IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 5, 2020, accepted February 18, 2020, date of publication February 28, 2020, date of current version March 10, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2976879

A Real-Time Naive Bayes Classifier Accelerator on FPGA

ZHEN XUE[1,2], JIZENG WEI[1], AND WEI GUO[1,2]
[1]College of Intelligence and Computing, Tianjin University, Tianjin 300354, China
[2]Tianjin Key Laboratory of Advanced Networking, Tianjin 300354, China

Corresponding author: Jizeng Wei (weijizeng@tju.edu.cn)

This work was supported in part by the Tianjin Key Laboratory of Advanced Networking (TANK), and in part by the Science and Technology Key Project of Tianjin under Grant 17YFZCGX01180 and Grant 18JCQNJC00400.

ABSTRACT In this paper, we propose a real-time hardware naive Bayes classifier (NBC) which is implemented on field programmable gate array (FPGA). We first use logarithm transformation based look-up table and float-to-fixed point process to simplify the calculations in naive Bayes classification algorithm. The methods clear up the multiplication and division operations of floating points completely. Based the simplified algorithm, we design our hardware architecture which includes both training and inference part. A novel format of logarithm look-up table with very limited items and a shifter in it are working together to calculate the logarithm value of any number. There are several processing element (PE) arrays in the accelerator where each PE in an array is running in parallel, which speed up the classification process remarkably. The experiments prove that the proposed accelerator has much better real-time efficiency than the general processor, some hardware Bayes classifiers and convolutional neural network (CNN) accelerators. It outperforms the NBC and semi-NBC accelerators and costs far less resources on chip than many CNN accelerators. Its utilization of LUT, FF and BRAM is only 10%, 0.05% and 2% of CNN accelerators on average. The experimental results over five datasets of different magnitudes show the accelerator has almost no loss of classification accuracy comparing with ARM Cortex-A9 processor. Their deviation of the classification accuracy is only 0.39% on average. What's more, it improves the performance of the training phase and the inference phase about 7.9+e4 and 8.3+e4 on average, respectively.

INDEX TERMS Accelerator, FPGA, hardware architecture, naive Bayes classifier.

I. INTRODUCTION

Recently, Artificial Intelligence & Internet of Things (AIOT) [28] has gained widely concern with the rapid development of 5G communication. AIoT is an integrate technology of artificial intelligence (AI) and Internet of Things (IoT), which produces and collects massive data to execute high-level AI applications on edge devices. As is well known, edge devices usually have limited resources such as storage space and computing power. Therefore, there are new and higher requirements to the implementation of machine learning (ML) algorithms [1] on edge devices. However, most of current ML applications running on general processors have not been able to meet the needs of AIoT. Fortunately, customized hardware accelerators can fit the scenarios perfectly because of the efficiency and low-cost. From this point of view, it is an inevitable trend to customize special hardware accelerators for ML algorithm.

As an important kind of ML methods, classification algorithms have been broadly studied and improved, including naive Bayes classifier (NBC) [18], [38], [41], neural network (NN) [19], [31], [40], deep learning (DL) [20], [39], [42], decision tree (DT) [21], [43], [44], etc. Naive Bayes classifier has been well developed because of its computational simplicity and high efficiency, which are widely used in various scenarios such as spam filtering [4], image classification [5] and so on. A NBC is a probabilistic machine learning model that is used for classification task. The mathematic crux of it is based on Bayes theorem. It calculates the posteriori probability of a test feature vector of each possible category and then selects the category with the maximum probability as the classification result. Although NBC is simpler than other ML algorithms, it can get fairly good classification precision in so many complex real situations. At the same time, there are only a few arguments in it

The associate editor coordinating the review of this manuscript and approving it for publication was Songwen Pei.

and is very highly scalable, which makes NBC can response in real time and pretty hardware-friendly. These good features make NBC a really ideal machine learning classification algorithm orienting hardware construction to maximize its performance.

At present, although NBC has been applied in various fields, the design of special hardware architecture is rare. Most naive Bayes Classifiers are implemented on general processors. It is not only inefficient and costly, but also can cause difficulties in the developing of AIoT, which involves large data processing such as edging computing and interaction.

With the rapid development of big data, there is massive data movement between edge devices and cloud in a AIoT system. Traditionally, edge devices collect a lot of data and send them to the cloud, where a ML algorithm, e.g. CNN trains the data and then the cloud sends the model to the edge device. Although CNN is a very efficient ML method, this processing pattern costs much time and area resources. As the training part of CNN is too costly to implement on a edge device, making the training and inference part have to be implemented apart in cloud and edge devices. However, to some lightweight applications which NBC algorithm can handle, the resources of edge devices are sufficient to implement both training and inference part. Thus, much data movement can be avoided, which can reduce a lot of time cost and resource consumption. Considering all the above, we propose a NBC accelerator including both training and inference part to make lightweight applications more efficient.

Motivated by all the above, we design our hardware NBC accelerator including both training and inference part. It completely avoids multiplications and divisions of floating points by shift operations and logarithm transformation Based a novel logarithm look-up table (LUT). As far as we know, we are the first who use shift operations and a logarithm LUT to clear up division. Firstly, we simplify NBC algorithm to make it more suitable for hardware implementation. The methods used are float-to-fixed point process and logarithm transformation basing LUT method. By using logarithm transformation, multiplications and divisions can be replaced by additions and subtractions. The logarithm LUT stores only a small quantity of data and can get the logarithmic value of any number. What's more, all the data in the hardware accelerator is converted to fixed-point numbers. These optimization methods working together help reduce resource consumption and simplify calculations greatly. Then, we design our hardware NBC basing on the optimized algorithm, which includes both training part and inference part. Roughly, the accelerator consists of many PE arrays and memories. Each PE in a PE array has the same function and architecture. When a PE array is running, PEs are executing in parallel so that the classifier can be speeded up. Therefore, the efficiency of NBC accelerator is improved remarkably. Finally, from experiments using several datasets, it can be found that our hardware accelerator has better performance than software implementation, some other NBC, semi-NBC and CNN accelerators under the same conditions. It not only has almost no loss of classification accuracy comparing with ARM Cortex-A9 processor, but also consumes very limited resources on chip.

The contributions of this paper are summarized as follows:
(1) We optimize naive Bayes classification algorithm to make it more friendly to hardware implementation. Float-to-fixed point process and logarithm transformation based LUT method are used to simplify the calculations in NBC. Moreover, a novel format of logarithm LUT is designed, which can get the logarithmic value of any number by storing only a few logarithmic data.
(2) Basing the optimized algorithm, we propose our NBC hardware architecture which includes both training and inference part. The NBC accelerator is sped up in many processes by running PEs in parallel in the same PE array. What's more, multiplication and divisions of floating points are completely avoided in the NBC accelerator. Moreover, shift operations and the logarithm LUT are used to replace division operations innovatively, which improves the performance greatly.
(3) A set of experiments demonstrate that our design has a much better real-time performance than software NBC, some hardware Bayes classifiers and CNN accelerators. It has almost no accuracy loss comparing with general processors implementation and outperforms the NBC and semi-NBC accelerator. Moreover, the NBC accelerator outperforms ARM Cortex-A9 processor 7.9+e4 and 8.3+e4 times in training and inference part, respectively. What's more, our design costs very limited resources comparing with many CNN accelerators. The utilization of LUT, FF and BRAM is only 10%, 0.05% and 2% of CNN accelerators on average.

This paper is organized as follows. Section II reviews some related works. We introduce the mathematical principles of naive Bayes Classifier in section III. In section IV, some methods are discussed to optimize naive Bayes Classifier algorithm, which include logarithm based LUT method and float-to-fixed point. We propose the hardware architecture in section V, where the training part and the inference part will be discussed in detail. Section VI shows some experiment results and analyzes the merits of our design. The factors affecting classification precision, time cost or resources utilization will also be exploited. Finally, in section VII, we will conclude the paper briefly.

