Comprehensive Integration of Hyperdimensional Computing with Deep Learning towards Neuro-Symbolic AI

Hyunsei Lee[1], Jiseung Kim[1], Hanning Chen[3], Ariela Zeira[2], Narayan Srinivasa[2], Mohsen Imani[3] and Yeseong Kim[1]
[1]Daegu Gyeongbuk Institute of Science and Technology, [2]Intel Labs, and [3]University of California Irvine
{wwhslee, js980408, yeseongkim}@dgist.ac.kr, {ariela.zeira, narayan.srinivasa}@intel.com, {hanningc, m.imani}@uci.edu

2023 60th ACM/IEEE Design Automation Conference (DAC) | 979-8-3503-2348-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/DAC56929.2023.10248004

***Abstract*—HD computing is a symbolic representation system which performs various learning tasks in a highly-parallelizable and binary-centric way by drawing inspiration from concepts in human long-term memory. However, the current HD computing is ineffective in extracting high-level feature information for image data. In this paper, we present a neuro-symbolic approach called NSHD, which integrates CNNs and Hyperdimensional (HD) learning techniques to provide efficient learning with state-of-the-art quality. We devise the HD training procedure, which fully integrates knowledge from the deep learning model through a distillation process with optimized computation costs due to the integration. Our experimental results show that NSHD provides high energy efficiency as compared to CNN, e.g., up to 64% with comparable accuracy, and can outperform the learning quality when more computing resources are allowed. We also show the symbolic nature of the NSHD can make the learning humnan-interpretable by exploiting the property of HD computing.**

***Index Terms*—HD Computing, Knowledge Distillation, Neuro-symbolic AI.**

I. Introduction

Deep learning models, namely Neural Networks (NNs), have proven to be very effective in solving various challenging problems. One of the most researched domains is computer vision and Convolutional Neural Networks (CNNs) have been especially remarkable in image-related tasks. The success can largely be attributed to their ability to extract high-quality, high-level features during training. This training, however, is a heavily time-consuming process. Training data is iteratively applied to the model and the model's parameters, which are commonly over the millions, are updated for every batch of training data. In consequence, NNs are expensive both in terms of computation and memory, making them unfit for deployment to resource-limited edge devices. Another concern is their "black-box" nature. Which is to say that although we can observe a networks's topology, weights, hyperparameters, etc., drawing conclusions on and explaining what individual components of complex networks are doing or how the network as a whole will perform is still very much ongoing research. This further delays the deployment of machine learning to some fields such as healthcare industries where many clinicians remain wary of AI techniques as they are responsible for safety critical tasks [1].

Recently, many research have put forth Hyperdimensional (HD) computing as an alternative learning model that addresses some of the concerns of traditional deep learning techniques [2][3]. HD computing is a symbolic representation system that draws inspiration from concepts in human long-term memory [4]. It maps data points to high-dimension, random vectors, called hypervectors. Hypervectors have dimensions in the thousands and use a holistic mapping, meaning information is encoded equally over a vector's components and the vector as a whole is seen as an entity. This is unlike traditional representation where bit positions hold specific information.

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 also supported in part by National Science Foundation #2127780, Semiconductor Research Corporation (SRC), 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 a grant from Intel.

The dimension-independent nature of HD computing allows implementations of highly parallel hardware accelerators [5], which is another key advantage of using HD computing. However, hypervectors and their i.i.d. components are not as effective in extracting high-level feature information like CNNs. In our experiments, for example, the state-of-the-art encoding algorithm, non-linear encoding [6], yields an abysmal 39.88% and 19.7% accuracy on the CIFAR-10 and CIFAR-100 [7] datasets, respectively.

Neuro-Symbolic AI is an emerging concept in AI that looks to combine NNs and other symbolic learning approaches [8]. The general approach is to use deep learning techniques to extract features which a symbolic model will then use to form a more human-like interpretation. Earlier works [9] [10] have conducted preliminary studies for a similar approach, i.e., HD computing-based learning methods which *encodes* features extracted by state-of-the-art deep learning models. However, there are two key technical concerns for neuro-symbolic AI. First, earlier work [10] use convolution layers at the beginning of deep learning models as feature extractors while simply abandoning subsequent layers, throwing away any valuable knowledge potentially stored in them. As a result, their approach only achieved sub-optimal performance as compared to the CNNs. Furthermore, the extracted features typically have an enormous number of values due to archetypal structures of the convolutional layers of the modern deep learning models, which have many channels. Many prior research point out that the encoding procedure is the main bottleneck in HD computing [2]. A large number of extracted features create not only a huge computational burden but also a large memory overhead to store the hypervectors during encoding and the whole HD learning process.

