This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2954472, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

QuantHD: A Quantization Framework for Hyperdimensional Computing

Mohsen Imani, Samuel Bosch, Sohum Datta, Sharadhi Ramakrishna, Sahand Salamat, Jan M. Rabaey, *Fellow, IEEE,* and Tajana Rosing, *Fellow, IEEE*

***Abstract*—Brain-inspired Hyperdimensional (HD) computing models cognition by exploiting properties of high dimensional statistics– high-dimensional vectors, instead of working with numeric values used in contemporary processors. A fundamental weakness of existing HD computing algorithms is that they require to use floating point models in order to provide acceptable accuracy on realistic classification problems. However, working with floating point values significantly increases the HD computation cost. To address this issue, we proposed QuantHD, a novel framework for quantization of HD computing model during training. QuantHD enables HD computing to work with a low-cost quantized model (binary or ternary model) while providing a similar accuracy as the floating point model. We accordingly propose an FPGA implementation which accelerates HD computing in both training and inference phases. We evaluate QuantHD accuracy and efficiency on various real-world applications, and observe that QuantHD can achieve on average 17.2% accuracy improvement as compared to the existing binarized HD computing algorithms which provide a similar computation cost. In terms of efficiency, QuantHD FPGA implementation can achieve on average 42.3× and 4.7× (34.1× and 4.1×) energy efficiency improvement and speedup during inference (training) as compared to the state-of-the-art HD computing algorithms.**

***Index Terms*—Brain-inspired computing, Hyperdimensional computing, Energy-efficiency, FPGA Acceleration**

I. Introduction

With the emergence of the Internet of Things (IoT), technological advances are continually creating more data than what we can handle [1]. Today, many IoT applications analyze data by running machine learning algorithms in data centers. However, it is well-known that existing learning algorithms may be overcomplex for many real-world applications [2]. Simpler algorithms can also deliver the same task with lower computational complexity and hardware/energy requirements. For instance, Deep Neural Networks (DNNs) are used for complicated classification problems such as image classification tasks, e.g., ImageNet dataset [3]. However, the computational complexity and memory requirements of DNNs makes them inefficient for a broad variety of real-life embedded applications. Therefore, it is crucial to design a light-weight learning method since in IoT systems sending all the data to the cloud for processing is not scalable, cannot guarantee real-time response [4].

Brain-inspired Hyperdimensional (HD) computing [5], [6] has been proposed as a light-weight learning methodology. HD computing is developed based on the fact that brains compute with *patterns of neural activity* which are not readily associated with numbers [5]. HD computing builds upon a simple set of operations with random HD vectors, is robust in the presence of hardware failures. It also offers an alternative computational paradigm that can be applied to learning problems [5], [6]. The first step in HD computing is to encode all data points to a high-dimensional space. During training, HD computing linearly combines the encoded hypervectors in order to create a hypervector representing each class. During inference, the classification task is performed by checking the similarity of an encoded *query hypervector* with all class hypervectors and returning the class with the highest similarity score. In this work, we argue that in most practical applications, HD computing algorithms require to be trained and tested using floating point values. HD computing with binary model provides significant low classification accuracy which often is not acceptable by users. On the other hand, working with floating point values increases the HD computation cost and hinders use of HD as a light-weight classifier.

In this work, we observe that the low classification accuracy of HD computing using binarized/quantized model is due to the weakness of the existing training procedure [7]. In this paper, we proposed QuantHD, a novel quantization framework which enables HD computing to be trained and tested on a low-cost binary/ternary model with high classification accuracy. The main contributions of the paper are listed below:

- To the best of our knowledge, this is the first HD computing framework which enables model quantization with minimal impact on the classification accuracy. We develop an iterative training approach which adapts the HD model to work with the quantized values.
- QuantHD significantly accelerates HD computing during training and inference by removing the majority of non-binary computations from the similarity check. Unlike the existing HD computing algorithms that require the use of floating point model, QuantHD performs similarity check with the quantized (binary/ternary) model. The quantized model simplifies the costly cosine similarity to more hardware-friendly metrics such as Hamming distance.
- We accordingly proposed a pipelined FPGA implementation which accelerates both training and inference by

M. Imani, S. Ramakrishna, S. Salamat, and T. Rosing are with the department of computer science and engineering, University of California San Diego, La Jolla, CA 92093, USA.
E-mail: {moimani, sbelachi, tajana}@ucsd.edu

S. Bosch is with the Physics Department, cole polytechnique fdrale de Lausanne (EPFL), Lausanne 1015, Switzerland.
E-mail: Samuel.Bosch@epfl.ch

S. Datta, and J. Rabaey are with Department Electrical Engineering and Computer Sciences at the University of California, Berkeley, CA 94720, USA.
E-mail: {sohumdatta, jan-rabaey}@eecs.berkeley.edu

reducing the cost of the associative search. We evaluate QuantHD on several practical classification problems, including face and activity recognition. Our evaluations show that QuantHD can improve the HD classification accuracy by 17.2% as compared to the existing HD computing algorithms [8], [9]. In terms of efficiency, QuantHD FPGA implementation can achieve on average 34.1× and 4.1× (42.3× and 4.7×) energy efficiency improvement and speedup during training (inference) as compared to the state-of-the-art HD computing algorithms [9], [8]. Comparing QuantHD with multi-layer perceptron and binarized neural network classifiers, we observe that QuantHD can provide 8.2× and 13.4× faster computing in training and testing respectively, while providing similar classification accuracy.

II. Hyperdimensional Computing

Hyperdimensional computing can be applied to different learning problems. Here we focus on classification, one of the most popular supervised learning algorithms. Figure 1a shows an overview of HD computing for classification. The first step of HD classification is to use the encoding module to map data points to a high-dimensional space. The training module combines the encoded hypervectors in order to create a model representing each class. The information in each class stored as a pattern of values distributed in $D = 10,000$ dimensions. The class hypervectors represents a trained model and are placed in the associative memory. During inference, the same encoding module maps the input data to high-dimensional space. The reasoning task finds a class hypervector which has the most similarity to a query hypervector. In the following, we explain the details of encoding, training, and similarity check used for inference.

A. Encoding

The first step of HD is to encode each data point into a hypervector. Original data point is assumed to have n features i.e., $\mathbf{f} = \langle f_1, \dots f_n \rangle$. Our goal is to encode each feature into a hypervector that has D dimensions, e.g., $D = 10,000$. Each feature vector in original domain stores the feature values and their corresponding positions.

Encoding alphabets: To differentiate the position of each feature, we exploit a set of randomly generated base hypervectors, i.e., $\{\mathbf{B}_i, \mathbf{B}_2, \dots, \mathbf{B}_n\}$, where n is the feature size of an original data point ($\mathbf{B}_i \in \{0,1\}^D$). Due to random generation and the size of hypervector, the base hypervectors are nearly orthogonal [10], meaning that the vectors only have about 50% common elements:

$$\delta(\mathbf{B}_i,\ \mathbf{B}_j) \simeq D/2 \qquad (0 < i, j \leq n,\quad i \neq j).$$

where δ is the Hamming distance similarity between the two hypervectors. The feature hypervectors are more likely to be orthogonal when dimensionality, D, is large enough as compared to the size of the feature vector in the original domain ($D >> n$).

We also differentiate the feature values using a set of hypervectors. We first find the minimum and maximum feature values across all training data points, say $\{f_{min}, f_{max}\}$, and then discretize the values to m different levels. Note that this discretization can happen linearly or non-linearly depending on the feature distributions. We select the discretization by looking at the distribution of the feature values. We generate a single hypervector represent each level, $\{\mathbf{L}_1, \mathbf{L}_2, \dots, \mathbf{L}_n\}$. For the first level, $\mathbf{L}_1$, we generate a single hypervector representing f_{min}. Then each time, we select D/m random bits and flip them to generate the next level hypervector. This process ensures that the last level hypervector, $\mathbf{L}_m$, is nearly orthogonal to $\mathbf{L}_1$, while other level hypervectors are correlated. For example, hypervectors assign to neighbor levels have a high correlation as they have at most D/m bits difference.

