HDVQ-VAE: Binary Codebook for Hyperdimensional Latent Representations

Austin J Bryant , *Graduate Student Member, IEEE* and Sercan Aygun , *Member, IEEE*

***Abstract*—Hyperdimensional computing (HDC) has emerged as a promising paradigm offering lightweight yet powerful computing capabilities with inherent learning characteristics. By leveraging binary hyperdimensional vectors, HDC facilitates efficient and robust data processing, surpassing traditional machine learning (ML) approaches in terms of both speed and resilience. This paper addresses key challenges in HDC systems, particularly the conversion of data into the hyperdimensional domain and the integration of HDC with conventional ML frameworks. We propose a novel solution, the Hyperdimensional Vector Quantized Variational Auto Encoder (HDVQ-VAE), which seamlessly merges binary encodings with codebook representations in ML systems. Our approach significantly reduces memory overhead while enhancing training by replacing traditional codebooks with binary $(-1,+1)$ counterparts. Leveraging this architecture, we demonstrate improved encoding-decoding procedures, producing high-quality images within acceptable peak signal-to-noise ratio (PSNR) ranges. Our work advances HDC technology by considering efficient ML system deployment to embedded system environments.**

***Index Terms*—Binary Encoding, Hyperdimensional Computing (HDC), Machine Learning (ML) Integration, Vector Quantized Variational Auto Encoder (VQ-VAE).**

I. Introduction

Hyperdimensional Computing (HDC) has recently received significant attention from researchers due to its ability to provide lightweight yet efficient computing coupled with certain learning characteristics [1]. HDC offers both speed and robustness compared to conventional machine learning (ML), owing to its unconventional data representation. Utilizing high-dimensional symbolic binary vectors, known as *hypervectors*, HDC performs various ML tasks [2]. These hypervectors possess the inherent property of holographic information storage [3]. By employing binary $(-1,+1)$ hyperdimensional vectors, HDC achieves holographic information storage [4], encompassing multiple incoming data within a single binary vector, with lower resource consumption than conventional binary radix systems. The dimensionality (D) of hypervectors typically ranges from $1K$ to $10K$, with vectors necessitating near orthogonality for unbiased symbol representation. This unique orthogonality property facilitates *bundling*, *binding*, and *permuting* operations for encoding purposes, enabling a single vector to represent multiple inputs [5]. Bundling hypervectors yields a hypervector most similar to all addends. The binding operation results in a hypervector most dissimilar to the set of hypervectors bound, akin to coordinate-wise multiplication. Permutation, a circular shift, generates uncorrelated hypervectors, preserving orthogonality [6].

While HDC is a promising path toward lightweight ML applications, much consideration must be given to how data is converted to its symbolic representation. This paper focuses on helping streamline the encoding process using unsupervised learning. In contrast to conventional ML methods, which often employ continuous, real-valued vectors such as those in Word2Vec [7], the HDC approach utilizes a discrete, symbolic representation. This enables robust, memory-efficient holographic information storage using hyperdimensional binary vectors, reducing the overhead associated with floating-point representations. In traditional ML approaches for symbol processing, vectors are processed based on element-wise positions, where each element in the vector has a specific meaning tied to its position. Similarly, HDC employs positional encoding in binary vectors. However, HDC also uses binding and bundling logic operations to combine information, preserving relationships within the data in a compact form. While conventional models utilize dimensionality reduction techniques to connect subgroups of vectors and measure similarity between words, HDC uses

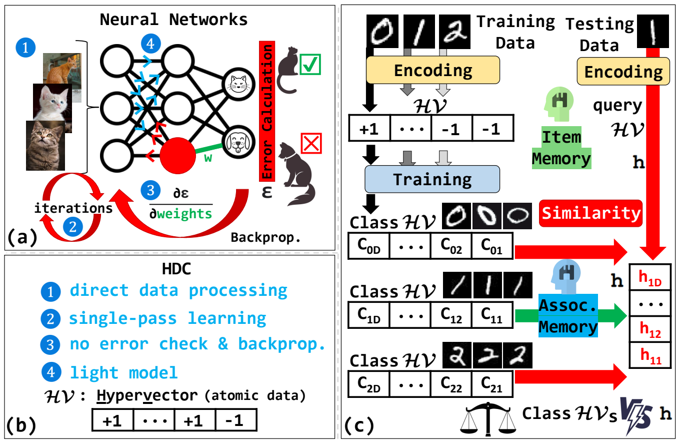

Fig. 1. From traditional ML to HDC: (a) Conventional multi-layered neural network, (b) HDC specs over the conventional ML, (c) Training and testing procedures in HDC; MNIST image dataset case study.