In this paper, we present a neuro-symbolic approach that integrates CNNs and Hyperdimensional (HD) learning techniques, addressing the two aforementioned concerns hindering accuracy and efficiency. We name this model Neuro-Symbolic HD (NSHD) and it utilizes a portion of well-performing CNN models for their superior feature extraction and integrates them with the HD model. We accelerate the efficient and highly parallelizable HD operations on GPGPU and FPGA for an efficient hardware realization. Unlike prior works based on the traditional HD training, which only take partial knowledge from CNN models, we further extend the HD training procedure to make NSHD achieve state-of-the-art learning quality. Our new training procedure fully integrates knowledge from the deep learning model through a distillation process between the heterogeneous models. The following summarizes our contributions:

- We propose NSHD, which aims for **accurate and efficient** symbolic learning with better explainability by integrating lightweight HD computing with state-of-the-art deep learning.
- **During training, NSHD explicitly exploits the learned knowledge of the pre-trained model throughout all the layers**, unlike prior works which take the extracted features only from several convolutional layers. In other words, NSHD trains the HD computing model by transferring the knowledge trained in both the selected convolution layers extracting features and the rest of the layers performing sophisticated feature assimilation and prediction. We formulate our solution by carefully altering HD learning methods and knowledge distillation techniques [3][11], i.e., the proposed training method used in NSHD intrinsically transfers the knowledge of the two heterogeneous learning models, deep learning and HD computing.

• **We propose a learning-driven feature compression method to create an effective information-preserving projection, making HD learning highly efficient.** The projection maps convolution-extracted features with extreme dimensions into a significantly smaller dimension before feeding them into the HD computing process. To this end, we developed a concrete theoretical foundation that tightly incorporates the backpropagation of deep learning and the state-of-the-art HD training procedure.

The evaluation results show that NSHD can provide the state-of-the-art quality for the vision tasks to the HD computing. For example, NSHD can achieve comparable prediction results with up to 64% of the execution time reduction. We also show that the NSHD learning procedure has the human-interpretable property, which can accelerate the development of the neuro-symbolic AI.

II. Related Work

HD fundamentals HD computing is a computing paradigm inspired by sparse distributed memory (SDM), a human long-term memory model studied in neuroscience. Kanerva explains the concept of HD computing and discusses its mathematical foundations in-depth in [4]; here, we briefly discuss the fundamentals of HD computing. HD representation encodes data points to symbolic hypervectors. The majority of research in the literature generate hypervectors by randomly sampling each dimension from bipolar values [12][2][6]. The underlying idea behind the high dimensions and random values is to reach orthogonality between unrelated data points and similarity between related ones. The relation of hypervectors is measured through a similarity metric; dot product similarity is most often used for bipolar hypervectors [9]. Two hypervectors are orthogonal if they have a zero dot product. If two hypervectors of dimension D taken at random are each generated with i.i.d. components, they will have a high probability of a $D/2$ overlap in bits with a standard deviation of $\sqrt{D/4}$. That is, when dimensions are in the thousands, they are quasi-orthogonal, meaning they represent distinct information.

Learning with HD computing Learning models using the HD representation system aggregate hypervectors belonging to the same class into a single centroid hypervector that represents the class [2]. Inference is done by comparing similarity between an input hypervector and the class centroids. HD learning algorithms have proved to be very successful in several learning tasks such as language recognition [13], speech recognition [12], and robotics [9].

The concept of combining NNs and HD models has also been explored in prior research. [14] converts the prediction layer's output of various CNN models to form an ensemble of HD models. [10] and [9] are similar to our work in that they use the first several layers of a CNN as a feature extractor for a HD model. However, prior works have two drawbacks. First, they do not utilize the potential knowledge held in the CNN's weights. For example, [10] discards all subsequent layers and also requires the retraining of the entire feature extractor alongside the HD model. It, consequently, dismisses valuable knowledge held in the discarded layers and calls for greater computational resources. Second, the extracted features need to be significantly reduced to keep the HD learning model and its operations memory and computation efficient. [9] uses locality sensitive hashing (LSH) with random hyperplanes to reduce feature dimensions. However, LSH does not allow radically small bucket sizes and, therefore, the hashing calculations and computations on the resulting hypervectors are not negligible. Nevertheless, these approaches still bring significant performance improvements over standalone HD learning algorithms.

III. Overview of NSHD

We present NSHD, a neuro-symbolic framework that combines modern deep learning models with a more intuitive and brain-like model, Hyperdimensional (HD) computing. Fig. 1 shows an overview of NSHD. NSHD first *symbolizes* an input image into a hypervector (Sec. IV). We utilize layers of well-performing CNNs for their feature extracting capabilities. NSHD then converts the features into hyperdimensional representations. Using the symbolic data, we can reason through the HD model with better explainability and efficiency.

