Efficient Hyperdimensional Learning with Trainable, Quantizable, and Holistic Data Representation

Jiseung Kim
DGIST
js980408@dgist.ac.kr

Hyunsei Lee
DGIST
wwhslee@dgist.ac.kr

Mohsen Imani
UC Irvine
m.imani@uci.edu

Yeseong Kim
DGIST
yeseongkim@dgist.ac.kr

***Abstract*—Hyperdimensional computing (HDC) is a computing paradigm that draws inspiration from human memory models. It represents data in the form of high-dimensional vectors. Recently, many works in literature have tried to use HDC as a learning model due to its simple arithmetic and high efficiency. However, learning frameworks in HDC use encoders that are randomly generated and static, resulting in many parameters and low accuracy. In this paper, we propose TrainableHD, a framework for HDC that utilizes a dynamic encoder with effective quantization for higher efficiency. Our model considers errors gained from the HD model and dynamically updates the encoder during training. Our evaluations show that TrainableHD improves the accuracy of the HDC by up to 22.26% (on average 3.62%) without any extra computation costs, achieving a comparable level to state-of-the-art deep learning. Also, the proposed solution is $56.4\times$ faster and $73\times$ more energy efficient as compared to the deep learning on NVIDIA Jetson Xavier, a low-power GPU platform.**

***Index Terms*—Hyperdimensional Computing, Quantization, Alternative Computing, Data Representation**

I. Introduction

Hyperdimensional (HD) computing is an alternative computing paradigm that draws inspiration from brain functions. It exploits a high degree of binary-centric operations and boasts highly parallel computations to realize highly lightweight learning. For example, prior research utilized HDC as an energy-efficient classifier alternative to sophisticated deep learning to solve various classification problems [1], [2] with high accuracy. Unlike conventional representation systems where specific positions of elements define the meaning. e.g., computing with 32/64-bit words, HDC uses holistically represented vectors of high dimensions, i.e., dimensions in thousands, called *hypervectors*, enabling noise-tolerant and highly-parallel learning. Based on the property of the hypervectors and distances in the high-dimensional space, we can *represent* different data and *learn* their relationships. The key idea is to first map (encode) original data into hypervectors and combine them with the lightweight HD operations, which perform brain-like cognitive functionalities, e.g., memorization and information association, eventually training a set of new hypervectors representing each class. We can then perform inference by computing hyperspace distances between the hypervector encoding's given input and each of the class hypervectors trained.

As aforementioned, when learning on HD models, data points are encoded to hyperspace — that is, we first convert raw input data into hypervectors. The encoding method plays a prominent role in the accuracy and complexity of models. The state-of-the-art encoding methods exploit a high-dimensional matrix of hypervectors, called *base hypervectors*, whose elements for each dimension are randomly generated before the training. The randomized generation ensures any member hypervectors are unrelated, as they are near-orthogonal in the hyperspace. The encoder associates factors of data points with the projection matrix to encode feature data into hypervectors.

There are various encoding methods proposed [1], [2], [3], [4], [5], [6], [7], [8], [9]; but once generated, all existing encoding methods use the *same* projection matrix for the entirety of the learning phase, posing critical technical issues. First, randomly drawn values result in hypervectors that disregard relations between features of input data points; but are also *learnable* during the training as performed in other state-of-the-art algorithms such as deep learning. Second, the HDC based on the existing encoders necessitates extremely *large dimensions*, e.g., $D = 10,000$ to guarantee high accuracy. This requirement is inevitable since the distinguishable hypervectors must be generated on purely random extraction. The large dimension consequently degrades the efficiency.

To sum up, the *static* nature of the encoding forces the utilization of higher dimensions and is widely considered as the reason behind the lower accuracy of HD learning models. In this paper, we propose the TrainableHD framework, an HDC method to improve accuracy by applying a *dynamic* encoder along with an efficient quantization method. During the training process, TrainableHD evaluates the quality of the trained models and updates base hypervectors used in the encoders by calculating the expected errors with HD arithmetic. It improves the accuracy significantly and makes use of lower dimensions in HDC. Since we do not introduce any additional operations for the inference, TrainableHD can provide high performance and efficiency, ensuring high prediction quality. The followings summarize our contributions:

1) **We propose TrainableHD, which enables learning of the appropriate HD encoder**. To the best of our knowledge, this is the first work that trains the encoder of HDC, addressing the static nature of existing encoders.
2) **We present an optimization technique to minimize the overhead of the encoder training**. We train the encoder only when necessary, significantly improving the performance.
3) **To realize the efficient inference, TrainableHD enables quantization for HD learning**. We also show an efficient inference framework on various acceleration platforms, including CPU with SIMD, Tensor Core (GPU), and FPGA.

Our evaluation shows that TrainableHD outperforms the accuracy of existing HD learning methods by up to 22.26% without introducing additional computational costs to the inference. Combined with quantization, the inference of TrainableHD on Jetson Xavier GPU (Zynq-7000 FPGA) is $56.4\times$ ($180.8\times$)

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government(MSIT) (No.2018R1A5A1060031), Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science. This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2022-0-00991, 1T-1C DRAM Array Based High-Bandwidth, Ultra-High Efficiency Processing-in-Memory Accelerator). This work was supported in part by National Science Foundation #2127780, Office of Naval Research grants #N00014-21-1-2225 and #N00014-22-1-2067, the Air Force Office of Scientific Research under award #FA9550-22-1-0253, and generous gifts from Cisco and Xilinx.