II. RELATED WORKS

As the simplest form of Bayes classifier, naive Bayes classifiers are implemented on hardware platforms for different application scenarios [23]–[27]. At present, many special hardware design of NBC has been proposed. Meng *et al.* [12] have proposed a novel hardware architecture of naive Bayes classifier for visual object recognition. They aim at binary feature vectors and assume the training data obeys Bernoulli distribution. By using this novel method, about half of the probability calculation can be avoided by doing subtractions. As the sum of probabilities of components value taking "1" and "0" is bound to be 1. What's more, they convert

multiplication of probabilities in inference part of NBC into addition and subtraction operations by using logarithm LUT. Therefore, the performance of the classifier is remarkably improved on the premise of a high accuracy.

To be more efficient, many acceleration methods are applied to hardware NBC. Usually, there is a need to use transcendental functions log() to simplify the computation to speed up a hardware accelerator. Therefore, avoiding the computation of log() is very necessary. So far, there are so many useful methods to solve the problem. References [2], [3], [6], [16], [17], [22], [45] Paul *et al.* [2] have shown an efficient approach to compute log() and antilog() in hardware. The approach is based on LUTs and followed by an interpolation step, resulting in an area-efficient, fast design. The main idea of their approach is to use LUTs along with linear or quadratic interpolation, which avoids multiplication and division subtly. Bariamis *et al.* [3] present a novel method called ALA (Adaptable Logarithm Approximation), which approximates the base-2 logarithm of integers at an arbitrary accuracy on a hardware platform. Their approach is implemented based on a piecewise liner approximation methodology. So the logarithm function can be approximated by an arbitrary number of linear segments.

GI Webb *et al.* [13] propose a new efficient technique that uses a weaker attribute independence assumption than NB called AODE (Aggregating One-Dependence Estimators), which averages all of a constrained class of classifiers. AODE improves prediction accuracy without undue computational overheads and it is particularly suited to incremental learning. Choi and Lee [14] designed and implemented a parallel hardware of semi-naive Bayes Classifier on FPGA. They proposed a method called parallel semi-naive Bayes (PSNB) classification based on stochastic discrimination (SD) theory innovatively, which simplifies the calculation process and improves the classification efficiency.

Instead of aiming at binary feature vectors like [12], the design proposed in this paper can deal any kind of numeric feature vectors. It is known that the main computation bottleneck of NBC algorithm is the division and multiplication computation in the training and the inference phase. To improve the efficiency and reduce the time and area cost of the NBC accelerator, we remove all the multiplication and division operations of floating points in our design, while [12] use a float-point division operator. What's more, we customize a novel logarithm LUT for our NBC accelerator to calculate logarithm values more efficiently. While the logarithm computation hardware architectures above mentioned are designs to calculate log() function rather than aiming at specific applications.

III. THE THEORY OF NAIVE BAYES CLASSIFIER

Assume there is a dataset $\{(X^i, y^i), \mathrm{i} = 1, 2, \ldots, \mathrm{N}\}$ from C categories. $X^i = (x_0^i, x_1^i, \ldots, x_{n-1}^i)$ is a n-dimensional feature vector and y^i is its label, i.e. the category, and $\{y^i = \mathrm{c}, \mathrm{c} = 0, 1, \ldots, \mathrm{C}-1\}$. There is a new feature vector $\widehat{X}$ whose label is unknown. Its label can be predicted as $\hat{y}$:

$$\hat{y} = \underset{c=0,1,\ldots,C-1}{\arg\max} P(y = c|\widehat{X}) \tag{1}$$

The Bayes theorem can be used to compute $\hat{y}$:

$$\hat{y} = \underset{c=0,1,\ldots,C-1}{\arg\max} \frac{P(y = c)P(\widehat{x_0}, \widehat{x_1}, \ldots, \widehat{x_{n-1}}|y = c)}{P(\widehat{X})} \tag{2}$$

There is a very important presupposition in naive Bayes, which is that all components in a feature vector are completely independent. Thus, the joint probability equals the product of their probabilities.

$$P(\widehat{x_0}, \widehat{x_1}, \ldots, \widehat{x_{n-1}}|y = c) = \prod_{k=0}^{n-1} P(\widehat{x_k}|y = c) \tag{3}$$

What's more, $P(\widehat{X})$ is always the same to a specified data set. Thus,

$$\hat{y} = \underset{c=0,1,\ldots,C-1}{\arg\max} P(y = c) \prod_{k=0}^{n-1} P(\widehat{x_k}|y = c) \tag{4}$$

The relative frequency can be used to estimate probability, i.e.

$$P(y = c) = \frac{Num_{y=c}}{Num} \tag{5}$$

If set $M_{k_i} = \{m_{k_i}|1 \leq i \leq t\}$ denotes the value set of component $x_k (0 \leq k \leq n-1)$ of a feature vector and $|M_k| = t$. Then the conditional probability can be got.

$$P(x_k = m_{k_i}|y = c) = \frac{Num_{y=c \cap x_k = m_{k_i}}}{Num_{y=c}} \tag{6}$$

In (5) and (6), Num is the number of the data set and $Num_{y=c}$ is the number of samples whose label is c. $Num_{y=c \cap x_k = m_{k_i}}$ denotes the number of samples whose label is c and the value of component x_k is m_{k_i}.

Laplace smooth is usually applied to (5) and (6) to avoid zero probabilities:

$$P(y = c) = \frac{Num_{y=c} + 1}{Num + C} \tag{7}$$

$$P(x_k = m_{k_i}|y = c) = \frac{Num_{y=c \cap x_k = m_{k_i}} + 1}{Num_{y=c} + t} \tag{8}$$

In conclusion, NBC get $Num_{y=c}$ and $Num_{y=c \cap x_k = m_{k_i}}$ by doing some counting operations through training data in the training phase. Then it calculates all kinds of probabilities using (7) and (8). NBC uses (4) to compute the prediction label $\hat{y}$ of a test feature vector in the inference phase. Table 1 summarizes some common arguments in this paper.

IV. HARDWARE DESIGN ORIENTED NBC ALGORITHM

Although NBC is a relatively simple ML classification algorithm, it involves a lot of complex calculations of floating points such as multiplications and divisions. All these computation will damage the efficiency of the NBC accelerator if they are implemented on the FPGA directly. Because of these factors, it is very necessary to simplify the NBC algorithm

TABLE 1. Common arguments in this paper.

Arguments	Description
N	The number of samples in the training set
C	The number of categories
c	The label of a sample
n	The dimension of feature vectors
m_{k_i}	The value of the kth component
t	The number of possible values of a component

to make it more suitable to hardware design. According to section III, the main computation in NBC is the divisions in the training phase and the multiplications in the inference phase.

To make NBC algorithm more friendly to hardware implementation, we clear up all the multiplications and divisions of floating points in it by using logarithm transformation based on LUT and float-to-fixed point process. According to section III, there are massive floating-point calculations of multiplications and divisions in both training and inference phase, which require lots of time and space. In general, edge devices in a AIoT system have very limited resources, including storage space, computation power, bandwidth and so on. Therefore, it will damage the efficiency of NBC certainly if the large quantities of complex floating-point computation operations are put into hardware directly. Under the premise of almost no loss of classification accuracy, we use logarithm transformation based on LUT method to simplify the calculations in NBC algorithm. In this way, all the computation operations in NBC are converted into additions and subtractions. What's more, all the floating-point numbers are converted into fixed-points, which fits real edge-device scenarios perfectly.

A. LOGARITHM TRANSFORMATION

We store many base-2 logarithm values in memory and using them to transform the multiplications and divisions into addition and subtraction operations. By using (5) and (6), it needs to do lots of divisions to get the various probabilities. There are also massive multiplications when computing posteriori probabilities in the inference part of NBC, according to (4). Thus, if all the divisions and multiplications are avoided, much time and space resources will be saved.

