Efficient Implementation of k-Nearest Neighbor Classifier Using Vote Count Circuit

Haiyan Shu, Rongshan Yu, Wenyu Jiang, Wenxian Yang

***Abstract*—k-nearest neighbor (k-NN) classification is a non-parametric method to classify the objects based on the training set. It is an instance-based classifier operating on the assumption that unknown instance is related to the known ones according to some distance/similarity functions. In this paper, a hardware-assisted algorithm, vote count, is introduced to approximate the k-NN classifier to provide a low cost classification solution. It is found that this hardware-assisted solution achieves similar performance as that of the k-NN classifier. In addition, it is highly scalable with respect to the training sample size which is essential for k-NN algorithm to deliver its full potential for real-life classification problems.**

***Index Terms*—k-nearest neighbor classifier, nonparametric classification, vote count, low-power design, Flash memory.**

I. Introduction

Classification is a task to classify a new observation (query vector or instance) into different categories on the basis that a training set of data containing observations (reference vectors or instances) whose category is known. k-NN classifier [1], [2] is one of the oldest and simplest approaches to solve the classification problem. In k-NN, an object is classified by a majority vote of its neighbors. Generally, a small positive integer k is considered when choosing the k nearest neighbors for final decision. k-NN classifier is a very flexible classification scheme as it does not involve any design phase (training) over the training data. Therefore it is particularly appealing for problems with a large training set, or the incremental learning problems with dynamically varying training set. In addition, the k-NN classifier deals with multiclass problem effortlessly. These make k-NN attractive in many applications.

Although k-NN classifier provides a simple solution for supervised classification scheme, it has poor scalability with respect to the size of training set since both its space requirement and the classification time are $O(n)$, which significantly limits the value of k-NN classifier to deliver its full potential for real-life classification problems, such as object and scene recognition where n can be large (e.g. $n = 8 \times 10^7$ training images in [3]).

Recently, the vote count circuit [4], [5] has been introduced in the context of multimedia search. The vote count circuit is a pattern matching circuit that can be implemented on a customized memory circuit. By exploiting the massive parallel processing latent in memory circuits, vote count algorithm has a complexity of $O(1)$, i.e., its query time is a constant regardless of the size of the training set. In addition, its latest version based on a novel design using NAND Flash memory structure, referred to as *enhanced* vote count (EVC) [5], is highly energy efficient. For example, it is found that for content based search at the same search throughput, EVC has the potential to deliver $100 \sim 1000$ times in energy saving on difficult databases (e.g., database with 1-trillion records with high SNR). Although the original purpose of designing vote count circuit is for content-based database search, it is possible to apply it to machine learning applications as well as pattern recognition, classification and regression. In this paper, we investigate the feasibility of implementing k-NN classifier on vote count circuit, and its performance in terms of classification accuracy, and complexity.

The concept of implementing k-NN algorithm on hardware can be found in [6]–[8]. In [6], [7], both NN and k-NN classifiers are implemented using analog electronic circuits on the basis of the probabilistic formulation of these classifiers. A learning algorithm was used to further optimize the training data set to minimize the classification error. In [8], a neural network model is proposed to identify the k largest or the k smallest ones in a data set and the algorithm is realized using an array architecture with low hardware complexity and high computational speed. Compared with those prior-art, the EVC circuit further provides the functions of storing and updating the training data set inherited from its memory based implementation. In addition, it can achieve high density by leveraging the mature Flash memory process technologies, and low power consumption by the low-power *interlocked* design in [5]. The rest of this paper is organized as follows. In Section II, we first review the k-NN algorithm, followed by the introduction of its approximation implementation using vote count algorithm and its supporting circuits. The simulation results as well as measurements on an FPGA-based prototype are presented in Section III, which is followed by a conclusion drawn in Section IV.

