CascadeHD: Efficient Many-Class Learning Framework Using Hyperdimensional Computing

Yeseong Kim
DGIST
yeseongkim@dgist.ac.kr

Jiseung Kim
DGIST
js980408@dgist.ac.kr

Mohsen Imani
UC Irvine
m.imani@uci.edu

***Abstract*—The brain-inspired hyperdimensional computing (HDC) gains attention as a light-weight and extremely parallelizable learning solution alternative to deep neural networks. Prior research shows the effectiveness of HDC-based learning on less powerful systems such as edge computing devices. However, the many-class classification problem is beyond the focus of mainstream HDC research; the existing HDC would not provide sufficient quality and efficiency due to its coarse-grained training. In this paper, we propose an efficient many-class learning framework, called CascadeHD, which identifies latent high-dimensional patterns of many classes holistically while learning a hierarchical inference structure using a novel meta-learning algorithm for high efficiency. Our evaluation conducted on the NVIDIA Jetson device family shows that CascadeHD improves the accuracy for many-class classification by up to 18% while achieving 32% speedup compared to the existing HDC.**

***Index Terms*—Hyperdimensional Computing, Many-class classification, Edge Computing**

I. Introduction

The introduction of edge computing has created a new computing paradigm that pushes various workloads closer to the location where target applications are needed [1]. Artificial intelligence is one of the candidates expected to be deployed into the edge devices. However, state-of-the-art machine learning (ML) algorithms, including deep neural networks (DNN), require a large amount of computing power and memory resources to provide better service quality. Besides, the types of ML problems in IoT-edge practice have become more challenging. Many-class classification is a representative problem [2]. Since the many-class classification tasks often involve a complex problem space, the DNN models need deeper and more complicated model architectures with large sizes, additionally burdening both computational workloads and resource requirements.

On the one hand, brain-inspired hyperdimensional computing (HDC) arises as an attractive alternative solution to enable lightweight learning on such less-powerful computing platforms [3]. This computing paradigm is based on sparse distributed memory model researched in neuroscience [4], [5], which is inspired by the fact that the brain operates by computing patterns rather than numbers the current computer uses. Thereby, the HDC mimics the human memory cognition with random high-dimensional vectors, e.g., $D = 10,000$ dimensions, called *hypervectors* in short, which represents patterns of information in a hyperspace. For the classification problems, we first map each training sample to a point in the high-dimensional space. The HDC-based learning trains different *class hypervectors*, each of which represents a per-class universal pattern in training data points as an ML model. During the inference, we can select the class that shows the highest similarity by measuring cosine distances between each class hypervector and given data mapped into the high-dimensional space. Many prior pieces of research solve practical application problems using HDC, e.g., language recognition [6], human activity recognition [7], and DNA pattern matching [8], robotics [9], showing high accuracy comparable to deep learning with superior efficiency. The highly parallelizable nature of HDC also enables various optimized platform designs, which can perform learning tasks in a more lightweight fashion, including low-power GPU and ASIC [10].

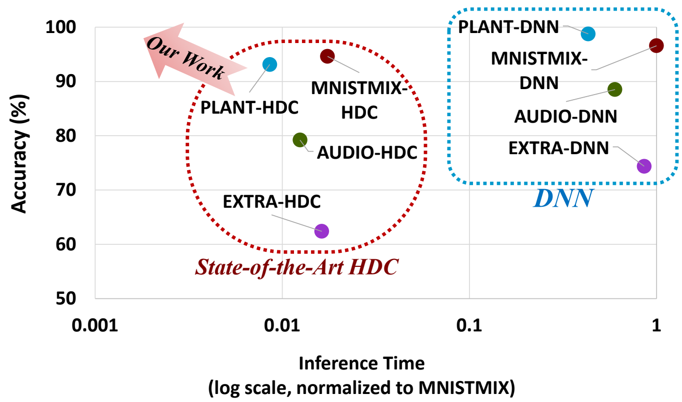


Fig. 1. Inference time and accuracy comparison of state-of-the-art HDC over DNN for many-class problems. All results are evaluated on NVIDIA Jetson Xavier device. We learn the HDC models following the training configuration in [10] with the HDC encoder [12]; the DNN models are selected by AutoKeras [13] which automates hyperparameter search to identify the optimal DNN model structure. Our work improves both accuracy and efficiency.

Although the collection of research works exhibits that the HDC can be a promising solution for the edge computing environment, the number of classes of the ML problems that prior studies have focused is usually small, e.g., less than $K = 10$. We have witnessed that the state-of-the-art HDC algorithms would not provide sufficient quality and efficiency for many-class classification problems. Figure 1 shows our preliminary evaluation, which compares the accuracy and energy efficiency of the HDC-based learning with deep learning. The results show that the accuracy of HDC is significantly degraded for many class problems, although it may improve the inference performance. For example, for Google's Audioset dataset ($K = 74$) [11], the HDC-based learning loses around the accuracy of 9.3% as compared to the DNN model.