978-3-9819263-7-8/DATE23/© 2023 EDAA

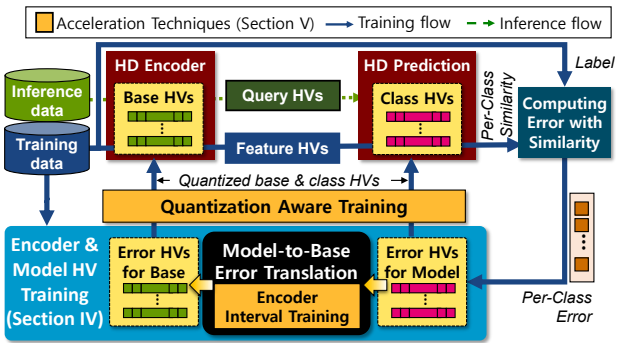


Fig. 1: Overview of TrainableHD

faster and 73× (167.8×) more energy efficient than deep learning on the Xavier GPU.

II. Related Work

HD computing has been implemented to solve various learning problems [9]. The encoding of data samples into hypervectors is the most important phase for the HD learning models as it has the most influence on the model's accuracy and complexity. However, existing encodings [1], [2], [3], [4], [5], [6], [7], [8], [10] do not utilize the knowledge obtainable from training samples. Recently, work in [11] proposed an alternative encoding method called ManiHD, which implements the manifold projection before the static HD encoder. However, ManiHD is a class information-agnostic method that does not consider the characteristics of each class and still utilizes the static encoder, resulting in suboptimal accuracy. In our measurement over various datasets, the manifold projection procedure also spent about 2.95× more time in the encoding process. Our work is different in that we learn the encoder itself dynamically without adding any extra overheads to the inference.

III. TrainableHD Overview

Figure 1 shows an overview of the proposed TrainableHD learning framework. The goal of our training is to identify two types of learned hypervector representations: (i) *base hypervectors* of the encoder, initially created with random components generated from Gaussian distribution, and (ii) *class hypervectors* that represent high-dimensional patterns for each class, initially having zero-value components. During training, using the current base hypervectors, we first encode the hypervector representation for the training samples, called *(encoded) feature hypervectors*. The training is proceeded by comparing the similarity between the feature hypervectors and every *class hypervector*. The HD module defines the class of training data as the class hypervector that showed the maximum similarity value. Based on the similarity values and ground-truth labels, the HD module computes per-class errors and updates the class hypervectors to reduce the error for later predictions. The core of TrainableHD is the encoder training technique discussed in Section IV, which *translates* the per-class scalar errors to per-feature hypervector errors to update the base hypervectors. The encoder training happens only when it is necessary based on an optimization technique called Encoder Interval Training (EIT) (Section V-A.) This process is repeated for all training samples with a mini-batch over multiple epochs. We also propose a method to train the model quantized with low precision hypervector elements, e.g., 8-bit integers, using quantization-aware training (QAT) (Section V-B.)

At the inference phase fully accelerated by the quantization, the *trained* encoder creates the *query hypervector* using inference data without extra computation costs upon the existing HD learning solutions. Then, the HD model performs the similarity computation between the query hypervector and class hypervectors to identify the sample's class.

IV. Dynamic Encoder Training with HD Model

A. Encoding Principle

Much like how the human brain has millions of neurons and synapses that activate upon input stimuli, HDC uses hypervectors to represent *any entities* in high dimensional space, or hyperspace. The hypervector is of a holistic representation [12] which distributes information equally over all its components. In the majority of related work, to map/encode raw values, e.g., features of training/testing samples, to hypervectors, they generate *base* hypervectors by randomly sampling each dimension from bipolar values {-1,1} [2] or Gaussian distribution $N(\mu, \sigma^2)$ for higher accuracy [4]. The reason HDC necessitates randomness and high dimensions is to achieve quasi-orthogonality. In other words, it assumes that different features are uncorrelated to each other, presenting near-zero similarity.[1] However, since the existing encoders do not touch the base hypervector once created, they cannot identify the accurate representation if different features are correlated.

Before explaining our encoder training method, we discuss key properties of the general encoding procedure, which we used to devise our proposed technique. Let $\vec{v}(\in \mathbb{R}^p)$ be a vector of scalars, $\langle v_1, \cdots, v_p \rangle$, to be encoded, and assume that a codebook $\mathbb{C}(\in \mathbb{R}^{p \times D})$ comprises the information of D-dimensional hypervector representations corresponding to each element, i.e., $\mathbb{C} = \langle \mathbf{C}_1, \cdots, \mathbf{C}_p \rangle$. The state-of-the-art encoding methods, e.g., random projection [2] and non-linear encoding [4], can be represented by

$$v_1 \otimes \mathbf{C}_1 \oplus v_2 \otimes \mathbf{C}_2 \oplus \cdots v_p \otimes \mathbf{C}_p$$

Here, $\otimes$ is the *binding* operation, which associates different information with element-wise multiplication, and $\oplus$ is the *bundling* operation to combine different information into a hypervector with element-wise addition. Then, the encoder usually applies an activation function, e.g., $cos(\cdot)$, $sign(\cdot)$, etc.