II. Implementation

A. k-NN classification

We consider a supervised classification problem with c classes and a training set $\Gamma = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ containing n observations. Each observation $(\mathbf{x}, y)$ consists of an instance $\mathbf{x} \in \chi^d$ and a label $y \in \omega$, where χ^d is the set of d dimensional vectors, and $\omega = \{1, \ldots, c\}$. A classifier $f : \chi \rightarrow \omega$ is a function that maps an input instance $\mathbf{x}' \in \chi^d$ to a class.

k-NN classification is an instance-based classification scheme that locates the nearest neighbor in instance space and

H. Shu, R. Yu, W. Jiang and W. Yang are with the Signal Processing Department, Institute for Infocomm Research, A*STAR, Singapore 138632. (email: hshu@i2r.a-star.edu.sg; ryu@i2r.a-star.edu.sg; wjiang@i2r.a-star.edu.sg; wyang@i2r.a-star.edu.sg)

labels the unknown vector with the same class label as that of the known neighbors. It is widely used to solve the above mentioned supervised classification problem. In a typical k-NN classification, the distances from instances $\mathbf{x} \in \chi^d$ in the training set to the input instant $\mathbf{x}'$ are computed, and the set D_k of k nearest neighbors of $\mathbf{x}'$ is identified. The input instant $\mathbf{x}'$ is then classified as the most frequent class among D_k. This process can be formulated as

$$f(\mathbf{x}') = \arg\max_{l \in \omega} |\{(\mathbf{x}_i, y_i) \in D_k | y_i = l\}|, \quad (1)$$

where $|\cdot|$ is the cardinality of a set.

It is shown by Fix and Hodegs [9] that when infinite number of samples are available, k-NN classifier achieves less than twice the Bayes error. However, in practice we never have infinite number of training samples. It has been shown that for finite sample size, k-NN classifier may lead to estimates with large biases and variance, in particular, when the dimensionality of data is large [10]. For this reason, k-NN classifier and its variances are more suitable to tasks where sample sizes are large. On the other hand, k-NN has poor scalability with respect to the training sample size since both its space requirement and the classification time are $O(n)$, which significantly limits the value of k-NN classifier to deliver its full potential for real-life classification problems.

B. Vote Count Implementation of k-NN

The state-of-the-art fast algorithm for NN search is the Locality Sensitive Hashing (LSH) algorithm [11], [12], which works by projecting a feature vector $\mathbf{x}$ onto a (pseudo-random) vector $\mathbf{a}_j$ as $\mathbf{x} \cdot \mathbf{a}_j$, quantizing it, and concatenating k quantized values from k projections to form a hash key. In this process, similar items are mapped to the same buckets with high probability. This scheme is widely used to find the nearest neighbor vectors of a certain query in a large database of reference vectors with high dimensions. Unfortunately, LSH works well with either large database or high robustness, but not both. A main reason for LSH's scaling difficulty is the *concatenation* of quantized projected values, which quickly reduces recall (probability of finding the right match) due to potential high bit error rate (BER) after random projection for nearest neighbors.

Instead of concatenating p quantized values from p projections and doing it R times, a vote count algorithm is used to count the matching for each reference vector. For each projection, the projection result is quantized and mapped into different bins. The counters of those training vectors that fall in the same bin with the query vector $\mathbf{x'}$ will increase by 1. Mathematically, the final outputs of the counters are given by

$$C_i = |\{\mathbf{a}_j | \mathcal{Q}(\langle \mathbf{x}_i, \mathbf{a}_j \rangle) = \mathcal{Q}(\langle \mathbf{x}', \mathbf{a}_j \rangle), j = 1, \ldots, L\}|, \quad i = 1, ..., n, \quad (2)$$

where C_i is the vote count corresponding to training data vector $\mathbf{x}_i \in \Gamma$ w.r.t. query vector $\mathbf{x}'$, $\mathbf{a}_j$'s are random projection vectors, $\langle \cdot, \cdot \rangle$ is the inner product, $\mathcal{Q} : \mathbb{R} \to \mathbb{Z}$ is the mid-riser quantization function defined as

$$\mathcal{Q}(t) = \lceil \frac{t}{\Delta} \rceil. \quad (3)$$

The flow chart for vote count implementation is given in Fig. 1. First, L random projection vectors $\mathbf{a}_j$'s are generated. For each such random vector, the inner products of all reference vectors ($\mathbf{x}$'s) and the query vector ($\mathbf{x'}$) are calculated, and the results are quantized. For each projection, the counters of those reference vectors that fall in the same quantization bin as that of the query vector are increased by 1. This procedure is repeated for other projections, for a total of L times. Next, the counters of reference vectors in the training set are checked. Reference vectors that have vote count amongst the highest k counts are identified and the k approximate nearest neighbors set D_k is formed. Finally, the most frequent class among these k nearest neighbors is assigned as the class of the query vector as in (1).