After taking logarithm of the right side of (4), the equality is still tenable. Meanwhile, the divisions and multiplications are converted into additions and subtractions. According to the inference phase of naive Bayes classifier in section III, the prediction result is the label which makes$P(c)\prod_{k=0}^{n-1}P(\widehat{x_k}|c)$ take the maximum value. In this sense, there is no need to calculate the precise posteriori probabilities as (4) is still right when its right side is monotonically transformed (It takes the minimum if it is transformed by a monotonic decreasing function). That is, $P(c)\prod_{k=0}^{n-1}P(\widehat{x_k}|c)$ can be the variable of any monotonic function as the transformation does not change the prediction label, i.e. the result of (4). Fortunately, logarithmic function is a monotonic function and it can be used to avoid multiplication and division operations. What's more, the probabilities are suitable for the definition domain of variables of logarithmic functions as they are non-negative values. Basing on the above-mentioned considerations, we use logarithm transformation to convert multiplications and divisions into addition and subtraction operations. We discuss the logarithm transformation in the inference phase firstly and then in the training phase in the following paragraphs.

In the inference part, all the multiplications are transformed into additions. According to section III, (4) is used to compute the prediction label of a feature vector. The label can also be got by taking the logarithm of the right side of the equation:

$$\hat{y}=\underset{c=0,1,\ldots,C-1}{\arg\max}\left[\log_2 P(y=c)+\sum_{k=0}^{n-1}\log_2 P(\widehat{x_k}|y=c)\right] \quad (9)$$

So these complex multiplications are converted into comparatively simple additions. To get the final prediction label, we need to obtain the various probabilities which are calculated in the training phase. Similarly, the logarithm values of the probabilities replace the probabilities in (7) and (8) by using logarithm transformation in the training phase.

$$\log_2 P(c)=\log_2(Num_{y=c}+1)-\log_2(Num+C) \quad (10)$$

$$log_2 P(x_k=m_{k_i}|c)=log_2(Num_{y=c\cap x_k=m_{k_i}})-log_2(Num_{y=c}+t) \quad (11)$$

So far, all the multiplications and divisions in NBC have been converted into additions and subtractions, which reduces the time and space cost remarkably.

B. LOGARITHM LUT

We get logarithm values of probabilities by using logarithm transformation and a novel format of logarithm LUT, which stores only a few logarithm values and can get the logarithm of any number by fetching and some addition operations. In section IV A, although logarithm transformation is used to simplify the computation in training and inference, the numerical calculation is still very complex because it involves transcendental functions, i.e. logarithm functions. Thus, to make it more friendly to edge-device computation, we build a LUT to store the logarithmic values rather than calculate them in real time. It is referred as logarithm LUT. In this way, a lot of computation operations in NBC can be replaced by memory access operations to reach higher efficiency and performance.

The basic idea of the logarithm LUT is selecting a few appropriate numbers in a small interval and storing their base-2 logarithm values in memory. The logarithm value of a number can be calculated by decomposing it into the product of a number in this interval and a power of two. According to (10) and (11), all data needing to take logarithm is the counting results after Laplace smoothing, which is denoted by N. Therefore, N can be equal to Num_C, $Num_{y=c}+1$, $Num_{y=c}+t$ or $Num_{y=c\cap x_k=m_{k_i}}+1$. The other count results are unknown before the counting work is finished excepting

$Num + C$ is constant to a training dataset. Easily, it can be known that the antilogarithm range is the integer interval $[0, Num + C]$. So the range of the logarithmic results is a real interval $\left[-\infty, \log_2 (Num + C)\right]$. Such a large number of values stored in on-chip memory not only occupies too many memory cells, but also lead to low utilization because of the dispersion of the count results. Considering all of these factors, we explore a way of appointing a small real interval $[a, b]$. It equidistantly takes out some numbers and then stores their base-2 logarithmic values in a LUT. Meanwhile, N is decomposed into the product of two numbers, m and 2^x. And m is a fixed-point real number in $[a, b]$.

$$log_2 N = log_2(m \times 2^x) \tag{12}$$

Equal to:

$$log_2 N = log_2 m + x \tag{13}$$

In this way, the depth and range of the logarithm LUT is very flexible and not limited by the number of training samples. The logarithm LUT stores the logarithm value of m. There is a need to fetch $log_2 m$ from the LUT and execute an addition operation to get the logarithm value of N. Moreover, we opt base-2 logarithm because any value stored in FPGA is binary, which makes it possible to decompose N by shift operations. The number of shift bits is denoted as x.

In our design, $[a, b] = [0.5, 1]$, e.g. $m \in [0.5, 1]$. When N is decomposed, we follow the rule that m should be less than *1* and the number of significant bits of it should be as many as possible. Therefore, the integer part of m is zero as m is bound to be in $(0, 1)$. Further more, m has very limited fractional bits as it is fixed-point. So the significant bits of m should be set as many as possible to make the logarithm value of N more accurate. All the fractional bits of m can be made being significant. In this sense, the first bit of the fractional part is always one, so $m \in [0.5, 1)$. While being decomposed, N is reduced by shifting bit by bit. Shift operations stop immediately once m is less than *1*. Thus, after the decomposition, the integral part of m is *0* and the fractional part's first bit is bound to be *1*. w can be used to denote the width of the rest part of the fractional part. m_b is the binary form of m.

$$m_b = 0.1\underbrace{addr}_{\text{w bits}} \tag{14}$$

There is 2^w possible values of m, so the logarithm LUT should store 2^w logarithmic results. What's more, there is no need doing extra computation to get the address of a logarithm value $log_2 m$. Because $addr$ can be used as the address of the logarithm LUT directly to obtain the corresponding value, which is the fixed-point form of $log_2 m$ and can be donated as l. According to logarithm functions, $log_2 m$ is in interval $[-1, 0)$ as $m \in [0.5, 1)$, so the integer part of l needs only one bit. It is important to note that the data in the logarithm LUT is the absolute value of $log_2 m$.

Figure 1 is an example of a logarithm LUT architecture. Assume the width of addresses of the logarithm LUT is 15-bit. Therefore, the LUT has 2^{15} items. Its width of the items is 8-bit, and the fractional part takes up 7 bits. There is an item whose address in the LUT is $0 \times 22b8$, i.e. $addr = 010\_0010\_1011\_1000$. Thus, $m_b = 0.1010\_0010\_1011\_1000$, and $log_2 m = 0.1010011$, which is the value in the logarithm LUT corresponding to the address $0 \times 22b8$.

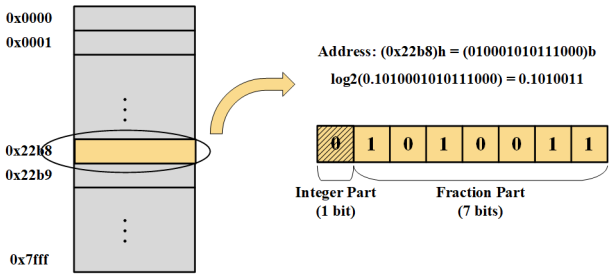

FIGURE 1. An Example of a Logarithm LUT. The logarithm LUT has $2^{15} = 32768$ items. The address of an item is $0 \times 22b8$, whose binary form is 010_0010_1011_1000, i.e. $addr = 010\_0010\_1011\_1000$, which is 15-bit. According to (14), the corresponding m_b is 0.1 $addr$ = 0.1010_0010_1011_1000. After taking logarithm, $log_2 m = 0.1010011$, which is the value of the item in the logarithm LUT. Each item in the LUT is 8-bit, including one-bit integer part and 7-bit fraction part. The shaded portion in the diagram is the integer part of the value.

V. NBC SPECIFIC HARDWARE ACCELERATOR

A. OVERVIEW

Basing on the simplified NBC algorithm in previous sections, we design our NBC accelerator considering both training and inference part, which is shown in Figure 2. Instead of dealing binary feature vectors only [12], our architecture is designed for any kind of numeric feature vectors, i.e. a component value of a feature can be equal to any number. Moreover, there are no floating-point multiplications and divisions in the NBC accelerator, while [12] use a floating-point division operator. Our design consists of a controller, a counting processing element array (CNT PEs), a counting adder-LUT array (CNT A-LUTs), a probability computation PE array (PC PE array), a probability LUT (P-LUT), a posteriori probability computation PE array (POS PEs) and a comparer.