streamlined binary similarity measures, like cosine similarity, to infer relationships between hypervectors. This work uses conventional ML to help select symbols for representing abstract data. To the best of our knowledge, for the first time in the literature, Vector Quantized Variational Auto Encoder (VQ-VAE) architecture is decorated with an HDC approach for an efficient latent space representation without the loss of generality. The VQ-VAE codebook's discrete aspect aligns with standard encoding techniques for HDC, where a discrete symbol list or *codebook* is used. In addition, the *encoder*, *decoder*, and *quantization* process of the VQ-VAE aligns with our goal of learning to map abstract data to a set of symbols in an unsupervised manner.

Once the codebook is obtained, the classification task becomes the second component of the overall learning architecture. Traditional systems typically rely on conventional floating-point memory representation involving complex operations such as *multiply-accumulate-activate*. However, in HDC-based classifiers, a single-pass learning strategy is employed, diverging from traditional ML systems. This study integrates a binary codebook trained using HDVQ-VAE with the learning strategy of the HDC paradigm. The latent space elements undergo further processing through binding and bundling to create a classifier system. Results indicate that this proposed framework offers an easily deployable architecture with reduced memory usage while maintaining accuracy levels comparable to conventional approaches. This paper presents the following contributions:

❶ Integration of HD mechanics into VQ-VAE architecture via codebook replacement.
❷ Enable the use of downstream HDC applications that take advantage of HD latent space.
❸ Reduction of latent space by 32× from the transition of float32 codebook to the binary codebook.
❹ Reduction of overall model size due to codebook replacement.

The structure of this paper is as follows: Section II presents the background, and the methodology of the HDVQ-VAE is given in Section III. Section IV reviews the results and compares them to conventional VQ-VAE, and Section V concludes the paper.

II. Background

A. Hyperdimensional Computing (HDC)

HDC is an emerging computing paradigm gaining attention, particularly in developing ultra-lightweight learning systems [8]. In comparison to conventional ML approaches depicted in Fig 1 (a), HDC, outlined in Fig 1 (b), offers direct data processing without the need for additional feature engineering, supports single-pass learning, eliminates the requirement for error optimizations and backpropagation, and achieves a lightweight model thanks to unary processing capabilities.

Manuscript received XXXXXXXXXX XX, 2024; accepted XXXXXXXXXX XX, 2024. Date of publication XXXXXXXXXX XX, 202X; date of current version XXXXXXXXXX XX, 202X. This article was recommended by Associate Editor XXXXXXXXX XXXXXXXXX. *(Austin J. Bryant and Sercan Aygun contributed equally to this work.) (Corresponding author: Austin J. Bryant.)*

This work was supported in part by the National Science Foundation under Grant 2019511 and by the start-up fund provided by the University of Louisiana at Lafayette for Dr. Sercan Aygun.

The authors are with the School of Computing and Informatics, University of Louisiana at Lafayette, Lafayette, LA, USA (E-mail: {austin.bryant1, sercan.aygun}@louisiana.edu).