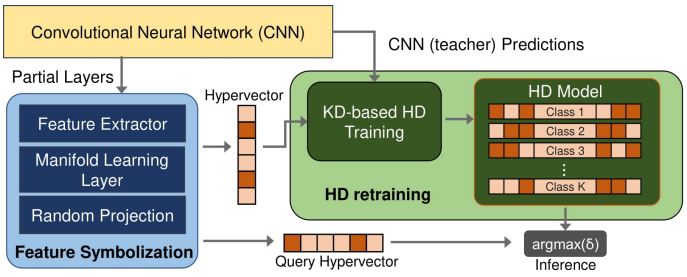


Fig. 1: An Overview of NSHD

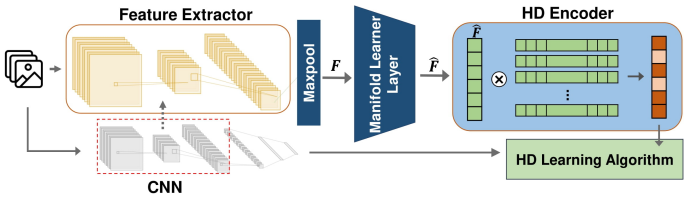


Fig. 2: NSHD Feature Symbolization Procedure

We can utilize the symbolic data, i.e., in our case, the hypervectors, for diverse learning task. In this paper, we particularly focus on the classification to better show the learning quality of NSHD as compared to the prior HD-based work, which usually handles the classification tasks. Unlike the prior HD computing-based training procedure, we employ knowledge distillation (KD) to take advantage of knowledge in the rest of the CNN's layers. KD is a model compression and acceleration technique that is used to train a smaller model (student) with the final prediction logits of a larger, more complex model (teacher). We devise a distillation procedure between the neural and symbolic models to integrate their knowledge more completely. With the distillation training procedure happening across CNN and HD computing, NSHD trains the HD model that consists of the representative hypervectors for each class, called *class hypervectors*.

Once the model is trained, we can perform the inference procedure by (i) computing the symbolized features called the *query hypervector*, and (ii) comparing the query hypervector with all class hypervector with a similarity metric. Then, we can take the inference result by selecting the class with the most similar to the query hypervector.

IV. Data Representation – Neural to Symbolic

A. Feature Extractor

NSHD can take virtually any deep learning model as its feature extractor. Advances in deep learning techniques such as convolutional layers have allowed CNNs to make great strides in this area. For the feature extractor, we turned to well-performing CNN models that are available "off-the-shelf" and pretrained. The models we show in this paper as our feature extractor are pretrained CNNs: Mobilenetv2 [15], Efficientnet b0, Efficientnet b7 [16], and VGG16 [17]. These models all end with fully-connected layers which are focused more towards the role of classification. A naive way to make a feature extractor is to use all the convolution layers. However, it is more beneficial to choose *earlier* layers for higher efficiency. It is still challenging to know at which layer a CNN model fully extracts features to the best of its capability, since human interpretation of NNs is still an ongoing research [18]. It is beyond the scope of our work here; fortunately, in practice, it is easy to empirically search for this layer. We take an intermediate layer near the end of the model that yields good accuracy. We remove all subsequent layers from the model, but use them later to transfer their knowledge during the HD model's training.

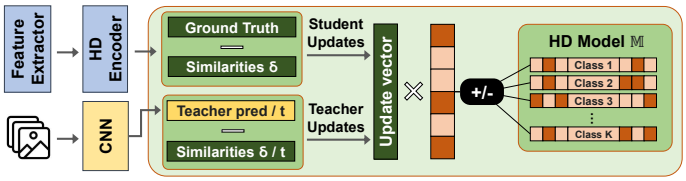


Fig. 3: Distillation Across Deep Learning and HD Computing

B. Encoding to a HD Representation

Learning models using HD representation map data to hypervectors in hyperspace such that related vectors will be closer in similarity while unrelated vectors remain orthogonal. In this paper, we use a popular encoding method, *random projection* [2]. Random projection encoding exploits binary random projection hypervectors $\mathbb{P}$ with F random bipolar base hypervectors of D dimensions $[\mathbb{P}_0 \cdots \mathbb{P}_F]^D$, where F is the number of features in a data sample, in this paper it is the flattened output size of the feature extractor. With $\mathbb{P}$ and an extracted feature vector $\mathbb{V} = \{V_1, \cdots, V_F\}$, the random projection encoding is defined as $\mathbb{H} = \Phi_{\mathbb{P}}(\mathbb{V}) = sign(V_1 \otimes \mathbb{P}_1 \oplus \cdots \oplus V_F \otimes \mathbb{P}_F)$, where $\oplus$ and $\otimes$ are HD arithmetic operations *bundling* and *binding*, respectively. Bundling combines two or more hypervectors into a single composite hypervector that is similar to its inputs and is most commonly implemented as an element-wise addition. Binding associates hypervectors by merging them into a resultant hypervector that is quasi-orthogonal to its inputs and is implemented as a scalar or element-wise multiplication.

C. Feature Reduction for Optimal Efficiency

Prior work using HD models for learning tasks have tackled simple workloads with relatively few features, i.e., less than 1000 [3]. Even so, the encoding process has traditionally been the main bottleneck of HD-based learning models. NSHD takes extracted features from intermediate layers of CNNs which could result in an extremely large number of features, e.g., the second to last convolution layer of VGG16 outputs 25,088 features. When the number of features is high, the binding of F features and $\mathbb{P}$ hypervectors, which are also of high dimensions, leads to an extraordinarily large amount of parameters. In order to reduce parameter count and make our model more efficient, we propose the use of *a manifold learning layer*. Our manifold layer first maxpools the output of the feature extractor with a window of 2, we then add a fully-connected layer that acts as a regressor to further reduce the number of pooled features to $\hat{F}$. We denote it with $\Psi(\cdot) : \mathbb{R}^F \rightarrow \mathbb{R}^{\hat{F}}$. To design this process, we draw inspiration from the learning procedure of work in [19], which maps between the original pooled feature space to another space. As a result, the manifold learner takes as input the pooled features flattened to a $1 \times F$ vector and outputs a fixed $\hat{F}$ features, which will be encoded to an HD representation with binary random projection. In summary, the symbolization process of NSHD is $\mathbb{H} = \Phi_{\mathbb{P}}(\Psi(\text{conv}(x)))$, where $\text{conv}(\cdot)$ is the CNN feature extractor and x is the input image. The key challenge here is how to train the feed-forward layer of the manifold learner, which is essential to achieve high efficiency in using HD computing. We discuss our approach in Sec. V-C.