Aggregation The encoding of each data point occurs by binding (XORing) each base hypervector with the corresponding level hypervector. For each feature, the level hypervector is selected as the nearest quantized level close to absolute feature value. Finally, we add all the results for all the features:

$$\mathbf{H} = \mathbf{B}_1 \oplus \overline{\mathbf{L}}_1 \ + \ \mathbf{B}_2 \oplus \overline{\mathbf{L}}_2 + \dots \ + \ \mathbf{B}_n \oplus \overline{\mathbf{L}}_n \qquad (1)$$

where $\overline{\mathbf{L}}_i \in \{\mathbf{L}_1, \dots, \mathbf{L}_m\}$, and $\oplus$ denotes an XOR operation.

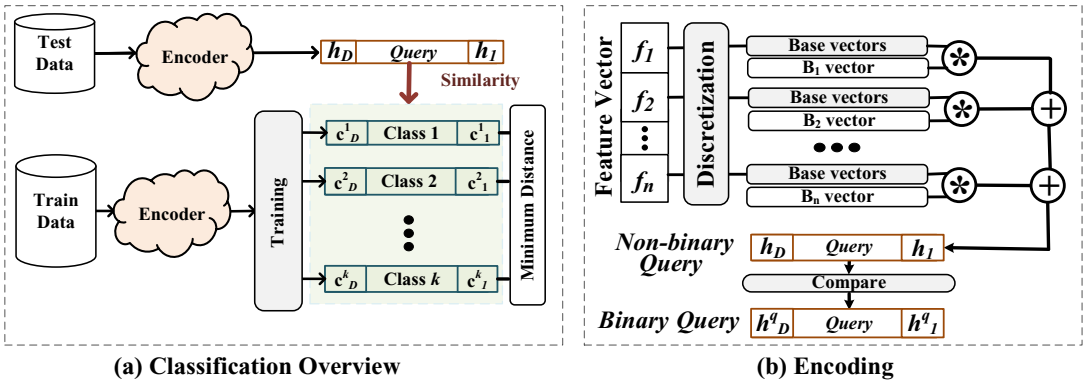
B. HD Training

The HD training happens with the encoded hypervectors. HD computing is its fast learning capability. Most existing HD computing algorithms perform the training in a single iteration. The training adds all the encoded data points that belong to the same class. For example of face recognition problem, HD computing creates two hypervectors representing "Face" and "No-face" classes. These hypervectors can be created by separately adding all encoded hypervectors which have the "Face" and "No-face" labels.

C. Inference: Non-binary or Binary Model?

In HD computing, the trained model has non-binarized elements with positive or negative floating point values. The existing HD computing methods [15], [16], [8], [17], [18] binarize the class hypervectors to eliminate costly *Cosine* operation used for the associative search. In addition, most of the existing hardware accelerators for HD computing only accelerate the binary model [16], [19]. We observe that HD computing using binary hypervectors cannot provide acceptable classification accuracy on the majority of practical problems. Table I reports the HD classification accuracy for four classification applications includingspeech recognition [11], activity recognition [12], physical monitoring [13], and face detection [14]. The results are reported for the based HD computing [9] using binary and non-binary class elements. Note that the baseline HD is using Ngram-based encoding which is different from one explained in Section II-A. In this table, N and n are the numbers of classes and features respectively. To get an acceptable classification accuracy, HD computing requires to use class hypervectors with non-binary elements. For example, for face recognition, HD computing with binary model gets 68.4% accuracy, which is much lower than 95.9% accuracy that HD using a non-binary model can achieve.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2954472, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems



(a) Classification Overview (b) Encoding

Fig. 1. Overview of HD computing for classification.

TABLE I
CLASSIFICATION ACCURACY AND EFFICIENCY OF HD RUNNING ON NON-QUANTIZED AND BINARY MODEL.

Applications	Features (n)	Classes (k)	Floating-point Model		Binary Model	
			Accuracy	*Execution (ms)*	*Accuracy*	*Execution (ms)*
Speech Recognition [11]	617	26	91.1%	11.8	88.1%	1.6
Activity Recognition [12]	561	6	93.8%	3.6	77.4%	0.6
Physical Monitoring [13]	52	12	88.9%	8.0	85.7%	1.1
Face Detection [14]	608	2	95.9%	4.7	68.4%	0.7

The results in Table I also compare the execution time of HD with binary and non-binarized models running on embedded devices (Raspberry Pi 3) using ARM Cortex A53 CPU. Our results show that HD with floating-point model provides on average 17.5% higher accuracy, but is 6.5× slower as compared to HD computing with the binary model. The lower efficiency of the non-binary model comes from the costly *Cosine* similarity metric which involves a large number of additions and multiplications.

III. PROPOSED QUANTHD

A. Overview

In this section, we present QuantHD, a novel framework for quantization of HD computing model during training. QuantHD enables quantizing (binarizing/ternarizing) of the HD model with no or minor impact on the classification accuracy. QuantHD consists of three main steps; (i) **Initial training:** it creates an initial HD model by accumulating all encoded hypervectors corresponding to each class. The initial training step is the same as conventional HD computing algorithms. (ii) **Quantization**, which projects the HD model to a binary or ternary model. Since the HD model has been trained to work with the floating-point values, the quantization results in a significant quality loss. (iii) **Retraining** compensates the quality loss due to the model quantization. QuantHD iteratively retrains the HD model such that it adopts to work with the quantized model.

B. QuantHD Framework

Initial Training: QuantHD trains the class hypervectors by accumulating all encoded hypervectors which belong to the same class. As Figure 2a shows, each accumulated hypervector represents a class. For example, for an application with k classes, the initial HD model contains k non-quantized hypervectors $\{\mathbf{C}_1, \ldots, \mathbf{C}_k\}$, where $\mathbf{C}_i \in \mathbb{N}^D$ (❶).

Model Projection: We develop a model projection method which maps this model to a quantized hypervectors, $\{\mathbf{C}_1^q, \ldots, \mathbf{C}_k^q\}$, with binary or ternary representation (❷). The binary and ternary models represent the class hypervectors using $\{0, 1\}$ and $\{-1,\ 0,\ +1\}$ elements respectively. The details of model quantization are explained in Section III-C.

Iterative Learning: Although the initial trained model provides high classification accuracy, the quantization of the model significantly degrades the accuracy. This accuracy degradation comes from mapping the binary domain which does not preserve distances between the vectors. To compensate for the possible quality loss, QuantHD supports a retraining procedure which iteratively modifies the HD model in order to adapt it to work with the quantization constraints. QuantHD keeps both quantized and non-quantized models. For each data point in the training dataset, say $\mathbf{H}$, we first quantize the encoded hypervector, $\mathbf{H}^q$, and then check its similarity with the quantized model. The similarity metric is the Hamming distance for the binary model and dot product for the ternary model (❸). If the quantized model correctly classifies $\mathbf{H}^q$, we do not update the model. However, if $\mathbf{H}^q$ is incorrectly classified, we only update the *non-quantized model*, while the quantized model stays the same. This update happens on two class hypervectors; a class that data is misclassified to ($\mathbf{C}_{miss}$), and a class that data point belongs to ($\mathbf{C}_{match}$). Since in HD the information stored as a pattern of distribution in high-dimensional space, the update of the non-quantized model can perform by(❹):

$$\mathbf{C}_{miss} = \mathbf{C}_{miss} - \alpha\mathbf{H} \quad \textit{and} \quad \mathbf{C}_{match} = \mathbf{C}_{match} + \alpha\mathbf{H}$$

where α is a learning rate ($0 < \alpha < 1$). Note that although

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2954472, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