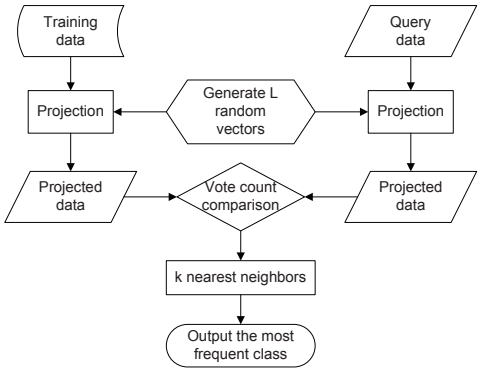

Fig. 1. Flow chart for vote count implementation.

In traditional k-NN, the most computationally demanding operation is the distance calculation when new query vector is introduced. The distances from the query vector to all the training vectors have to be calculated when new query vector is introduced. When the size of training set is large, this process presents high computation and is hard to provide scalability. In vote count based classification, high computation issue for large training set can be easily solved. The projections for each reference vector in the training set is done once. When new query is received, only the projection of the query vector is needed, and no additional projection of those reference vectors is required. This reduces the computational complexity on the distance computing in k-NN classification.

However, the updating of *all* vote counters would be formidably slow when the training set is very large if it is done in software. To ensure high speed in vote count, Singh and Jiang proposed to implement it in modified memory devices like Dynamic Random Access Memory (DRAM) and Flash memory [4]. Furthermore, Jiang and Yu proposed a NAND Flash memory structure to provide a low power consumption solution [5]. These memory devices have a 2D matrix layout, and have the interesting property that once a row is opened for read access, the entire row's data become available. So all m vote counters can be updated simultaneously, leading to very fast search time. Details will be introduced below.

C. Vote Count Circuit

In practically any memory circuit, there is a word-line for each row to access memory cell on that row, and there is a

bit-line for each column to sense one of the memory cells on that column. Vote count circuit may be implemented using several different memory architectures. The first is a parallel architecture based on a DRAM or NOR Flash, where cells in the same column connect to their bit-line in parallel. As shown in Fig. 2, a DRAM memory cell has a 1-transistor 1-capacitor (1Tr1C) structure. The charge and the voltage of the capacitor depend on its stored value (0 or 1), influence the bit-line (BL) voltage, and thus control the read out during a DRAM read access.

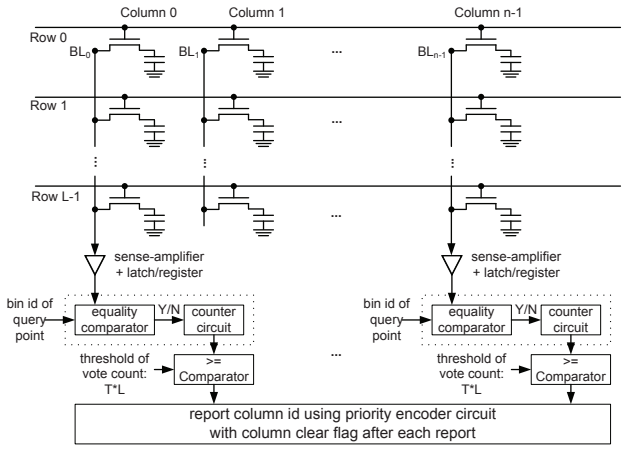

Fig. 2. DRAM based vote count implementation.

In our design, DRAM is modified to implement the vote count algorithm efficiently by leveraging the property of DRAM that when a single cell is being read, the values of all cells in that row on the same *matrix grid* (often referred to as *DRAM macro*) become available simultaneously. In DRAM, each column has its own sense-amplifier to sense and read the bit value stored in the cell belonging to that column. The output of the sense-amplifier is further sent to a latch or register for temporary storage. The sense-amplifier and latch/register are shared across rows to reduce the overhead of such circuitry. As shown in Fig. 2, standard DRAM is modified by adding vote count related circuitry. The resulting circuit has a *matrix* structure of the size $L \times n$ cells where L is the number of the random vectors used by vote count projection and n is the number of instances in the (training) dataset. In practice, an n-column memory matrix can be built from multiple 2D memory arrays each with far narrower width, but accessed simultaneously during vote counting comparison.