V. Lightweight Learning with HD Computing

A. NSHD Retraining

Early implementations of HD models bundled data samples under the same class to form a single hypervector, called class hypervectors $\mathbb{C} = \sum_{i=1}^{n} \mathbb{H}_i$, and they would represent the class. The inference procedure was to simply compare incoming data to class hypervectors, i.e., $argmax(\delta([\mathbb{C}_0 \cdots \mathbb{C}_{k-1}], \mathbb{H}))$, where k is the number of classes. For higher accuracy, many prior works *retrain* the class hypervectors through multiple iterations. In this paper, we utilize Many-class Similarity Scaling (MASS) retraining, proposed in [3]. MASS further tunes the class hypervectors such that the class hypervector with the correct label would become more similar to the input sample while other class hypervectors would grow more dissimilar. A unique advantage of MASS compared to prior retraining methods is that it updates the class hypervector based on *class-wise similarity differences*. It is performed by calculating an update vector $\mathbf{U}$ by taking the one-hot encodings for training samples and subtracting the similarity values of the training sample hypervector, $\mathbb{H}$, i.e., $\mathbf{U} = one\_hot - \delta(\mathbb{M}, \mathbb{H})$, where $\mathbb{M} = [\mathbb{C}_0 \cdots \mathbb{C}_{k-1}]$. $\mathbf{U}$ now holds update values that would bring larger changes for erroneous classifications. The training sample $\mathbb{H}$ is finally scaled with the update values and a learning rate, λ, and bundled to the class hypervectors: $\mathbb{M} = \mathbb{M} + \lambda \mathbf{U}^\intercal \mathbb{H}$.

By itself, HD learning models are very efficient as their operations are highly parallelizable [5]. However, as discussed in Sec. II valuable information stored in the dropped layers are never utilized in prior works. As a means to retain knowledge from the sophisticated training procedure of deep learning models and to compress our model, we implement knowledge distillation [11] as an extension to the MASS retraining algorithm.

B. Knowledge Distillation – Deep Learning to HD

Fig. 2 illustrates NSHD's learning procedure. We employ the teacher-student KD framework [11], which has traditionally been between models of similar NN architectures, but in this work, we present a process that distills knowledge from a deep learning architecture to a HD model. This enables the transfer of knowledge in the unused layers, allowing for the selection of earlier and efficient, but worse-performing intermediate layers with minimal performance degradation.

Algorithm 1 NSHD Knowledge Distillation Procedure

1: $\mathbb{M} = [\mathbb{C}_0 \cdots \mathbb{C}_{k-1}]$
2: **for** hypervector $\mathbb{H}$ in a training dataset **do**
3: similarity_values $= \delta(\mathbb{M}, \mathbb{H})$
4: soft_pred $=$ similarity_values$/\mathbf{t}$
5: soft_labels $= softmax($teacher_pred$)/\mathbf{t}$
6: distilled_updates $=$ soft_labels $-$ soft_pred
7: $\mathbf{U} = (1 - \alpha) \times$ (one_hot $-$ similarity_values)
8: $\mathbf{U} = \mathbf{U} + \alpha \times$ distilled_updates
9: $\mathbb{M} = \mathbb{M} + \lambda \mathbf{U}^\intercal \mathbb{H}$

Algorithm 1 shows the complete retraining procedure with knowledge distillation. We use the original, uncut CNN as the teacher and NSHD with an earlier but less accurate layer as the student. The primary change from the MASS procedure is to calculate $\mathbf{U}$, i.e., *class-wise similarity difference* based on the teacher's prediction results, which have more useful knowledge than the simple one-hot encoding vector in describing the training sample. As shown in Line 4, we calculate the student's soft prediction value by softening the similarity values with a hyperparameter $\mathbf{t}$. Similarly, we take the teacher's softened predictions by taking the softmax output of the last prediction layer and softening it with $\mathbf{t}$. With these values we can get the distilled update vector by subtracting the two. The final update vector, $\mathbf{U}$, is a weighted sum between the updates using the teacher as the target and the ground-truth, one-hot encoding based updates. Finally, the updated training sample $\mathbb{H}$ is bound to the class hypervectors $\mathbb{M}$ updating all class hypervectors at each iteration.

C. Training the Manifold Learner

As discussed in Sec. IV-C, NSHD optimizes efficiency by introducing the manifold layer to reduce the number of encoded features. A naive way to implement the manifold layer is to instrument the original CNN model and retrain; however, this requires the costly retraining of either the entire CNN model or the rest of the layers after the manifold layer at the very least. We address the issue by decoding the errors happening in the class hypervectors and back-propagating them to the manifold layer across the HD encoder.