The NBC controller coordinates the other parts of the NBC accelerator to make them run orderly, which receives some state signals from other parts and then sends controlling signals. The timeline in Figure 2 shows the execution sequence of all the parts of the NBC accelerator:

t1: The CNT PE array counts the number of all kinds of labels and components to get $Num_{y=c}$ and $Num_{y=c \cap x_k = m_{k_i}}$. There are several counting PEs working in parallel. In every clock cycle, each counting PE counts the label or a component of a training sample. Meanwhile, its address is computed and sent to the corresponding adder-LUT. The counting adder-LUT array has as many elements as the counting PE array, as shown in Figure 2. When receiving an address, an adder-LUT adds one to the corresponding item in the LUT.

t2: The probability computation PE array computes the logarithm probabilities in (10) and (11) by using the

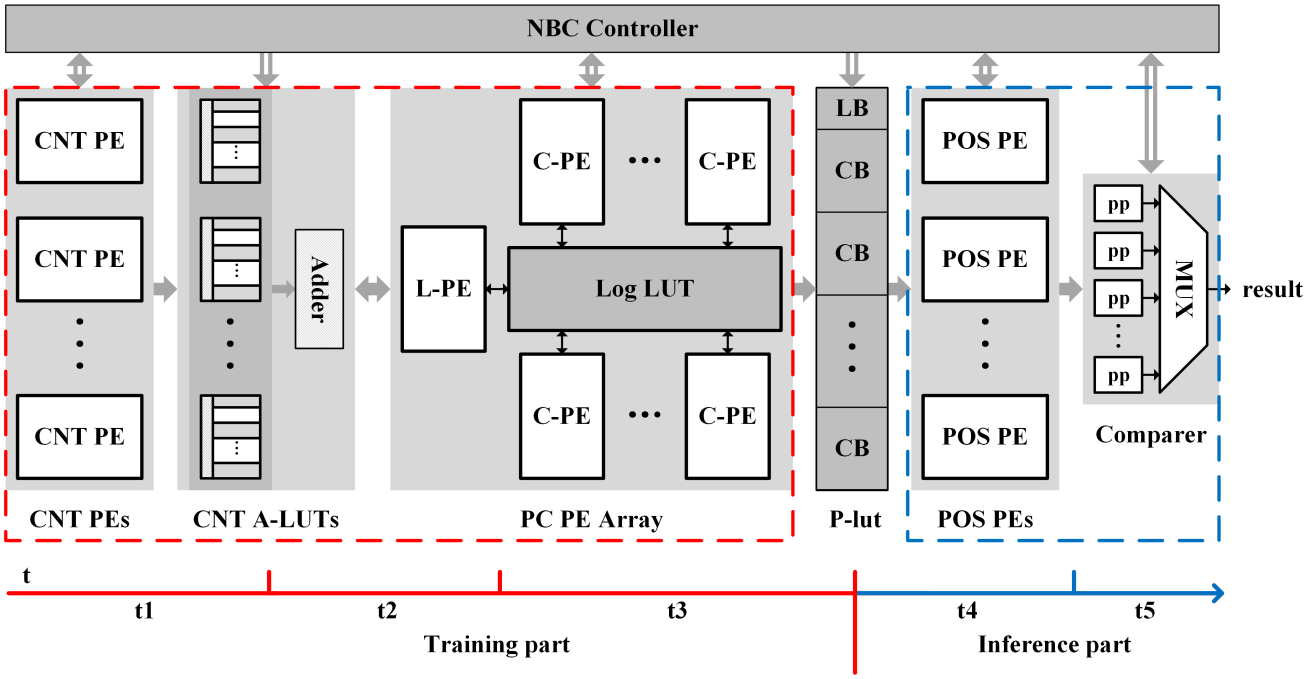

FIGURE 2. The architecture of the NBC accelerator.

logarithm LUT. After counting operations are finished, the label PE (L-PE) in the PC PE array calculates logarithm values of all kinds $P(c)$ in (10) and sends them to the probability LUT. When receiving a reading address, the adder in CNT A-LUTs sums up all the items corresponding to the same address in each adder-LUT and sends them to the PC PE array. That is, Num, $Num_{y=c}$ and $Num_{y=c \cap x_k=m_{k_i}}$ are sent to the PC PE array. The probabilities of labels are calculated in the L-PE firstly because $Num_{y=c} + t$ are also computed in it, which will be used to calculate $log_2 P(x_k = m_{k_i}|c)$ in the component PEs, according to (11).

t3: There are several component PEs (C-PE) in the PC PE array calculating logarithm probabilities of components. All the component PEs are running in parallel. Once working out a logarithm probability, the PC PE array sends it to the probability LUT, which stores the training results for the inference part.

t4: The POS PE array has several POS PEs, which calculate the posteriori probabilities of different labels. To a test feature vector, each PE computes the posteriori probability corresponding to one label simultaneously. After that all the posteriori probabilities are sent to the comparer.

t5: The comparer selects the maximum posteriori probability, whose corresponding label is the prediction result.

In Figure 2, the parts running during t1, t2 and t3 form the training part of the NBC accelerator. The parts running during t4 and t5 belong to the inference part. The hollow arrows are controlling signals and the solid arrows are data paths.

B. COUNTING ADDER- LUT AND PROBABILITY LUT

We combine adders and LUT to avoid too much data transference. The LUT part of a counting adder-LUT stores all kinds of $Num_{y=c}$ and $Num_{y=c \cap x_k=m_{k_i}}$. When receiving a writing address, a counting adder-LUT accesses the corresponding location and increases its value by one. There is an adder in each adder-LUT for increasing the items when writing. There is also an adder for all the adder-LUTs to get the whole counting result when reading, as Figure 2 shows. In this way, the counting adder-LUT array outputs the sum of the counting values from every adder-LUTs while receiving a reading address. Each counting PE relates an adder-LUT and they work in parallel. If we use only an adder-LUT, it can cause data conflicts because many counting PEs may produce the same address. Thus, there is a need to change the item of the same location at one time. Rather, the data can be read by sending the same address to each adder-LUT concurrently. So an adder is enough to sum and output the whole counting result.

The LUT part of an adder-LUT is divided into many blocks. There are one label block(LB) and several component blocks(CB). The label block stores the numbers of training samples having different labels, i.e. $Num_{y=c}$, and the component blocks stores the numbers of samples corresponding to different component values, *0, 1, …* i.e. $Num_{y=c \cap x_k=m_{k_i}}$. Therefore, the label block at least has at least C items. What's more, there are at least t component blocks and $C \times n$ items in each of them.

We arrange all kinds of the counting results in a specific order. In the counting adder-LUT, the label block is located at the lower addresses and the component blocks are located at the higher addresses. Each of the CBs corresponds to a component value, and the CB corresponding to 0 is at the lower addresses. In the label block and every component block, the counting results having different labels are stored in ascending order, from lower to higher addresses. What's more, the counting results corresponding to the same label are arranged in order of increasing dimension (0 to *n-1*) in each component block. For a training sample whose label is c and the value of the k-th component is m_{k_i}. Its address in a counting adder-LUT corresponding to its label is c, which locates in the label block. The address corresponding to the component c is

$$LB_{size} + CB_{size} \times m_{k_i} + c \times n + k \tag{15}$$

LB_{size} and CB_{size} are the number of items in the label block and a component block in a counting adder-LUT, respectively.

The probability LUT has the same structure as the LUT part of a counting adder-LUT. It also has one label block and several component blocks, which have the same number of items as a counting adder-LUT. That is because there is a one-to-one corresponding relationship between the probabilities and the counting results.

C. COUNTING PE

Each counting PE goes through every training sample in its local training data memory to do counting operations until all the training samples are dealt with. Figure 3 is the structure of a counting PE. As is shown, there are two local memories in a counting PE, which store training labels and training feature vectors, respectively. And their items of the same location correspond to the same training sample. The local memories are implemented by BRAMs. The address generator increases the training data memory address to fetch a new training sample when the counter finishes the last one. What's more, the address generator checks whether the address reaches the bound of the training data memory by comparing the current address with the maximum address of the training data memory. If all the training data are gone through, the generator will not increase the address.

In every clock cycle, the counter computes an address to update the counting adder-LUT using (15) in subsection V B. To a training sample, the address of the label is calculated in the first clock cycle, and then the address of each component is calculated in the following clock cycles. The label or the component data is selected by a multiplexer to determine which kind of addresses to compute. The address computer (Addr Computer) in the counter calculates the address corresponding to a label or a component using (15).

