FSL-HD: Accelerating Few-Shot Learning on ReRAM using Hyperdimensional Computing

Weihong Xu, Jaeyoung Kang, and Tajana Rosing
University of California San Diego, La Jolla, CA 92093, USA
Email: {wexu, j5kang, tajana}@ucsd.edu

***Abstract*—Few-shot learning (FSL) is a promising meta-learning paradigm that trains classification models on the fly with a few training samples. However, existing FSL classifiers are either computationally expensive, or are not accurate enough. In this work, we propose an efficient in-memory FSL classifier, FSL-HD, based on hyperdimensional computing (HDC) that achieves state-of-the-art FSL accuracy and efficiency. We devise an HDC-based FSL framework with efficient HDC encoding and search to reduce high complexity caused by the large dimensionality. Also, we design a scalable in-memory architecture to accelerate FSL-HD on ReRAM with distributed dataflow and organization that maximizes the data parallelism and hardware utilization. The evaluation shows that FSL-HD achieves 4.2% higher accuracy compared to other FSL classifiers. FSL-HD achieves $100-1000\times$ better energy efficiency and $9-66\times$ speedup over the CPU and GPU baselines. Moreover, FSL-HD is more accurate, scalable and $2.5\times$ faster than the state-of-the-art ReRAM-based FSL design, SAPIENS, while requiring 85% less area.**

***Index Terms*—In-memory processing, Few-shot learning, Hyperdimensional computing**

I. Introduction

Few-shot learning (FSL) is a data-efficient learning paradigm that relies on memory-augmented neural networks [1–3] to quickly adapt to unseen data by using a few labeled samples. Compared to deep learning models trained on a million-scale dataset, the FSL model only needs < 10 training samples per class. State-of-the-art FSL models consist of a front-end CNN feature extractor and a back-end classifier that retrieves the associated class (Fig. 1). Existing works [4–6] demonstrated that fixing the front-end CNN and just training the back-end classifier can provide fast and accurate learning capability. In this sense, the FSL boils down to designing the back-end classifier. Given a new learning episode with n classes and k labeled samples per class, the back-end classifier first learns from the image embeddings of support (training) set $\mathbb{S}$ that contains the nk samples. Then it performs classification on the query (test) set $\mathbb{Q}$. The learning episode with a balanced support set is defined as the n-way k-shot FSL problem.

In reality, since FSL is normally applied to on-device continuous learning with a limited hardware budget, designing an efficient and accurate FSL classifier imposes challenges on both algorithm and hardware. Existing FSL algorithms [1–5] fails to leverage both high accuracy and low complexity. The approaches based on k-nearest neighbor (kNN)- [1, 5, 7] with L1 distance are less complex but not very accurate. MAML [3] has a linear layer as a classifier while MLP [4] uses a two-layer perceptron to obtain excellent accuracy. However, the complicated gradient-based training incurs 10-100$\times$ complexity compared to L1-based kNN. Although [2] uses a weighted linear combination of cosine distance to reduce the training overhead as compared to MAML and MLP, the cosine distance is still $\approx 5\times$ more complex than L1.

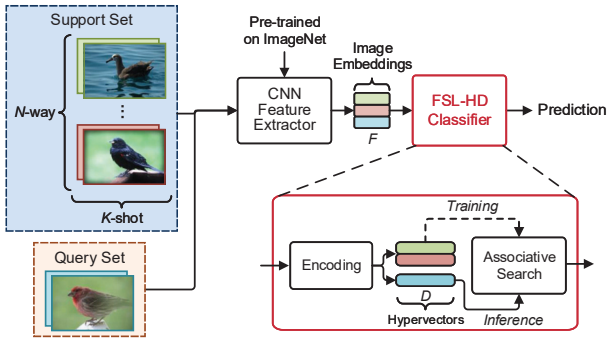


Fig. 1: Few-shot image classification using FSL-HD.

On the hardware side, existing studies [7–9] show that FSL classifiers are memory-intensive, becoming a bottleneck during the inference of FSL. To cope with the memory-intensive nature, recent works accelerate the classifiers using processing in-memory (PIM). SAPIENS [7] implements L1-based kNN classifier in 2T2R ReRAM and achieves sub-mW power consumption. Robust-MANN [8] and AiM [9] are attention-based FSL accelerators on nonvolatile devices that exhibit superior efficiency. These designs choose PIM-friendly but inaccurate FSL classifiers, thus having huge accuracy gap vs. gradient-based classifiers [3, 4]. It is challenging to implement these classifiers using PIM [10] since the gradient computation needs costly computing and memory resources. Moreover, existing PIM-based FSL accelerators [7, 8] are constrained by their classification algorithms and limited memory space, thereby lacking flexibility to support more FSL settings.