Each instance in the training set is assigned to a column and each random vector is assigned to a row. After each of L rounds of projections, the comparison result may affect the counter for each column. This forms the basic hardware architecture implementing the vote count algorithm. It has both a relatively simple structure and fast speed. However, this implementation has a drawback of high power consumption because each column will draw current during read.

The second architecture, a low power version vote count circuit is proposed based on the structure of NAND Flash memory [5]. This circuit, referred to as *enhanced* vote count (EVC), is shown in Fig. 3. In this implementation, instead of comparing 1 quantized projected value at a time, m quantized

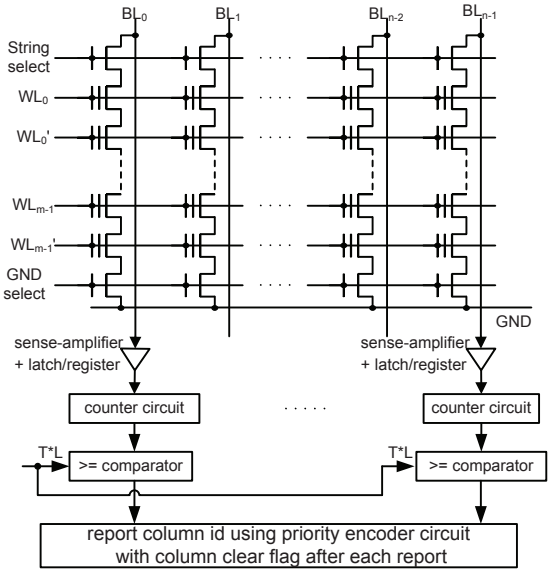

Fig. 3. A NAND Flash based EVC implementation. For brevity, only 1 set of NAND string is shown for each data entry (bit-line) in the database. In actual implementation, L' NAND strings will be connected to each bit-line in a way similar to Fig. 2 to support L' comparison and counter update operations.

projected values are compared at a time. And the EVC architecture is designed such that a column draws current only if all m values match with the query's, using what is referred to as the *interlocked* design, which is detailed in [5]. Power consumption is thus reduced accordingly significantly.

The building block of Flash memory is the floating gate transistor (fGT) as shown in Fig. 4. When a floating gate (FG) stores no net charge, conventionally a "1" or *erased* state, a low V_G [1] (denoted mid) is enough to make the fGT conduct. But when FG contains a net (negative) charge, conventionally a "0" or *programmed* state, a high V_G (denoted hi) is needed. Note that for ease of drawing, the net charge is drawn beside the FG, where in practice it resides on the FG itself.

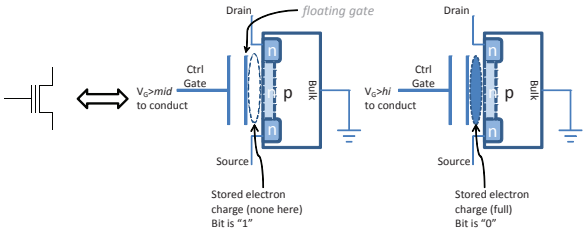

Fig. 4. Workings of floating gate transistor (fGT).

In contrast to the parallel version in DRAM in Fig. 2, NAND Flash is formed by connecting several fGTs in series (together called a NAND string). During reading, all other fGTs are given a *hi* V_G to ensure they conduct, but a *mid* V_G is applied to the fGT of interest (denoted F1). The *hi*'s ensure all other fGTs form conductive channels, and if F1 is erased, a *mid* V_G is enough to form a conductive channel in F1, and current will flow in the series. If F1 is programmed, *mid* will *not* be

[1] V_G denotes Control Gate (CG) voltage. But in Integrated Circuit design, V_G can also be used in lieu of CG to Source voltage to estimate on/off state of a transistor, that resides inside a series of transistors (i.e. a NAND string).

enough to form a conductive channel in F1, and the series will *not* conduct. Thus the presence of current (or lackof) decodes charge state of F1 and its respective bit value. With the matrix layout, the *mid* and *hi* are applied to all fGTs on the same row, respectively. The line controlling one row is called a word-line.