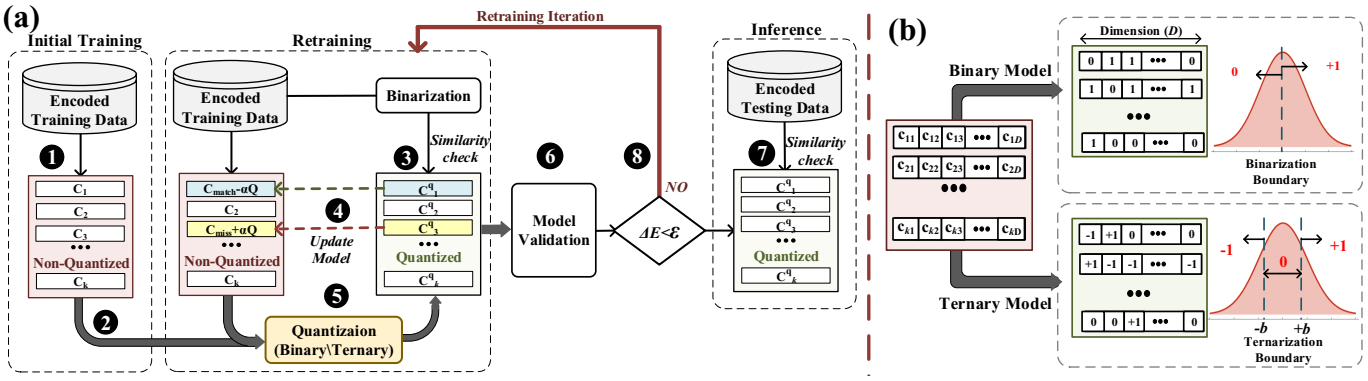


Fig. 2. (a) QuantHD framework overview. (b) Binarizing and ternarizing the trained HD model.

the similarity check performs on the quantized model, we only update the non-quantized model, while the quantized model stays the same. Similarly, after once epoch or iteration across all training examples, the new non-quantized models are written back to the quantized model (❺).

Model Validation: We examine the classification accuracy of the projected model on the validation data, which is 5% of the training data(❻). If the projected model accuracy is changed less than ε, we send the new model to inference to perform the reset of computation; otherwise, we start retraining the quantized model by checking the similarity of all training data points and accordingly updating the non-quantized model (❼). Note that the QuantHD stops after a pre-defined number of iterations if the convergence condition does not satisfy. For all experiments in this paper, we use $\varepsilon = 0.01$ and limit the maximum number of iterations to 30.

Figure 3a,b show the classification accuracy of QuantHD with a binary model during different retraining iterations. The results are reported for speech recognition (ISOLET) and face recognition (FACE) applications using three different learning rates. Note that, here we use the encoding module introduced in Section II-A which provides higher accuracy than the baseline HD computing reported in Table I. Our evaluation shows that QuantHD using small learning rate slow down the learning process. For example, QuantHD with $\alpha = 0.01$ cannot get the maximum accuracy in 30 retraining iterations. Increasing the learning rate to $\alpha = 0.05$ improves the learning speed and the final classification accuracy. However, using a learning rate of $alpha = 0.3$ or larger increases the fluctuation on the accuracy during the retraining phase and can cause possible divergence. Figure 3c,d shows the impact of learning rate on the classification accuracy of QuantHD using binary and ternary models. Our result shows that QuantHD provides maximum classification accuracy using a learning rate of around $\alpha = 0.05$. Note that retraining the original QuantHD model with non-quantized values requires $\alpha > 1$ ($\approx 1.5 - 2$) for fast and stable training.

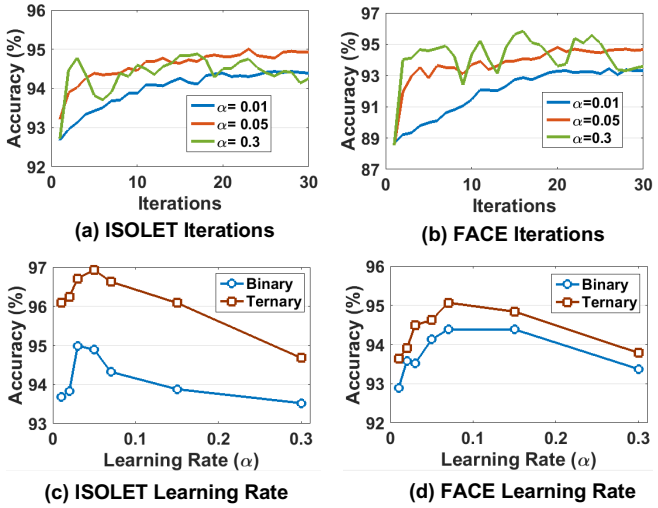


Fig. 3. QuantHD classification accuracy during (a,b) different retraining iterations, and (c,d) using different learning rates.

C. Details of Model Quantization

After training the HD model, the class hypervectors is represented using non-quantized (integer or floating point) elements which can take negative or positive values. The goal of model quantization is to map these values to a new domain where the computation can be performed with much higher efficiency. Here we explain how we quantize the hypervectors to binary and ternary domains.

Binary Model: QuantHD binarizes the model using the sign function by assigning all positive and negative elements to 1 and 0 bits respectively. In fact, binarization maps each class hypervector from $\mathbb{N}^D$ to $\{0,1\}^D$. Unlike the fixed-point/floating-point model which uses costly cosine metric, binary model exploits Hamming distance for similarity check.

Ternary Model: The HD model can also be mapped into a ternary domain where each class element can take $\{-1,\ 0,\ +1\}$ values. Ternarization gives more flexibility to a model to select more suitable weights during quantization (shown in Figure 2b). In the ternary model, the similarity check performs using the dot product between the hypervectors.

Ternarization determines the sparsity of the model by selecting a boundary where the elements can get -1, 0, and +1 values. One naive way is to normalize the class hypervectors and linearly split a range between minimum and maximum feature into three equal regions. Each class elements can be assigned to one of the regions depending on the dimension

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2954472, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

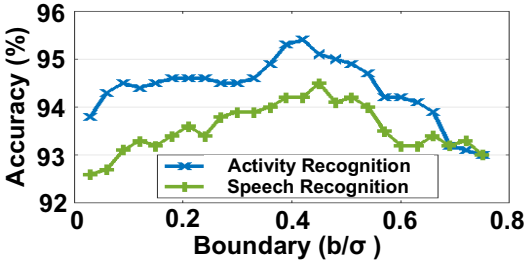


Fig. 4. Impact of the ternarization boundary on the classification accuracy.

values. For example, we can assign numbers in rage of $[-b,+b]$ to zero, while $[-1,-b]$ and $[+b,1]$ are assigned to -1 and +1 respectively (see Figure 2b). This method does not properly work since the class values are not uniformly distributed. Instead, we observe that in all of our tested datasets the class hypervectors have a Gaussian distribution. We can select the ternarization boundary to balance the portion of dimensions that are assigned to 0 value.

The boundary values can set based on the data distribution and its variance (σ). Figure 4 also shows the impact of the ternarization boundary on the classification accuracy of two applications: activity recognition (UCIHAR) and speech recognition (ISOLET). The x-axis in a graph shows the ternary boundary which changes from 0 to 0.8σ. Increasing the b boundary at first improves the classification accuracy by giving more flexibility to the weights to be zero. This also results in higher sparsity of the ternarized model. However, increasing the b boundary more than a specific value results in lower classification accuracy, since high sparsity in the trained model may result in information loss.