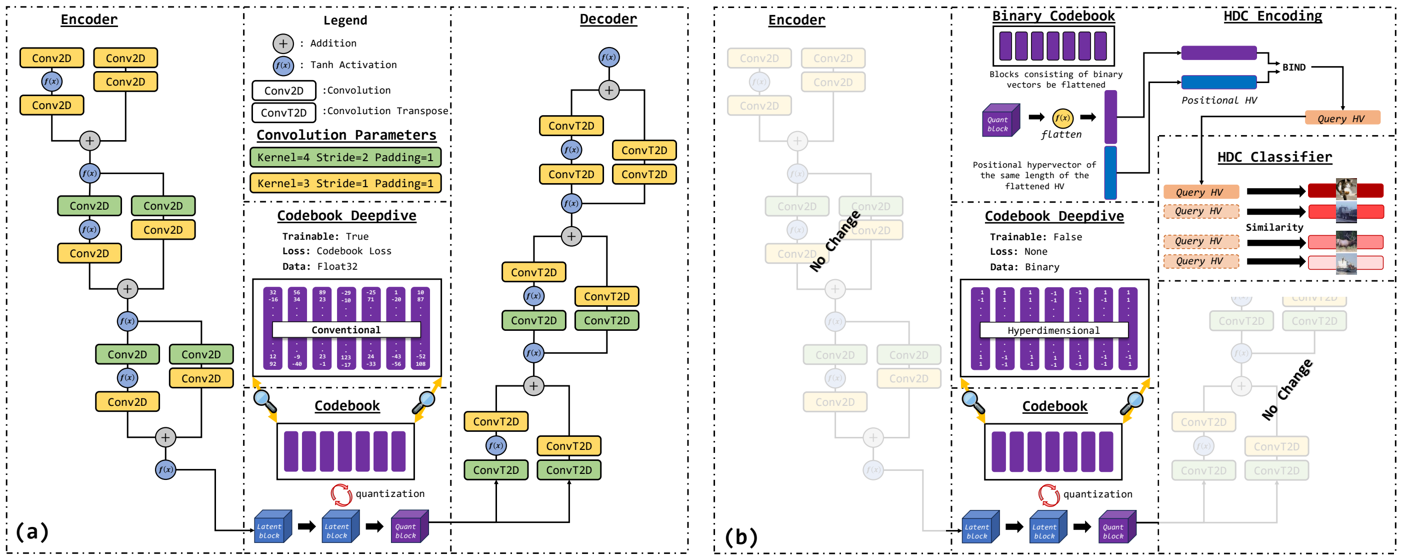

Fig. 2. Vector Quantized Variational Auto Encoder: (a) Conventional architecture fine-tuned for this work, (b) HDVQ-VAE: Proposed architecture using HDC for latent spaces; the new classifier part is in HDC learning approach.

Encoding data into hypervectors and applying several logic operations results in the learned class of the training sample, unlike the learning steps (*forward pass - backward pass*) in traditional neural network systems, which require several optimization steps.

HDC excels in representing symbols, such as letters, image pixel positions, and 1-D signal timestamps, by leveraging orthogonal binary vectors. Each vector comprises D dimensions and consists of randomly occurring $+1$ and -1 values (interpreted as logic-1 and logic-0 in a hardware environment).

For symbol representation, since it is not a numerical value, a vector evaluation follows the unbiased probability point between 0 and 1. Assuming the middle point of the overall probability is 0.5 for each symbol vector representation, no bias exists between generated vectors. The probability presents having a logic value of 1 over the total vector size, thus having equal probability for each symbol.

On the other hand, scalar values, including 1-D signal amplitude and image pixel intensity, are also effectively encoded within HDC systems. The minimum and maximum scalar values within the space are mapped to $-1-1-1-1...-1$ (minimum hypervector) and $+1+1+1+1...+1$ (maximum hypervector), respectively. For example, when encoding grayscale image pixel intensities, which range from 0 to 255 for 8-bit representation, a pixel intensity of 0 is represented as $-1-1-1-1...-1$ in hypervector space, while a pixel intensity of 255 is represented as $+1+1+1+1...+1$. Any value within a range is represented by randomly flipping bits from -1 to $+1$ (or vice versa) based on the magnitude of the value. Larger values correspond to patterns resembling the maximum hypervector, while smaller values resemble the minimum hypervector [9].