Compared with DRAM implementation in Fig. 2, since m projections are compared simultaneously for each reference instance, the number of comparison and counter update operations is reduced significantly. In addition, the power consumption per comparison is also significantly reduced since only the matched NAND string will draw current and hence consume power. Therefore, the power consumption of EVC will be significantly lower than DRAM based VC implementation.

The final counter output of EVC circuit will be different to VC due to its different comparison and counter updating process. Specifically, denote the random projection vectors in EVC as $\{\mathbf{a}_{j,v}|v = 1, \ldots, m, j = 1, \ldots, L'\}$ where $\mathbf{a}_{j,v}, v = 1, \ldots, m$ are the m concurrent random projections to be compared at j-th comparison and counter updating operation. Since the counter will be increased by 1 only when the all m quantized projected values from the NAND string match, the output of the counter after L' comparison is now given by:

$$\begin{aligned} C_i &= |\{\mathbf{a}_{j,1}|\mathcal{Q}(\langle \mathbf{x}_i, \mathbf{a}_{j,v}\rangle) = \mathcal{Q}(\langle \mathbf{x}', \mathbf{a}_{j,v}\rangle), \\ &\quad v = 1, \ldots, m, j = 1, \ldots, L'\}|, \\ &\quad i = 1, \ldots, n \end{aligned} \tag{4}$$

where C_i is the counter output of i-th bit-line corresponding to training data vector $\mathbf{x}_i \in \Gamma$. Note that the total number of random projections is $L = m \times L'$ in EVC although the total number of comparison and counter updating operations is L'.

In traditional k-NN implementation, the time complexity is around $n \cdot d \cdot \tau$, where n is the size of training data, d is the dimension of the feature vectors, and τ is the amortized time for processing 1 out of d dimensions, with typically $\tau \approx$ 1ns. In the DRAM based vote count (VC) implementation, the time complexity is $L \cdot \tau_{par}$, where τ_{par} is the DRAM read access time, with typically $\tau_{par} \approx 25 - 50$ns. In the NAND Flash based EVC implementation, the time complexity is $L' \cdot \tau_{ser}$, where $L' = \frac{L}{m}$ with m as the number of simultaneously compared projections, and τ_{ser} is the NAND Flash read access time with typically $\tau_{ser} \approx 5 - 50\mu$s. Although for ease of drawing, Fig. 3 only shows a NAND string per column, in practice there would be L' NAND strings per column, each connecting to the bit-line in parallel while each NAND string is itself a serial circuit.

Note that to achieve the same accuracy, the L in EVC implementation will generally be larger than the L in VC implementation, implying higher memory transistor usage but with the benefit of significantly lower power and energy consumption. Since L is usually no more than a few hundred, it is obvious that, when $n \cdot d$ is around 10000 to 100000, k-NN scheme has similar time complexity to that of the VC and EVC implementation. When $n \cdot d$ increases further, k-NN would present increasingly higher time complexity than that of vote count, thus making vote count circuit a more favor solution. For example, in [3], $n = 8 \times 10^7$ training images ($d = 19$ largest PCA dimensions out of 3072) are used to provide object and scene recognition, implying at least 1.5sec (or 30sec per [3]) to recognize one query image. Although multiple processors can be used to reduce k-NN time complexity, it would proportionally add to system cost. In contrast, even with a conservative $L' = 50$ and $\tau_{ser} = 50\mu$s, EVC would only take $\sim$2.5 ms. Furthermore, the high data density of memory chips, especially NAND Flash, enables scalability, and is the additional advantage of vote count implementation.

III. Experimental Results

In this section, the actual power and speed of vote count implementation is examined, followed by its accuracy compared with k-NN classifier.

A. FPGA-based Prototype

We have implemented an EVC prototype using a matrix (32×32) of discrete nMOSFETs with two different threshold voltages to emulate fGTs[2] and an FPGA[3] as the controller. Note that in all off-the-shelf Flash memories, including those in FPGAs, their accessing circuits (e.g. row address decoder for generating word-line voltages) are hard-wired and cannot be modified using FPGA to implement VC or EVC algorithm.