In principle, we can view the HD encoding as an *interdimensional mapping* across domains represented with different vector bases. In other words, it maps information in real coordinate space of p dimensions to another hyperspace, whose basis can be any set of hypervectors, $\mathbb{C}$. We define the interdimensional mapping function, $\mathbf{H} = \phi_{\mathbb{C}}^{p \to q}(\vec{v})$, as a procedure that transmits identical information stored in $\vec{v}$ into a hypervector, $\mathbf{H}(\in \mathbb{R}^q)$, by referring codewords in $\mathbb{C}$. For example, the original encoder maps raw features with a random codebook of $\mathbb{B}$; we can also map the error values in scalar vectors to the domain of class hypervectors with another encoding, $\phi_{\mathbb{K}}^{k \to D}(\vec{v})$, by using the codebook of the class hypervectors, $\mathbb{K}$. We utilize the HD encoding properties to translate the information of per-class scalar errors to the base hypervector errors.

B. HD Model Training

Algorithm 1 describes the steps of the training of TrainableHD. The TrainableHD algorithm first encodes the feature hypervector, $\mathbf{H}$, with the $sign(\cdot)$ function activation for binarization (Ⓐ). It then updates (i) the class hypervectors, $\mathbb{K}$, (Ⓑ) and (ii)

[1] We use the dot product, denoted with $\delta(\cdot)$, for the similarity measure.

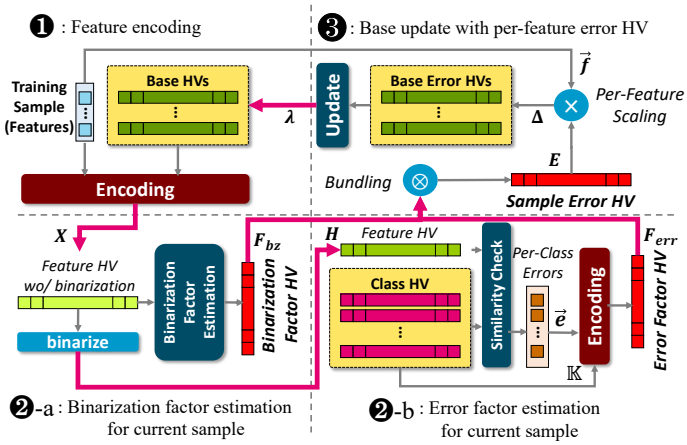


Fig. 2: An Illustration of Base Hypervector Training

the base hypervectors, $\mathbb{B}$ (❻). To train the class hypervectors, we compute the per-class errors, $\vec{e}$, using the similarities in hyperspace calculated by the dot product with $softmax(\cdot)$ normalization and the one-hot-encoding vector, $\vec{o}$, given by the ground-truth label. We then scale the feature hypervector for each class considering the amount of per-class error, i.e., $\vec{e} \times \mathbf{H}$, and adjust the class hypervectors with a learning rate, λ.

Algorithm 1 Training procedure of TrainableHD

1: **for** $\vec{f}$ in training datasets **do**
2: // ❹ **Encoding**
3: $\mathbf{X} \leftarrow \phi_{\mathbb{B}}^{f \rightarrow D}(\vec{f})$; $\mathbf{H} \leftarrow sign(\mathbf{X})$
4: // ❺ **Updating class HVs**
5: $\vec{s} \leftarrow softmax(\delta(\mathbf{H}, \mathbb{K}))$; $\vec{e} \leftarrow \vec{o} - \vec{s}$; $\Theta \leftarrow \vec{e} \times \mathbf{H}$
6: $\mathbb{K} \leftarrow \mathbb{K} \oplus (\lambda \times \Theta)$
7: // ❻ **Updating base HVs**
8: $\mathbf{F}_{bz} \leftarrow \mathbf{I} - tanh(\mathbf{X})^2$; $\mathbf{F}_{err} \leftarrow \phi_{\mathbb{K}}^{k \rightarrow D}(\vec{e})$
9: $\mathbf{E} \leftarrow \mathbf{F}_{err} \otimes \mathbf{F}_{bz}$; $\mathbf{\Delta} \leftarrow \vec{f} \times \mathbf{E}$
10: $\mathbb{B} \leftarrow \mathbb{B} \oplus (\lambda \times \mathbf{\Delta})$

Figure 2 illustrates how TrainableHD updates the base hypervectors at the next stage. Our goal is to translate the information of per-class errors to the per-feature errors due to the incorrectness of the base hypervector, $\mathbb{B}$. TrainableHD accomplishes this with two steps: (i) encoding the sample-wise error in a hypervector, $\mathbf{E}$, called the *sample error hypervector*, (ii) estimating the per-feature error hypervector from $\mathbf{E}$.