Once vector generation is complete, the data undergoes subsequent encoding steps, such as binding, shifting, permutation, and bundling tailored to the specific application. Applications primarily use two types of encoding: n-gram-based and record-based encoding. In symbol-based applications like language processing, n-gram operation involves handling n symbols (letters) alongside their corresponding hypervectors. Any n group of symbols are combined using shifting and permutation of their corresponding hypervectors [10]. For instance, n=3-gram operations on text processing take three sub-sets of letter hypervectors in a sentence, shift and permute each 3-group vector, and multiply them through the binding operation. Multiplications of $+1$ and -1 binary values are implemented using XOR operations in hardware, and the final result is another hypervector. The resultant hypervectors (n-gram vectors) from each n-gram sub-set are then cumulatively aggregated (achieved through population count operations in hardware) to produce a single vector from multiple n-gram inputs. This accumulation is a bundling operation and facilitates the holistic learning property of HDC. The final step involves binarization of the accumulated values back to binary form. This process, customized for each dataset class, generates a unified 1-D vector representing the respective class. Each new sample contributes to this resultant class hypervector by undergoing the same encoding steps [11]. Inference follows the same encoding steps outlined above; classification entails a similarity assessment between the test sample vector and the class hypervectors. A higher similarity score (measured through metrics such as cosine similarity, Hamming distance, dot product, etc.) indicates a closer match between the class and the test hypervector, yielding classification results. Fig 1 (c) illustrates the training and inference flow within the context of image processing using HDC. Each image class in MNIST dataset contributes to the corresponding class hypervector in associative memory. During testing, following the same encoding steps, all class hypervectors are sequentially compared via similarity scores with the query hypervector from item memory.

B. Vector Quantized Variational Auto Encoder (VQ-VAE)

Variational Auto Encoders (VAE) are used in many applications, such as generative image diffusion models, to produce a compressed latent space that is computationally less expensive to perform operations. While conventional VAEs produce continuous compressed representations, the VQ-VAE architecture learns compressed *discrete* representations. The VQ-VAE architecture introduces a quantization process that replaces the reparametrization trick of conventional VAEs. [12]. The VQ-VAE's quantization process transforms the continuous representations into discrete ones. This quantization is performed by calculating the distance between features and a codebook, where a codebook is defined as a finite symbol list. Once distances are calculated, each feature is replaced with the nearest symbol. This design choice provides an opportunity to utilize the codebook to represent discrete outputs in a vector space. The VQ-VAE compresses and reconstructs abstract data through the structured interplay between the encoder and decoder, in which data is encoded into a symbolic representation and then decoded back into its original representation [12]. This paper is motivated by the shared goal of encoding for HDC and the VQ-VAE encoding process, mapping data to a discrete symbolic representation. Fig. 2 (a) illustrates conventional VQ-VAE architecture.

Conventional VQ-VAE uses three main loss metrics to guide its training: *Mean Square Error* (MSE), *Codebook Loss*, and *Commitment Loss*. The MSE is calculated between the reconstructed and original data. Codebook loss and commitment loss are nearly identical. The only difference between them is which set of vectors is detached to force the other to be considered during backpropagation. Both are represented as the MSE of the encoder output vectors and the nearest codebook vectors (*quantized* vector). The codebook loss detaches the gradients of the raw encoder output. This detachment of the gradients forces the codebook or symbol list to move toward the raw output of the encoder. The commitment loss works the other way around, detaching the quantized or symbol representation. This forces the encoder to minimize the distance between its output and the codebook. These detachments are used to train the codebook and encoder at separate rates. The codebook should be restrained from adjusting drastically to stabilize training. This is enforced by scaling down the codebook loss. The commitment loss forces the encoder to 'commit' to the symbols in the codebook. In what follows, the losses in the conventional VQ-VAE are given:

$$\begin{aligned}\textbf{MSE Loss} &= \tfrac{1}{N}\textstyle\sum_{i=1}^{N}(x_{\text{recon},i}-x_i)^2\\ \textbf{Codebook Loss} &= \text{MSE}(\text{quantized}, x_{\text{detach}})\\ \textbf{Commitment Loss} &= \text{MSE}(\text{quantized}_{\text{detach}}, x)\end{aligned}$$