A root cause is that the current HDC-based learning fails to distinguish confusing classes due to the intensified difficulty in identifying precise decision boundaries between class hypervectors when the number of classes becomes larger. Also, when the learning problem has many classes, the HDC inference needs to compute the distances from all the class hypervectors, degrading performance and energy efficiency.

This paper revisits the state-of-the-art HDC learning technique to address many-class classification for high efficiency and accuracy. We propose a novel efficient many-class learning framework, called CascadeHD, tailoring two orthogonal technical solutions. The first solution, called *MAny class Similarity Scaling (MASS)*, revises the iterative hypervector fine-tuning procedure to consider interdependency of different classes, which the current HDC techniques neglect. It significantly increases the classification accuracy, e.g., by more than 10% for the PLANT dataset. To improve inference performance when having many classes, we propose another meta-learning solution, dubbed *waterfall* training, which builds a cascade tree structure of the class hypervectors. It automatically identifies confusing classes from the learned hypervector model and dataset, and creates a hierarchical model by merging similar class hypervectors as a decision node. It reduces the number of similarity comparisons during the inference, improving the inference throughputs and achieving high efficiency.

We evaluate the proposed CascadeHD with various many-class problems on the NVIDIA Jetson device family equipping low-power GPU suitable for edge computing. Our evaluation results show that CascadeHD significantly outperforms classification accuracy compared to the state-of-the-art HDC algorithm, providing comparable accuracy to the DNN. We also observe that CascadeHD combining the MASS and waterfall training improves the energy efficiency (performance) by 21.3% (22.4%) and 76.0 $\times$ (65.6 $\times$) over the prior HDC and DNN inference, respectively.

II. CascadeHD Classification Overview

Figure 2 shows the outline of the CascadeHD classification learning procedure. The first step is to map the original training samples into the points in the high-dimensional space, i.e., hypervectors. During the training, we identify the common element-wise patterns included in the hypervectors for each class. To this end, we perform arithmetic operations with hypervectors, e.g., combining hypervectors using element-wise addition and suppressing non-crucial patterns using subtraction. Here, we propose a new model training method, dubbed *MASS*, which significantly increases the prediction accuracy for many-class classification problems. Once trained the model, i.e., *class hypervectors* representing each class, we can perform the inference by computing the similarities. We convert the trained class hypervector model to a hierarchical classification procedure using a new meta-learning method, called the *waterfall training*. Given a query hypervector encoded for a testing sample, we compute the cosine similarity between each class hypervector and the query hypervector and find the most similar class throughout the hierarchical waterfall model. In the rest of the section, we describe the CascadeHD learning workflow with the discussion of the key highlights as compared to the prior HDC-based learning algorithms.

A. Encoding

The goal of the encoding is to map a data point in the original space of the training data into another high-dimensional space while preserving the similarities of the data point across different spaces. The property of the high-dimensional space makes the classification training workflow efficient, e.g., relatively simple linear separation is much highly accurate than the linear classification in the original space. Many different encoding methods have been proposed; our CascadeHD work exploits the non-linear encoding method recently proposed in [12].

Let us assume that a feature vector in the training/testing data is given by $\mathbf{F} = \langle f_1, \cdots, f_F \rangle$ where F is the number of the features of the dataset. To encode the feature vector offline, we prepare F *base hypervectors*, $\mathbf{B_i}$, whose D elements are sampled from the normal distribution, where D is the size of the target high-dimensional space. We multiply F_i by B_i and then combine the results using the element-wise addition to create a single hypervector representation, including non-linear feature interactions. Finally, we apply the cosine function for each hypervector element to add a high degree of non-linearity for the response values. In the HDC, the binarized hypervectors are often preferred due to their high efficiency in the computation. In this case, we can take the sign bit of the cosine results, i.e., assigning all positive elements to +1 and zero/negative elements to -1. This encoding method can explicitly excavate non-linear latent interactions between different features in the original space. In our experiments, it significantly outperforms other existing encoding methods such as ID-Level encoding [6]. However, even with this encoding, HDC often does not achieve high enough accuracy for many-class classification problems. It leads us to develop a new training method.

B. Class Hypervector Training

The very early research in HDC uses a single-pass training to create the class hypervectors [14]. The underlying assumption is that averaging different vectors of training samples may extract the common patterns for each class. Thus, the single-pass training is to combine all the encoded hypervectors for each class using the element-wise addition: $\mathbf{C}_k = \sum_i \mathbf{H}_i^k$ where $\mathbf{C}_k$ is the class hypervector for k-th class, and $\mathbf{H}_i^k$ is the encoded hypervector for the i-th sample in k-th class. To achieve higher accuracy, we need to refine the class hypervectors through an iterative procedure. The iterative procedure is often referred to, *retraining*, which was originally proposed in [15] for the first time. For a training sample, $\mathbf{H}' = \mathbf{H}_i^k$, it checks whether $\mathbf{H}'$ has the highest cosine similarity with the k-th class hypervector. The class hypervectors are updated only when it misclassifies, by $\mathbf{C}_k += \mathbf{H}'$ and $\mathbf{C}_w -= \mathbf{H}'$ where w is the misclassified class index in the similarity computation.