In this paper, we propose FSL-HD, a PIM accelerator for FSL back-end classifier with enhanced accuracy and efficiency over previous PIM designs [7, 8]. The aforementioned issues are addressed via exploiting hyperdimensional computing (HDC) [11] that encodes data into high-dimensional binary vectors (called *hypervectors* (HVs)), thus replacing complex fixed-point arithmetic with lightweight binary operations. We also dramatically improve latency and energy efficiency of HDC's encoding in ultra-high vector dimensions compared to the state of the art [11] by designing a novel method called *tensorized encoding*, with a *progressive search scheme* for low-latency inference. It reduces the encoding complexity by 22-31$\times$ and search latency by 1.9-2.8$\times$ compared to existing HDC baseline [11]. As a result, our FSL-HD runs at much lower cost than the existing low-cost FSL algorithms [4]. Also, FSL-HD achieves 4.2% higher accuracy than [1]. Compared to the state-of-the-art FSL solution [4], FSL-HD has 10-100$\times$

978-3-9819263-7-8/DATE23/© 2023 EDAA

lower inference complexity and memory consumption with only 1.5% accuracy gap. Our FSL-HD accelerator on ReRAM with optimized data organization and dataflow yields 2.5× speedup and consumes 85% less chip area while providing more flexible FSL support compared to existing PIM design [7].

II. Proposed FSL-HD Algorithm

Fig. 1 illustrates the pipeline of FSL-HD algorithm, composed of the front-end feature extractor and the back-end FSL-HD classifier. The feature extractor uses a pre-trained CNN model, and its weight parameters are fixed for the whole process as other FSL algorithms [7]. The front-end extracts image embeddings, which are a few hundred-dimensional vectors (e.g., 512 dimensions for ResNet-34). The back-end module performs classification given the extracted image embeddings.

FSL-HD classifier exploits HDC [11] to improve the accuracy and efficiency. FSL-HD classifier consists of encoding, training, and inference stage. The encoding step converts image embeddings into HVs (binary high-dimensional vectors). During the training phase, we generate the class HVs for support set. In turn, the inference phase predicts the class given the query HV.

A. Tensorized Encoding of Image Embeddings

FSL-HD processes FSL on HV, i.e., image embeddings from the feature extractor need to be encoded to binary HV. One representative encoding method is signed random projection (RP) due to its good accuracy [11]. The signed RP is a linear projection of the input vector $\mathbf{x} \in \mathbb{R}^F$ expressed as: $\mathbf{h} = \text{Encode}(\mathbf{x}) = \text{sign}(\mathbf{R}^T\mathbf{x})$, where $\mathbf{R} \in \{-1, 1\}^{F\times D}$ represents the binary projection matrix that transforms $\mathbf{x}$ into the corresponding HV $\mathbf{h} \in \{-1, 1\}^D$. $\text{sign}(\cdot)$ denotes the sign operation, and the projection matrix $\mathbf{R}$ is generated from the uniform distribution. Nevertheless, the RP-based encoding consumes excessive complexity. The RP encoding is a vector-matrix multiplication with $\mathcal{O}(DF)$ computation and memory complexity, where D is the HV dimension while F is the image embedding size. D normally ranges from 1K to 10K to get sufficient accuracy. Hence, the RP encoding step takes 95% or more of the total run time [11]. Fig. 2(a) shows that the complexity (number of binary operations) of RP encoding is 50-200× higher than Hamming search. Besides, the memory size to store RP matrix $\mathbf{R}$ is 25-100× than the space to store the class HVs for Hamming search. For example, storing the projection matrix $\mathbf{R}$ requires 1Mb for $D = 2048, F = 512$, far exceeding the existing 64kb ReRAM [7].

We propose the **tensorized encoding** that significantly reduces the encoding overhead. We decompose the large projection matrix $\mathbf{R}$ into multiple small sub-matrices, thus improving performance, lowering the total memory size and encoding complexity. The original RP is an orthogonal projection. Instead of directly generating $\mathbf{R}$, we use the Kronecker product [12] to construct the projection matrix that preserves the orthogonality. Specifically, the projection matrix $\mathbf{R}$ is decomposed into the Kronecker product of M sub-matrices (i.e., order) as:

$$\mathbf{h} = (\otimes \mathbf{r}_{i=1}^M)^T\mathbf{x} = (\otimes_{i=2}^M \mathbf{r}_i)^T \text{mat}(\mathbf{x}, F/f_1, f_1)\mathbf{r}_1^T, \quad (1)$$

where $\otimes$ denotes the Kronecker product and $\mathbf{R} = \otimes \mathbf{r}_{i=1}^M$. $\mathbf{r}_i \in \mathbb{R}^{f_i\times d_i}$ satisfies $\prod_{i=1}^M f_i = F$ and $\prod_{i=1}^M d_i = D$. $\text{mat}(\mathbf{A}, B, C)$ denotes the reshape operation to convert the vector (or matrix) $\mathbf{A}$ to a $B \times C$ matrix. This is to align the shape of intermediate matrices with the projection sub-matrices.

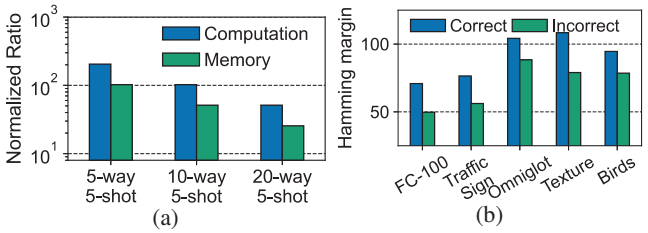


Fig. 2: (a) Computation complexity and memory size ratios between RP encoding and Hamming search. (b) Average Hamming margin of correct and incorrect predictions on five datasets for 20-way 5-shot problem.