Step 1: Encoding the sample-wise error in a hypervector: TrainableHD encodes the sample error hypervector, $\mathbf{E}$, which includes the hypervector-type information of *how much error occurs for a single sample*. The encoded hypervector, $\mathbf{X}$, eventually contributes to the scalar errors, $\vec{e}$, through two following computations: (i) the binarization, (i.e., due to $\mathbf{H} \leftarrow sign(\mathbf{X})$ in ❹ of Algorithm 1) and (ii) the discrepancy with the class hypervectors, (i.e., due to $\delta(\mathbf{H}, \mathbb{K})$ in ❺). TrainableHD represents the two *factors* in a form of hypervectors, $\mathbf{F}_{bz}$ and $\mathbf{F}_{err}$. Figure 2❷ illustrates how we compute each factor.

- (❷-a) The first factor due to the binarization, $\mathbf{F}_{bz}$, is computed by $\mathbf{I} - tanh(\mathbf{X})^2$ where I is a hypervector whose every element is 1 and $tanh(\cdot)$ is the hyperbolic tangent. Since the binarization function, $sign(\cdot)$, amplifies the hypervector element of $\mathbf{X}$ in the range of $[-1, +1]$, an element value closer to 0 may create higher errors in the prediction, where the impact of each element on the error is bound within the same range. We exploit the square of $\tanh(\cdot)$, which is suitable to explain these impacts of the hypervector elements on the errors.

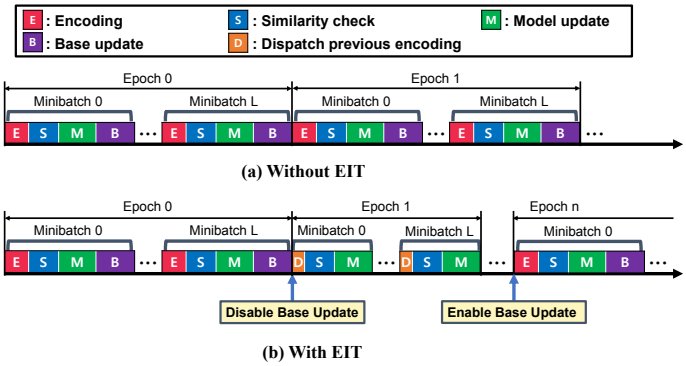


Fig. 3: An example of Encoder Interval Training

- (❷-b) Next, we compute the second factor due to the class hypervectors, $\mathbf{F}_{err}$. As discussed in Section IV-A, we can see the HD encoding as an interdimensional mapping. Thus, we encode the per-class error of $\vec{e}$ with the basis of $\mathbb{K}$ by $\mathbf{F}_{err} = \phi_{\mathbb{K}}^{k \rightarrow D}(\vec{e})$, meaning that $\mathbf{F}_{err}$ *bundles all per-class error hypervectors* scaled with the corresponding error value. With the two hypervector-encoded factors, TrainableHD can compute the sample error hypervector by associating(binding) the information through $\mathbf{E} \leftarrow \mathbf{F}_{err} \otimes \mathbf{F}_{bz}$.

Step 2: Estimating the per-feature errors in hypervectors: In this step (❸), TrainableHD generates *base error hypervectors*, denoted as $\mathbf{\Delta}$, which estimates per-feature errors that occur from each base hypervector. Using the sample error hypervector that represents the error for each sample, we distribute the per-sample error into the feature domain, assuming that higher feature values in the raw training sample contribute to higher error in hypervectors. It can be described by $\mathbf{\Delta} = \vec{f} \times \mathbf{E}$. Then, the base error hypervectors, $\mathbf{\Delta}$, has f hypervectors, each of which has the information of the amount of the per-feature error. We can finally update the base hypervectors by bundling the base error hypervector with the learning rate.

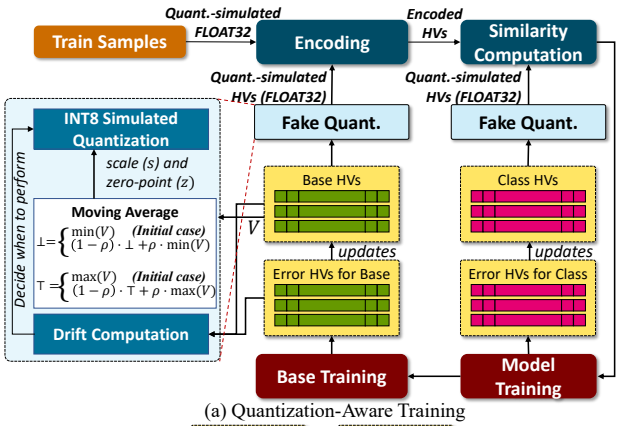
V. Optimized Implementation for Accelerators

A. Encoder Interval Training

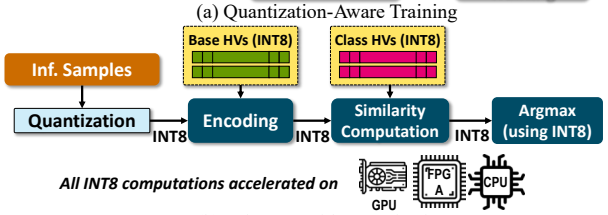
Encoding feature hypervectors with the updated base hypervector every time may add extra complexity to the existing HD learning. We address this issue with an optimization technique called *Encoder Interval Training* (EIT). EIT enables the reuse of feature hypervectors to reduce the encoding time of TrainableHD training. Figure 3 describes an example of the EIT. For the first iteration, we perform TrainableHD with encoding feature hypervectors. The EIT stores generated feature hypervector values into memory. For the next (n-1) iterations, where n is a hyperparameter that defines the EIT period, we perform the training using the previously stored feature hypervectors. Note that the base hypervectors keep updating during the reuse iterations, which have the capability of encoding better feature hypervectors; in our observation, the noise-tolerant nature of HD can compensate as long as we update the feature hypervectors when sufficiently changed. At every nth, we re-encode and perform HD model and base hypervector training. EIT is repeated until the training is terminated.

