ACCELERATING RANDOM FOREST TRAINING PROCESS USING FPGA

Chuan Cheng *Christos-Savvas Bouganis*

Department of Electrical and Electronic Engineering
Imperial College London
email: {chuan.cheng09, christos-savvas.bouganis}@imperial.ac.uk

ABSTRACT

Random Forest (RF) is one of the state-of-art supervised learning methods in Machine Learning and inherently consists of two steps: the training and the evaluation step. In applications where the system needs to be updated periodically, the training step becomes the bottleneck of the system, imposing hard constraints on its adaptability to a changing environment. In this work, a novel FPGA architecture for accelerating the RF training step is presented, exploring key features of the device. By combing a fine-grain data-flow processing at low-level and by exploiting parallelism available at high level inherent in the algorithm, significant acceleration factors are achieved. Key to the above gains is a novel FPGA FIFO based merge sorter module, a core component in the architecture, that exhibits high efficiency in memory utilisation; as well as a batch training strategy that enable full exploitation of the high memory bandwidth offered by the on-chip memory featured on FPGA devices. The proposed system achieves speed-up factors of up to 230x over a 3GHz Intel Core i5 processor when an Altera Stratix IV device is utilised under classification problems drawn from VOC2007.

1. INTRODUCTION

Random Forest (RF) [1] is one of the state-of-art supervised learning methods in machine learning [2] with comparable classification accuracy with well-known support vector machine (SVM). The method infers a predicting hypothesis from labeled training data in order to predict the label of any unseen new input data. Similar to any supervised machine learning method, RF design is divided into two phases: Training and Evaluation. Training and evaluating an accurate RF classifier for large amount of data (several thousands to several millions) can take hours in modern CPUs due to scarce memory bandwidth and limited capability of parallel processing. An indication of the computational cost required for large-scale data is shown in Table 1 [3]. The measurements are based on classification problems collected from UCI machine learning repository, and are the total times required for the learning and testing of the classifiers. The

Table 1. Time measurements in seconds for RF operations (training and evaluation steps). The results are based on a fully utilized 48-core processor.

Dataset	Number of examples	1k trees	10k trees	100k trees
Car	1728	0.97	9.1	77.86
Kr-vs-Kp	3196	2.43	22.95	225.11
Mushroom	8124	3.38	26.79	229.40
Spambase	4601	5	48.64	488.01
Splice	3190	60	34.03	343.12

presented work targets applications where both training and evaluation steps needs to be invoked repeatedly during the operation; these are typically involved in continuous machine learning problems where concept drift is of importance. In these cases, the training step becomes the dominant component of the total operation in terms of runtime. Acceleration of the training step of the algorithm will have a direct impact on the practical use of the algorithm on applications where continuous learning is required.

Researchers have already investigated the use of multi-core CPUs [3] and GPU [4] platforms for the acceleration of the training stage of the RF algorithm. However, the RF training stage is dominated by intense memory access with large degree of data dependences. These characteristics limit the full exploitation of the SIMD architecture of modern GPUs, where the general architecture of modern CPUs is not tuned to the characteristics of the RF training stage. On contrary, algorithms with intense memory access with large degree of data dependence have been demonstrated to map well in modern FPGAs [5]. The flexible logic elements on FPGAs allow customization in fine-grain parallelisation, where the large number of embedded memories in high-end FPGA devices offer high on-chip memory bandwidth. In this work, a novel FPGA architecture for the training stage is presented, which achieves significant speed-ups over modern CPU-based systems. Key to the proposed system is a novel FPGA FIFO based merge sorter that minimises the re-