Consider an example of 2-order tensorized encoding with $F = 256, D = 1024$. To align with RP matrix dimension based on $\prod_{i=1}^M f_i = F$ and $\prod_{i=1}^M d_i = D$, the dimensions of sub-matrices are $f_1 = f_2 = 16, d_1 = d_2 = 32$. Instead of directly computing $\mathbf{R}^T\mathbf{x}$ with $\mathcal{O}(DF)$ complexity, our encoding method first reshapes the image embedding into a 16×16 matrix and sequentially computes $M = 2$ small matrix multiplications with $\mathbf{r}_1$ and $\mathbf{r}_2$. The tensorized encoding speeds up the encoding by over 20× and uses 100× less memory space compared to RP encoding [11] (see Sec. IV-A). Furthermore, our encoding can easily scale to support different image embedding sizes F and HDC dimensions D by changing the sub-matrices dimensions f_i and d_i.

B. Single-pass FSL Training

The training stage generates the representative HV for each class. For the n-way k-shot FSL, the generation of the j-th class HV, $\mathbf{C}^j$, is given as:

$$\mathbf{C}^j = \text{sign}\left(\sum\nolimits_{i=1}^k \mathbf{h}_i^j\right), \quad (2)$$

where $\mathbf{h}_i^j$ is the i-th support HV of the j-th class. The summation is a point-wise addition operation and the class HV has the same dimension as $\mathbf{h}_i^j$ (i.e., $\mathbf{C}^j \in \{-1, +1\}^D$). Our HDC training is simple and does not require back-propagation. Also, the training is performed in a single pass so that FSL-HD sees the training data only once. These two benefits greatly reduce the complexity of training and the amount of dataflow since no additional memory is required to store the training samples and intermediate data.

C. Inference: Progressive Search

HDC inference aims to find the class HV $\mathbf{C}$ most similar to the query HV $\mathbf{Q}$. Hamming similarity is used to measure the similarity. The class HV with the largest similarity is regarded as the prediction result. The naive Hamming search is conducted in an exhaustive manner, where the pairwise Hamming similarity between the query and n class HVs is computed, requiring $\mathcal{O}(ND)$ computation complexity. The search overhead is costly when the HV dimension D and the number of ways n are large.

FSL-HD inference tries to make correct predictions at the minimal computation cost. Fig. 2(b) shows that the correct predictions have a larger average Hamming similarity margin

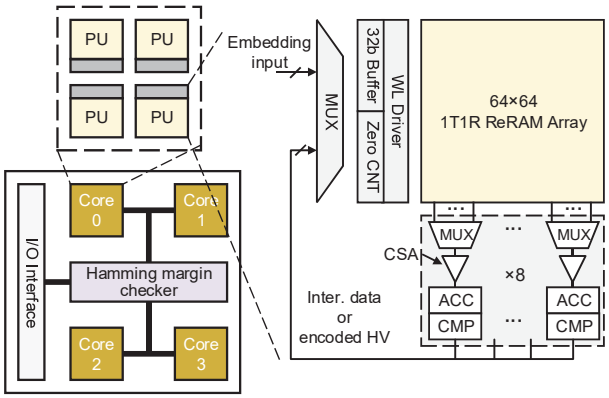
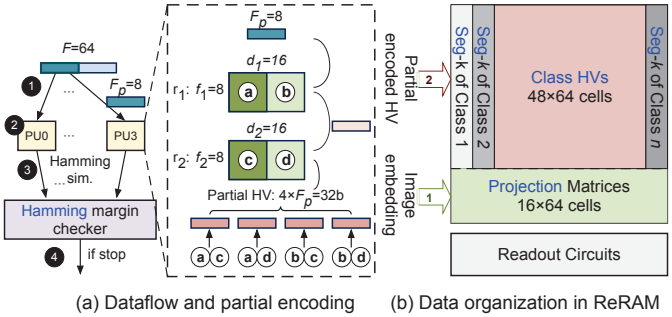


Fig. 3: Diagram of FSL-HD's 64kb in-memory architecture. $N = 16$ processing units (PUs) are implemented.



(a) Dataflow and partial encoding (b) Data organization in ReRAM

Fig. 4: (a) FSL-HD dataflow and partial encoding scheme, (b) data organization in PU.

than the incorrect ones. Hamming similarity margin is defined as the distance difference between the 1st and 2nd matched class HVs. A large margin value suggests that exhaustively computing the Hamming similarity for all bits is not always necessary to get correct predictions. Instead, we can use partial comparisons to obtain sufficiently accurate results, which reduces the search complexity and latency. Based on the observation, we propose **progressive search** that computes the minimum bits of Hamming similarity for the prediction. These minimum bits are defined using the Hamming margin. At each time step, the Hamming similarity for each segment is computed and added to the partial Hamming similarity. Then the Hamming margin is computed and compared to the threshold. Once the threshold is exceeded, the search process terminates and the class with the largest partial Hamming similarity is returned as the result.