III. HDVQ-VAE Framework

The proposed architecture builds upon a well-known conventional ML module: the VQ-VAE, with an update of binary data representation to incorporate orthogonal vectors, resulting in the HDVQ-VAE. This adaptation offers a lightweight, easily deployable, and more efficient classifier utilizing the encoding dynamics of the HDC paradigm. Our architecture comprises *down blocks*, *up blocks*, and *quantization* modules. The down blocks consist of dual sequential modules, each comprising two convolution layers through which the input passes before undergoing addition and being fed through the *tanh* activation function. The first of these dual sequential modules features a pre-activation *tanh* function situated between the convolution layers. Fig. 2 (a) illustrates the conventional, and Fig. 2 (b) shows the proposed HDVQ-VAE architecture.

By replacing the convolution layers in the down block with convolutional transpose layers, we created our up block that aims to reverse the operation performed by the down block. The quantization module holds the codebook and is responsible for mapping latent vectors to this codebook. In our proposal, we replace the conventional codebook with a binary $(-1, +1)$ representation for our architecture. This allows us to utilize hypervectors or sections of hypervectors as our codebook. We initialize these codebook vectors using pseudo-random sequences. In HDC, where orthogonality is essential for symbol representation, random sources provide this orthogonality.

Once the codebook is initialized, it remains fixed throughout the overall architecture without further updates, unlike a conventional VQ-VAE. This enables us to eliminate the codebook and commitment loss. We maintain the straight-through estimator to propagate gradients from one side of the quantization module to the other [12]. The codebook can comprise either hypervectors or sections of hypervectors. If the codebook consists of sections of hypervectors, then the latent space must contain multiple vectors that, when flattened, form a hypervector. Alternatively, if the codebook directly employs hypervectors, the quantized latent will encompass multiple hypervectors.

We have significantly streamlined the training process by eliminating the codebook and commitment loss in our modified VQ-VAE model. The static codebook does not require updates during training, making the codebook loss unnecessary. Similarly, the binary nature of the codebook diminished the benefits of a commitment loss. In addition to the conventional loss function, we have opted to create loss functions ($_L$) from the well-known metrics to help guide the network: peak signal-to-noise ratio (PSNR) [13], structural similarity (SSIM), learned perceptual image patch similarity (LPIPS) [14], and perplexity (PPLX) [15]:

$$\textbf{PSNR}_L = 1.0 - \frac{\text{PSNR}(x_i, x_{\text{recon}})}{48} \quad || \quad \textbf{SSIM}_L = 1.0 - \text{SSIM}(x_i, x_{\text{recon}})$$
$$\textbf{LPIPS}_L = \frac{1}{N}\sum_{i=1}^{N}\text{LPIPS}(x_i, x_{\text{recon},i}) \quad || \quad \textbf{PPLX}_L = \frac{1}{\exp(-\sum p_i \log p_i)}$$

In our experiment, the training procedure uses the loss of the reconstructed images and their originals to guide the model toward producing meaningful latent representations. The training procedure is as in Algorithm 1.

In addition to utilizing HDC for binary latent space representation, the classifier component is also retained within the HDC domain to further leverage the latent space. Fig. 2 (b) further illustrates the HDC classifier specifically designed for the codebook in the HDC domain. The process involves multiplying (binding) and accumulating flattened vectors with positional hypervectors for each corresponding class. The positional vectors coincide with our latent block's spatial dimensionality of 24×24. Initializing our positional vectors involves assigning each spatial position a distinct binary. The distinctness of these vectors is ensured by the random assignment of bits to 0 or 1 with a 0.5 probability of either. Similarly to the quantized latent block, we flatten the positional vectors into a single hypervector that is then bound with the flattened quantized latent block. When evaluating (inference phase) incoming query samples, they are similarly encoded in the hypervector domain and compared to each previously generated class hypervector. The highest score determines the classification result. Consequently, this paper introduces two primary proposals to be tested in the subsequent section: binary latent spaces and codebook classification using HDC classifier.