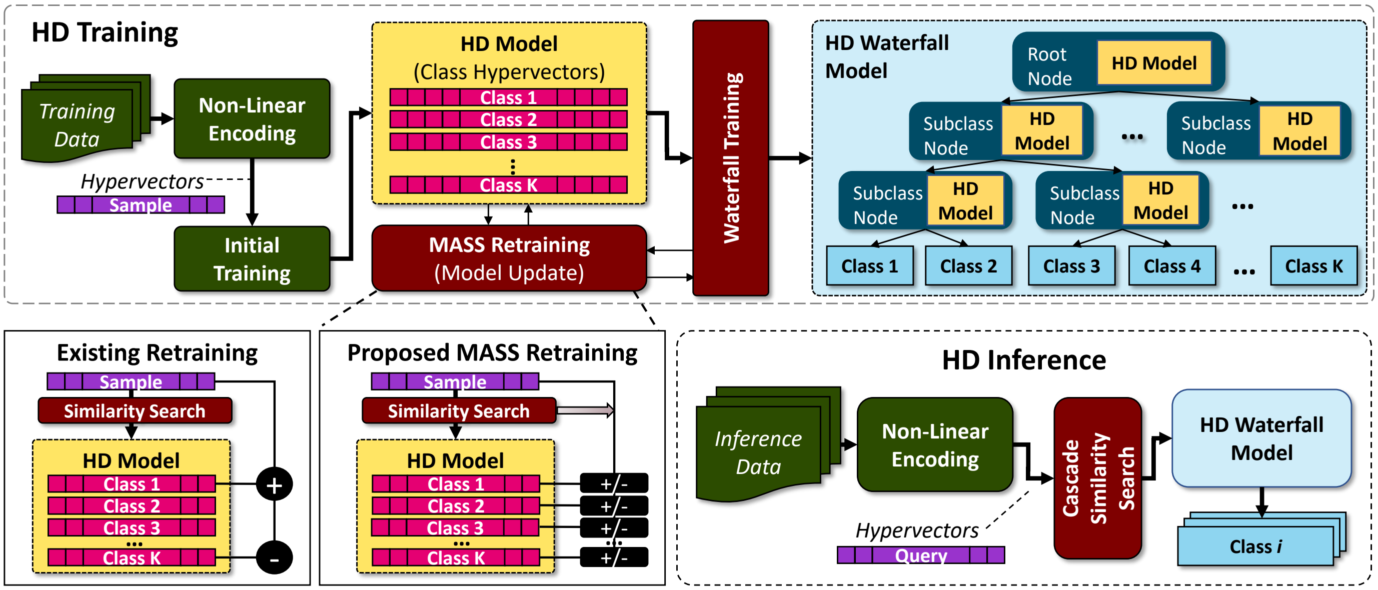
Since the retraining procedure is proposed in [15], most research works have exploited it without many notable changes over the last several years as a standardized procedure. However, we observe this procedure can train the class hypervectors in a suboptimal way, and the issue is more significant for the many-class problems. A crucial drawback of this approach is that it updates only the two hypervectors for the misclassification cases. Since the many-class classification problem has a much larger number of classes than two, this retraining procedure significantly slows down the training time and often does not avoid the local optima issue. Another complication is that, once the model predicts the right classes of all the training samples, we can not update the model anymore. It happens even though it does not create sufficient margins between class hypervectors, which are more required in the many-class problems to avoid confusion between classes.

As shown in Figure 2b, CascadeHD addresses this issue by updating the class hypervectors gradually based on the computed cosine similarities. The higher difference from the target similarity, the higher updates happen in the hypervectors. Our proposed method updates all the class hypervectors simultaneously, guaranteeing high convergence for many-class problems. Besides, instead of updating the model only when observing misclassified training samples, CascadeHD updates the model every time. We call this training technique as *MAny class Similarity Scaling (MASS)*. We elaborate the details in Section III-A.