III. FSL-HD PIM Accelerator

We accelerate FSL-HD on the single-level cell (SLC) 1T1R ReRAM device [13] that has fast wake-up, high-density, non-volatile and low-power to overcome data movement overhead. It has been physically verified to realize highly parallel and stable computing for various inference tasks [13, 14].

Fig. 3 depicts the high-level diagram of ReRAM-based FSL-HD, composed of $N_C = 4$ cores and one Hamming margin checker. The core is used for receiving query image embeddings, encoding into HVs, and doing the progressive search. The Hamming margin checker compares the Hamming margin metrics and decides the termination of progressive search. Each FSL-HD core contains $N_{PU} = 4$ processing units (PUs), where each PU has a 64×64 1T1R ReRAM sub-array. The total size of $N = N_C \times N_{PU} = 16$ PUs is 64kb. The PU performs either HDC encoding or Hamming search with the aid of peripheral circuits. The peripheral circuits include the WL driver, 32b buffer, zero counter (CNT), 8:1 column MUXs, current sense amplifier (CSA), accumulator (ACC), and comparator (CMP). Each pair of cascaded CSA, ACC, and CMP is multiplexed by one column MUX. In this case, $N_{BL} = 8$ bit lines (BLs) are activated simultaneously. We set the number of simultaneously activated world lines (WLs) to $N_{WL} = 8$ as the previous works [13, 14] have verified that up to 9 activated WLs can provide accurate computation.

A. FSL-HD Dataflow

Storing or moving HVs between PUs incurs a large buffer and bandwidth overhead. As such, we design the efficient and distributed dataflow (Fig. 4).

Query distribution and progressive search. In FSL-HD, the image embedding query with F dimension is divided into multiple segments, where each segment has a size of F_p. Rather than process the entire query in a single PU, FSL-HD distributes the query to all N PUs in a round-robin manner. Step ❶ in Fig. 4(a) gives an example of $F = 64, F_p = 8, N = 4$. For each round, $N \times F_p = 32$ points of the query embedding are distributed and computed. Each PU receives a short query segment with dimension $F_p = 8$. Each query segment is processed locally within the PU as Step ❷. Each PU first computes the tensorized encoding for the received query segment and then returns the obtained partial Hamming similarity to the Hamming margin checker as shown in ❸. During Step ❹, the Hamming margin checker performs the progressive search. It first receives n partial Hamming similarity values from each PU, where n denotes the number of class HVs. It updates the collected partial similarities and checks whether the Hamming margin condition (the largest similarity exceeds the 2nd largest similarity by the threshold value) is satisfied and whether terminates the inference.

Partial encoding. As described above, each PU only receives and encodes a query segment with dimension F_p. The right side of the Fig. 4(a) shows the partial encoding scheme of the query segment with a $M = 2$ order tensorized encoding with sub-matrices dimension of $f_1 = f_2 = 8, d_1 = d_2 = 16$. The partial encoding scheme resolves the imbalances between projection matrix dimension and the BL parallelism. The second dimension of projection sub-matrices d_i is normally larger than the activated BL number $N_{BL} = 8$. If we directly multiply the query segment with entire $\mathbf{r}_i$, the intermediate data need to be cached in additional buffers. To reduce the need for extra buffering, we divide the two projection sub-matrices into four blocks of size $F_p \times N_{BL}$. The query segment each time is only multiplied with one block of each sub-matrix. Each block multiplication is realized using the in-memory in Sec. III-C. In this way, partial HV can be generated in four steps (Fig. 4(a)). The PU loads each F_p-b partial HV output and searches against the stored class HVs immediately as it is generated.

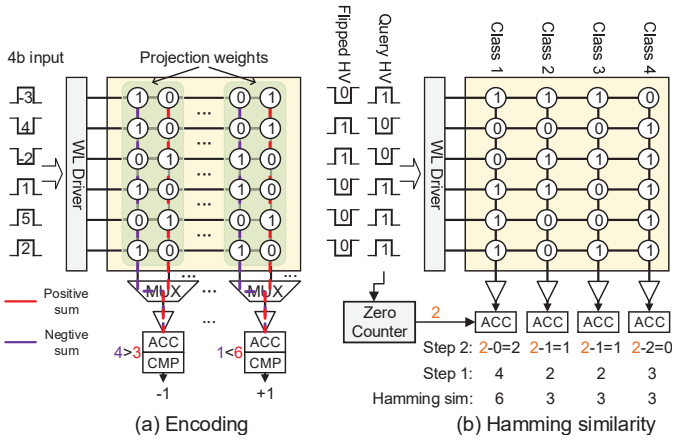


Fig. 5: In-memory (a) Tensorized encoding scheme and (b) two-step Hamming similarity computation.