TABLE I
Speed and power of FPGA-based EVC implementation

	m	L	I_{match}	V_{dd}	τ_{ser}
EVC-FPGA	4	16	0.7-0.8 mA	3.3V	200μs

Table I shows its speed and power, where I_{match} is the current per matching column[4]. Power consumption of the matrix (P_{matrix}) is $N_{match} \times I_{match} \times V_{dd}$, where N_{match} is expected number of matched columns per m-projection comparison. That is, only the matched columns draw non-negligible current. Assuming uniform data distribution, $N_{match} \approx n \times 2^{-m}$.

Search time is $L' \cdot \tau_{ser}$, hence energy consumption of the matrix per search (E_{search}) is $P_{matrix} \cdot L' \cdot \tau_{ser}$. Although a conservative $\tau_{ser} = 200\mu$s is used, additional testing on circuit timing and stability suggests that $\tau_{ser} = 20\mu$s is feasible, which would imply proportionally faster operating frequency and lower E_{search}. Power consumption of the controller (e.g. vote counters, priority encoder) is estimated to be small and cannot be feasibly separated from the rest of the FPGA board, thus no corresponding measurements are available.

B. Algorithmic Simulation

Because the FPGA-based prototype is ROM-like and cannot change its database or configuration parameters m and L easily, the evaluation of vote count algorithm performance is done by simulations on software, and binary quantization is considered. The datasets are chosen from UCI Machine Learning Repository [13].

The first testset is "seeds". In this dataset, there are totally 210 objects in 3 classes. The feature dimension is 7. The simulation result is shown in Table II.

[2]This is because there are no commercially available discrete fGTs.

[3]Specifically, Xilinx Virtex-4 LX25LC is used.

[4]I_{match} is relatively large due to use of discrete MOSFETs, and is expected to be much smaller in IC-based implementation currently under development.

TABLE II
PERFORMANCE COMPARISON OF k-NN AND VOTE COUNT CLASSIFICATION FOR "SEEDS".

	m	L	1-NN	3-NN	5-NN	7-NN	9-NN
k-NN			0.910	0.919	0.919	0.905	0.914
VC		50	0.884	0.893	0.896	0.904	0.896
VC		100	0.9	0.9	0.905	0.9	0.895
VC		1000	0.896	0.891	0.905	0.907	0.910
EVC	4	40	0.864	0.884	0.896	0.897	0.895
EVC	4	200	0.896	0.892	0.903	0.909	0.909
EVC	8	80	0.882	0.894	0.9	0.896	0.897
EVC	8	400	0.894	0.892	0.900	0.905	0.905

TABLE III
PERFORMANCE COMPARISON OF k-NN AND VOTE COUNT CLASSIFICATION FOR "VERTEBRAL".

	m	L	1-NN	3-NN	5-NN	7-NN	9-NN
k-NN			0.806	0.777	0.797	0.781	0.781
VC		50	0.794	0.793	0.801	0.803	0.802
VC		100	0.813	0.777	0.806	0.8	0.803
VC		1000	0.802	0.806	0.796	0.804	0.803
EVC	4	40	0.775	0.783	0.785	0.796	0.791
EVC	4	200	0.795	0.801	0.800	0.804	0.794
EVC	8	80	0.788	0.786	0.793	0.787	0.788
EVC	8	400	0.803	0.803	0.806	0.799	0.801

In the simulation, 10-fold cross validation are used, and the value of k ranges from 1 to 9. Based on this setting, k-NN classifier can reach around 90% accuracy under different k value selections. For vote count implementation, three projection numbers (L=50, 100, 1000) are considered. As shown in Table II, when L=50, the classification result is not as good as that of the k-NN for some k's. When the value of L increases, the classification accuracy of the vote count algorithm improves, and reaches that of k-NN when L=100 and $L = 1000$. This shows that vote count implementation can be used to efficiently approximate the k-NN classifier when the projection number is sufficiently large. In addition, it shows that further increasing L will not improve the performance of vote count significantly when the vote count circuit already delivers similar performance to that of k-NN.

For EVC implementation, m=4 and 8 are selected, i.e., a total of 4 or 8 projections are compared simultaneously in each round. Thus, $\frac{L}{m}$ rounds of comparison are required to implement L projections. The performances of EVC are also shown in Table II. It can be seen from those results that more projections are required in order for EVC to achieve the similar performance compared to vote count. For example, when $m = 8$, $L = 400$ projections are required for EVC to provide similar performance as that of k-NN. Note that in this case the number of comparing round is actually $400/8 = 50$.