IV. FPGA Implementation

HD computing can be implemented in different hardware platforms such as CPU, GPU, or FPGA. Due to a large amount of bit-wise operations exist in training and inference of HD computation, FPGA is a suitable candidate for efficient HD computing acceleration. QuantHD has three main phases: training, retraining, and inference. These phases are sharing similar blocks. For example, the encoding is a commonly used block in all phases. Similarly, the retraining and inference are using associative search block on their computation.

A. Encoding Acceleration

The encoding happens based on two sets of pre-generated hypervectors: level hypervector ($\{\mathbf{L}_1,\ldots,\mathbf{L}_m\}$) and base hypervectors ($\{\mathbf{B}_1,\ldots,\mathbf{B}_n\}$) [7], [9], [8]. Both base and level hypervectors are binary, $\mathbf{B}_i,\mathbf{L}_i \in \{0,1\}^D$. This binary representation enables FPGA to store each dimension of level and base hypervector using a single bit. Figure 5 shows an overview of the FPGA implementation of the encoding module. Encoding module first reads the feature values, assigns them to pre-generated level hypervectors (Ⓐ). Each feature value is compared with m quantized feature values and then assigned to a level with the closest distance (Ⓑ). For each feature, the encoding module performs the XOR operation between the level hypervector and the corresponding base hypervector, i.e., $\mathbf{B}_i \oplus \overline{\mathbf{L}}_i$ for i^{th} feature index, where $\overline{\mathbf{L}}_i \in \{\mathbf{L}_1,\ldots,\mathbf{L}_m\}$ (Ⓒ). Finally, for each dimension, the results of XOR operations are accumulated using an adder working in a pipeline structure (Ⓓ). All encoding operations, including the XOR and adders are implemented using LookUp Tables (LUTs) and Flip Flops (FFs).

Depending on the number of features, n, and hypervectors dimensionality, D, FPGA may not have enough resources to generate all D dimensions of encoded hypervector once at a time. In that case, our implementation performs encoding only on d dimensions of the level and base hypervectors ($d < D$). We further discuss about the encoding throughput at Section IV-B)

B. Training Acceleration

Training involves in the accumulation of all encoded hypervectors corresponding to a class, where each hypervector represents using D non-quantized values. FPGA performs the addition of the encoded hypervectors using DSP blocks. Since the training does not share many resources with the encoding module, they can be run in parallel to accelerate the training. Figure 5 shows an overview of the FPGA implementation of the training module. When an encoding module generates the d dimensions of the encoded hypervector, DSPs accumulate the previously d encoded dimensions with the corresponding class hypervector. The encoding and training modules are working in a pipeline structure, which results in hiding the latency of the encoding module. Depending on a class that the data point belongs to (a tag bit shown in Figure 5), FPGA reads the class hypervectors (Ⓔ). Then, this class hypervector is accumulated with the encoded hypervector in a tree-based adder (Ⓕ) To accelerate the training, we sort the data such that all data points corresponding to a class process sequentially. Note that this sorting can have negative impact on the classification accuracy. As it has been shown by several work [20], [21], training with randomly shuffled data reduces the chance of over-fitting; thus results in providing a higher classification accuracy. For example, in ISOLET, we observe that QuantHD provides 0.7% higher accuracy if it trains on shuffled data. Here, we sort training data point in order to improve the FPGA efficiency. This sorting enables the accumulation operation happens on a class hypervector without accessing another class hypervector from the internal FPGA block RAM (BRAM). Finally, the accumulated class hypervector can be written back into the BRAM block (Ⓖ).

The value d can be limited either by the encoding or training module. In the encoding module, the maximum d which can be generated at each iteration depends on the features size, and it is limited by the number of available LUT/FF resources. On the other hand, the maximum d value that training module can process depending on the number of available DSPs on the FPGA. For applications such as physical monitoring with $n = 75$ features and $k = 6$ classes, the number of DSPs limits the d value to 64. However, for face recognition with $n = 608$ features and $k = 2$ classes, the d value is limited to 192 by

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2954472, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

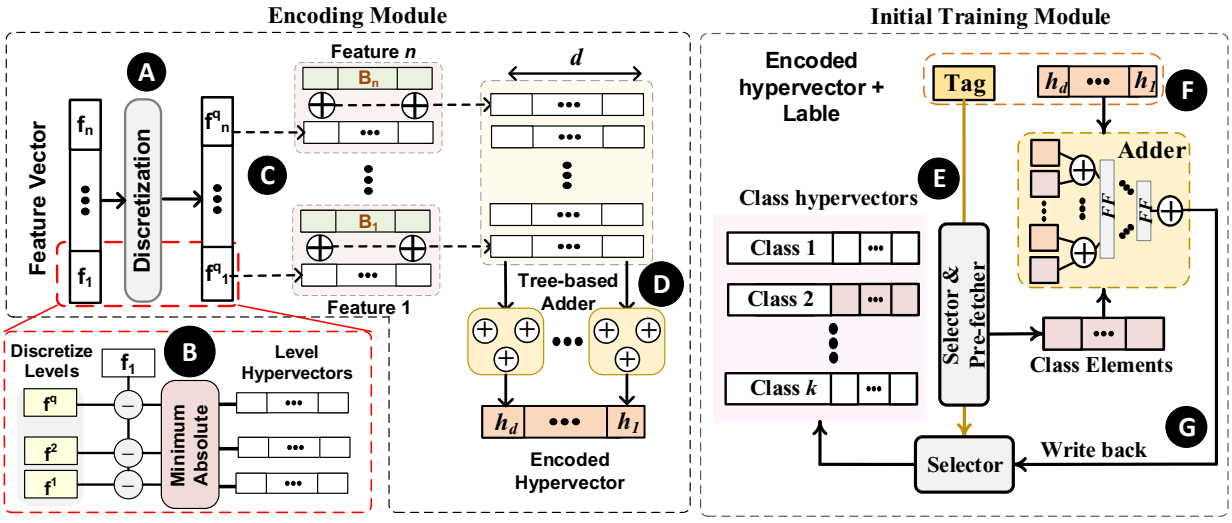


Fig. 5. Hardware acceleration of the encoding and initial training modules.

LUT utilization. This training process continues over the entire training data points until generating a hypervector for each existing class.

C. Retraining & Inference Acceleration

QuantHD uses similar hardware to accelerate retraining and inference modules. The associative search is the main functionality of QuantHD during inference and retraining. Retraining also uses another module to update the class hypervectors and quantize the model, when a train data point misclassifies with a model. Here we explain the FPGA implementation of each module in details.

Model Quantization: After training the non-quantized model, FPGA quantizes the model by comparing each class dimension with a threshold value. This process is performed by reading class hypervectors from distributed RAMs and comparing each dimension with a threshold value(s) (shown in Figure 6c). For binarization, all dimensions with a smaller value than a μ threshold are assigned to "0" bit, while other dimensions get "1" bit. Similarly, ternarization happens similarly by comparing each class element with two threshold values. As we discussed in Section III-C, we assign values smaller than a $TH_1 = \mu - 0.42\sigma$ to "01", values larger than $TH_2 = \mu + 0.42\sigma$ to "11", and other values to "00". FPGA stores both quantized and fixed-point models to perform the retraining.

Associative Search: During retraining and inference, HD requires to perform the associative search over the training and testing data respectively. The existing HD computing algorithms [8] perform the retrain on the non-quantized model. Thus, they require to use costly cosine similarity during retraining. In contrast, QuantHD associative search happens on the quantized model. The search over quantized model significantly accelerates the retraining/inference procedure since it avoids the costly cosine similarity between a query and non-quantized model.