Algorithm 1 HDVQ-VAE Training Procedure

```
1:  Initialize:
2:      Model parameters use default Pytorch initialization
3:      ★codebook is initialized as a stack of non-trainable binary vectors
4:  for each epoch do
5:      for each batch in training data do
6:          Encode:
7:              Input data is $B \times 3 \times 96 \times 96$
8:              Result is latent block of size $B \times 128 \times 24 \times 24$
9:          Quantize:
10:             Compute distances to ★codebook vectors
11:             Assign each latent vector to the nearest ★codebook vector
12:         Decode: Generate reconstruction from quantized latent block
13:         Calculate Losses ❇ :
14:             Compute Reconstruction loss as $MSE(x_{\text{recon}}, x)$
15:             Compute Perplexity loss as $\frac{1}{\exp(-\sum p_i \log p_i)}$
16:             Compute PSNR loss as $1.0 - \frac{\text{PSNR}(x, x_{\text{recon}})}{48}$
17:             Compute SSIM loss as $1.0 - \text{SSIM}(x, x_{\text{recon}})$
18:             Compute LPIPS loss as $\frac{1}{N}\sum_{i=1}^{N}\text{LPIPS}(x_i, x_{\text{recon},i})$
19:             Add all losses to calculate the total loss for the batch
20:         Backpropagate: Update model parameters based on total loss
21:     end for
22: end for
    ★ The codebook consists of binary vectors, unlike conventional VQ-VAEs.
    ❇ Codebook loss is not used in HDVQ-VAE.
    Binary vector size is taken as 128.
```

IV. Experiments and Results

In this section, we run two groups of experiments considering our two main contributions: the performance of HD latent space for reconstruction and latent space performance in the classification problem. Using the architecture in Fig. 2 (b), we first consider the performance of the binary HD latent space. The images are constructed based on the STL10 dataset [16], and the construction performance of the HDC-based approach is in the acceptable performance range.

The experiments were conducted using Intel i7-9750h CPU, 16GB of RAM at 2400MHz, and an Nvidia Geforce GTX 1650. Pytorch and Torchvision provided the functionality needed for the experiments. Our experiments use the STL10 dataset [16]. We chose the STL10 dataset for its larger 96×96×3 ($W\times H\times C$) size compared to MNIST and CIFAR datasets. The larger size allowed us to test the reconstruction of higher-quality images where the codebook vectors may need to represent more details with a challenging problem. In the case of our HDVQ-VAE, where the codebook does not need to be adjusted, encoding high-quality vectors is crucial for passing vital detail information to the decoder.

The additional losses focused on detail help guide the model in the selection of codebook vector to best pass detail information from the encoder to the decoder. The HDVQ-VAE can reconstruct images acceptably, as can be seen in Fig. 3.

Once trained, we can produce binary latent vectors using the encoder and binary codebook in Fig. 2 (b). We use the latent vectors to test the classification abilities of the conventional method with respect to the convolutional neural network (CNN). The CNN classifier's architecture consists of three convolution 2D layers, a flattened layer, and two fully connected linear layers connected sequentially. Specifically, the architecture includes convolutional layers with input-output channel pairs (128, 64, 32, and 8), each followed by a *tanh* activation function. Then, a single flattened layer is placed at the end of the convolutional layers. The flattened output is passed through two fully connected linear layers, transitioning from 72 to 32 features, followed by a ReLU activation and then from 32 to 10 features. The CNN classifier was trained using a '*reduce on plateau*' scheduler starting at a learning rate of 0.003 with a scale factor of 0.5 and a threshold of 0.01 for our cross-entropy loss. As seen in Fig.s 4 (a) and (b), the CNN classifier nearly doubled in evaluation accuracy while more than tripling in training accuracy. Fig.s 4 (c) and (d) report the testing phase confusion matrix for conventional and HD approaches, respectively.