TABLE I: Design Acceleration On Xilinx ZCU104

	LUT	FF	BRAM	URAM	DSP
Total	84.9K	146.5K	224	40	844
Available	230.4K	460.8K	312	96	1728
Utilization	36.87%	31.80%	71.79%	41.67%	48.84%
Frequency	200MHz				
Power	4.427W				

Let us recall Algorithm 1 describing the knowledge distillation procedure. $\mathbb{E} = \lambda \mathbf{U}^{\intercal}\mathbb{H}$ is the form of the class-wise error hypervectors. We denote each per-class error hypervector by $\mathbf{E_i} \in \mathbb{E}$, where $0 \leq i < K$, and our goal is to *decode* $\mathbf{E_i}$ into the original feature space ($\in \mathbb{R}^{\hat{F}}$). During the decoding procedure, since the sign function is not differentiable in the backpropagation, we approximate its effect with a straight-through estimator usually used for training binary neural networks [20]. We then apply the HD decoding [2], which applies the binding with the binary random projection hypervectors, $\mathbb{P}$, and the dot-product operation in turn. As a result, we can estimate the errors in the output of the manifold layer, and update the fully-connected layer through the typical backpropagation procedure of deep learning.

VI. Implementation

A. GPGPU Implementation

We implement the training and inference procedure on a GPGPU. We can use common tensor frameworks, e.g., TensorFlow or PyTorch, to run the feature extractor for the given training/test inputs. One of the key merits of NSHD is that it uses the weights pretrained in the original CNN model without any modification for both the training and inference procedure. Thus, we can eliminate the costly backpropagation procedure of CNN, and furthermore can use any advanced acceleration platform, e.g., tensor engines or external hardware accelerators. In this work, we utilize the NVIDIA TensorRT framework, which performs high-performance deep learning inference based on quantization and specialized hardware units in GPGPU.

Once the features are extracted, we perform the rest of the training/inference procedure on the common GPGPU system. Our current implementation is based on the PyTorch; but we enhance its capability with custom operations that best utilize the binary-centric nature of the hypervector operations in an optimized fashion. For example, there are two primary binary computations: (i) the HD encoding procedure, which is mainly composed of binding operations with the binary random projection hypervectors, and (ii) the similarity computation for the class hypervectors, which takes symbolized binary hypervectors as its input. We optimize the binary computations on CUDA so that it utilizes the constant memory, a type of GPGPU memory that is significantly faster than the common GPU global memory (GDDR) due to the dedicated cache hierarchy. During the binary HD computation, we load the binary hypervectors on the constant memory while loading the other types of the data, e.g., integer or floating-point, from the global memory to the shared memory whose performance is similar to the L1 cache. Then, we can perform the computations eventually without multiplication, i.e., only using relatively lightweight arithmetic operations, i.e., addition or subtraction depending on the sign bit for each binary hypervector element in the constant memory. In addition, since the hypervectors stored in the constant memory are binary, we can significantly reduce the memory footprints.

B. FPGA Implementation

We also implement the hardware acceleration of the NSHD inference procedure on FPGGA, Xilinx Zynq UltraScale+ MPSoC ZCU104 (ZCU104). The host program running on ZCU104 PS side (ARM Cortex 53) is written with Python, and the communication between PS and PL is based on AMBA AXI interface. For hardware accelerator on the ZCU104 PL side, we import the Xilinx deep learning unit (DPU) IP. We exploit NSHD into DPU using Vitis AI framework.

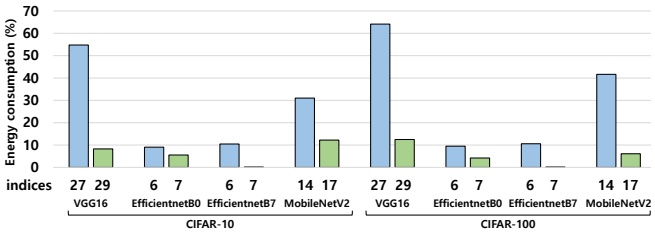


Fig. 4: Percentage Improvements on Energy Efficiency

The trained entire NSHD model can be viewed as a special type of the neural networks, where the hypervectors and HD operations are quantized data and tensor operations. Thus, we can compile and map the trained NSHD model into the Vitis AI framework so that it runs the convolution layers and HD computing-related computation on the same platform. Note that the Vitis AI framework quantizes the given model for better efficiency, and in our observation, the quantization has very minor impacts on the prediction quality.

VII. Experimental Results

A. Experimental Setup

We implement the proposed NSHD on NVIDIA Xavier Platform, which aims to the low-power edge systems, and Xilinx Zynq ZCU104 platform. To measure power consumption of the GPGPU, we use the NVIDIA system management interface (nvidia-smi). Table I summarizes the resource utilization of Xilinx DPU IP on the MPSoC programmable logic (PL) side. Compared to traditional GPU-based accelerators, FPGA-based accelerator consumes less power.

Models used as feature extractors are Mobilenetv2 [15], Efficientnet b0, Efficientnet b7 [16], and VGG16 [17]. We label layers of the CNN by their indices; Efficientnet is divided by their blocks, Mobilenetv2 by operators, and VGG16 by each convolution, pooling, and activation layers. Each fully-connected layer in all models have their own index. We use datasets Cifar-10 and Cifar-100 [7] for performance evaluations and set hypervector dimensions at 3,000.