Figure 6a shows an overview of the FPGA implementing the associative search block. Regardless of using a binary or

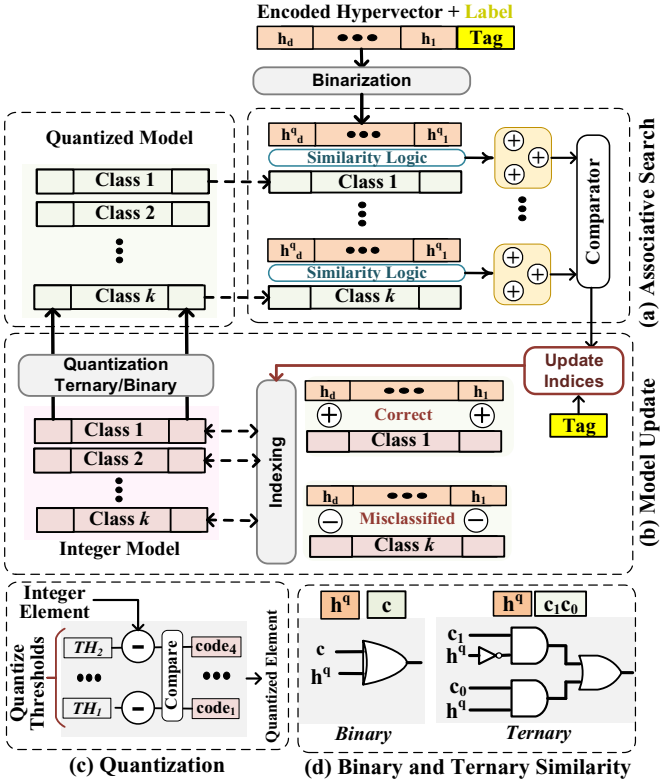


Fig. 6. Hardware acceleration of inference and retraining; (a) associative search of the binary query with the quantized model. (b) Updating non-quantized model in case of misclassification. (c) Quantization of the model. (d) similarity metric used for quantized model.

a ternary model, our approach uses binary query hypervector ($\mathbf{H}^q$) to perform the similarity check. The binarization of the query hypervector happens right after the encoding module by comparing each element with $n/2$, where n is the number of features in the application. In our implementation, the encoding and associative search modules are working in a pipeline structure. As we explained, the encoding module

cannot generate all $D = 10,000$ dimensions of a query hypervector in a single iteration. Therefore, when associative search performs the similarity check on d dimensions of a query hypervector, the encoding module generates the next d dimensions of a query hypervector. In associative search, we first read the first d dimensions of all class hypervectors and then perform the similarity of the binary query with each class hypervector. Figure 6d shows the FPGA implementation supporting dot product between a single class and a query hypervector. Depending on the model, i.e., binary or ternary, the class elements can be represented using one or two bits. For the binary model, class elements represent using a single bit. Thus, the similarity check between a query and the class hypervectors simplifies to Hamming distance. As Figure 6d shows, this operation can be implemented using a single `XOR` gate. For ternary model, each class element represents using two bits, where $\{01, 00, 11\}$. In the ternary model using two bits to represent each class value, the similarity search can be implemented using two `AND`, a `NOT`, and an `OR` gates. The similarity logic in both binary and ternary models create a single bit for each dimension. The results of similarity logic, which are d bits, are added together in a d-bits tree-based adder (Shown in Figure 6a). This d window sequentially moves through all class/query dimensions to cover all D dimensions.

Model Update: In the case of an incorrect match, the retraining uses another module to update the non-quantized model (Figure 6b). For each data point in training data, QuantHD checks the result of a similarity search with the label of training data. In the case of misclassification, the "Update Indices" generates the address of two classes that need to be updated. FPGA reads those class hypervectors which are already stored in distributed RAM blocks and updates them by addition and subtraction of them with a query hypervector ($\mathbf{H}$). Finally, the class hypervectors are written back into distributed RAM blocks. QuantHD continues the search operation on the next encoded data points using the same quantized model. Finally, QuantHD quantizes the model (Figure 6b) after going over the entire training data. This process continues iteratively for a pre-defined number of iterations.

V. Evaluation

A. Experimental Setup

We implement and verify the functionality of QuantHD training and inference using Verilog. We synthesize the code on Xilinx Vivado Design Suite [22] and implement it on the Kintex-7 FPGA KC705 Evaluation Kit. We used Vivado XPower tool to estimate the device power. All QuantHD software support including training, retraining, and inference have been implemented on CPU. For CPU, the QuantHD has been written in C++ and optimized to provide the maximum performance on embedded devices (Raspberry Pi 3) using ARM Cortex A53 CPU. As a baseline, we compare the accuracy and efficiency of QuantHD with state-of-the-art HD computing algorithm [9], [7], multi-level perceptron [23], and binary neural network [24] implemented on FPGA. To show the advantage of QuantHD in both algorithm and hardware aspects, we implement the baseline HD computing using the same implementation as QuantHD, explained in Section IV. Table II summarizes the evaluated datasets. The tested benchmarks range from relatively small datasets collected in a small IoT network, e.g., PHYSICAL, to a large dataset which includes hundreds of thousands of images of facial and non-facial data. Note that in image-like data, QuantHD works on features which are extracted from the original data. For example, we use Histogram of Oriented Gradients (HOG) feature extraction for Face Detection problem. In contrast, the convolution layer in neural networks can directly extract information from the original image. Our future work in to include feature extraction as a part of the encoding method.

B. Accuracy

Table III compares the classification accuracy of QuantHD using binary and ternary models with the state-of-the-art HD computing algorithm using binary and non-quantized models [9]. The baseline uses Ngram-based encoding, while QuantHD uses the encoding proposed in Section II-A. To have a fair comparison, we give an advantage to the baseline HD to retrain the non-quantized model for the same number of iterations as QuantHD model. The results also reported for the baseline HD with the binary model, when the HD models have been turned into binary once after the training. For QuantHD, the models have been retrained for 40 iterations with $\alpha = 0.05$ learning rate. For ternary models, we use ternary boundary $b = 0.42\sigma$ which results in maximum classification accuracy.

Our evaluation shows that the baseline HD provides high classification accuracy using non-quantized model. However, in baseline model, both training and retraining are significantly costly. The training process involves retraining that involves several iterations of the similarity check over non-quantized model. Similarly, in the inference phase, the associative search between query and trained model required costly consine metric. The binarization of the baseline model has been proposed to reduce the inference cost, by replacing cosine with Hamming distance similarity. However, this binarization used inthe baseline HD computing [9], [8] has two main disadvantages: (i) it results in a significant drop in the classification accuracy, as the model never trained to work with this constraint. (ii) The retraining is as costly as the non-quantized model as the similarity check needs to perform using the cosine metric.

QuantHD addresses the several existing issues in the HD computing algorithms. QuantHD provides an iterative procedure which enables the HD to learn to work with binary or ternary models. In addition, QuantHD defines a learning rate for training procedure which further improves the HD classification as compared to prior work with no learning rate ($\alpha = 1$). Our evaluation on Table III shows that QuantHD using binary and ternary models can provide comparable accuracy to the non-quantized model. The accuracy is higher for the ternary model as it gives more flexibility to the training module to select better class values. Our evaluation shows that QuantHD using binary and ternary models provide on average 16.2% and 17.4% higher accuracy as compared to the baseline HD computing [9] using a binary model (See Table III). In

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2954472, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

TABLE II
DATASETS (n: FEATURE SIZE, k: NUMBER OF CLASSES).

	n	K	Data Size	Train Size	Test Size	Description/State-of-the-art Model
ISOLET	617	26	19MB	6,238	1,559	Voice Recognition [11]
UCIHAR	561	12	10MB	6,213	1,554	Activity recognition(Mobile)[12]
PAMAP2	75	5	240MB	611,142	101,582	Activity recognition(IMU)[13]
FACE	608	2	1.3GB	522,441	2,494	Face recognition[14]
EXTRA	225	4	140MB	146,869	16,343	Phone position recognition[25]

TABLE III
COMPARISON OF QUANTHD CLASSIFICATION ACCURACY WITH THE STATE-OF-THE-ART HD COMPUTING.