B. FSL-HD Data Organization

Fig. 4(b) shows FSL-HD's data organization in ReRAM array. Two types of data need to be stored: projection sub-matrices $\mathbf{r}_i$ for encoding and class HVs $\mathbf{C}^j$ for search. For the weights of projection sub-matrices, each PU needs to access the projection weights in their ReRAM array, respectively, since the query segment is distributed and the encoding is processed in every PU. The projection weights are duplicated for each ReRAM array and stored in the 16×64 cells. In this way, each PU can encode using its local projection weights. The duplication of projection weights incurs storage overhead. Storing the M projection sub-matrices needs at least $2 \cdot \sum_{i=1}^{M} d_i \times f_i$ cells for each ReRAM. However, thanks to the significant weight reduction of tensorized encoding method, the 1kb preserved space can support various sub-matrices sizes as well as encoding for different embedding sizes.

Each PU stores the bits of all class HVs corresponding to the partially encoded query. A segment of HVs for all n classes is stored in each PU's ReRAM (Fig. 4(b)). These segments of all n class HVs are stored in the remaining 48×64 cell area. The consecutive bits of each segment are organized horizontally along the BLs to maximize the parallelism for similarity computation.

C. In-memory Tensorized Encoding

Tensorized encoding requires M small matrix multiplications. FSL-HD uses PU's ReRAM array to realize the in-situ matrix multiplication. As shown in Fig. 5(a), each binary $\{-1, +1\}$ weight from the projection sub-matrices $\mathbf{r}_i, (i = 1, \ldots, M)$ is stored in 2-bit 2's complement form within the two consecutive columns in the ReRAM array (i.e., 10 for -1 and 01 for +1). The input of encoding is quantized with 4b and cached in the near-memory 32b buffer. The input data are loaded into the WL driver in a bit-serial manner. The encoding computation is separated into two parts: positive sum and negative sum. When computing the negative sum, the BL columns storing the first bit of weights are continuously activated for 4 cycles to compute the 4b input data. The CSA senses the signal from BL current and accumulates the results in ACC. After the negative sum is obtained, the positive sum is computed similarly using the BL columns that store the second bit of weights. CMP generates the binary encoded output by comparing the values of positive sum and negative sum. Note that the output is not binarized when generating the intermediate results between two projection sub-matrices. In this case, the CMP is bypassed and the output is 4b data.

D. In-memory Progressive Search

We propose a two-step Hamming similarity computation scheme. Fig. 5(b) gives an example of two-step Hamming similarity computation to compute the query HV (100111) against 4 class HVs. First, it counts the number of matched 1s in query and class HV. The query HV is loaded into the WL driver and the number of 0s in query HV is stored in the zero counter. The multiplication results (number of matched 1s) are measured by the CSAs from the BL current. The second step counts the number of matched 0s through loading the flipped bits of query HV into the WL driver to multiply with class HVs. The output of CSA equals to the number of unmatched 0s in query and 1s in class HVs. Hence, the matched 0s for query and class HVs can be calculated by subtracting the digitalized CSA results from the stored zero counting value. Note that the hardware modification cost is low because only one additional zero counter is required for each PU.

E. In-memory FSL-HD Training

The in-memory training process (Eq. 2) is realized via reusing the in-memory tensorized encoding scheme without additional circuits. The encoded query HV bits are stored in the 32b buffer and then written into the ReRAM array. The query HV from the same class is written to the same ReRAM row. The number of rows to aggregate equals to the FSL shot, k. Considering k is normally ≤ 5, the aggregation and binarization can be performed using the same circuits in the encoding.

IV. Results

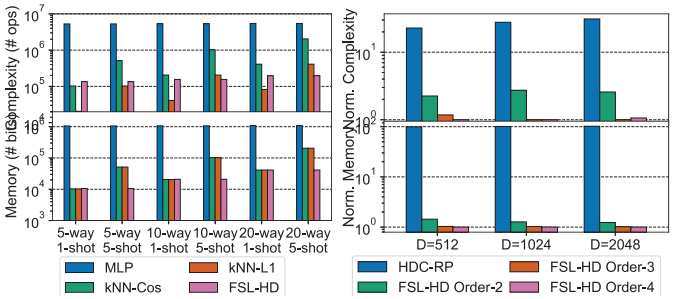
Hardware modeling. We use the 40nm 1T1R cell [13] to construct ReRAM array. The peripheral circuits of PU and Hamming margin checker are implemented using Verilog HDL and synthesized by TSMC 40nm CMOS library. The CSA is from [15] and scaled to 40nm. We input the component characteristics obtained from synthesis into an in-house simulator that emulates the execution behavior at the cell-level granularity.

Benchmarks. We use ResNet-34 [16] (output embedding size $F = 512$) as a front-end feature extractor with ImageNet pre-trained weights. Five popular meta-datasets [17] are used to evaluate the FSL algorithms: Traffic Sign [18], CUB-200 Birds [19], FC100 [20], Texture [21], and Omniglot [22].