C. Cascade Inference

Once the model is trained, we can perform the inference by measuring the cosine similarity from each class hypervector and a query hypervector encoded. As shown in Figure 2c, when many classes are compared, the exhaustive similarity calculation requires high computation costs. CascadeHD addresses this issue by training a hierarchical comparison flow at offline, inspired by the hierarchical DNN learning approach shown in [16]. We train the inference tree using a new meta-learning technique called *waterfall training*. With a trained class hypervector model, this technique automatically identifies similar classes, i.e., which would be confusing during the

(a) Overview of CascadeHD (Training)



(b) HD Retraining Method Comparison

(c) CascadeHD Inference Procedure

Fig. 2. CascadeHD Overview

inference, and then merges them into a latent category, creating a new decision node. By repeating this procedure, we can build the tree structure, which routes samples with challenging classes to the new decision models, focusing more on confusing classes. We perform the inference procedure from the root node with the trained tree model until it reaches the leaf node representing a single class. The proposed cascading inference provides two primary benefits. First, since the new decision node performs a binary classification mostly for confusing classes, which are relatively easier than multi-class problems, it could improve the prediction accuracy. Second, the multi-level cascading procedure reduces the number of similarity computations, i.e., calculating similarities for the merged categories rather than class hypervectors of all the original classes, accelerating the inference performance. We explain the details of the waterfall training in Section III-B.

III. CascadeHD Learning Techniques

CascadeHD employs two learning techniques, MASS and waterfall training, on top of the conventional HDC-based learning workflows. MASS is the new retraining procedure that replaces the conventional retraining procedure; the waterfall training is a meta-learning algorithm, which converts the flat list of class hypervectors into a hierarchical inference structure. The two techniques address the issues of many-class classification problems regarding both the prediction quality and implementation efficiency.

A. Many-class Similarity Scaling (MASS)

Suppose we have the class hypervectors, C_k, for a K-class problem using the single-pass training. The goal of the MASS training is (i) to measure the distances in the cosine similarity (denoting by δ) from each class hypervector for a given training sample, i.e., $\delta(\mathbf{C}_k, \mathbf{H}')$, and (ii) update each class hypervector to reach the target similarity. In HDC, two hypervector are referred to be completely dissimilar if they are orthogonal [5], i.e., the cosine similarity should be zero for completely different data points. In contrast, if two hypervectors are the same, the cosine similarity should have one. Thus, we can define the vector, which represents the target similarities, using one-hot encoding. For example, if a training sample is in the k-th class, the

Algorithm 1: MASS retraining algorithm

$\mathbf{M} = [\mathbf{C}_0 \cdots \mathbf{C}_{K-1}]^\intercal$
for $\mathbf{X} = [\mathbf{H}_i^{k_1} \cdots \mathbf{H}_{i+B-1}^{k_B}]^\intercal$ *in the training dataset* **do**
  $\mathbf{S} = \delta(\mathbf{M}, \mathbf{X})$
  $\mathbf{Y}_{code} \leftarrow$ *one_hot_encoding(*$\mathbf{K} = \{k_1, \cdots, k_B\}$*)*
  $\mathbf{S}' \leftarrow normalize(ReLU(\mathbf{S}))$
  $\mathbf{U} = \mathbf{Y}_{code} - \mathbf{S}'$
  $\mathbf{M} = \mathbf{M} + \lambda \mathbf{U}^\intercal \mathbf{X}$
end

target similarity vector is $\langle 0, \cdots, 1, \cdots, 0 \rangle$ where the single 1 is set at the k-th index. The MASS retraining updates the class hypervectors so that it could yield the cosine similarity values closer to the target similarity vector in the next inference.

Algorithm 1 shows the complete procedure of the MASS training for a single iteration, where $\mathbf{M}$ is the $K \times D$ matrix that includes the class hypervectors, $\mathbf{X}$ is a B-size batch of the encoded hypervectors for training samples, i.e., $B \times D$ matrix, and $\mathbf{S}$ is the $B \times K$ matrix of the cosine similarity values for all pairs of the class hypervectors and encoded hypervector. We then compute $\mathbf{Y}_{code}$, which is composed of the target similarity vectors for each sample class (label) using the one-hot encoding. After taking the ReLU function to ignore negative similarity values, which rarely happens due to the random nature of HDC but would impact the computation stability, we perform min-max normalization to update the class hypervectors with sufficient class-wise discrimination. Lastly, we add the batched hypervectors to the model $\mathbf{M}$ with the calculated degree of the updates in $\mathbf{U}$, i.e., so that the higher difference from the target similarity, the larger addition or subtraction happens in the model updates. For example, since misclassified samples are likely to have a higher difference to the target similarity vector, a higher amount of changes will happen for the class hypervector model. Here, we update the hypervector with a learning rate, λ, to control the iterative retraining speed. Note that, unlike the conventional retraining procedure, it updates all class hypervectors every time considering the relative distance in the cosine similarity, achieving higher accuracy even for the many class problem with a guarantee of faster convergence.