For the knowledge distillation process, we perform hyperparameter search to identify the best combination of T and α discussed in Sec. V-B; we also report the full search results in Sec. VII-C2 as an example. We observe that another hyperparameter, $\hat{F}$, which is the feature size produced by the manifold layer, should be sufficiently large enough, i.e., at least as large as the number of classes, to produce accurate prediction results. We empirically set $\hat{F}$ as 100 in our evaluations. For the hypervector dimension, we utilize $D = 3,000$ by default, which is relatively smaller than the dimension used in most prior work, i.e., $D = 10,000$, since there are very negligible changes in accuracy; we also show how the accuracy changes over different dimensions in Sec. VII-D.

B. NSHD Efficiency

1) Energy Efficiency Comparison: In this evaluation, we compare the energy efficiency of NSHD models during inference by comparing with the respective CNN model. For NSHD, we empirically select two convolution layer indices for each CNN model as the layer extracting features so that the accuracy loss is less than 10%. Figure 4 shows energy consumption improvements of NSHD at different intermediate layers as feature extractors on the Cifar-10 and Cifar-100 dataset. The results show that NSHD can save energy efficiency significantly when selecting an early convolution layer for feature extraction, e.g., using VGG16 as the feature extractor at layer 27 uses 64% less energy than the original CNN model. NSHD achieves higher energy efficiency when selecting earlier convolution layers. NSHD can take advantage of the powerful state-of-the-art feature extraction capabilities of modern deep learning models while being more scalable, even more so if we are willing to consider trade-offs (discussed in Sec. VII-C).

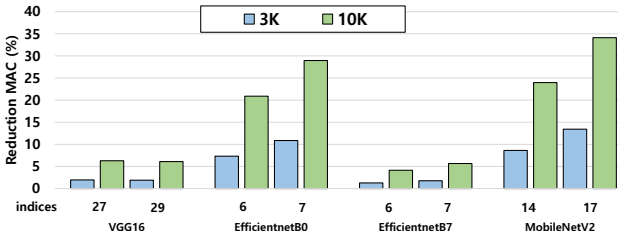


Fig. 5: Impact of Manifold Learner on MACs for 3K: $D = 3,000$ and 10K: $D = 10,000$. The numbers on X-axis for each model are the layer indices used for the feature extractor.

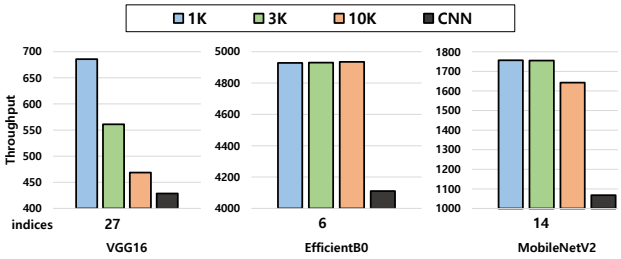


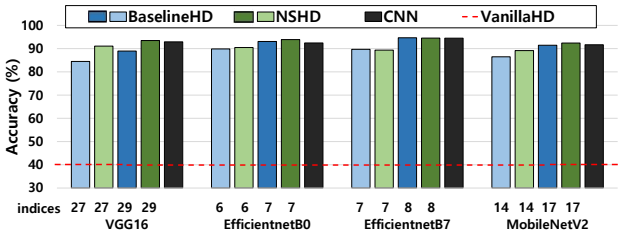
Fig. 6: Throughput (FPS) of FPGA Implementation

2) Comparison with Prior HD work: As discussed in Section II, there were a couple of prior HD works which try to use the existing DNN models as a feature extractor [9]. We name this approach BaselineHD. The key difference of NSHD from BaselineHD is the proposed manifold layer, which learns effective compression strategies to reduce the feature sizes eventually processed in the HD encoding procedure. To understand how much computations are saved using the manifold learner, we measured the number of multiply-accumulate (MAC) operations by assuming that the binding/bundling operations are element-wise multiplication/addition. Figure 5 summarizes the experimental results. We observe that the manifold learner plays an important role in keeping NSHD efficient. For example, NSHD requires 20.9% and 28.95% fewer computations for Efficientnetb0 using layers 6 and 7 as the feature extractor, respectively. Since the encoding overhead increases as the hypervector dimension increases, we observe higher savings for $D = 10,000$ than the $D = 3,000$ cases, e.g., up to 34% for Mobilenetv2 at the 17th layer.

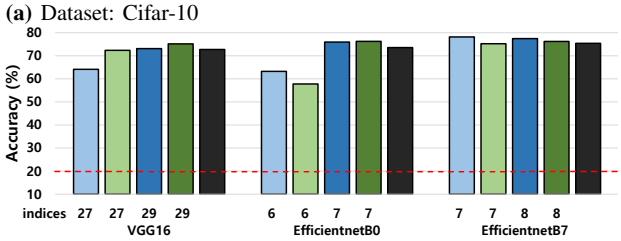
3) FPGA Acceleration: To examine the practical value of NSHD, we next examine the throughput of the FPGA accelerator. Figure 6 presents the throughput of NSHD on the ZCU104 platform as compared to the CNN model running on the same DPU accelerator. Here we choose to use frame per second (FPS) as the inference throughput metric. For this evaluation, we selected the earliest layer used in Sec. VII-B1 and measure the throughput over different hypervector dimension settings. Compared to the original CNN model, NSHD is hardware friendly, therefore achieving, on average 38.14% improvement in inference throughput.