978-1-4799-0004-6/13/$31.00

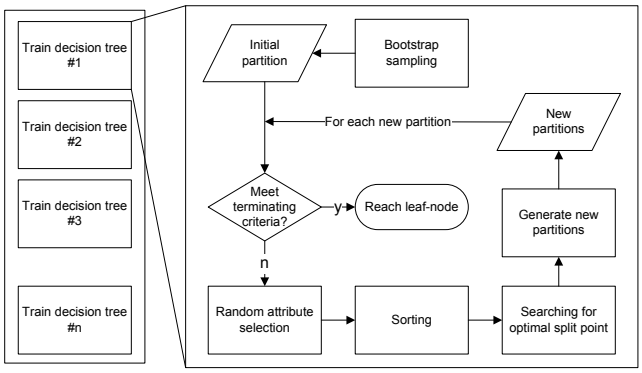

Fig. 1. Work-flow of training RF classifier

quired hardware resources, leading to high acceleration factors for a given device area. To fully exploit the high memory bandwidth featured on FPGA devices, a batch training strategy is also proposed, enabling a significant acceleration to be achieved in large-scale training problem.

2. BACKGROUND

2.1. Summary of Random Forest training algorithm

Random Forest classifier is an ensemble of decision tree classifiers [6]. Thus, any conventional decision tree training method can be applied to training RF [7][8]. Fig. 1 illustrates the work-flow of RF training process as a sequence of single decision tree training processes. The key concept is to split the data to two partitions according to a certain objective function. The goal is to produce certain type of partitions at the leaves of the decision tree. In each splitting of the training data a *node* in the tree classifier is generated with information needed to construct the RF classifier.

The splitting continues until certain criteria is met. Finding the optimal splitting point requires $O(n^n)$ memory accesses, where n is the number of training instances in the partition. By sorting the training instances prior to the selection of the splitting point, a typical $O(nlogn)$ memory access is achieved, speeding up the training process significantly. [9] For the completion of the RF training, two additional stages are adopted compared to the conventional decision tree training. The first stage is to perform bootstrap sampling before training each decision tree, where the second extra stage is to randomly select a number of attribute(s) to be used at each non-leaf node as potential splitting attributes.

2.2. Bootstrap sampling

Originally, the size of bootstrap sampling was set to be the same as that of original data set, but further investigations [10][11] suggested that such setting is not optimal for biased machine learning problems. To address the above issue several approaches have been proposed including over-sampling the minority class, sub-sampling the majority class and constructing weighted splitting function. In this work we adopt the bootstrap sub-sampling approach. Based on empirical experiments, sub-sampling is demonstrated to be an effective way to avoid the impact from biased training data set, where in addition it is compatible with the batch learning strategy that is utilised in the proposed system in order to leverage on the high bandwidth offered by the on-chip FPGA memories.

2.3. Random attribute selection

For the Random attribute selection process, two main techniques are employed. In the first technique, a set F of attributes is randomly selected, where the cardinality of F is much smaller than the total number of attributes available in the training data set. During the learning process, the optimal splitting point is determined by searching through all F attributes. In the second technique, a linear combination of attributes is formed where the attributes are randomly selected from the set of all attributes. In this work, the first technique is adopt, where the size of the set is fixed to one. The selection of a single attribute is based on the investigation reported in [1], where it is demonstrated that the average difference between the error rate using a single attribute and using a larger number of attributes is less than 1%. Furthermore, the optimal number of attributes is data-dependent hence it is only found by fine-tuning through multi-fold training.

2.4. Splitting measurement

Different measurements can be used for determination of optimal splitting point, typical options include the Gini impurity and entropy impurity (information gain) [6] metrics. In this paper we use the default Gini measurement, as defined in Eq.(1).

$$i(N) = \sum_{i \neq j} P(\omega_i)P(\omega_j) = \frac{1}{2}\left[1 - \sum_j P^2(\omega_j)\right] \quad (1)$$

$P(\omega_i)$ and $P(\omega_j)$ refer to the proportion of instances with label i and j in the partition respectively. The splitting point that yields the maximum drop in impurity is considered as the optimal splitting point. The drop in impurity for binary split is defined in Eq.(2).

$$\triangle i(N) = i(N) - P_L i(N_L) - (1 - P_L) i(N_R) \quad (2)$$

N_L and N_R represent the newly generated partitions, left and right partition respectively.