B. Waterfall Training

The second learning technique used in CascadeHD is the waterfall training, which reduces the number of similarity computations required in the inference. Figure 3 shows how CascadeHD builds the hierarchical model from the class hypervectors. Before the waterfall training procedure, we first split the training data samples into two groups, the *training subset* and *validation subset* following the standard ML practice. We employ a bottom-up approach to learn the underlying relationship among the many classes automatically. The initial class hypervectors are trained with the *training subset* using the MASS training procedure, yielding K class hypervectors, each of which corresponds to each class representation (A). We can regard the class hypervectors as a two-level tree, where the classification performs with the in-node model at the root node. We then evaluate the model with the validation subsets to identify two similar classes, i.e., which were confusing on the validation subset (B). Suppose we obtain a confusion matrix $\mathbf{F}$ from the evaluation with the validation dataset. We set the diagonal entries of $\mathbf{F}$ by zero, and make it symmetric by $\mathbf{D} = \frac{1}{2}(\mathbf{F}+\mathbf{F}^{\intercal})$. The entry $\mathbf{D}_{ij}$ measures how difficult it is to discriminate the class (or category) i and j due to the class hypervector similarity. We select the entry, saying the class p and q with the highest value, and combine the two classes using the element-wise addition. It creates a merged hypervector, which represents a new decision node for a latent category. To classify the samples routed to the decision node, we also train a new *in-node* binary classification model for the merged two classes using the MASS training (C).

We repeatedly apply this procedure to create many hierarchical categories. We observe that the waterfall training may increase the accuracy for the validation set in the early merging stage by performing the fine-grained in-node classification for confusing classes. It also keeps the accuracy when combining similar-enough classes, but a significant accuracy drop may occur when merging too distinct classes. Thus, we stop the procedure when we observe an accuracy drop exceeding a threshold value, τ (D).

Training Proxy Decision Node: An issue of the hierarchical classification model is that the accuracy highly depends on the decision nodes close to the root, as the mistake that happens at higher-level nodes cannot be corrected once it routes a sample into an incorrect category. We resolve this issue by allowing some nodes, called *proxy decision nodes*, to classify more than two classes. We select the proxy decision nodes by evaluating the validation set again after learning the entire tree structure. In this evaluation stage, we count how many samples are incorrectly routed and select the node that routes a specific class with the highest error rate. For the selected proxy node, we retrain its in-node classification model with a new additional class that was likely to be misclassified. This procedure is also repeated until either (i) it selects P proxy nodes where P is the user-configurable parameter and (ii) the accuracy drop exceeds the threshold value, τ, with an added proxy node.

IV. Experimental Results

A. Experimental Setup

We have implemented the proposed CascadeHD using PyTorch 1.6. We measured the performance and energy consumption of different learning algorithms on NVIDIA Jetson TX2 and Xavier, targeting edge computing platforms. The power consumption of the GPU devices is collected using the jtop tool. We compare CascadeHD with (i) the state-of-the-art HDC learning method, which uses the existing retraining mechanism based on the same PyTorch platform, and (ii) the DNN models whose hyperparameters are optimized using AutoKeras [13] to get the best accuracy. We tested 100 different DNN models for the hyperparameter search, which have up to three layers and 1024 neurons, trained for 100 epochs. To evaluate the HDC retraining procedure, we trained each model with 30 epochs. During the waterfall modeling, we used the training dataset of 20% as the validation subset. Table I summarizes datasets used in our evaluation. We include three many-class problems, PLANT, MNISTMIX, and AUDIO, along with two medium-size-class problems, EXTRA and MNISTMIX, to verify the generality of CascadeHD.

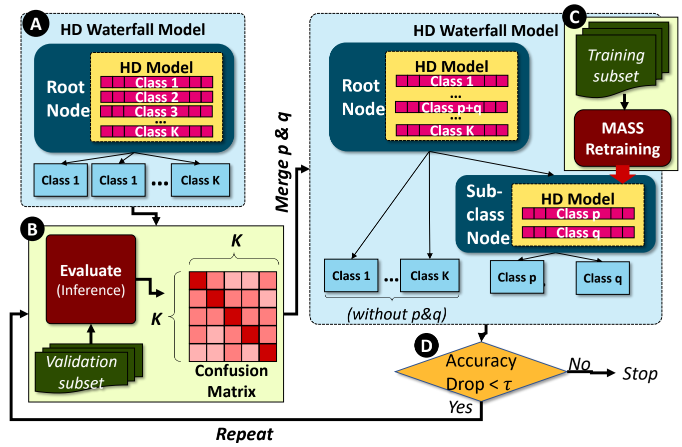


Fig. 3. Waterfall Training

TABLE I
Evalution Datasets

Name	Description	N_{train}	N_{test}	K	F
PLANT [17]	Plant classification	1440	160	100	64
EXTRA [18]	Human activity recognition	12332	5286	26	226
ISOLET [17]	Voice recognition	6238	1559	26	617
MNISTMIX [19]	Multi-language classification	52720	6000	100	784
AUDIO [11]	Google's audioset classification	14802	1645	74	1280