B. Acceleration with Quantization

The proposed trainable encoder exploits the floating-point base hypervectors for high accuracy in a similar way to the state-of-the-art HD encoders [4], even though the encoded hypervectors are binarized. To offer high efficiency for infer-



(a) Quantization-Aware Training



(b) Inference with Quantization

Fig. 4: TrainableHD Quantization

ence, TrainableHD employs quantization, which enables integer operation-only accelerator designs by completely eliminating the costly floating-point operations. We propose an optimized Quantization-Aware Training (QAT) method for HDC, which makes precision adjustments during training.

Training Quantized Hypervectors Figure 4a shows the TrainableHD learning procedure modified with QAT. To model the effects of quantization on the base hypervectors, the original input features and current base hypervectors are *fake-quantized*, i.e., it clamps and rounds their elements to produce an approximate version of the inputs in 8-bit integers (INT8) but stores them as the floating-point (FLOAT32) data type. We apply the fake quantization for the class hypervectors in the same manner. At the end of the training, we finally convert the trained base and class hypervectors with INT8 representations, so the inference can be performed completely on the INT8 domain as shown in Figure 4b.

One challenge to training in floating-point with quantization is how to set the quantization parameters for either the original inputs or hypervectors. We use a transformation method, known as *affine transformation*, which performs the quantization for x using two parameters, s and z as follows:

$$quant(x) = min(max(round(s \cdot x + z), 2^{b-1} - 1), -2^{b-1}))$$

where s is the scale factor, z is the value to be mapped to zero in the quantized form, and $b = 8$ for INT8. Since the upper and lower bounds of the input x are changed, e.g., due to the base and class hypervectors updates, we also learn the two parameters during the training iterations. To this end, we observe the *moving averages* of the minimum and maximum ($\perp$ and $\top$) for the given hypervectors to be quantized, as shown in Figure 4a. Here, we empirically selected the decay rate for the moving average by $\rho = 0.01$. Then, we can set $s = 2^b/(\top - \perp)$ and $z = \perp - round(-2^{b-1}/s)$, i.e., dividing the range of the moving averages, $[\perp, \top]$, equally into the quantized points.

QAT performance optimization To minimize the QAT costs mainly coming from the inserted fake quantization process, we propose an optimization technique called Drift-Aware Update (DAU), which decides when to perform the fake quantization. As discussed in Section IV-B, we update the base and class hypervectors using the hypervector-type errors with a learning rate, i.e., $\mathbf{\Theta}' = \lambda \times \mathbf{\Theta}$ and $\mathbf{\Delta}' = \lambda \times \mathbf{\Delta}$. Since HDC is robust to noise, a few differences in the base and hypervectors would not significantly affect the training results. We perform the fake quantization *only when* observing a large number of changes accumulated by $\mathbf{\Theta}'$ and $\mathbf{\Delta}'$, called *drift*. More formally, the proposed DAU optimization invokes the fake quantization procedure for the class hypervectors if $\sum_i |\mathbf{\Theta}'_i|/|\mathbf{K}| > \epsilon$ and for the base hypervectors if $\sum_j |\mathbf{\Delta}'_j|/|\mathbf{B}| > \epsilon$, which the fake quantization was skipped at the previous iteration i and j. In our current implementation, we conservatively set the threshold ϵ by 0.01 (1%) to guarantee the QAT to run with the correct simulation during most iterations.

TABLE I: Evalution Datasets

Name	Description	N_{train}	N_{test}	k	f
EMOTION [13]	Emotion recognition from ECG signal	1705	427	3	1500
FACEA [14]	Face recognition	22441	2494	2	512
FACE [14]	Face recognition	22441	2494	2	608
HACT [15]	Human activity recognition	7352	2947	6	1152
HEART [16]	MIT-BIH Arrhythmia dataset	119560	4000	5	187
ISOLET [15]	Voice recognition	6238	1559	26	617
MAR [17]	Plant classification	1440	160	100	64
MNIST [18]	Hand-written digit classification	60000	10000	10	784
PAMAP2 [19]	Physical activity monitoring dataset	16384	16384	5	27
SA12 [20]	Smartphone-based activity recognition	6213	1554	12	561
TEX [17]	Plant classification	1439	160	100	64
UCIHAR [15]	Human activity recognition	7352	2947	6	561

VI. Experimental Results

A. Experimental Setup

We have implemented the training procedure of the TrainableHD framework using PyTorch running on NVIDIA GeForce RTX 3090. The inference procedure was implemented on various acceleration platforms that support both floating-point and integer vector operations, including CPU (Intel Xeon Silver 4110), low-power GPU (Nvidia Jetson Xavier), and FPGA (Xilinx Zynq-7000). We measured the execution time and power consumption using Intel RAPL for CPU, Nvidia Nsight for GPU, and Xilinx Vitis toolkit for FPGA.