The several counting PEs in the NBC accelerator are running in parallel. Each of them is related to a counting adder-LUT. In our design, the capacity of each local training memory is the same. The counting process is finished only when all the counting PEs finish their counting operations.

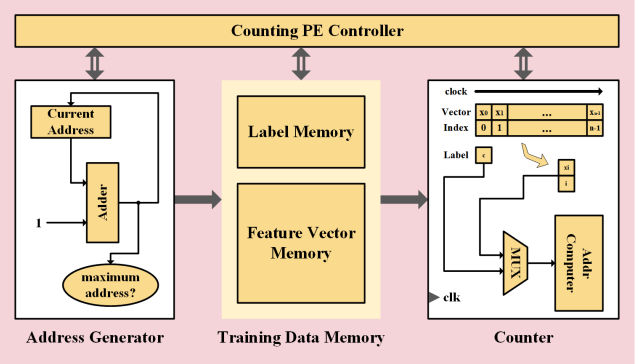

FIGURE 3. The architecture of a counting PE.

D. PROBABILITY COMPUTATION PE

The probability computation PE array calculates the logarithm probabilities in (10) and (11) and sends them to the probability LUT. There are many PEs in the PC PE array, including one label PE and several component PEs. They are used to calculate the logarithm probabilities of labels and components, respectively. After the counting operations are completed, the PC PE array gets counting results from the counting adder-LUTs. Then it calculates their logarithm values by shift operations and the logarithm LUT, according to subsection IV B. After that, the PC PEs work out all kinds of logarithm probabilities using (10) and (11) and store them in the P- LUT. The computation there only involves fixed-point additions and subtractions.

Figure 4 (a) is the structure of a component PE. There are five adders, a multiplexer, a label register array, a shifter and a probability computer in the component PE, which are yellow-shaded. The dotted rectangles are parts outside of the PE. Using logarithm data from the log LUT and the number of shift bits (SFT bits) from the shifter, the probability computer calculates the logarithm probabilities and sends them to the P-LUT. As the data in logarithm LUT is absolute values, additions and subtractions should be selected as appropriate when computing the logarithm value of a number. According to (10), (11) and (13), all probabilities can be get in this way:

$$log_2 P(C) = log_2(Num + C) + log_{y=c} - sft\_bits \tag{16}$$

$$log_2 P(x_k = m_{k_i}|c) = log\_reg_c + log_{y=c \cap x_k = m_{k_i}} - sft\_bits \tag{17}$$

where $log_{y=c}$ and $log_{y=c \cap x_k = m_{k_i}}$ are the corresponding item in the logarithm LUT to label c and component x_k. log_reg_c is equal to $log_2(Num_{y=c} + t)$ in (11). Process 1-5 depict the data flow in a probability computation PE.

Process ❶ - ❺ depict the data flow in a probability computation PE.

❶: After the last logarithm probability is calculated, the address adder (Addr Adder) increases the current counting address (CNT Addr) by one to access the next counting result in the counting adder-LUT array.

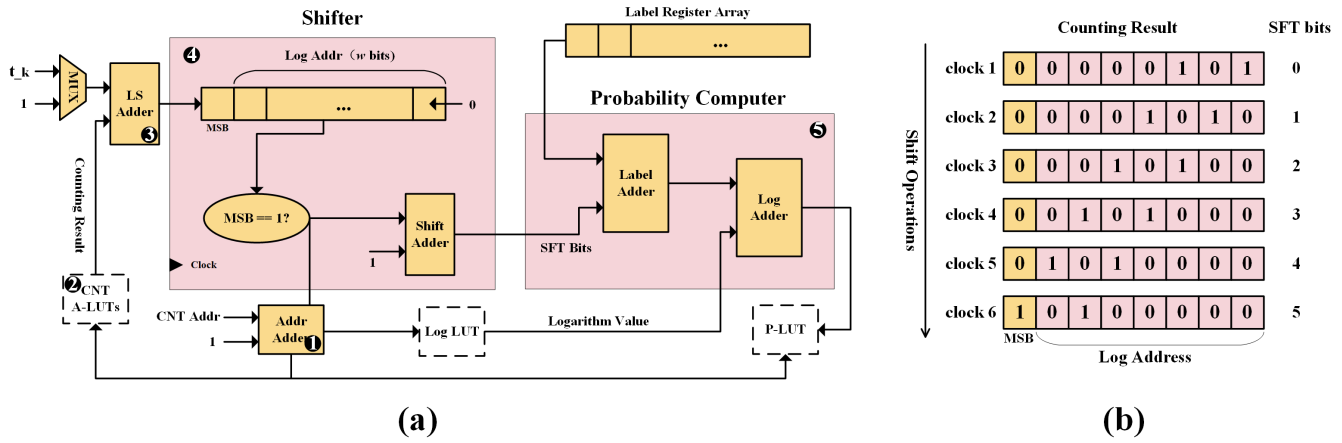

FIGURE 4. (a) The architecture of a component PE in the PC PE array. (b) An example of the shift process.

❷: When receives a new address, the counting adder-LUT array sends the corresponding counting result to the Laplace smooth adder (LS Adder).

❸: The multiplexer and the LS Adder work together to do Laplace smooth. They add 1 or t to the counting result if it is a label count, and add 1 if it is the number of the k-th component of a feature vector.

❹: The shifter reduces the received number through left shift operations. In each clock cycle, the shifter left shifts the number and checks whether the most significant bit(MSB) is *1*. If it is not, the shift adder in the shifter increases the shift bits by one. Instead, the shifter outputs the shift bits to the probability computer. Meanwhile, it outputs the logarithm address to fetch the corresponding log value, which is the low w bits of the shifted number. Figure 4 (b) is an example to explain this shift process. Supposing that there is a 8-bit number (0000_0101)b. In every clock cycle, it is left shifted one bit and the shift bits is increased if the MSB is not *1*. After 5 clock cycles, its MSB turns to be *1*. Then shift operations are over. Thus, the address of the logarithm LUT is the low *7* bits, which is (1010_0000)b. And the number of the shift bits is *5*.

❺: According to (17), there are an addition and a subtraction operation to get a component probability. The label adder does the first addition, which is the sum of log_reg_c and $log_{y=c \cap x_k=m_{k_i}}$. The log adder does the second addition to get $log_2 P(x_k = m_{k_i}|c)$.

There is a difference between the label PE and the component PEs, as the computation ways of the two kinds of probabilities are not the same, according to (16) and (17). The label PE also computes $log_2(Num_{y=c} + t)$ in (11), which are stored in a register array referred as label register array. Therefore, the label PE should run before the component PEs, as the latter need the values in the label register array to compute logarithm probabilities.

E. INFERENCE PART

For a test feature vector, the inference part computes the posteriori probabilities of every label and then get the prediction result, which is the label taking the maximum posteriori probability. The posteriori probability computation PE array does some additions to get the posteriori probabilities, where each POS PE computes the posteriori probability of a label. In every clock cycle, a POS PE gets the value of a component of the feature vector, and then computes its corresponding address in the P-LUT by using (15). Therefore, the hardware structure of the address computer in a POS PE is the same as the counter of a counting PE in Figure 4 (a). Meanwhile, the posteriori probability (PP) is accumulated. After the feature is gone through completely, the label probability is added to the current PP. What's more, all the PEs run in parallel.

After working out the posteriori probability of each label, the comparer selects the maximum. The corresponding label of it is the inference result. This highly parallel inference structure makes the NBC accelerator very suitable for real-time applications running on edge devices.

TABLE 2. The basic properties of the five datasets.

Dataset	N^*	n^*	C^*	t^*
MNIST	60000	784	10	2
Car	1728	6	4	4
Connect	67557	42	3	3
Voting	435	16	2	3
Monk	432	8	2	4

*N: The number of samples in the training set; n: The dimension of feature vectors; C: The number of categories; t: The number of possible values of a component. The arguments are all shown in Table1.