FSL baselines. We compare FSL-HD with three baselines: MLP [4], kNN [17] with L1 distance (kNN-L1) [1] and Cosine similarity (kNN-Cosine) [5]. The MLP with two fully connected layers and 512 hidden nodes following the original paper [4]. The k value for kNN is set to 1 [17]. We use six different combinations of FSL parameters: the number of ways is set to 5, 10, 20 and the number of shots is 1 and 5. The test query size is 15 per class and accuracy is calculated using the average of 300 randomly generated FSL tasks. MLP[4], kNN-L1 and kNN-Cosine [17], and FSL-HD algorithms are implemented using PyTorch on a system with Intel i7-8700K with 64GB RAM

TABLE I: Average accuracy gap and ranking

Algorithm	5-way 1-shot	5-way 5-shot	10-way 1-shot
MLP [4]	−1.11% (2.0)	0.00% (1.0)	−0.98% (2.0)
kNN-L1 [1]	−5.94% (4.8)	−6.92% (4.4)	−5.94% (5.0)
kNN-Cosine [5]	−1.56% (2.6)	−3.63% (3.2)	−0.55% (2.0)
HDC-RP	−2.45% (3.0)	−3.47% (3.2)	−1.73% (2.8)
FSL-HD	−1.80% (2.6)	−2.95% (3.2)	−2.00% (3.2)
Algorithm	**10-way 5-shot**	**20-way 1-shot**	**20-way 5-shot**
MLP [4]	0.00% (1.0)	−1.74% (2.8)	−0.33% (1.8)
kNN-L1 [1]	−8.09% (4.6)	−4.83% (4.2)	−7.07% (4.0)
kNN-Cosine [5]	−4.84% (3.0)	−0.41% (1.6)	−3.65% (3.0)
HDC-RP	−3.40% (3.4)	−1.83% (3.4)	−2.44% (2.8)
FSL-HD	−3.03% (3.0)	−1.56% (3.0)	−2.18% (3.4)



(a) (b)

Fig. 6: (a) Inference complexity and memory size comparison of MLP[4], kNN-Cosine[5], kNN-L1[1], and FSL-HD. (b) Complexity and memory size of RP and Tensorized encoding.

and NVIDIA Geforce GTX 1080Ti. The energy consumption of CPU and GPU are measured using Intel Power Gadget and `nvidia-smi` command, respectively.

HDC configurations. We evaluated our design using HV dimensionality, $D \in [512, 4096]$. The tested orders of tensorized encoding are $M = 2, 3, 4$. According to Eq. 1, the dimension of sub-matrices should satisfy $\prod_{i=1}^{M} f_i = F$ and $\prod_{i=1}^{M} d_i = D$. We fix the first dimension of sub-matrix to $f_1 = f_2 = f_3 = 8$ for $F = 512$. The second dimension of sub-matrix is $d_1 = d_2 = d_3 = 8$ for $D = 512$, $d_1 = d_2 = 8, d_3 = 16$ for $D = 1024$, and $d_1 = 8, d_2 = d_3 = 16$ for $D = 2048$, respectively. These f_i and d_i dimension sizes align with the activated word line numbers in the ReRAM version of FSL-HD.

A. FSL-HD Algorithm Evaluation

Comparison to FSL algorithms. Table I compares the accuracy gaps of our proposed FSL-HD, original RP-based HDC (HDC-RP) and three FSL baselines (MLP [4], kNN-L1 [1], kNN-Cosine [5]). The accuracy is quantified using the average accuracy gap between the best accuracy and the average ranking among the five algorithms. FSL-HD yields practically usable accuracy at $D = 2048$. MLP achieves the lowest average accuracy gap on the six FSL problems, while FSL-HD shows the second lowest accuracy gap: 1.5% lower than MLP and 4.2% higher than kNN-L1. Our algorithm offers better accuracy than HDC-RP with significantly lower overhead.

We compare the inference complexity and memory consumption for three FSL baselines and FSL-HD in Fig. 6(a). The inference complexity is defined as the number of binary operations needed for inference. Although MLP [4] achieves the highest average accuracy, MLP consistently requires $10\times$ to $100\times$ higher inference complexity and memory size as compared to the other algorithms. This is due to the complicated matrix multiplications during inference. For 1-shot FSL problems, the complexity of the proposed FSL-HD is higher than of kNN-L1, but in all other settings FSL-HD's complexity and memory requirements are lower since the complexity and memory size of kNN methods grow linearly with the number of the way and the shot. As a result, our algorithm scales well with memory size as the problem sizes scale up.

Effectiveness of tensorized encoding. We evaluate the HDC-based FSL algorithms with and without proposed tensorized encoding. Fig. 6b shows the normalized encoding complexity in terms of the number of binary operations needed and memory size requirements for tensorized encoding with different orders of projection sub-matrices under D from 512 to 2048. As the order M increases, the required complexity and memory size needed for encoding both decrease. Our encoding lowers complexity by $22 - 31\times$ and reduces memory size by $100\times$ compared to the random projection (RP) encoding. For $D = 2048$, we choose order-3 tensorized encoding since it yields the lowest encoding complexity with good accuracy.