Implementation Methodologies To implement the accelerated inference on CPUs, we exploited the state-of-the-art library, Facebook's FBGEMM library, supporting optimized INT8 operations based on x86 SIMD instructions and multithreads. We expanded the FBGEMM library to support the sign function. since the base and class hypervectors are constant once deployed, we can reorganize the stored order of the hypermatrix elements in advance to make the memory access pattern fully sequential in GEMM operations. For the GPU acceleration, we use Tensor Cores in NVIDIA's Jetson Xavier. We extended the XCelHD [8], which is the CUDA impemetaion of HD computing, to accomplish the quantized execution of trainableHD. We mapped HD operations to cuBLAS APIs, and also implemented an in-place element update function for the intermediate results to support the sign function without uncoalesced memory accesses. The proposed quantization method enables the highly-accurate and efficient implementation on FPGA without using the resource-hungry DSP units unlike existing work [2]. We utilized the Xilinx Vitis framework to implement the GEMM and reduction operations on a systolic array structure that performs most computations using LUTs. Since the base and class hypervectors are invariant during inference, we load them into the buffer of the systolic array in advance and reuse them for multiple inputs without extra host communications.

Baselines and datasets We compare TrainableHD with (i)

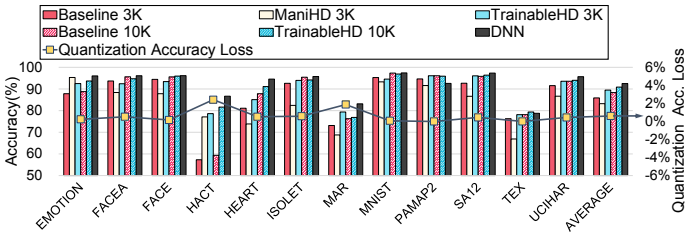


Fig. 5: Accuracy Comparison

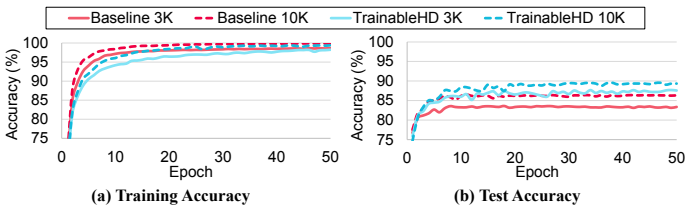


Fig. 6: Learning Accuracy Changes over Epochs

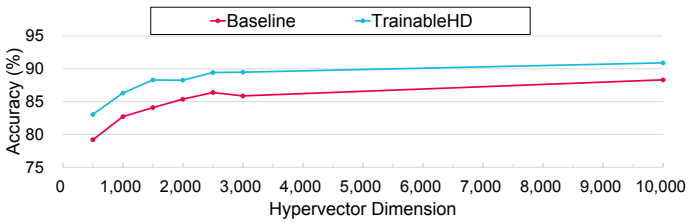


Fig. 7: Impact of Dimension Reduction

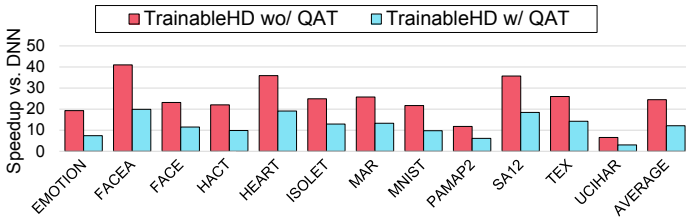


Fig. 8: Training Speedup of TrainableHD over DNN

the state-of-the-art HD learning method, which retrains class hypervectors using the static nonlinear encoder (Baseline) [4], (ii) ManiHD, which applied the manifold projection on HDC (ManiHD) [11], and (iii) the deep learning models optimized using Ray Tune to achieve the best accuracy (DNN) by performing the hyperparameter search, which have up to 64 batch sizes, 5 layers, and 512 neurons, trained for 50 epochs. To evaluate TrainableHD, Baseline, and ManiHD frameworks, we retrained each model with 50 epochs with $\lambda = 0.01$ and $n = 10$ emperically selected. Table I summarizes the details of the datasets, which include a wide range of practical problems.

B. Classification Accuracy of TrainableHD

Figure 5 presents the accuracy comparison results over different learning methods. For Baseline and TrainableHD, we measured under two different hypervector dimensions 3K and 10K. The result showed that (i) TrainableHD achieves higher accuracy levels than Baseline, one of the state-of-the-art encoding methods that use a nonlinear encoder, and (ii) the QAT has a very minor impact on the accuracy. For example, for HACT, the classification accuracy of Baseline is 57.28% and 59.35% for $D = 3,000$ and $D = 10,000$, respectively. In contrast, TrainableHD achieved 78.62% on $D = 3,000$ and 81.61% on $D = 10,000$. In the same dimension, TrainableHD achieved 3.62% and 2.58% higher accuracy on average as compared to Baseline with $D = 10,000$ and $D = 3,000$, respectively, which are comparable to the state-of-the-art deep learning models tuned for high accuracy. When comparing TrainableHD with ManiHD for $D = 3,000$, we observed that TrainableHD outperforms ManiHD with the exception of one dataset. On average, TrainableHD achieves 6.27% higher accuracy than ManiHD. However, ManiHD exploits the manifold projection, which is a non-negligible overhead during inference.