For our HDC classification test, we reshape the binary latent vectors from our HDVQ-VAE, flattening them into a much higher-dimensional vector, which forms our hypervector. The flatting operation brings our latent representation from $B \times 128 \times 24 \times 24$ to $B \times 73728$ where B is our batch dimension, and 128 is the binary vector size. Our new hypervector is bound with a static positional hypervector initialized

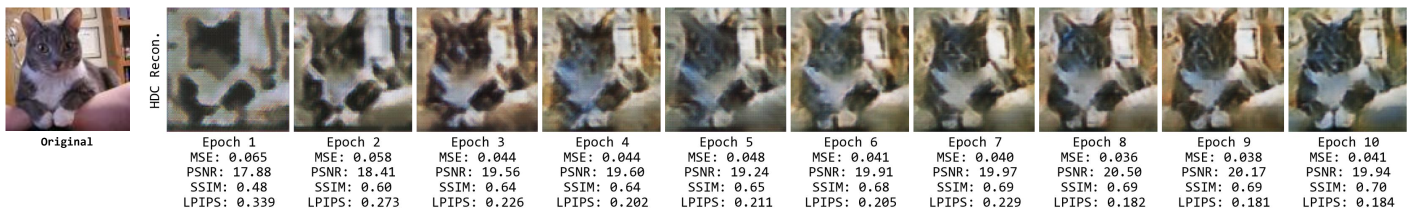
Original
HDC Recon.
Epoch 1 MSE: 0.065 PSNR: 17.88 SSIM: 0.48 LPIPS: 0.339
Epoch 2 MSE: 0.058 PSNR: 18.41 SSIM: 0.60 LPIPS: 0.273
Epoch 3 MSE: 0.044 PSNR: 19.56 SSIM: 0.64 LPIPS: 0.226
Epoch 4 MSE: 0.044 PSNR: 19.60 SSIM: 0.64 LPIPS: 0.202
Epoch 5 MSE: 0.048 PSNR: 19.24 SSIM: 0.65 LPIPS: 0.211
Epoch 6 MSE: 0.041 PSNR: 19.91 SSIM: 0.68 LPIPS: 0.205
Epoch 7 MSE: 0.040 PSNR: 19.97 SSIM: 0.69 LPIPS: 0.229
Epoch 8 MSE: 0.036 PSNR: 20.50 SSIM: 0.69 LPIPS: 0.182
Epoch 9 MSE: 0.038 PSNR: 20.17 SSIM: 0.69 LPIPS: 0.181
Epoch 10 MSE: 0.041 PSNR: 19.94 SSIM: 0.70 LPIPS: 0.184

Fig. 3. Generative performance of the latent spaces in HDC domain. The original cat image is used as a reference for the performance metric calculations.

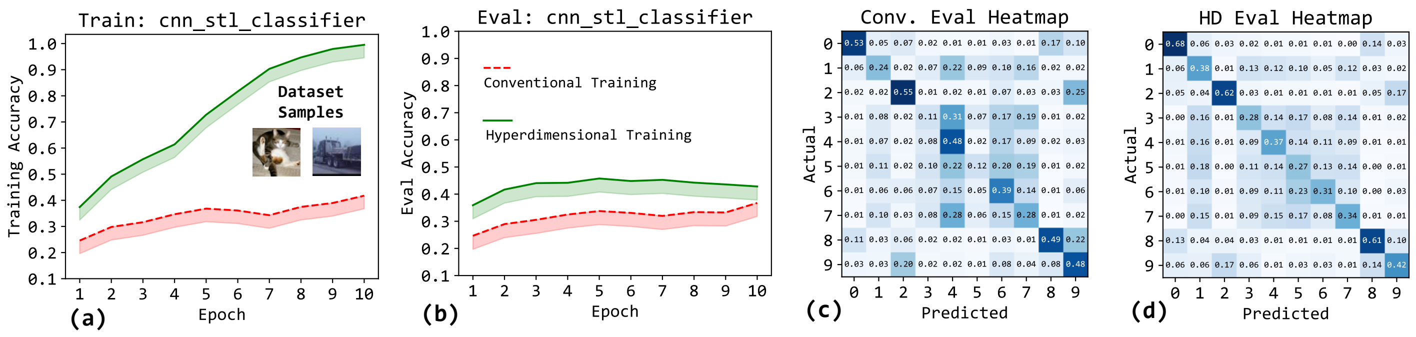

Fig. 4. CNN-based conventional classifier performance over the scalar latent space and HD latent space classification using the CNN model. (a) Training accuracy for conventional and HD models, (b) Testing accuracy for conventional and HD models, (c) Testing phase confusion matrix of the conventional model, and (d) Testing phase confusion matrix of the HD model.