2.5. Related works

Despite the successful mapping of several well-known classifiers on FPGA such as SVM [12] and Naive Bayes classifier [13], to the best of the authors' knowledge, this is the first work that investigates the potential of FPGAs in the acceleration of the training stage of the RF algorithm. The closest work to this one is by Narayanan [14], where an FPGA architecture is proposed for training a single decision tree classifier. The authors propose to map to FPGA the step of finding optimal split point and let the host PC to perform the rest of training steps. By exploiting parallel processing on FPGA, a speedup of 5.6x in comparison to a PC-alone implementation is reported. However, this work is dedicated to accelerate a single decision tree classifier, which is a subset of the overall training of an RF classifier.

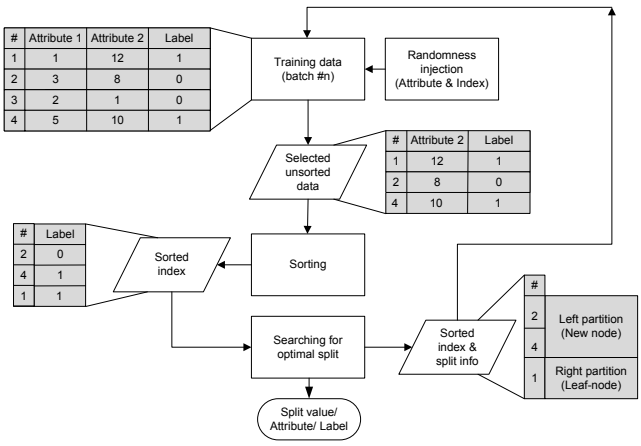

Fig. 2. Overview of the training process

3. ACCELERATE RF TRAINING ON FPGA

3.1. Batch learning

Training a RF involves intense memory access making the access to the off-chip memory the bottleneck of an FPGA-based system. Moreover, the on-chip memory space on FPGAs is not large enough to store the whole training dataset for real applications that would require a significant acceleration. To fully utilize the high memory bandwidth of the on-chip memory and hide the latency caused by fetching data from the external memory, a batch learning strategy is proposed in this work. In the proposed strategy, the original training dataset is stored on external memory and is split into several partitions. The training data are sent to the on-chip memory in batches and the system trains decision trees on each batch in turn. The size and the number of batches as well as the number of decision trees to be trained on each batch are parametrized and left to the user. The impact of the proposed batch training on the accuracy of RF classifier has been investigated in a selection of binary classification problems and the relevant results are reported in the performance evaluation section (Section 4).

3.2. Overview of the proposed training process

Each decision tree of the ensemble is trained on the training data (batch) in a circled dataflow as shown in Fig. 2. The processing starts with selecting a part of the training data and feeding it into a sorting operation. The selection is based on bootstrap sampling and random attribute selection, which are represented by the *Randomness injection* block in Fig. 2. The selected data ready to be fed into sorting unit is notated by *Selected unsorted data*. In the example shown in Fig. 2, the batch n contains four training instances (examples), each instance with two attributes; after random selection item # 1,2,4 with labels and values corresponding to attribute two are fed into the sorting unit. The output of sorting unit is a list of sorted index numbers with corresponding labels as represented by *Sorted index* in the figure. The output is generated sequentially, and is sent into the searching unit. In the searching unit, an optimal split point is found and the original partition is split into two new partitions as represented by *Left partition* and *Right partition* in the figure. After the splitting, the sorted index list that is previously the content of initial partition is now the combination of two new partitions. In order to distinguish between different partitions, a pair of tags are also produced, indicating the boundary of each partition. To start the next round of splitting, new partitions are in turn sent back to the starting point of the dataflow. The splitting is performed recursively until all the nodes are labeled as leaf-node, which indicates the completion of the training of a single decision tree.