VI. EXPERIMENTS

The proposed hardware accelerator is implemented on a Xilinx ZYNQ 7020 device, using Verilog HDL and Vivado 2018.1. We make experiments on five data sets of different magnitudes. They are MNIST data set and some UCI data sets which are Car Evaluation Dataset (Car), Connect-4 opening dataset (Connect), 1984 United States Congressional Voting Records Dataset (Voting) and The Monk's Problems DataSet (Monk). All the datasets are shown in Table 2.

All the data sets are preprocessed into a uniform format. Their label values are quantified into integers from zero

to *C-1* and the value of a component x_k is also a integer of interval $[0, t-1]$. In order to apply MNIST dataset into naive Bayes classifier more effectively, we convert the original images of 28*28 pixels into binary vectors with size of 784-bit [15]. Not only fitting binary feature vectors like [12], our design also fits all kinds of classification datasets.

As CNN accelerators ([9], [10], [35]) are very popular ML accelerators these days, we compare our design with some state-of-the-art CNN accelerators [32]–[34], [36]. Table 3 shows the resources usage of the NBC accelerator is very limited. The CNN accelerators and our design are all implemented on Xilinx boards. The NBC accelerator are configured with six counting PEs, five computation probability PEs, including one label PE and four component PEs, and ten posteriori probability computation PEs.

TABLE 3. Hardware resource consumption of our design and the CNN accelerators.

Resource	[32]	[33]	[34]	[36] A	[36] B	Ours
Board	ZC706	ZC706	ZC706	Zynq XC7Z045	Zynq XC7Z045	Zynq 7020
LUT	90000	155000	93000	149037	93977	11243
FF	92000	153000	96000	95284	57033	49
BRAM	540	732	528	211.5	99.5	1

As shown in Table 3, the resource consumption of the NBC accelerator is much less than the CNN accelerators. Its LUT utilization is about 10% of the CNN consumption on average. The FF and BRAM is only 0.05% and 2% on average, respectively. Comparing with the CNN hardware implementations, our NBC accelerator costs far limited resources, which makes it more suitable for edge devices in the AIoT.

A. PERFORMANCE

We compare the time cost of the training and the inference part of the NBC accelerator with the software implementation on a general purpose CPU integrated in Xilinx Zynq-7000 SoC ZC702 board, i.e., ARM Cortex-A9 processor. Table 4 and Table 5 show their time cost. The clock frequency of ARM Cortex-A9 is 400MHz and the NBC accelerator is 100MHz.

TABLE 4. Time cost of the training part.

Dataset	ARM Cortex-A9		NBC Accelerator	
	CC	RT (ns)	CC	RT (ns)
MNIST	2.7+e10	6.8+e10	8.0+e6	8.0+e7
Car	3.3+e7	8.3+e7	3.7+e3	3.7+e4
Connect	2.0+e11	5.0+e11	5.4+e5	5.4+e6
Voting	1.2+e7	3.0+e7	1.8+e3	1.8+e4
Monk	4.5+e6	1.1+e7	1.1+e3	1.1+e4

As Table 4 and Table 5 show, the time cost of the training and the inference part of the NBC accelerator are both far less than the counterparts of the ARM Cortex-A9 processor. As to clock cycles (CC), our design improves the efficiency of the training part up to 3.7+e5 times and 7.9+e4 times on average. What's more, the NBC accelerator improves the performance of the inference part 8.3+e4 times on average, up to 3.5+e5 times. Similarly, the running time (RT) of the training and the inference part are reduced 2.0+e4 times and 2.1+e4 times on average, respectively. Therefore, the proposed NBC accelerator improves the training and the inference efficiency so remarkably, comparing with the software implementation.

TABLE 5. Time cost of the inference part.

Dataset	ARM Cortex-A9		NBC Accelerator	
	CC	RT (ns)	CC	RT (ns)
MNIST	3.9+e7	9.8+e7	800	8.0+e3
Car	6.2+e4	1.6+e5	13	1.3+e2
Connect	1.7+e7	4.2+e7	48	4.8+e2
Voting	1.5+e5	3.8+e5	21	2.1+e2
Monk	3.9+e4	9.8+e4	13	1.3+e2

Generally, the response time in the order of microseconds can be referred as a real-time response [29], [30]. As it shows, the time cost of the inference part of our design shown in Table 5 are all in the order of microseconds or nanoseconds. So the inference part of our NBC accelerator can satisfy the real-time requirement of AIoT platforms.

We implement the NBC accelerator in [12] and the semi-NBC accelerator in [14] on the Xilinx ZYNQ 7020 board to compare their performance (Per. in Table 6) with our design. Table 6 shows the resource consumption of the three designs and the running time (*TC (ns)*) of inferring a feature vector from the dataset *Connect* and the classification accuracy (*Acc. (%)*). To compare the performance of the two accelerators and our design, we use *Time Cost / (LUT * BRAM)* to evaluate the time cost of each unit resource consumption. The experimental results shows that our design outperforms the other two accelerators, as it has the smallest unit time cost.

TABLE 6. Hardware resource consumption, time cost, classification accuracy and performance of the NBC accelerator in [12], the semi-NBC accelerator in [14] and our design.

Accelerator	LUT	FF	BRAM	IO	TC (ns)	Acc. (%)	Per.
Design [12]	6785	32	2	42	3.6+e3	71.89	0.27
Design [14]	36776	89	16	46	7.2+e4	83.36	0.12
Our Design	11243	49	1	38	4.8+e2	72.21	0.04

The simplicity of NBC algorithm makes its hardware implementation cost less resources than the counterpart of semi-NBC. As semi-NBC relaxes the independence assumption, it has higher classification accuracy. However, there are much more complex computation such as multiplications in the semi-NBC accelerator than NBC hardware implementation. Therefore, the design [14] consumes more hardware resources and time cost than our design, which leads to higher unit time cost.

Because of the parallel PE processing and no multiplication and division operations of floating points in our design, the NBC accelerator can reach a higher performance.

The accelerator in [12] has no parallelism strategy, which makes it consume more time when inferring a test feature vector. What's more, the design in [12] has a float-point divider in its hardware implementation, which can cost much hardware resources and damage the efficiency. We use a novel format of logarithm LUT and a shifter working together to avoid division operations completely. Moreover, we convert all the floating points in our design into fixed points. All these methods make our NBC accelerator have a lower unit time cost, that is to say, have higher performance.

As a popular image dataset, MNIST is widely used to many CNN hardware accelerators, Table 7 shows their time cost of the inference and classification accuracy. The NBC accelerator outperforms the design [11], which is the most efficient CNN accelerator of the four. The inference time cost of our design is only 31% of it, with classification accuracy sacrifice of 12.4%. If we use *accuracy / time cost (A/T)* to measure the overall performance of a hardware accelerator, our design is the best. It's A/T is about 2.8 times as much as the design [11]. Considering the time and resources cost, the NBC accelerator is more suitable for the AIoT system to some lightweight applications which NBC can handle.

TABLE 7. The time cost and classification accuracy of the CNN accelerators and the NBC accelerator.

Performance	[37] A	[37] B	[37] C	[11]	Ours
Time Cost(us)	928	637	637	25	8
Accuracy(%)	96.33	94.67	88.00	96.8	84.44
A/T	0.10	0.15	0.14	3.87	10.56

The classification performance is affected by the number of counting PE. By changing the number of counting PEs in the counting PE array, we explore the influence in the time cost of the training part and resource consumption. Basing the dataset Voting, different numbers of counting PEs are configured to discuss the changes of training time cost and some key resources consumption, including LUT, FF, BRAM and IO. As Figure 5 shows.

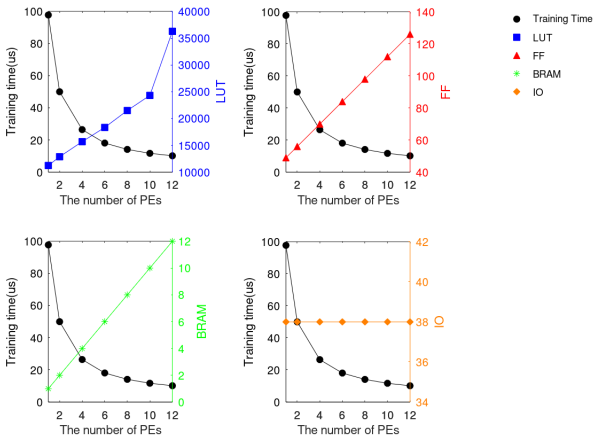

FIGURE 5. The training time cost and resources consumption under different numbers of counting PEs.