B. Accuracy of MASS Retraining

We first evaluate the accuracy of the CascadeHD method, focusing on the MASS retraining procedure. We compare CascadeHD with the state-of-the-art HDC learning, which uses the two-class update retraining method, and the DNN models, which are optimized for accuracy using AutoKeras. Figure 4 presents our evaluation results of the top-5 and top-1 accuracy for each dataset. The results show that the CascadeHD, which uses the mass retraining, achieves comparable accuracy to the highly-optimized DNN models. CascadeHD also significantly outperforms the state-of-the-art HDC-based learning. For example, for the AUDIO dataset, the baseline HDC learning loses 14.4% accuracy compared to the DNN model; however, the MASS retraining procedure effectively addresses the issue, offering the same level of accuracy. In summary, CascadeHD achieves highly comparable accuracy to the DNN models, only with a loss of 0.78% for the top-5 accuracy and 1.36% for the top-1 accuracy.

To better understand how the proposed MASS retraining behaves differently from the existing HDC learning method, we conducted a detailed analysis of the updated class hypervectors. As discussed in III-A, the goal of the retraining procedure is to update the class hypervectors to obtain similarity values close to the target similarity, i.e., Y_{code}. In this experiment, we measure the differences between the computed similarity and target similarity for all training samples over 30 training epochs. Figure 5 summarizes the results for three representative datasets. We observe that the existing retraining method is in general prolonged during the training, e.g., PLANT, since it updates only two classes for each sample in an epoch. The results also show that the updating procedure often diverges, e.g., MNISTMIX,

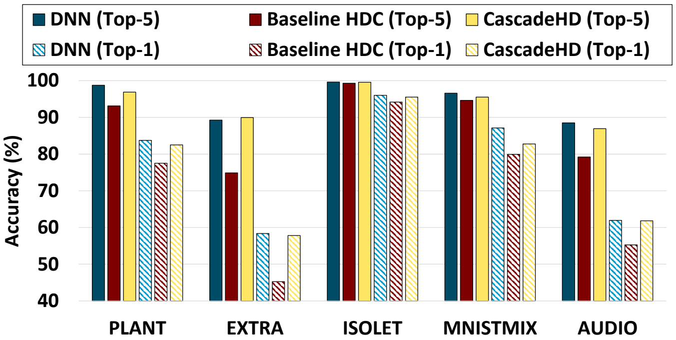


Fig. 4. Accuracy Comparison. We ran CascadeHD without the waterfall modeling to evaluate the impact of the MASS training procedure.

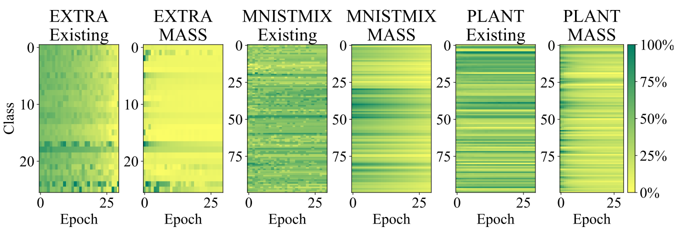


Fig. 5. Changes of distances to the target similarity over 30 training epochs. The distances are normalized for each case. The lower is, the better. "Existing" is the results obtained by the state-of-the-art, prior retraining method; "MASS" is the proposed method.

or fails to train the model throughout the epochs, e.g., PLANT. In contrast, the proposed MASS training procedure gradually updates all the class hypervectors based on the similarity differences, achieving appropriate convergence to the target similarity.

C. Efficiency of Waterfall Model

The proposed waterfall modeling method creates the hierarchical classification models mainly to optimize inference efficiency. Figure 6 summarizes the speedup and energy efficiency improvements of the baseline state-of-the-art HDC and CascadeHD without using the proxy nodes as compared to the DNN models. We observe that the proposed CascadeHD achieves significant speedup and energy efficiency improvement over the DNN approaches. For example, compared to the DNN inference running on TensorFlow, CascadeHD provides 61.4×/65.6× speedup and 69.7×/76.0× energy efficiency improvement on NVIDIA Jetson TX2/Xavier. The results also show that the hierarchical model structures effectively optimize the runtime and energy consumption of the HDC-based inference. It is because the baseline HDC has to compute the cosine similarity values between all K class hypervectors and each sample, whereas CascadeHD reduces the number of similarity computations with the tree-based model. We observe that the waterfall model-based inference reduces the runtime by 22.4% on average while achieving an energy consumption saving of 21.3%.

In CascadeHD, we can achieve higher efficiency by turning the hyperparameter value, τ, which is the threshold used for stopping the node merging process. Figure 7 summarizes the accuracy and efficiency tradeoff. The results show that CascadeHD could provide higher efficiency with a small amount of accuracy loss by building a deeper and wider tree-based model. For example, when $\tau = 3\%$, CascadeHD further optimizes the inference time by 40.5% at the cost of accuracy loss of 3.7%.