	Baseline HD		QuantHD		
	Non-Quantized	*Binary*	*Non-Quantized*	*Binary*	*Ternary*
ISOLET	91.1%	88.1%	95.8%	94.6%	95.3%
UCIHAR	93.8%	77.4%	98.1%	96.5%	97.2%
PAMAP2	88.9%	85.7%	92.7%	91.3%	92.7%
FACE	95.9%	68.4%	96.2%	94.6%	95.4%
CARDIO	93.7%	90.9%	97.4%	95.3%	97.7%
EXTRA	70.2%	66.7%	74.1%	72.6%	74.0%
Average	**88.9%**	**79.5%**	**92.4%**	**90.8%**	**92.1%**

addition, we observe that QuantHD accuracy using binary and ternary models is 1.9% and 3.1% higher than baseline HD using non-quantized model.

C. Training Efficiency

We compare the QuantHD with the baseline HD computing in terms of training/retraining efficiency. All HD-based designs have the same performance/energy during the generation of the initial training model. However, during retraining which involves the significant training cost, they have different computation efficiency. In the baseline HD, the retraining is performed by checking the similarity of each training data point with the non-quantized model. This search significantly increases the cost of retraining, since the associative search in the non-quantized domain is much more costly than binary or ternary domains. In contrast, the retraining in QuantHD performs by checking the similarity of training data points with the binary/ternary model. After each similarity check, QuantHD updates the non-quantized model by adding and subtracting a query hypervector from two class hypervectors. Figure 7 compares the energy consumption and execution time of the baseline HD and QuantHD. Our evaluation shows that QuantHD with the binary (ternary) model can achieve on average 36.4× and 4.5× (34.1× and 4.1×) energy efficiency improvement and speedup as compared to the baseline HD computing algorithm.

D. Inference Efficiency

Figure 8 compares the energy consumption and execution time of running a single query in the baseline HD computing with QuantHD using binary and ternary models. All reported results are the average energy and execution time of a single prediction, processed on the entire test data. In QuantHD, the encoding and associative search modules are working in a pipeline stage. Therefore, the execution time of the encoding module hides under the execution time of the associative search search. However, FPGA still needs to pay the cost of energy consumption in the encoding module (as shown in top graph in Figure 8).

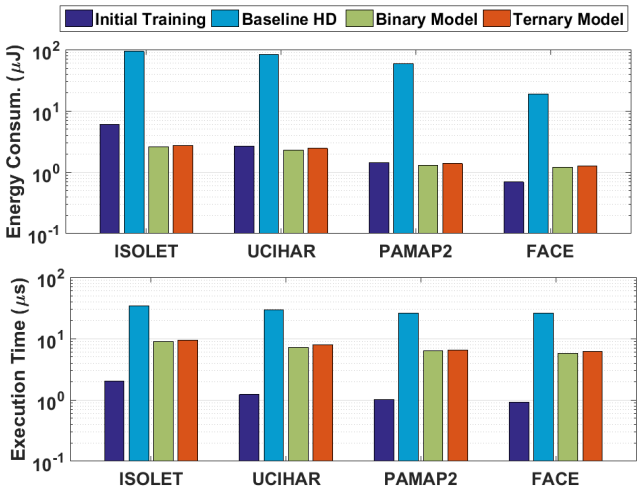


Fig. 7. Energy consumption and execution time of QuantHD, conventional HD and BNN during training

We used Vivado XPower tool to estimate the device power. Our evaluation shows that HD using the non-quantized model is the most inefficient design due to its significant cost during the associative search. Regardless of the training procedure, the binary models provide the maximum efficiency during inference. We also observe that for most of the applications, both binary and ternary models can provide higher efficiency than BNN due to their lower number of computations. The results show that QuantHD using binary and ternary model can achieve 45.7× and 42.3× energy efficiency improvement and 5.2× and 4.7× speedup as compared to baseline HD while providing comparable accuracy.

Figure 9 shows the energy consumption and execution time of our FPGA implementation with AMD Radeon R390 GPU and ARM Cortex A53 CPU during the inference. All platforms run the baseline HD code using non-quantized model with $D = 10,000$ dimensions. For each application, the results of energy and execution time are normalized to GPU running HD computing with $D = 10,000$ dimensions. Our evaluation shows that for all tested applications, FPGA can provide on average 7.6× (5.9×) lower energy consumption and 1.7× (41.5×) faster computation as compared to the GPU (CPU) when running HD in full dimension. The higher efficiency of the FPGA comes from its optimized implementation, high level of parallelism, and storing the HD model close to the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2954472, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

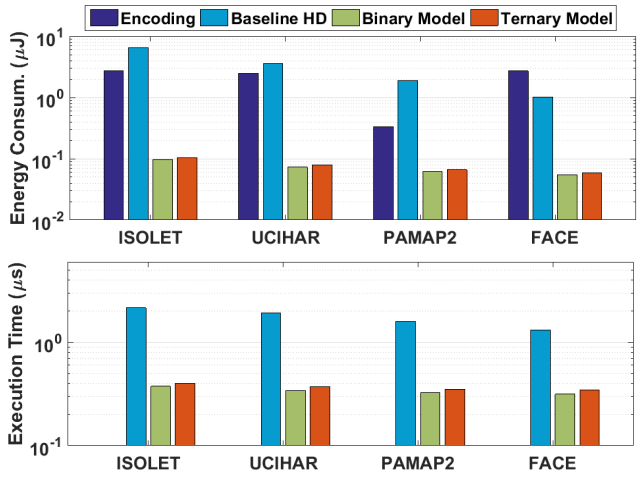


Fig. 8. Energy consumption and execution time of QuantHD during inference

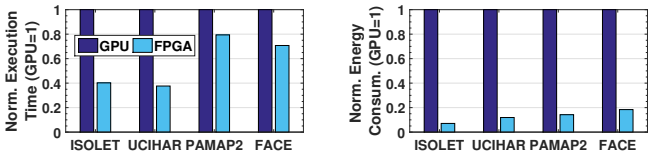
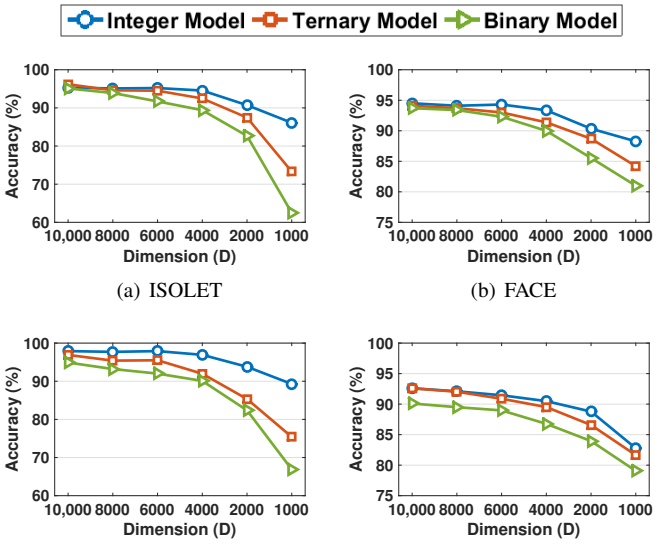


Fig. 9. Normalized Energy consumption and execution time of our FPGA as compared to GPU.

computing units.