TABLE II: Model Size (Learning Parameters) Comparison

Model	Layer	CNN	NSHD	BaselineHD
VGG16	27	537.2MB	69.61MB	87.17MB
	29	537.2MB	69.05MB	96.61MB
Efficientnetb0	5	16.08MB	5.76MB	11.75MB
	6	16.08MB	12.36MB	15.16MB
	7	16.08MB	15.69MB	20.38MB
	8	16.08MB	20.79MB	39.67MB
Efficientnetb7	6	255.25MB	164.382MB	170.01MB
	7	255.25MB	251.03MB	260.45MB
	8	255.25MB	264.52MB	302.3MB
Mobilenetv2	14	8.94MB	3.52MB	5.85MB
	17	8.94MB	8.55MB	13.24MB



(a) Dataset: Cifar-10



(b) Dataset: Cifar-100

Fig. 7: Accuracy Comparison

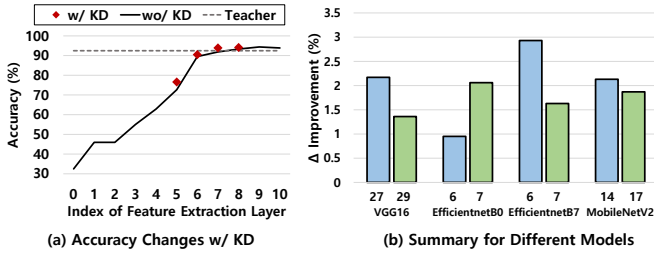


Fig. 8: Impact of KD on the Learning Accuracy

4) Model size: Table II shows a comparison of parameters in terms of their size in bytes for the original CNN and NSHD at different intermediate layers as feature extractors. The results show that NSHD can reduce the model size effectively without sacrificing accuracy. Note that the NSHD model is significantly smaller than BaselineHD for many cases thanks to the manifold layer, which effectively reduces the amount of features passed from the convolution layers in the feature extractor to the HD model. Without it, a naive approach would have had 96.61M parameters for VGG16 at the 29th layer which is a 39.91% increase from what we have with the manifold learner.

C. NSHD Learning Accuracy

1) Accuracy Comparison: We evaluate the accuracy of NSHD by comparing its counterparts: (i) VanillaHD, the HD model that does not use any feature extractor, (ii) BaselineHD, the HD model that uses the convolution feature extractor in a similar fashion to prior work [9] but without the manifold layer and knowledge distillation, and (iii) CNN, the original CNN models. Figure 7 shows the comparison results. Our results show that, unlike VanillaHD, NSHD can solve image-related learning tasks in a neuro-symbolic way. Indeed, NSHD is able to achieve similar accuracy levels to the respective CNN model at least, and furthermore, with sufficient feature extraction layers, NSHD reliably outperforms the CNN model. For example, at layer 7 in Efficientnet b0, NSHD outperforms the original model while at layer 6, NSHD shows similar performance as the original model but is more efficient. Also, NSHD outperforms BaselineHD significantly. Even though BaselineHD typically uses much more learning parameters as discussed in Sec. VII-B4, NSHD achieves better accuracy based on the knowledge distillation.

2) Impact of Knowledge Distillation: NSHD utilizes knowledge distillation (KD) to fully integrate the knowledge of deep learning and symbolic HD learning. To better illustrate the impact of KD, Figure 8 (a) shows the results of taking each layer as the feature extractor for

Alpha \ Temp	12	13	14	15	16	17
0	0.6786	0.6786	0.6786	0.6786	0.6786	0.6786
0.1	0.6858	0.6861	0.6787	0.683	0.6876	0.6932
0.2	0.6986	0.7015	0.6949	0.6941	0.692	0.6917
0.3	0.7201	0.7147	0.7132	0.7254	0.7163	0.7155
0.4	0.7342	0.738	0.7352	0.7433	0.7424	0.7346
0.5	0.7421	0.7449	0.7432	0.7446	0.7486	0.7457
0.6	0.7485	0.7455	0.7509	0.7459	0.7439	0.7445
0.7	0.7462	0.7459	0.7525	0.7505	0.7473	0.7498
0.8	0.7412	0.7394	0.7455	0.745	0.7466	0.7483
0.9	0.7371	0.7399	0.7417	0.7425	0.7468	0.7464

Fig. 9: Accuracies for Hyperparameter Search in KD.

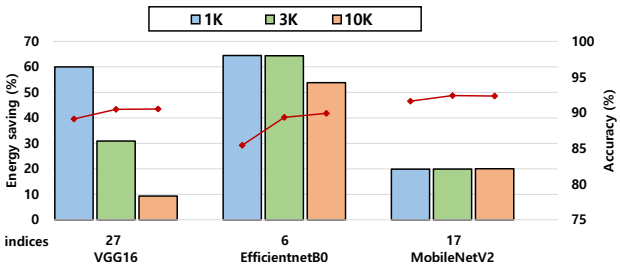


Fig. 10: Efficiency and Accuracy Tradeoff on FPGA.