Effectiveness of progressive search. We study the impact of the Hamming margin threshold values (from progressive search) on the accuracy and latency reduction for 20-way 5-shot problem (see Fig. 7). The Hamming margin threshold ranges from 20 to 60 based on the observation in Fig. 2(b). The accuracy is compared with the exhaustive search method across five datasets. Smaller margin threshold leads to more significant latency reduction with lower accuracy. The margin threshold $= 30$ yields 39.8% to 51.3% latency reduction with negligible accuracy loss. The trend is similar on other FSL problem settings. The progressive search provides a valuable tradeoff between searching latency and accuracy.

B. Performance and Energy Evaluation

Fig. 7 shows the performance and energy comparison of MLP [4], kNN [1, 5] and FSL-HD algorithms on CPU, GPU, and ReRAM with six FSL settings. FSL-HD-ReRAM and FSL-HD-ReRAM-Prog. denote the proposed PIM accelerator without and with progressive search enabled, respectively. The speedup and energy efficiency improvement are computed by normalizing the worst latency and energy consumption of each FSL setting to 1. FSL-HD-ReRAM is $32-66\times$ faster than FSL-HD running on CPU and GPU and achieves $10-66\times$ speedup over kNN-CPU and $7-9\times$ speedup over kNN-GPU. Although MLP's has over $10\times$ complexity than kNN and FSL-HD, the optimized matrix multiplication on CPU and GPU make the speedup of kNN and FSL-HD over MLP less significant.

FSL-HD-ReRAM is over $100\times$ and $1000\times$ more energy efficient as compared to kNN-CPU and FSL-HD-GPU, respectively. As the FSL problem size grows, the energy efficiency gain of FSL-HD-ReRAM over CPU and GPU baselines is more significant thanks to the efficient in-situ ReRAM computing that reduces the data movement overhead. Moreover, enabling the progressive search for FSL-HD-ReRAM-Prog. generates additional $1.9\times$ to $2.8\times$ speedup and energy efficiency improvements over FSL-HD-ReRAM due to reduced computations.

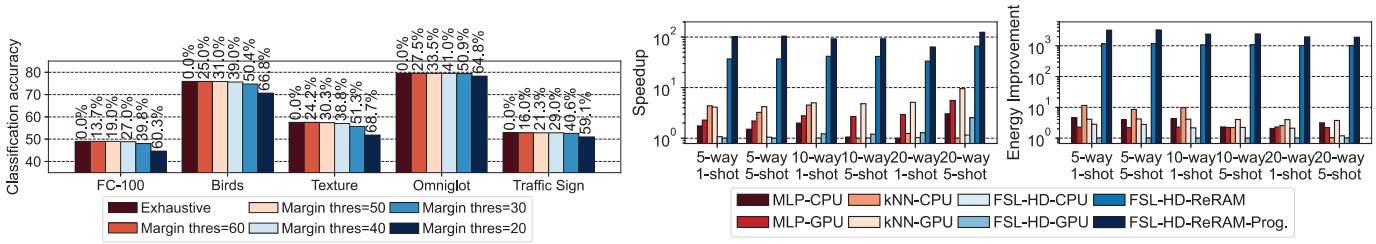


Fig. 7: **Left**: Classification accuracy and latency reduction (% over each bar) using progressive search with different Hamming margin thresholds for the 20-way 5-shot problem. **Right**: Performance and energy comparison.

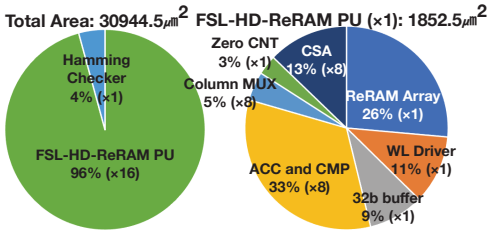


Fig. 8: Area breakdown of FSL-HD-ReRAM.

C. Comparison with PIM-based FSL Classifiers

Scalability comparison. Our proposed accelerator has better scalability compared to the PIM accelerator, SAPIENS [7]. SAPIENS only supports image embedding size $F = 32$, maximum $n = 32$ ways, and only one-shot learning. Due to the linearly growing memory consumption of kNN (see Fig. 1), it is difficult to fit into the limited memory space when F and FSL problem sizes increase. In comparison, the memory requirement of FSL-HD is independent of F and the number of FSL shots. It is a function of HV dimensionality D and the number of FSL way n. FSL-HD-ReRAM can support various image embedding sizes that range up to $F = 1024$ through changing the sub-matrices dimension f_i of tensorized encoding. For example, for $48 \times 64 \times 16 = 48$kb memory, the supported class number is from $n = \frac{48k}{2k} = 24$ for $D = 2048$ to $n = \frac{48k}{512} = 96$ for $D = 512$. It implies that our design scales well with other CNN-based feature extractors that have output feature dimensions from 512 to 1000.

Latency comparison. FSL-HD-ReRAM without progressive search has shorter processing latency under the same image embedding size compared to SAPIENS [7]. The latency of SAPIENS is linearly scaled with the embedding size F. To process the image embedding with size $F = 512$, SAPIENS needs 10,240 ns for inference while FSL-HD-ReRAM only needs 4,160 ns, which is 2.5× faster.