Another test set is "vertebral", which has 310 objects in 2 classes, and the dimension of feature vectors is 6. The simulation results are shown in Table III. Results similar to those of Table II can be observed. For this test set, the k-NN classifier achieves a classification accuracy less than 80%. With vote count implementation, the classification accuracy can reach 80% when L=50. Beyond that, the increasing of projection number L does not improve the classification accuracy significantly. For EVC, 200 and 400 projections are required respectively for $m = 4$ and $m = 8$ in order to achieve similar performance.

Based on the simulation results shown above, we can see that the vote count implementation can achieve similar performance as that of k-NN classifier with sufficiently large number of random projections. When the size of training set and the dimension of feature vectors are large, the benefit of the hardware-assisted vote counting on time efficiency can be significant when compared with k-NN classifier.

IV. CONCLUSION

In this paper, hardware-assisted vote count algorithm is introduced to implement k-NN classifier. With the help of the parallel accessing mechanism of memory circuit, the vote count algorithm can be implemented very efficiently. More significantly, for problems with large sizes of training sets, the proposed solution can perform classification tasks essentially at complexity $O(1)$ regardless to the sizes of the training set. In addition, by using NAND Flash memory, the power consumption of the circuit can be reduced significantly. Simulation results verify that the vote count implementation can achieve similar performance as that of k-NN classifier. For these reasons, the proposed design is a very attractive solution for implementing k-NN classifier where computational complexity of traditional k-NN algorithm is a concern.

REFERENCES

[1] E. Fix and J. Hodges, Jr., "Discriminatory Analysis: Nonparametric Discrimination: Consistency Properties," *Report No.4, USAF School of Aviation Medicine*, Feb. 1951.

[2] E. Fix and J. Hodges, Jr., "Discriminatory Analysis: Nonparametric Discrimination: Small Sample Performance," *Report No.11, USAF School of Aviation Medicine*, Feb. 1951.

[3] A. Torralba, R. Fergus, and W. Freeman, "80 million tiny images: a large dataset for non-parametric object and scene recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 11), PAGES = 1958-1970, YEAR = 2008,.

[4] V. Singh and W. Jiang, "An Algorithm and Hardware Design For Very Fast Similarity Search in High Dimensional Space," *Symposium on Foundations and Practice of Data Mining, part of Conference on IEEE Granular Computing*, Aug 2010.

[5] W. Jiang and R. Yu, "High-Performance, Very Low Power Content-based Search Engine," *ICME 2013 International Workshop on GREEN multimedia: Energy-efficient Multimedia Computing, Communication and Presentation*, Jul. 2013.

[6] K. Urahama and N. Takeshi, "Analog circuit implementation and learning algorithm for nearest neighbor classifiers," *Pattern recognition letters*, vol. 15, no. 7, pp. 723–730, 1994.

[7] K. Urahama and Y. Furukawa, "Gradient descent learning of nearest neighbor classifiers with outlier rejection," *Pattern Recognition*, vol. 28, no. 5, pp. 761 – 768, 1995.

[8] J.-C. Yen, J.-I. Guo, and H.-C. Chen, "A new k-winners-take-all neural network and its array architecture," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 901 – 912, 1998.

[9] T. M. Cover and P. E. Hart, "Nearest Neighbor Pattern Classification," *IEEE Trans. Information Theory*, vol. 13, pp. 21–27, Jan. 1967.

[10] K. Fukunaga and D. M. Hummels, "Bias of Nearest Neighbor Error Estimates," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 9, pp. 103–112, Jan 1987.

[11] J. Haitsma and T. Kalker, *A Highly Robust Audio Fingerprinting System*. ISMIR, 2002.

[12] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-Sensitive Hashing Scheme Based on p-stable Distributions,"*Proceedings of the Twentieth Annual Symposium on Computational Geometry*, pp. 253–262, 2004.

[13] K. Bache and M. Lichman, "UCI machine learning repository." http://archive.ics.uci.edu/ml, University of California, Irvine, School of Information and Computer Sciences, 2013.