3.3. Summary of hardware architecture

An FPGA architecture has been designed to efficiently implement the proposed training process described in the previous section. Fig. 3 depicts a top-level diagram of the architecture. The system consists of a memory array and a processing element (PE). The memory array contains several memory blocks, which are mapped to embedded memory blocks, and each block stores training instances corresponding to one attribute, where an additional memory block is used to store two sets of random indices. Bootstrap sampling and random attribute selection are performed by loading these indices. The PE block has two main components: A FIFO based merge sorter and an arithmetic logic unit (ALU) evaluating the Gini measurement. The sorted indices together with the boundary tags are stored in a FIFO buffer and are ready to be sent back to a memory address generation unit. The indices are used to address directly the related data point. In each round of splitting, the optimal

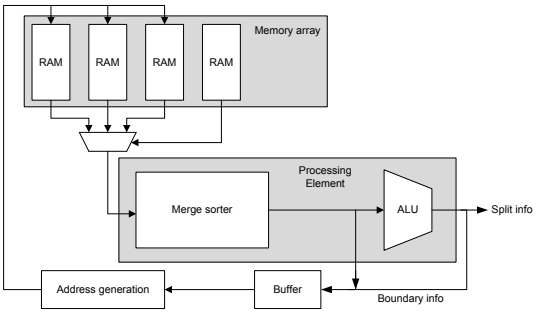

Fig. 3. Top-level FPGA architecture

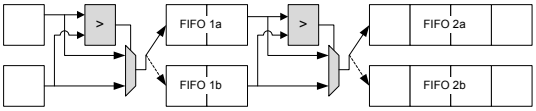

Fig. 4. Benchmark merge sorter design

splitting point is obtained after the entire partition passes through the ALU. Before this point, the boundary of the new partitions is still pending and hence the partitions can not be fed back to the PE for further processing, leading under-utilised hardware resources. In order to achieve maximum utilization of the resources allocated to each PE, each PE trains two decision trees simultaneously, resulting to around 100% utilization of the hardware resources.

Arithmetic calculation involved in the system are based on fixed point representation. The scaling factor and corresponding width of whole part and fractional part are also based on the format of the training data hence it is application dependent. The optimal scaling factor is obtained by fine-tuning through multi-fold training.

3.4. Efficient FIFO based merge sorter

The FIFO based merge sorter features fast output throughput and low resource utilization and is considered one of the most appropriate sorter type for FPGA devices [15]. A common implementation of the FIFO based merge sorter is in the form of cascaded units as illustrated in Fig. 4. In this work, a modification is proposed to the sorter that reduces the memory utilization by 50%. Previous attempts have been made to reduce memory usage by using linked list FIFO memory [16]. The drawback is that linked list memory needs extra storage to record the 'link' for each memory location, which causes waste in memory utilization. The proposed approach, on the other hand, does not need additional memory space for the 'link', instead it dynamically moves the content among the memory locations to ensure the memory address is still calculable.

Fig. 5 illustrates the working principle of the proposed merge sorter. Each row shown in the figure represents a segment of FIFO in the cascaded structure. The sorter starts operation by filling the empty FIFO with the first sorted data list (list A, shown in the upper row in Fig. 5 as A1 to A8). After that ,elements in the second sorted data list (list B, B1 to B8) start to arrive. Whenever an element from list A is sent to the next stage, its memory location will be occupied by a newly incoming element from list B (as shown in the middle row in the figure where memory location 0 to 2 have been occupied by the elements from list B); similarly when an element from list B is sent out, its location will be occupied by the following incoming element (again as shown in the middle row, location 0 was previously occupied by B1 after beating A1 but now is the place where B3 is located since B1 was sent out after comparison) Consider one case where A1 A2 and A3 are beaten solely by B1, memory location 0, 1 and 2 are occupied by B1, B2 and B3 respectively after sending out A1 to A3; to read B1 to B3 in order, the memory address needs to be generated in sequence of 0, 1 and 2, which is calculable. Consider a different case as illustrated in the middle row in Fig. 5, B1 and A3 have been sent out, their memory location is now occupied by B3 and B4; to read B2 to B4 in order, the memory address needs to be generated in sequence of 1, 0, 2. The memory address becomes incalculable here because the sequence of the memory address depends on the results of comparison, which is not known. Our approach is to ensure a calculable memory address sequence. A generalized case is shown in the bottom row in Fig. 5. When the continuity of memory address is broken, the element stored in the 'wrong' location will be moved to the 'correct' location to recover the continuity. A brief pseudo algorithm for dynamic element movement is given in Table 2 and is illustrated by an example in the bottom row, Fig. 5. By careful arrangement of two I/O ports available on each FIFO block, the movement of the data is achieved without interfering the normal dataflow operation.