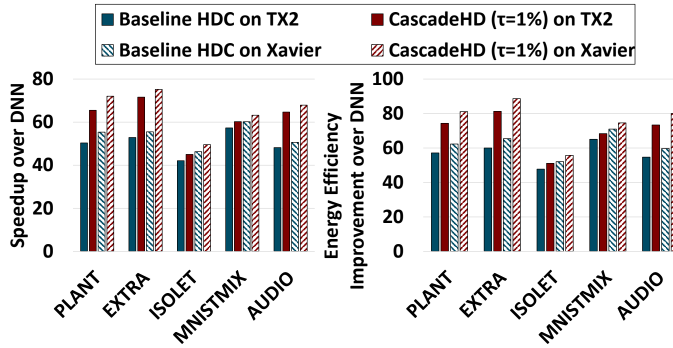


Fig. 6. Efficiency Comparison. All results show the improvements over the DNN models generated with AutoKeras running on Tensorflow 2.3.1. The results are evaluated using $\tau = 1\%$ to minimize the accuracy loss due to the cascading inference procedure.

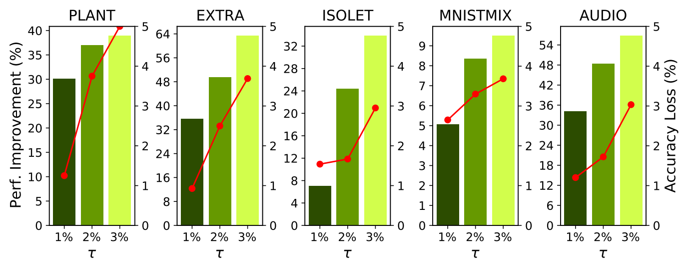


Fig. 7. Efficiency and accuracy tradeoff. Each bar shows the performance improvements over the baseline HDC; the red lines present the accuracy loss changes over different τ values. The inference time is measured on Jetson TX2.

D. Impacts of Waterfall Modeling on Accuracy

The inference efficiency improvement stems from the hierarchical model structure with a slight degradation of the accuracy. In this section, we show a detailed analysis of accuracy during the waterfall modeling procedure.

Accuracy changes during the waterfall training Figure 8 shows how the accuracy changes when merging a different number of decision nodes. We observe that the waterfall training can increase the accuracy at the beginning by generating the decision nodes that focus on confusing classes with relatively easier binary classifications. This characteristic thereby gives us a choice to create a model with fewer merging nodes if the goal is to increase the prediction quality even if it does not provide a significant performance improvement.

Proxy Decision CascadeHD has an additional configurable parameters, P, which select P decision nodes to give a second chance when a mistake happens during the cascading inference, as discussed

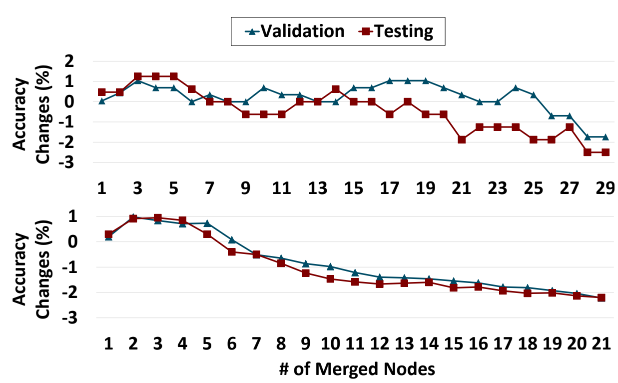


Fig. 8. Efficiency and Accuracy Tradeoff

TABLE II
IMPACTS OF THE PROXY NODES ON THE ACCURACY AND PERFORMANCE. ALL EXPERIMENTS ARE REPORTED WHEN $\tau = 2\%$. THE OVERHEAD INDICATES HOW MUCH IT LOSES THE PERFORMANCE IMPROVEMENT OVER THE CASE THAT DOES NOT USE THE PROXY NODE.

		PLANT	EXTRA	ISOLET	MNISTMIX	AUDIO
Accuracy Gain (%)	P=5	0.63%	0.73%	1.73%	0%	0.99%
	P=10	0.63%	1.30%	N/A	0.02%	1.87%
	P=15	1.25%	N/A	N/A	0.05%	N/A
Max. Overhead		2.70%	$<$1%	$<$1%	3.30%	2%

in Section III-B. Since increasing P to a very large number may degrade performance, we empirically select $P \leq 15$, which shows a marginal impact on the performance overhead for the all tested dataset. Our experiments are summarized in Tabel II[1]. The results show that, for some datasets, the technique which uses proxy nodes may compensate the accuracy loss incurred by the waterfall model training. For example, by changing ten nodes to perform the multi-class classification for AUDIO, it increases accuracy by 1.87%, which achieves the same accuracy level as the model using the MASS retraining procedure. Note that even in this case, we can expect efficiency improvements by 48% for this dataset, as compared to the baseline HDC.

