr. 2020.

[9] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?" *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 3133–3181, Jan. 2014.

[10] R. E. Schapire and Y. Freund, *Boosting: Foundations and Algorithms*. Cambridge, MA, USA: MIT Press, 2012.

[11] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and Regression Trees*. Boca Raton, FL, USA: CRC Press, 1984.

[12] B. Parhami, "Voting algorithms," *IEEE Trans. Rel.*, vol. 43, no. 4, pp. 617–629, Dec. 1994.

[13] K. Qin, K. Xu, F. Liu, and D. Li, "Image segmentation based on histogram analysis utilizing the cloud model," *Comput. Math. Appl.*, vol. 62, no. 7, pp. 2824–2833, 2011.

[14] G. Tzanetakis and P. Cook, "Musical genre classification of audio signals," *IEEE Trans. Speech Audio Process.*, vol. 10, no. 5, pp. 293–302, Jul. 2002.

[15] Y. LeCun, C. Cortes, and C. Burges. "MNIST Handwritten Digit Database." 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist

[16] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning* (Springer Series in Statistics), 2nd ed. New York, NY, USA: Springer, 2009.

[17] J. R. Struharik, "Implementing decision trees in hardware," in *Proc. IEEE 9th Int. Symp. Intell. Syst. Informat.*, Subotica, Serbia, 2011, pp. 41–46.

[18] F. Saqib, A. Dutta, J. Plusquellic, P. Ortiz, and M. S. Pattichis, "Pipelined decision tree classification accelerator implementation in FPGA (DT-CAIF)," *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 280–285, Jan. 2015.

[19] M. Barbareschi, S. Del Mario Prete, F. Gargiulo, A. Mazzeo, and C. Sansone, "Decision tree-based multiple classifier systems: An FPGA perspective," in *Proc. 12th Int. Workshop Multiple Classifier Syst. (MCS)*, 2015, pp. 194–205.

[20] M. Barbareschi, S. Barone, and N. Mazzocca, "Advancing synthesis of decision tree-based multiple classifier systems: An approximate computing case study," *Knowl. Inf. Syst.*, vol. 63, no. 6, pp. 1577–1596, 2021.

[21] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge, MA, USA: MIT Press, 2001.

[22] G. B. Hacene, V. Gripon, N. Farrugia, M. Arzel, and M. Jezequel, "Efficient hardware implementation of incremental learning and inference on chip," in *Proc. 17th IEEE Int. New Circuits Syst. Conf. (NEWCAS)*, Munich, Germany, 2019, pp. 1–4.

[23] M. Lunglmayr, "MNISTRF." Accessed: Oct. 28, 2021. [Online]. Available: https://github.com/mlunglma/MNISTRF