FSL-HD area and overhead. The hardware area breakdown of 40nm FSL-HD-ReRAM is shown in Fig. 8. FSL-HD-ReRAM with total of 16 PUs requires 30,944.5 μm^2 area. The peripheral circuitry accounts for over 70% of the FSL-HD PU area. Compared to the standard 1T1R ReRAM computing macro [13], the 32b buffer, zero CNT, and the top-level Hamming checker are the three additional modules in FSL-HD-ReRAM. These modules incur 18.6% area overhead. We compare 64kb FSL-HD-ReRAM with the state-of-the-art kNN-L1-based FSL design (64kb), SAPIENS [7]. FSL-HD-ReRAM has 84% less area while yielding higher accuracy. This is because SAPIENS constructs the ReRAM for multi-bit L1 distance search using 2T2R, which decreases the memory density.

V. Conclusion

This paper presents FSL-HD, an in-memory acceleration framework for FSL based on HDC. Our algorithm provides comparable accuracy to the state-of-the-art MLP, but runs faster speed, with less memory and complexity. We also design novel tensorized encoding, and progressive search during inference. We use the physically-verified 40nm 1T1R ReRAM [13] to accelerate FSL-HD. Our in-memory dataflow, organization, and energy reduction schemes improve FSL-HD's efficiency in PIM. The experiments demonstrate that FSL-HD-ReRAM achieves 4.2% accuracy improvement over kNN classifier [1]. Also, it is 2.5× faster than state-of-the-art ReRAM design, SAPIENS [7], while requiring 84% less area.

Acknowledgements

This work was supported in part by CRISP, one of six centers in JUMP (an SRC program sponsored by DARPA), SRC Global Research Collaboration (GRC) grant, and NSF grants #1826967, #1911095, #2003279, #2112665, #2112167, and #2100237.

References

[1] J. Snell *et al.*, "Prototypical networks for few-shot learning," *NeurIPS*, 2017.
[2] O. Vinyals *et al.*, "Matching networks for one shot learning," *NeurIPS*, 2016.
[3] C. Finn *et al.*, "Model-agnostic meta-learning for fast adaptation of deep networks," in *ICML*, 2017.
[4] A. Chowdhury *et al.*, "Few-shot image classification: Just use a library of pre-trained feature extractors and a simple classifier," in *ICCV*, 2021.
[5] W.-Y. Chen *et al.*, "A closer look at few-shot classification," in *ICLR*, 2019.
[6] Y. Tian *et al.*, "Rethinking few-shot image classification: a good embedding is all you need?" in *ECCV*. Springer, 2020.
[7] H. Li *et al.*, "Sapiens: A 64-kb rram-based non-volatile associative memory for one-shot learning and inference at the edge," *TED*, 2021.
[8] G. Karunaratne *et al.*, "Robust high-dimensional memory-augmented neural networks," *Nature communications*, vol. 12, no. 1, pp. 1–12, 2021.
[9] D. Reis *et al.*, "Attention-in-memory for few-shot learning with configurable ferroelectric fet arrays," in *ASP-DAC*, 2021.
[10] M. Imani *et al.*, "Floatpim: In-memory acceleration of deep neural network training with high precision," in *ISCA*. IEEE, 2019.
[11] J. Morris *et al.*, "Locality-based encoder and model quantization for efficient hyper-dimensional computing," *TCAD*, 2021.
[12] X. Zhang *et al.*, "Fast orthogonal projection based on kronecker product," in *ICCV*, 2015.
[13] S. D. Spetalnick *et al.*, "A 40nm 64kb 26.56 tops/w 2.37 mb/mm 2 rram binary/compute-in-memory macro with 4.23 x improvement in density and¿ 75% use of sensing dynamic range," in *ISSCC*, vol. 65. IEEE, 2022, pp. 1–3.
[14] C.-X. Xue *et al.*, "A 22nm 4mb 8b-precision reram computing-in-memory macro with 11.91 to 195.7 tops/w for tiny ai edge devices," in *ISSCC*, 2021.
[15] T. Na *et al.*, "Offset-canceling current-sampling sense amplifier for resistive nonvolatile memory in 65 nm cmos," *JSSC*, vol. 52, no. 2, pp. 496–504, 2016.
[16] K. He *et al.*, "Deep residual learning for image recognition," in *CVPR*, 2016.
[17] E. Triantafillou *et al.*, "Meta-dataset: A dataset of datasets for learning to learn from few examples," *arXiv preprint arXiv:1903.03096*, 2019.
[18] S. Houben *et al.*, "Detection of traffic signs in real-world images: The german traffic sign detection benchmark," in *IJCNN*, 2013.
[19] P. Welinder *et al.*, "Caltech-UCSD Birds 200," Caltech, Tech. Rep., 2010.
[20] B. Oreshkin *et al.*, "Tadam: Task dependent adaptive metric for improved few-shot learning," *NeurIPS*, 2018.
[21] M. Cimpoi *et al.*, "Describing textures in the wild," in *CVPR*, 2014.
[22] B. M. Lake *et al.*, "Human-level concept learning through probabilistic program induction," *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.