V. Related Work

Hyperdimensional computing (HDC) is a brain-inspired computing approach that enables lightweight learning based on theoretical neuroscience [5]. To solve classification tasks, HDC maps training data to points in a high-dimensional space and performs learning by combining the points, creating the class hypervectors, which include binary or non-binary patterns commonly observed for each of the learning classes. Compared to DNN, HDC has several superb properties, including fast training and parallelizable operations suitable for specialized hardware design such as ASIC, FPGA, and GPGPU [10].

Through many endeavors over recent several years, the HD research community improves the classification accuracy and efficiency of HDC-based learning, e.g., developing different encoding methods [12] and hardware-level optimizations by trading off quality and system cost [20]. However, since the work in [15], which proposed an iterative training method of class hypervectors, most works have relied on the iterative procedure as a standardized procedure without much improvement. Unfortunately, the iterative procedure often does not effectively handle the many-class classification problems, which are beyond the scope of the HDC research mainstream. In this paper, we discussed the limitation of the existing training method and proposed a new training procedure that significantly increases the accuracy along with improved efficiency in classifying the many-class learning tasks.

VI. Conclusion

In this paper, we propose CascadeHD, which improves the accuracy and efficiency of the HDC-based learning for many-class classification problems. The two key enablers of CascadeHD are the mass training and waterfall training, which respectively improve the accuracy of the HDC comparable to the DNN models, and enhances the performance and energy efficiency by building a hierarchical classification structure. Our evaluation results show that CascadeHD can achieve 65.6× speedup and 76.0× energy efficiency improvement on NVIDIA Jetson Xavier device with a minimal top-5 accuracy loss of 0.78% as compared to the DNN algorithm.

[1]The table includes N/A fields since CascadeHD stops adding more proxy nodes due to the accuracy drop larger than τ with the P setting.

Acknowledgment

This work was partially supported by Semiconductor Research Corporation (SRC) Task No. 2988.001 and Department of the Navy, Office of Naval Research, grant # N000142112225.

References

[1] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[2] F. Abramovich and M. Pensky, "Classification with many classes: Challenges and pluses," *Journal of Multivariate Analysis*, vol. 174, p. 104536, 2019.

[3] G. Karunaratne, M. Le Gallo, G. Cherubini, L. Benini, A. Rahimi, and A. Sebastian, "In-memory hyperdimensional computing," *Nature Electronics*, pp. 1–11, 2020.

[4] P. Kanerva, *Sparse distributed memory*. MIT press, 1988.

[5] ——, "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] 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 *2016 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2016, pp. 16–1.

[7] 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*, 2018, pp. 1–6.

[8] Y. Kim, M. Imani, N. Moshiri, and T. Rosing, "Geniehd: Efficient dna pattern matching accelerator using hyperdimensional computing," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 115–120.

[9] P. Neubert, S. Schubert, and P. Protzel, "An introduction to hyperdimensional computing for robotics," *KI-Künstliche Intelligenz*, vol. 33, no. 4, pp. 319–330, 2019.

[10] S. Datta, R. A. G. Antonio, A. R. S. Ison, and J. M. Rabaey, "A programmable hyper-dimensional processor architecture for human-centric IoT," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 3, pp. 439–452, Sep. 2019.

[11] J. F. Gemmeke, D. P. Ellis, D. Freedman, A. Jansen, W. Lawrence, R. C. Moore, M. Plakal, and M. Ritter, "Audio set: An ontology and human-labeled dataset for audio events," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 776–780.

[12] M. Imani, S. Pampana, S. Gupta, M. Zhou, Y. Kim, and T. Rosing, "Dual: Acceleration of clustering algorithms using digital-based processing in-memory," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 356–371.

[13] H. Jin, Q. Song, and X. Hu, "Auto-keras: An efficient neural architecture search system," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 1946–1956.

[14] F. R. Najafabadi, A. Rahimi, P. Kanerva, and J. M. Rabaey, "Hyperdimensional computing for text classification," in *Design, Automation Test in Europe Conference Exhibition (DATE), University Booth*, 2016, pp. 1–1.

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

[16] Z. Yan, H. Zhang, R. Piramuthu, V. Jagadeesh, D. DeCoste, W. Di, and Y. Yu, "Hd-cnn: hierarchical deep convolutional neural networks for large scale visual recognition," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2740–2748.

[17] "Uci machine learning repository," https://archive.ics.uci.edu/.

[18] Y. Vaizman, K. Ellis, G. Lanckriet, and N. Weibel, "Extrasensory app: Data collection in-the-wild with rich user interface to self-report behavior," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 2018, pp. 1–12.

[19] W. Jiang, "Mnist-mix: A multi-language handwritten digit recognition dataset," *arXiv preprint arXiv:2004.03848*, 2020.

[20] 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*, 2019, pp. 1–6.