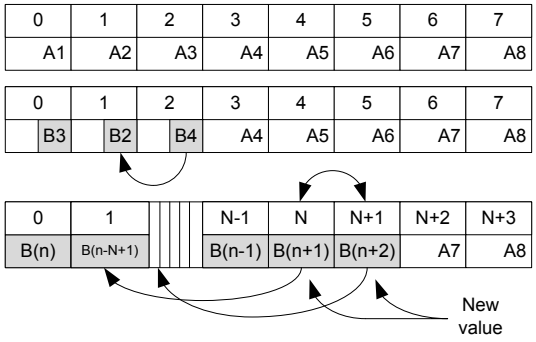

Fig. 5. Proposed merge sorter implementation

3.5. Evaluation of the splitting measurement

Searching for the optimal split starts on arrival of the first output from the sorting unit. In the proposed work, the ar-

Table 2. Pseudo algorithm for dynamic element movement

if continuity of memory address is broken
{*record* the boundary s(start) and e(end) of the gap;
(s = '1' , e = 'N-1' in the example)
(notate by r the elements to the right of the gap
and from list B;)
record f, the number of elements in r;
(2 in the example)
setup a cycled counter k starting from
'e+1' and reset every f-1 steps;
for (i = s, i <e+1, i++)
{ *wait until* element in location i is exported;
move element in location k to location i;
write new incoming element to location k;
k++;}
swap the elements in r until they are in order;
}

rived instances as considered as those allocated to the left partition while the instances yet to come are those allocated to the right partition. Every time a new input arrives, the distribution of instances allocated to the two partitions are changed. The ALU then evaluates the Gini measurement for the current distribution and records the distribution so far that produce the biggest drop in Gini impurity. After the last sorted element arrives ALU, the final round of Gini measurement is also finished, producing the optimal split point. To efficiently evaluate the Gini measurement onto FPGA, Eq.(2) is simplified. Eq.(2) can be transformed as in Eq.(3a). $size_{Li}$ refers to the number of instances that are allocated to the left partition and with label i, $size_{Lj}$ represents the number of instances in the left partition but with label j; similarly $size_{Ri}$ and $size_{Rj}$ notate the number of instances with different labels in the right partition. The biggest drop in Gini impurity is found when a minimum value of Eq.(3a) is obtained. This proposed simplification was originally proposed in [14], and it is not claimed as a contribution of this work.

$$\begin{aligned}\triangle i(N) &= P(\omega_{Li})P(\omega_{Lj})size_L + P(\omega_{Ri})P(\omega_{Rj})size_R \\ &= \frac{size_{Li} \cdot size_{Lj}}{size_{Li} + size_{Lj}} + \frac{size_{Ri} \cdot size_{Rj}}{size_{Ri} + size_{Rj}}.\end{aligned} \tag{3a}$$

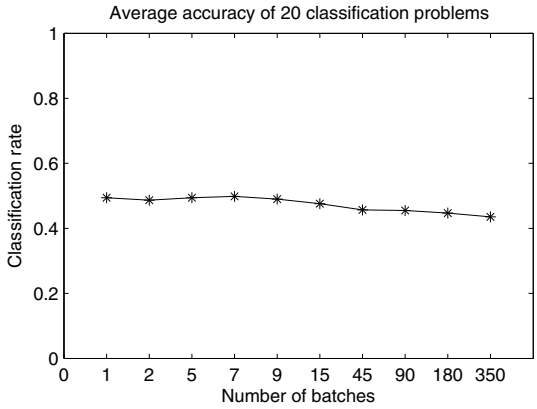

Fig. 6. Average classification accuracy as a function of the number of batches use for training