When the number of counting PEs are increasing, the time cost of the training part becomes less while the consumption of LUT, FF and BRAM become higher. Increasing the number of counting PEs does not involve the usage of IO, so there is no change of the cost of IO resource. As the fourth figure shows. Moreover, the time cost curve tends to be smoother with the increase of the counting PE array scale. Because the interactions between the CNT PEs and CNT A-LUTs will make more contributions to the training time cost when the number of the counting PEs becomes larger. Thus, in practical applications, the number of counting PEs should be selected considering both efficiency and resources cost.

B. PREDICTION ACCURACY

Under the configuration mentioned above, and each item in the logarithm LUT has 8 bits composed by one-bit integer part and seven-bit fraction part, we get the prediction accuracy of every dataset in Figure 6.

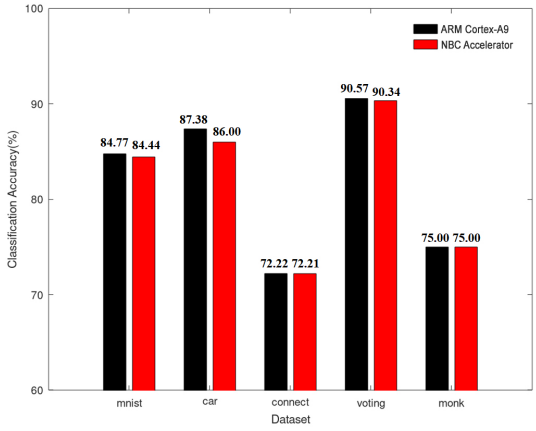

FIGURE 6. The classification of each dataset running on the NBC accelerator and ARM Cortex-A9 processor.

Figure 7 above shows that the NBC accelerator almost has no loss of classification precision, comparing with NBC running on ARM Cortex-A9 processor. The average deviation of classification accuracy between the two kinds of NBC implementation is about 0.39%, which is very little.

The fractional width of the logarithm LUT is a main factor influencing the classification accuracy. We explore the relationship between them using dataset MNIST and Connect by changing the number of the bits of items in the logarithm LUT. In section IV, we convert float-point logarithm values to fixed-point to reduce area and time cost on FPGA. The integer part of each item in the LUT is one-bit. The width of the fractional part depends on the fix-point processing strategy selected. Figure 7 shows the classification accuracy under different width of fractional bits of the logarithm LUT.

As can be seen from Figure 7, the classification accuracy rises at the beginning and then flattens gradually with the increase of the width of the logarithm LUT. When the width is 4-bit or more, the classification accuracy of MNIST is not

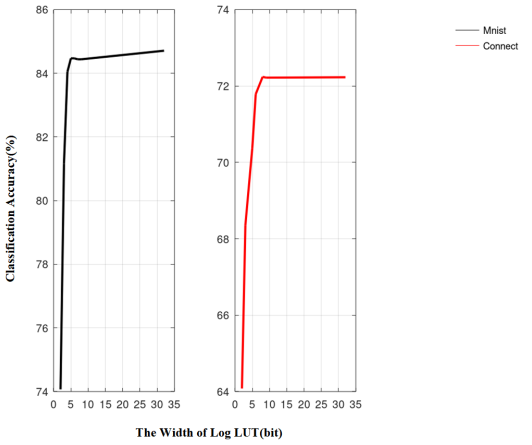

FIGURE 7. The classification accuracy under different width of the logarithm LUT.

improved significantly. For dataset Connect, the classification accuracy remains unchanged if the width of the logarithm LUT is more than 8 bits. By doing the same experiment, it can be got that one or two bits are enough to the other three datasets. Thus, to ensure the classification accuracy and reduce the resources consumption, we set the logarithm LUT's fractional part to 7 bits, and integer part to 1 bit.

VII. CONCLUSION

Motivated by higher requirements put forward by the emergent AIoT, we design our NBC specific hardware accelerator, which can complete training and inference in the AIoT environment. We use logarithm transformation basing on LUT and float-to-fixed point process to simplify the NBC algorithm. Due to removing multiplication and division operations in original NBC algorithm, these technologies help the NBC more effectively to hardware implementation. After the optimization, we design the NBC specific accelerator which is comprised of a controller, a training part and an inference part. The controller coordinates the other parts in the accelerator to make them running orderly. Multiple parallel processing methods are widely used to sped up the training and inference part. Moreover, a logarithm LUT with a novel format and shift operations are working together to compute the logarithm value of a number efficiently. The experiments use five datasets of different magnitudes to prove our design outperforms the general processor, many state-of-the-art hardware Bayes classifiers and CNN accelerators. The proposed accelerator has almost the same classification accuracy as the general processor. What's more, it utilizes very limited hardware resources comparing with the CNN accelerators.

In the future, we will make the NBC accelerator configurable by packing it into a AXI IP. As an edge device, the variables in the NBC accelerator such as the training data, feature vector dimension and so on can be sent to the IP through the AXI bus. Therefore, there is no need to synthesize the core code when the dataset is changed.

REFERENCES

[1] E. Alpaydin, *Introduction to machine learning*. Cambridge, MA, USA: MIT Press, 2004, pp. 33–39.
[2] S. Paul, N. Jayakumar, and S. P. Khatri, "A fast hardware approach for approximate, efficient logarithm and antilogarithm computations," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 2, pp. 269–277, Feb. 2009.
[3] D. Bariamis, D. Maroulis, and D. K. Iakovidis, "Adaptable, fast, area-efficient architecture for logarithm approximation with arbitrary accuracy on FPGA," *J. Signal Process. Syst.*, vol. 58, no. 3, pp. 301–310, May 2009.
[4] O. M. E. Ebadati and F. Ahmadzadeh, "Classification spam email with elimination of unsuitable features with hybrid of GA-naive Bayes," *J. Inf. Knowl. Manage.*, vol. 18, no. 1, Mar. 2019, Art. no. 1950008.
[5] J. M. Haut, M. E. Paoletti, J. Plaza, J. Li, and A. Plaza, "Active learning with convolutional neural networks for hyperspectral image classification using a new Bayesian approach," *IEEE Trans. Geosci. Remote Sens.*, vol. 56, no. 11, pp. 6440–6461, Nov. 2018.
[6] M. Ha and S. Lee, "Accurate hardware-efficient logarithm circuit," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 64, no. 8, pp. 967–971, Aug. 2017.
[7] S. Saurav, "Hardware accelerator for facial expression classification using linear SVM," in *Advances in Signal Processing and Intelligent Recognition Systems*. Cham, Switzerland: Springer, 2016, pp. 39–50.
[8] X. Lian, Z. Liu, Z. Song, J. Dai, W. Zhou, and X. Ji, "High-performance FPGA-based CNN accelerator with Block-Floating-Point arithmetic," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 8, pp. 1874–1885, Aug. 2019.
[9] L. Lu, Y. Liang, R. Huang, W. Lin, X. Cui, and J. Zhang, "Speedy: An accelerator for sparse convolutional neural networks on FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, New York, NY, USA, Feb. 2019, p. 187.
[10] W. You and C. Wu, "A reconfigurable accelerator for sparse convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, New York, NY, USA, Feb. 2019, p. 119.
[11] Y. Zhou and J. Jiang, "An FPGA-based accelerator implementation for deep convolutional neural networks," in *Proc. 4th Int. Conf. Comput. Sci. Netw. Technol. (ICCSNT)*, Harbin, China, Dec. 2015, pp. 829–832.
[12] H. Meng, K. Appiah, A. Hunter, and P. Dickinson, "FPGA implementation of naive Bayes classifier for visual object recognition," in *Proc. CVPR*, Colorado Springs, CO, USA, Jun. 2011, pp. 123–128.
[13] G. I. Webb, J. R. Boughton, and Z. Wang, "Not so naive Bayes: Aggregating one-dependence estimators," *Mach. Learn.*, vol. 58, no. 1, pp. 5–24, Jan. 2005.
[14] S.-W. Choi and C. H. Lee, "A FPGA-based parallel semi-naive Bayes classifier implementation," *IEICE Electron. Express*, vol. 10, no. 19, 2013, Art. no. 20130673.
[15] K. Appiah, A. Hunter, P. Dickinson, and H. Meng, "Binary object recognition system on FPGA with bSOM," in *Proc. 23rd IEEE Int. SOC Conf.*, Las Vegas, NV, USA, Sep. 2010, pp. 254–259.
[16] R. Shrestha and R. P. Paily, "VLSI design and hardware implementation of high-speed energy-efficient logarithmic-MAP decoder," *J. Low Power Electron.*, vol. 11, no. 3, pp. 406–412, Sep. 2015.
[17] A. Klinefelter, J. Ryan, J. Tschanz, and B. H. Calhoun, "Error-energy analysis of hardware logarithmic approximation methods for low power applications," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Lisbon, Portugal, May 2015, pp. 2361–2364.
[18] D. M. Chickering, D. Heckerman, and C. Meek, "Large-sample learning of Bayesian networks is NP-hard," *J. Mach. Learn. Res.*, vol. 5, pp. 1287–1330, Oct. 2004.
[19] T. Kohonen, "An introduction to neural computing," *Neural Netw.*, vol. 1, no. 1, pp. 3–16, Jan. 1988.
[20] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, Jul. 2006.
[21] J. R. Quinlan, "Introduction of decision trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, 1986.
[22] M. S. Ansari, B. F. Cockburn, and J. Han, "A hardware-efficient logarithmic multiplier with improved accuracy," in *Proc. Design Autom. Test Eur. Conf. Exhibit. (DATE)*, Florence, Italy, Mar. 2019, pp. 928–931.
[23] M. Li and K. Liu, "Causality-based attribute weighting via information flow and genetic algorithm for naive Bayes classifier," *IEEE Access*, vol. 7, pp. 150630–150641, 2019.
[24] L. Li, Y. Zhang, W. Chen, S. K. Bose, M. Zukerman, and G. Shen, "Naïve Bayes classifier-assisted least loaded routing for circuit-switched networks," *IEEE Access*, vol. 7, pp. 11854–11867, 2019.