E. QuantHD Trade-off: Dimensionality

HD computing has been designed to work with the pattern of vectors in high-dimensional space, i.e., $D = 10,000$. The correct dimensions depend on the actual dataset. However, we can reduce the hypervectors dimension to accelerate both training and testing computation. Figure 10 shows the impact of dimensionality on the classification accuracy of different applications using non-quantized, ternary, and binary models. Our evaluation shows that QuantHD using binary/ternary model can provide similar accuracy as QuantHD with the non-quantized model when the dimensionality is high. However, QuantHD using all models starts losing accuracy when the hypervector dimensions reduce. This accuracy drop is higher for the binary and ternary model since they use low accurate metric, i.e., Hamming distance, for similarity check. Dimension reduction also increases the gap between the accuracy of the non-quantized and quantized model. In particular, we observe a large accuracy drop on QuantHD with the binary model when the dimensionality gets lower than 8000. The results show that QuantHD using $D = 8000$ can achieve similar classification accuracy as full dimensionality while providing 17.6% energy efficiency and 14.3% speedup. In addition, QuantHD using ternary model can provide 26.4% and 19.8% (34.9% and 27.9%) energy efficiency and speedup while providing 1% (2%) quality loss, as compared to QuantHD with full dimension.



(a) ISOLET (b) FACE

(c) UCIHAR (d) PAMPA

Fig. 10. Impact of dimension reduction on the accuracy of QuantHD with binary and ternary models.

TABLE IV
RESOURCE UTILIZATION OF FPGA DURING TRAINING, RETRAINING, AND INFERENCE.

		d	LUT	FF	DSP	BRAM
Training	*Initial*	128	96%	22%	79%	32%
Retraining	*Baseline*	32	23%	18%	93%	31%
	Binary	128	92%	16%	8%	10%
	Ternary	192	95%	18%	8%	13%
Inference	*Baseline*	32	16%	12%	88%	31%
	Binary	224	93%	32%	0%	7%
	Ternary	192	97%	34%	0%	9%

F. Breakdown

Table IV compares the resource utilization of QuantHD using non-quantized, binary, and ternary models. The results are reported for UCIHAR datasets with $n = 561$ features, $k = 12$ classes and using $D = 10,000$. In terms of resource utilization, all models are using the same utilization for encoding and initial training. The difference of these methods is on the associative search where the non-quantized model takes larger resource to implement a similarity check. The larger associative memory in ternary model reduces the encoder size since these blocks are both implemented using the same LUT and FFs logics. Therefore, the lower number of dimensions can be generated by the encoder at each time. Our results in Table IV show that while in QuantHD with the non-quantized model, the DSP utilization is the main bottleneck of retraining and inference performance. However, QuantHD using binary/ternary model can provide higher performance by removing the necessity of DSPs to perform the associative search. Therefore, QuantHD with the quantized model processes a larger d in parallel.

G. QuantHD vs. Other Algorithms

As a light-weight classifier, we compared QuantHD accuracy and efficiency with the state-of-the-art light-weight clas-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2954472, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

TABLE V
COMPARISON OF QUANTHD WITH MLP AND BNN IN TERMS OF ACCURACY, EFFICIENCY, AND MODEL SIZE.

	MLP/BNN Topologies	Accuracy			CPU Training (s)			FPGA Inference (us)			Model Size		
		MLP	*BNN*	*HD*	*MLP*	*BNN*	*HD*	*MLP*	*BNN*	*HD*	*MLP*	*BNN*	*HD*
ISOLET	617-512-256-26	95.8%	96.1%	95.8%	2.08	17.69	0.31	27.39	5.24	0.40	1.81MB	56.7KB	65.0KB
UCIHAR	561-512-256-12	97.3%	95.9%	97.2%	1.04	8.32	0.12	21.43	5.18	0.37	1.68MB	52.7KB	30.0KB
PAMAP2	75-512-256-6	95.8%	94.2%	92.7%	0.61	4.75	0.07	13.07	3.78	0.35	0.68MB	21.3KB	15.0KB
FACE	608-512-256-2	96.1%	96.1%	95.4%	0.56	4.30	0.04	17.68	5.11	0.34	1.77MB	55.3KB	5.0KB

sifiers including Multi-Layer Perceptron (MLP) and Binarized Neural Network (BNN). For MLP and BNN, we used the similar models proposed in [24] and made a small modification in input and output layers to run different applications (shown in Table V). We use Keras with Tensorflow backend to train the MLP (BNN) models using Adam optimizer for 10 (100) epochs and learning rate of 0.01. Dropout with a drop rate of 0.5 is applied to DNN layers. Table V lists the classification accuracy and the training/inference execution time of all algorithms. The training results are reported on an embedded device (Raspberry Pi 3) using ARM Cortex A53 CPU. All testing results are reported on Kintex-7 FPGA. For testing, we used the same BNN implementation available on [24] to synthesize the networks on Kintex-7 FPGA. For BNN models, we used the best design parameters (# of SIMD and processing elements) which result in maximum resource utilization and performance. For MLP models, we used the framework proposed in [23] to implement MLP high-level code on FPGA efficiently.

Our evaluation shows that QuantHD can provide comparable accuracy to advanced algorithms while providing much faster computation in both training and testing. For example, QuantHD is on average 8.2× and 67.4× faster than MLP and BNN during training, when running on ARM Cortex A53 CPU. In addition, using FPGA acceleration for training, QuantHD can further achieve 48.9× speedup as compared to ARM CPU. Similarity during testing, QuantHD achieves on average 48.1× and 13.4× speedup as compared to the MLP and the BNN models, while providing the similar classification accuracy. Table V also compares the QuantHD, MLP, and BNN in terms of model size. Our evaluation shows that QuantHD can provide on average 52.2× and 1.6× smaller model size as compared to MLP and BNN respectively.

VI. RELATED WORK

The idea of HD computing has been mapped into several practical problems including supervised, semi-supervised, and unsupervised learning tasks [26], [27], [28], [29]. For classification, many existing approaches use HD computing for a single-pass training [9], [30]. For example, the work in [9] proposed a text classification algorithm based on random indexing. However, this method of training provides significantly lower classification accuracy on practical applications. Work in [8] proposed a retraining approach which improves the HD computing accuracy by iteratively updating the HD model over the training dataset. However, this approach forces both retraining and inference phases to use associative search on the fixed-point model which requires costly cosine metric for similarity check. In addition, the lack of definition of learning rate results in low stability and possible divergence during the retraining. Prior work tried to binarize the model after the retraining to reduce the inference cost [7], [8]. However, this approach results in a significant drop in classification accuracy. Moreover, the retraining is still expensive as it needs to be processed on the fixed-point model. In contrast, we propose a novel framework which enables both retraining and inference phases to perform on a quantized model using hardware friendly similarity metrics, i.e., Hamming distance. Our approach introduces the definition of the learning rate in retraining and adapts the HD model to work with the quantization constraints.

On the other side, prior work tried to design efficient hardware to accelerate HD computing, focusing on binary hypervectors. Work in [19], [31], [16], [32], [33] designed in-memory architecture based on emerging non-volatile memory to accelerate HD computation. For example, work in [16] showed three memory-centric hardware to accelerate the associative search in 10,000 dimensions. However, since these hardware are working with binary hypervectors, they provide very low classification accuracy on practical applications (explained in Section II-C). Work in [34], [35] proposed an efficient implementation of HD computing on FPGA, by exploiting the sparsity to enhance computation efficiency. Our proposed framework enables model quantization, and it opens a new opportunity for binary-based hardware to be used for a wide range of classification problems. In addition, unlike prior work that accelerates HD computing at inference, we proposed FPGA implementation which accelerates all phases on HD computing including training, retraining, and inference.