Efficientnetb0 as an example. The results show that KD effectively fills the accuracy gap by eliciting the knowledge stored in the CNN to the HD model. Figure 8(b) summarizes the impact of KD on the accuracy over other models. We observed the same trend over all models, and thus concluded that distilling knowledge from deep learning to the symbolic model brings out the full potential of both architectures.

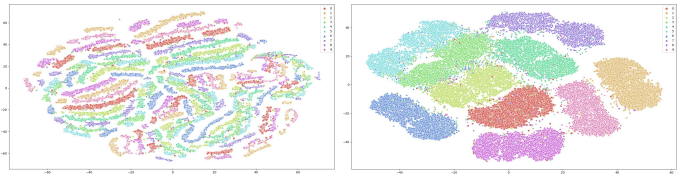
To best utilize the knowledge distillation, a process of hyperparameter search should be performed. As an example, Figure 9 shows detailed results of the hyperparameter search for Efficientnetb7 layer 7. The result shows that KD boosts the accuracy by 7.39% while utilizing the knowledge of the teacher model, in our case, the original CNN. In our experiments, the two hyperparameters, T and α vary for different models; but are typically found in the range of 14 to 16.

D. NSHD Dimensionality and Efficiency Tradeoff

In this section, we show the ablation study on how different dimensions affect NSHD. Figure 10 illustrates the tradeoff relationship between efficiency and accuracy over different dimensionality sizes. Traditionally, work in HD computing have defaulted to hypervectors with a dimension of 10,000 as higher dimensions allows for easier discrimination between hypervectors [4]. However, we do not require as high a dimension to achieve maximum accuracy. For most cases, the dimensionality larger than $D = 3,000$ is sufficient to regenerate the quality of the CNN model, while $D = 1,000$ would degrade the inference quality. For example, the parameters for the HD section of NSHD can be reduced by 70% by going from 10,000 to 3,000 and a further 20% improvement can be had with 1,000 dimensions with a relatively small loss in accuracy of an average of 1.64%.

E. NSHD Explainability

In this section, we look to explain HD learning and hypervectors through a visual representation to verify the potential of HD computing for human-like interpretation. Based on the t-SNE projection, Figure 11a depicts a 2D representation of hypervectors at the first iteration for the CIFAR-10 dataset using 7th layer of Efficientnet b0 as the feature extractor. The HD model plots a pattern that is somewhat vague and difficult to interpret. Knowledge in HD computing is mathematical in nature and we can add and subtract from these vectors of high dimensions to pull class hypervectors towards their data samples. Similarly, we can bring related hypervectors closer such that when we convert new data they will plot near their class. As shown in Figure 11b, in the final training iteration the training samples form several close clusters for each class, meaning that the class hypervector eventually represents the target class.



(a) First retraining iteration **(b)** Final results after 50 iterations

Fig. 11: Explainability of HD computing with t-sne Analysis.

VIII. Conclusion

In this paper, we propose NSHD, which provides accurate symbolic learning with better explainability and higher efficiency. We integrate lightweight HD computing with state-of-the-art deep learning by using a learning-driven feature compression method and transferring knowledge between different learning approaches. The experimental results show that the proposed method yields very comparable results to deep learning while optimizing the efficiency and memory footprint, e.g., by up to 64%.

References

[1] Jeremy Petch, et al. Opening the black box: The promise and limitations of explainable machine learning in cardiology. *Canadian Journal of Cardiology*, 38(2):204–213, 2022.

[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, 2019.

[3] 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, 2021.

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

[5] Mohsen Imani, et al. Fach: Fpga-based acceleration of hyperdimensional computing by reducing computational complexity. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, pages 493–498, 2019.

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

[7] Alex Krizhevsky, et al. Cifar-100 (canadian institute for advanced research).

[8] Zachary Susskind, et al. Neuro-symbolic ai: An emerging class of ai workloads and their characterization, 2021.

[9] Peer Neubert, et al. An introduction to hyperdimensional computing for robotics. *KI - Künstliche Intelligenz*, 33(4):319–330, Dec 2019.

[10] Arpan Dutta, et al. Hdnn-pim: Efficient in memory design of hyperdimensional computing with feature extraction. New York, NY, USA, 2022. Association for Computing Machinery.

[11] Geoffrey Hinton, et al. Distilling the knowledge in a neural network, 2015.

[12] Mohsen Imani, et al. Voicehd: Hyperdimensional computing for efficient speech recognition. In *2017 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8, 2017.

[13] Mohsen Imani, et al. Low-power sparse hyperdimensional encoder for language recognition. *IEEE Design Test*, 34(6):94–101, 2017.

[14] Peter Sutor, et al. Gluing neural networks symbolically through hyperdimensional computing, 05 2022.

[15] Mark Sandler, et al. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018.

[16] Mingxing Tan et al. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019.

[17] Shuying Liu et al. Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 730–734, 2015.

[18] Artur d'Avila Garcez et al. Neurosymbolic ai: The 3rd wave, 2020.

[19] Adriana Romero, et al. Fitnets: Hints for thin deep nets, 2014.

[20] Matthieu Courbariaux et al. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *ArXiv*, abs/1602.02830, 2016.