[25] P. Valdiviezo-Diaz, F. Ortega, E. Cobos, and R. Lara-Cabrera, "A collaborative filtering approach based on Naïve Bayes classifier," *IEEE Access*, vol. 7, pp. 108581–108592, 2019.
[26] R. Zhu, Y. Dai, T. Li, Z. Ma, M. Zheng, Y. Tang, J. Yuan, and Y. Huang, "Automatic real-time mining software process activities from SVN logs using a naive Bayes classifier," *IEEE Access*, vol. 7, pp. 146403–146415, 2019.
[27] E. Manino, L. Tran-Thanh, and N. R. Jennings, "On the efficiency of data collection for multiple Naïve Bayes classifiers," *Artif. Intell.*, vol. 275, pp. 356–378, Oct. 2019.
[28] L. C. Chunjie, "AIoT bench: Towards comprehensive benchmarking mobile and embedded device intelligence," in *Proc. Int. Symp. Benchmarking, Measuring Optim.* Cham, Switzerland: Springer, 2018, pp. 31–35.
[29] M. Ben-Ari, "Principles of concurrent and distributed programming," in *The Metropolis*, Singapore: Pearson, 2006, pp. 150–174.
[30] Y. Zhao, J. Liu, and E. A. Lee, "A programming model for time-synchronized distributed real-time systems," in *Proc. 13th IEEE Real Time Embedded Technol. Appl. Symp. (RTAS)*, Bellevue, WA, USA, Apr. 2007, pp. 259–268.
[31] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Salt Lake City, UT, USA, Jun. 2018, pp. 6848–6856.
[32] Y. Liang, L. Lu, Q. Xiao, and S. Yan, "Evaluating fast algorithms for convolutional neural networks on FPGAs," in *Proc. IEEE 25th Annu. Int. Symp. Field-Program. Custom Comput. Mach.*, Apr. 2017, pp. 101–108.
[33] L. Lu and Y. Liang, "SpWA: An efficient sparse winograd convolutional neural networks accelerator on FPGAs," in *Proc. 55th ACM/ESDA/IEEE Des. Autom. Conf. (DAC)*, San Francisco, CA, USA, Jun. 2018, pp. 1–6.
[34] H. Wang, W. Liu, T. Xu, J. Lin, and Z. Wang, "A low-latency sparse-winograd accelerator for convolutional neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Brighton, U.K., May 2019, pp. 1448–1452.
[35] S. Pei, T. Shen, X. Wang, C. Gu, Z. Ning, X. Ye, and N. Xiong, "3DACN: 3D augmented convolutional network for time series data," *Inf. Sci.*, vol. 513, pp. 17–29, Mar. 2020.
[36] X. Hu, Y. Zeng, Z. Li, X. Zheng, S. Cai, and X. Xiong, "A resources-efficient configurable accelerator for deep convolutional neural networks," *IEEE Access*, vol. 7, pp. 72113–72124, 2019.
[37] T.-H. Tsai, Y.-C. Ho, and M.-H. Sheu, "Implementation of FPGA-based accelerator for deep neural networks," in *Proc. IEEE 22nd Int. Symp. Des. Diag. Electron. Circuits Syst. (DDECS)*, Cluj-Napoca, Romania, Apr. 2019, pp. 1–4.
[38] I. Rish, "An empirical study of the naive Bayes classifier," in *Proc. Workshop Empirical Methods Artif. Intell. (IJCAI)*, 2001, pp. 41–46.
[39] Z. M. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "State-of-the-Art deep learning: Evolving machine intelligence toward Tomorrow's intelligent network traffic control systems," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2432–2455, May 2017.
[40] D. B. Sam, S. Surya, and R. V. Babu, "Switching convolutional neural network for crowd counting," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Honolulu, HI, USA, Jul. 2017, pp. 4031–4039.
[41] S. Mukherjee and N. Sharma, "Intrusion detection using naive Bayes classifier with feature reduction," *Procedia Technol.*, vol. 4, pp. 119–128, Jan. 2012.
[42] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Honolulu, HI, USA, Jul. 2017, pp. 1251–1258.
[43] G. Ke, "Lightgbm: A highly efficient gradient boosting decision tree," in *Proc. NIPS*, 2017, pp. 3146–3154.
[44] R. E. Banfield, L. O. Hall, K. W. Bowyer, and W. P. Kegelmeyer, "A comparison of decision tree ensemble creation techniques," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 1, pp. 173–180, Jan. 2007.
[45] J. Chen and X. Liu, "A fast and accurate logarithm accelerator for scientific applications," in *Proc. IEEE 28th Int. Conf. Appl.-Specific Syst., Archit. Processors (ASAP)*, Seattle, WA, USA, Jul. 2017, p. 208.

ZHEN XUE received the B.S. degree from Tianjin University, China, in 2018, where she is currently pursuing the M.S. degree with the College of Intelligence and Computing. Her main research interests include computer architecture and AI accelerator.

JIZENG WEI received the B.S. degree from the Harbin Institute of Technology, in 2004, and the M.S. and Ph.D. degrees in computer science from Tianjin University, Tianjin, China, in 2007 and 2010, respectively. He is currently an Associate Professor with the College of Intelligence and Computing, Tianjin University. His research interests include computer architecture, heterogeneous processor design, AI accelerator, and embedded systems.

WEI GUO received the M.S. degree from Louisiana State University, in 1991. Since 1991, she has been as Senior Engineer, a Senior Staff Engineer, and a Principal Staff Engineer with Motorola Ltd., for 12 years, where she had become an IC Design Expert. She is currently a Professor and the Director of the VLSI Research Laboratory, College of Intelligence and Computing, Tianjin University. Her research interests include SoC design technology, computer architecture, and multimedia processing.