4. PERFORMANCE EVALUATION

4.1. Impact of batch training on accuracy

The impact of the proposed batch training on the classification accuracy of the RF is evaluated using a software version of the RF training stage. As case study, twenty binary classification problems from VOC2007 dataset [17] were selected. Fig. 6 demonstrates the achieved average classification rate over the twenty binary classification problems of the RF algorithm as a function of the number of batches used to perform the training, when a RF with an ensemble of 200 decision trees are used. A number of batch of one, indicates the original algorithm where the whole dataset is used at once, and its performance is considered as the baseline. The rest of the point depict the results that correspond to RF trained on different number of batches. The results indicate that the proposed batch training leads to a classification performance that does not deviate significant from the baseline performance where all the training data are considered simultaneously. Further experiments demonstrated that the classification accuracy starts to drop in extreme cases where the size of each batch becomes too small to capture any information from the dataset that would lead to an informative decision tree classifier. Please note that the achieved classification rate accuracy, is in line with [18], and it depends on the generation of the features used in the classification, which is out of the scope of this work.

4.2. FPGA Acceleration

The performance of the FPGA system is also evaluated by utilising the twenty binary classification problems in the VOC2007 data set. The system is mapped on an Altera Stratix IV FPGA (EP4SGX70HF35C2) and it is clocked at 200MHz. The performance of the proposed system utilising a single PE is compared to a C++ software implementation of the

Table 3. Speedup in 20 binary classification problems with average accuracy = 0.48

Object	Speedup	Object	Speedup
aeroplane	168	bicycle	148
bird	151	boat	211
bottle	170	bus	196
car	195	cat	180
chair	142	cow	192
dining table	163	dog	159
horse	157	motorbike	167
person	159	potted plant	165
sheep	232	sofa	152
train	167	tv monitor	149

Table 4. Hardware utilisation (based on batches with 512 instances and 500 attributes)

Resource type	Utilisation	(%)
Combinational ALUTs	2776	5%
Memory ALUTs	148	<1%
Registers	4596	n/a
DSP block 18-bit elements	9	2%
Block memory bits	4102k	62%

RF training running on an Intel Core i5 3.2GHz processor (running on single core). Table. 3 presents the achieved speedups for the twenty classification problems. An average speedup of 171x is achieved over the employed problems. An average accuracy of 0.48 over twenty problems is achieved. The reduction in the classification accuracy is due to batch learning approach, which according to the previous results is small.

An instance of the hardware utilisation of the system when its batch contains 512 examples of 500 attributes is given in Table. 4. The results show that the proposed system scalability is limited due to the memory requirements to store the working batch.

4.3. Discussion on the scalability of the system

The utilised FPGA system so far contains one memory array connected to a single PE, as shown in Fig. 3. As the embedded memory blocks in modern FPGA devices are dual ported, each memory array can be connected to two PEs. Thus, an extra 2x speedup in addition to the acceleration shown in Table. 3 can be achieved.

Given enough memory space on FPGA, multiple memory arrays and PEs can be instantiated, letting each memory array process one of the batches. Thus, the aim is to use the minimum size of batch ensuring acceptable loss in classification rate accuracy. The system can load a batch to each memory array in a pipeline fashion from the external memory, and each memory array will train a set of decision trees on that batch. For instance, let's consider the training of a RF classifier for the 'person' problem. The training data are partitioned into 25 batches and 16 decision trees are trained by the proposed system on each batch. Fig. 7 shows the overall runtime as a function of the number of utilised memory arrays. The black block in the first bar indicates the runtime distribution of external memory access.