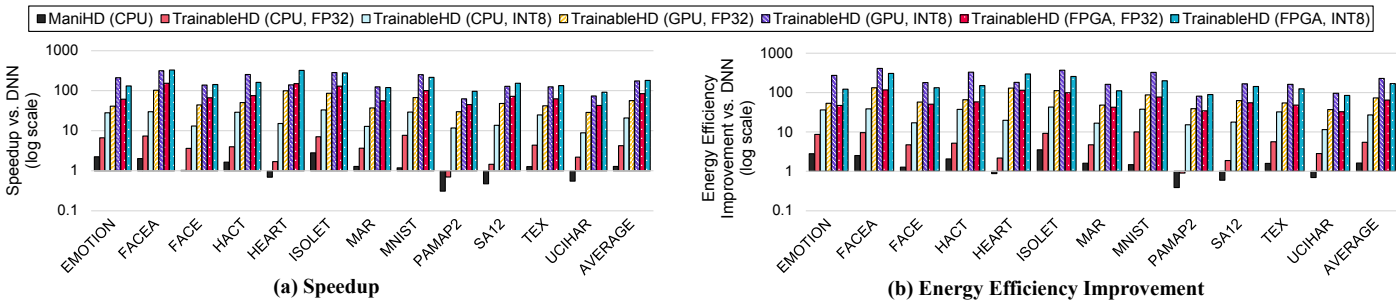
Learning Performance To understand why TrainableHD outperforms the state-of-the-art HD learning methods, we evaluate the training/testing accuracy changes over epochs. One of the important properties of HD learning is that it can achieve a sufficient amount of classification performance with a small number of epochs. As shown in Figure 6, the Baseline and TrainableHD provided a high degree of training accuracy within the first several epochs. Compared to Baseline, the training accuracy of TrainableHD is lower. However, it is because the Baseline tends to overfit the training datasets, whereas TrainableHD prevents such an issue in the earlier training phase, eventually achieving higher testing accuracy.

Dimension Reduction Figure 7 next shows the testing accuracy of TrainableHD and Baseline over different hypervector dimensions. We observe that TrainableHD is more robust to the dimension reduction. The accuracy of the Baseline at $D = 3,000$ is 85.87% which is comparable to the accuracy of TrainableHD at $D = 1,000$ which is 86.31%. Note that TrainableHD with $D = 3,000$ still outperforms the accuracy of BaselineHD with $D = 10,000$, implying that we can effectively reduce the dimension with minimal loss to accuracy.

C. Efficiency of TrainableHD

Training Efficiency We evaluated the training efficiency of TrainableHD as compared to DNNs. Figure 8 shows the comparison results using two varients of TrainableHD, one training without QAT and the other one with QAT, over the DNN model. The results show that TrainableHD without quantization improves training performance by $24.48\times$ on average. We can also enable quantization to yield INT8-quantized models, achieving higher efficiency when deployed for inference. The quantization simulation adds additional computation times, however, it still achieves $12.13\times$ speedup as compared to the DNNs.

Inference Efficiency Figure 10 summarizes the comparison results for speedup and energy efficiency improvements over the DNN inference on GPU. We use $D = 3,000$ that TrainableHD still shows higher quality than Baseline with $D = 10,000$ The results show that TrainableHD achieves a significantly high learning efficiency as compared to DNN. For example, when using the same GPU platform, TrainableHD is $56.4\times$ faster and $73\times$ more energy efficient even when not using quantization. Enabling quantization, TrainableHD additionally improves the performance by $3.1\times$ without accuracy loss. We also observe that the lightweight characteristics of TrainableHD enable high efficiency even on CPU for edge or cloud computing, assuming there are no on-device accelerators. TrainableHD with quantization provides $20.7\times$ faster than DNN on GPU. When using on-device FPGA units, it can offer $180.8\times$ higher speedup and $167.8\times$ better energy efficiency as compared to DNN. The computation costs during inference are the same as the Baseline as it does not add any extra computation procedure. TrainableHD with FP32 (INT8) also shows $3.3\times$ ($16.1\times$) better performance than ManiHD since ManiHD has non-negligible



(a) Speedup **(b) Energy Efficiency Improvement**

Fig. 9: Inference Comparison

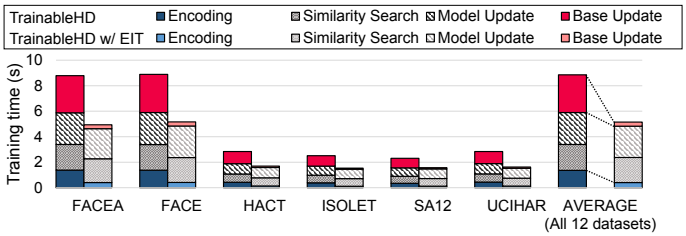


Fig. 10: Impact of Encoder Interval Training

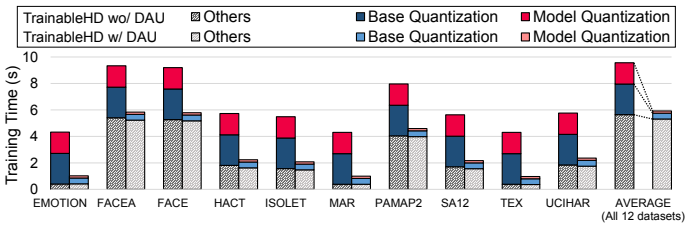


Fig. 11: Training Time Reduction of Drift-Aware Update

overheads to achieve higher accuracy due to the preprocessing CPU overhead of the manifold projection.