VII. CONCLUSION

In this paper, we propose a novel framework for model quantization of Hyperdimensional computing. In contrast to prior work that quantization results in a significant quality loss, QuantHD enables binarization and ternarization of the HD model with minimal impact of the accuracy. QuantHD is an adaptive framework which retrains the HD model to compensate for the possible quality loss due to quantization. Our framework is general and can be used for any types of quantization or enabling sparsity in HD computing. We observe that the gain of quantization is more evident in high-dimensional space. Going toward more restricted quantization, e.g., binarization, dimension reduction has a more destructive impact on the quality of the model. Therefore, a designer can

devise to either process HD computing using a low dimensional non-quantized model or high-dimensional quantized model, depending on the underlying platform.

VIII. Acknowledgment

This work was supported in part by CRISP, one of six centers in JUMP, an SRC program sponsored by DARPA and NSF grants #1911095, #1826967, #1730158 and #1527034; and in part by E2CDA-NRI, a funded center of NRI, a Semiconductor Research Corporation (SRC) program sponsored by NERC and NIST.

References

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[2] M. Denil, B. Shakibi, L. Dinh, N. De Freitas, *et al.*, "Predicting parameters in deep learning," in *Advances in neural information processing systems*, pp. 2148–2156, 2013.

[3] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[4] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang, *et al.*, "Ese: Efficient speech recognition engine with sparse lstm on fpga," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 75–84, ACM, 2017.

[5] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive Computation*, vol. 1, no. 2, pp. 139–159, 2009.

[6] M. Imani, Y. Kim, S. Riazi, J. Messerly, P. Liu, F. Koushanfar, and T. Rosing, "A framework for collaborative learning in secure high-dimensional space," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pp. 435–446, IEEE, 2019.

[7] M. Imani, C. Huang, D. Kong, and T. Rosing, "Hierarchical hyperdimensional computing for energy efficient classification," in *Proceedings of the 55th Annual Design Automation Conference*, p. 108, ACM, 2018.

[8] M. Imani, D. Kong, A. Rahimi, and T. Rosing, "Voicehd: Hyperdimensional computing for efficient speech recognition," in *International Conference on Rebooting Computing (ICRC)*, pp. 1–6, IEEE, 2017.

[9] A. Rahimi, P. Kanerva, and J. M. Rabaey, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, pp. 64–69, ACM, 2016.

[10] P. Kanerva, J. Kristofersson, and A. Holst, "Random indexing of text samples for latent semantic analysis," in *Proceedings of the 22nd annual conference of the cognitive science society*, vol. 1036, Citeseer, 2000.

[11] "Uci machine learning repository." http://archive.ics.uci.edu/ml/datasets/ISOLET.

[12] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," in *International workshop on ambient assisted living*, pp. 216–223, Springer, 2012.

[13] A. Reiss and D. Stricker, "Introducing a new benchmarked dataset for activity monitoring," in *Wearable Computers (ISWC), 2012 16th International Symposium on*, pp. 108–109, IEEE, 2012.

[14] Y. Kim, M. Imani, and T. Rosing, "Orchard: Visual object recognition accelerator based on approximate in-memory processing," in *Computer-Aided Design (ICCAD), 2017 IEEE/ACM International Conference on*, pp. 25–32, IEEE, 2017.

[15] A. Rahimi, S. Datta, D. Kleyko, E. P. Frady, B. Olshausen, P. Kanerva, and J. M. Rabaey, "High-dimensional computing as a nanoscalable paradigm," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 9, pp. 2508–2521, 2017.

[16] M. Imani, A. Rahimi, D. Kong, T. Rosing, and J. M. Rabaey, "Exploring hyperdimensional associative memory," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*, pp. 445–456, IEEE, 2017.

[17] A. Rahimi, A. Tchouprina, P. Kanerva, J. d. R. Millán, and J. M. Rabaey, "Hyperdimensional computing for blind and one-shot classification of eeg error-related potentials," *Mobile Networks and Applications*, pp. 1–12, 2017.

[18] M. Imani, T. Nassar, A. Rahimi, and T. Rosing, "Hdna: Energy-efficient dna sequencing using hyperdimensional computing," in *2018 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI)*, pp. 271–274, IEEE, 2018.

[19] T. Wu, P. Huang, A. Rahimi, H. Li, J. Rabaey, P. Wong, and S. Mitra, "Brain-inspired computing exploiting carbon nanotube fets and resistive ram: Hyperdimensional computing case study," in *IEEE Intl. Solid-State Circuits Conference (ISSCC)*, IEEE, 2018.

[20] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting.," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[21] C. Zhang, O. Vinyals, R. Munos, and S. Bengio, "A study on overfitting in deep reinforcement learning," *arXiv preprint arXiv:1804.06893*, 2018.

[22] T. Feist, "Vivado design suite," *White Paper*, vol. 5, 2012.

[23] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh, "From high-level deep neural models to fpgas," in *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*, pp. 1–12, IEEE, 2016.

[24] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 65–74, ACM, 2017.

[25] Y. Vaizman, K. Ellis, and G. Lanckriet, "Recognizing detailed human context in the wild from smartphones and smartwatches," *IEEE Pervasive Computing*, vol. 16, no. 4, pp. 62–74, 2017.

[26] M. Imani, J. Morris, J. Messerly, H. Shu, Y. Deng, and T. Rosing, "Bric: Locality-based encoding for energy-efficient brain-inspired hyperdimensional computing," in *Proceedings of the 56th Annual Design Automation Conference 2019*, p. 52, ACM, 2019.

[27] Y. Kim, M. Imani, and T. S. Rosing, "Efficient human activity recognition using hyperdimensional computing," in *Proceedings of the 8th International Conference on the Internet of Things*, p. 38, ACM, 2018.

[28] M. Imani, Y. Kim, T. Worley, S. Gupta, and T. Rosing, "Hdcluster: An accurate clustering using brain-inspired high-dimensional computing," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1591–1594, IEEE, 2019.

[29] M. Imani, S. Bosch, B. Rouhani, X. Wu, F. Koushanfar, and T. Rosing, "Semihd: Semi-supervised learning using hyperdimensional computing," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, IEEE, 2019.

[30] A. Rahimi, S. Benatti, P. Kanerva, L. Benini, and J. M. Rabaey, "Hyperdimensional biosignal processing: A case study for emg-based hand gesture recognition," in *Rebooting Computing (ICRC), IEEE International Conference on*, pp. 1–8, IEEE, 2016.

[31] H. Li, T. F. Wu, A. Rahimi, K.-S. Li, M. Rusch, C.-H. Lin, J.-L. Hsu, M. M. Sabry, S. B. Eryilmaz, J. Sohn, *et al.*, "Hyperdimensional computing with 3d vrram in-memory kernels: Device-architecture co-design for energy-efficient, error-resilient language recognition," in *Electron Devices Meeting (IEDM), 2016 IEEE International*, pp. 16–1, IEEE, 2016.

[32] S. Gupta, M. Imani, and T. Rosing, "Felix: Fast and energy-efficient logic in memory," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–7, IEEE, 2018.

[33] M. Imani, X. Yin, J. Messerly, S. Gupta, M. Nemier, S. Hu, and T. S. Rosing, "Searchd: A memory-centric hyperdimensional computing with stochastic training," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

[34] S. Salamat, M. Imani, B. Khaleghi, and T. Rosing, "F5-hd: Fast flexible fpga-based framework for refreshing hyperdimensional computing," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 53–62, ACM, 2019.

[35] M. Imani, S. Salamat, B. Khaleghi, M. Samragh, F. Koushanfar, and T. Rosing, "Sparsehd: Algorithm-hardware co-optimization for efficient high-dimensional computing," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 190–198, IEEE, 2019.