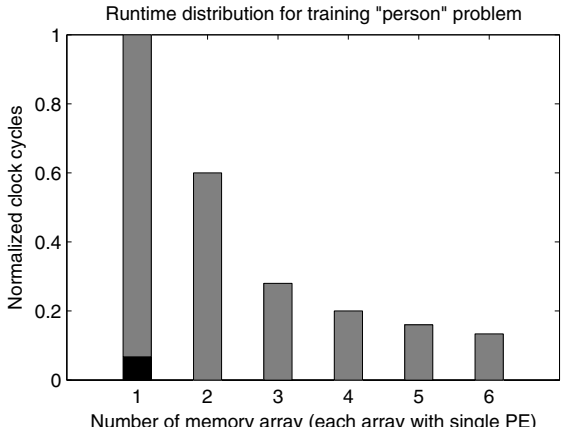

Fig. 7. Runtime with different number of memory arrays

5. CONCLUSION

In this paper, a novel FPGA architecture performing the Random Forest training stage is presented. The presented architecture is capable of achieving a speedup of up to 230x when compared to a software implementation on a CPU processor without any loss in classification accuracy. Key elements of the proposed architecture is a novel FIFO based merge sorter that reduces the required memory requirements compared to the existing state-of-the-art architecture, which is translated to a larger parallelisation factor for a given FPGA area, and a batch training strategy that allows the exploitation of the on-chip memory blocks, removing the off-chip memory access bottleneck.

6. REFERENCES

[1] L. Breiman, "Random forest," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001.

[2] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in *ICML '06 Proceedings of the 23rd international conference on machine learning*, 2006, pp. 161–168.

[3] H. Bostrm, "Concurrent learning of large-scale random forests," in *Proceedings of the Scandinavian conference on artificial intelligence*, 2011, pp. 20–29.

[4] T. Sharp, "Implementing decision trees and forests on a gpu," in *Proceedings of 10th European conference on computer vision*, Oct. 2008, pp. 595–608.

[5] B. Cope, P. Cheung, W. Luk, and L. Howes, "Performance comparison of graphics processors to reconfigurable logic: a case study," *IEEE transactions on computers*, vol. 59, no. 4, pp. 433–448, Apr. 2010.

[6] R. Duda, P. Hart, and D. Stork, "Pattern classification (2nd edition)," pp. 473–484.

[7] J. Quinlan, "C4. 5: programs for machine learning. vol. 1," *Morgan kaufmann*, 1993.

[8] L. Breiman, J. Friedman, R. Olshen, and C. Stone, "Classification and regression trees," *Monterey, CA: Wadsworth and Brooks/Cole Advanced Books and Software*, 1984.

[9] U. Fayyad and K. Irani, "On the handling of continuous-valued attributes in decision tree generation," *Machine learning*, vol. 8, no. 1, pp. 87–102, Jan. 1992.

[10] C. Chen, A. Liaw, and L. Breiman, "Using random forest to learn imbalanced data."

[11] M. Golawala and J. Hulse, "An empirical study of learning from imbalanced data using random forest," in *ICTAI 2007 Tools with artificial intelligence*, vol. 2, Oct. 2007, pp. 310–317.

[12] M. Papadonikolakis and C. Bouganis, "A heterogeneous fpga architecture for support vector machine training," in *Field-Programmable Custom Computing Machines(FCCM) 2010 18th IEEE Annual international symposium on*, 2010, pp. 211–214.

[13] H. Meng, K. Appiah, and P. Dickinson, "Fpga implementation of naive bayes classifier for visual object recognition," in *Computer Vision and Pattern Recognition Workshop(CVPRW), 2011 IEEE computer society conference on*, June 2011, pp. 123–128.

[14] R. Narayanan, D. Honbo, G. Memik, A. Choudhary, and J. Zambreno, "An fpga implementation of decision tree classification," in *DATA '07 Design, automation and test in europe conference and exhibition*, Apr. 2007, pp. 16–20.

[15] R. Marcelino, H. Neto, and J. M. Cardoso, "Sorting units for fpga-based embedded systems," *Distributed Embedded Systems: Design, Middleware and Resources*, pp. 11–22, 2008.

[16] D. Koch and J. Torresen, "Fpgasort: a high performance sorting architecture exploiting run-time reconfiguration on fpgas for large problem sorting," in *FPGA '11 Proceedings of the 19th ACM/SIGDA international symposium on field programmable gate arrays*, 2011, pp. 45–54.

[17] M. Everingham, C. L. Van-Gool, J. W. Williams, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results."

[18] K. Chatfield, V. Lempitsky, A. Vedaldi, and A. Zisserman, "The devil is in the details: and evaluation of recent feature encoding methods," in *BMVC 2011 22nd British machine vision conference*, Aug. 2011.