D. Impact of Optimization Techniques

Encoder Interval Training The EIT technique runs the encoder training on a regular basis. We measure the inference time without EIT and compare it with the case using EIT. Figure 10 shows the breakdown of training times for each case using the representative datasets. The EIT technique effectively reduces the training time to update the base hypervectors (labeled with Base Update) and the overhead of the repeated encoding procedure (labeled with Encoding). In our evaluation, the EIT technique reduces the training time for the base updates and encoding by 89.20% and 69.89%, respectively.

Drift-Aware Update for QAT Our second optimization technique is DAU (Section V-B), which performs the simulated quantization selectively only when sufficient changes happen in the hypervector updates. We verify the effect of DAU by comparing to the case of disabling the DAU, i.e., performing QAT and fake quantization for every update. As shown in Figure 11 showing the ten representative datasets, DAU significantly reduces the QAT overhead by 84.50% on average, which is essential for highly efficient inference. Note that we observed very minimal accuracy changes when using DAU, thanks to the holistic representation.

VII. Conclusion

In this paper, we proposed TrainableHD, an efficient and effective dynamic encoder for learning in HDC with the capability of the quantization. TrainableHD address the limiting factors in prior HD learning's training process, i.e., the static encoder, by continuously updating the encoder during the training process for better performance. Our evaluations show that TrainableHD improves the accuracy of the HDC by up to 22.26% (on average 3.62%) without any extra computation costs. It also achieves comparable accuracy to deep learning and is $56.4\times$ faster and $73\times$ more energy efficient on the NVIDIA Jetson Xavier.

References

[1] Yeseong Kim, et al. Cascadehd: Efficient many-class learning framework using hyperdimensional computing. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 775–780. IEEE, 2021.

[2] Mohsen Imani, et al. A framework for collaborative learning in secure high-dimensional space. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 435–446. IEEE, 2019.

[3] Yeseong Kim, et al. Efficient human activity recognition using hyperdimensional computing. In *Proceedings of the 8th International Conference on the Internet of Things*, pages 1–6, 2018.

[4] Mohsen Imani, et al. Dual: Acceleration of clustering algorithms using digital-based processing in-memory. In *53rd IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 356–371, 2020.

[5] Yang Ni, et al. Algorithm-hardware co-design for efficient brain-inspired hyperdimensional learning on edge. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 292–297. IEEE, 2022.

[6] Sohum Datta, et al. A programmable hyper-dimensional processor architecture for human-centric iot. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(3):439–452, 2019.

[7] Yeseong Kim, et al. Geniehd: Efficient dna pattern matching accelerator using hyperdimensional computing. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 115–120. IEEE, 2020.

[8] Jaeyoung Kang, et al. Xcelhd: An efficient gpu-powered hyperdimensional computing with parallelized training. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 220–225. IEEE, 2022.

[9] Denis Kleyko, et al. A survey on hyperdimensional computing aka vector symbolic architectures, part ii: Applications, cognitive models, and challenges. *arXiv preprint arXiv:2112.15424*, 2021.

[10] Zhuowen Zou, et al. Scalable edge-based hyperdimensional learning system with brain-like neural adaptation. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. ACM.

[11] Zhuowen Zou, et al. Manihd: Efficient hyper-dimensional learning using manifold trainable encoder. *DATE, IEEE*, 2021.

[12] Pentti Kanerva. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive computation*, 1(2):139–159, 2009.

[13] Jordan J Bird, et al. Mental emotional sentiment classification with an eeg-based brain-machine interface. In *Proceedings of theInternational Conference on Digital Image and Signal Processing (DISP'19)*, 2019.

[14] Yeseong Kim, et al. Orchard: Visual object recognition accelerator based on approximate in-memory processing. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 25–32, 2017.

[15] Dheeru Dua et al. UCI machine learning repository, 2017.

[16] Mohammad Kachuee, et al. Ecg heartbeat classification: A deep transferable representation. In *2018 IEEE International Conference on Healthcare Informatics (ICHI)*, pages 443–444. IEEE, 2018.

[17] Charles Mallah, et al. Plant leaf classification using probabilistic integration of shape, texture and margin features. *Pattern Recognit. Appl.*, 3842, 02 2013.

[18] Yann LeCun, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[19] Attila Reiss et al. Introducing a new benchmarked dataset for activity monitoring. In *2012 16th international symposium on wearable computers*, pages 108–109. IEEE, 2012.

[20] Jorge-L Reyes-Ortiz, et al. Transition-aware human activity recognition using smartphones. *Neurocomputing*, 171:754–767, 2016.