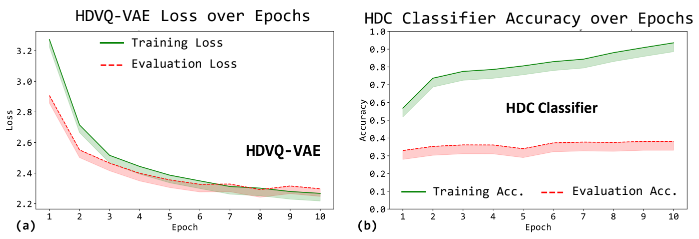

Fig. 5. (a) HDVQ-VAE loss performance and (b) Plain HDC classifier accuracy performance for the binary latent space.

at the beginning of training. Once bound, we do a single pass to accumulate vectors for their respective classes before signing. As we begin training, we take similarity measurements between the bound hypervector (*latent* $\times$ *position*) and our class vectors. We adjust the class vectors based on incorrect predictions, pushing away from the incorrectly predicted class and toward the correct one. At the end of each epoch, we once again sign the class vectors.

While training an HDC classifier does not use backpropagation, it outperformed conventional classifier models on loss and accuracy scores, as seen in Fig. 5. VQ version of VAE is more compliant with HDC than CNN due to the intricate learning of the quantization process. Not only is HDC computationally less expensive, but it is also lightweight, making it a suitable candidate for embedded systems.

The binarization of the codebook for our HDVQ-VAE brings memory savings and an overall reduction in the model size. Going from float32 to a binary representation shrinks the codebook size by 32×, and the information passed to the decoder is 32× smaller than that of a conventional VQ-VAE. Despite this 32× reduction in the information flow, comparing downstream latent classification, both the conventional and HDC classifiers perform better when using the latent of the HDVQ-VAE over the conventional VQ-VAE.

V. Conclusions

This study introduces a novel approach to enhance vector quantized variational autoencoders by incorporating hyperdimensional vector representations, aiming for a more efficient and lightweight solution in generative artificial intelligence (AI) and latent space classifications. By leveraging a binary hypervector space for latent representations, the codebook within the vector space can be utilized for image reconstructions or classifications. This research marks the inception of hyperdimensional computing (HDC) in machine learning (ML), suggesting its potential as a valuable addition to the conventional ML framework in future endeavors. While HDC alone may not suffice for end-to-end generative AI tasks, its potential applications in compact and efficient designs for embedded environments are promising. This study demonstrates a memory-efficient model through binarization, with the classifier component integrated via HDC learning dynamics, resulting in a lightweight yet improved classifier compared to conventional approaches such as convolutional neural networks.

Acknowledgments

The open-source code for this work can be found at https://github.com/tMLEClab/HDVQ-VAE.

References

[1] L. Ge and K. K. Parhi, “Classification using hyperdimensional computing: A review,” *IEEE Circuits and Systems Magazine*, vol. 20, pp. 30–47, 2020.
[2] S. Aygun, M. S. Moghadam, M. H. Najafi, and M. Imani, “Learning from hypervectors: A survey on hypervector encoding,” *arXiv preprint arXiv:2308.00685*, 2023.
[3] C. Yeung, Z. Zou, and M. Imani, “Generalized holographic reduced representations,” 2024, arXiv 2405.09689.
[4] P. Kanerva, “Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors,” *Cognitive Computation*, vol. 1, no. 2, pp. 139–159, Jun 2009.
[5] M. Stock, D. Boeckaerts, P. Dewulf, S. Taelman, M. V. Haeverbeke, W. V. Criekinge, and B. D. Baets, “Hyperdimensional computing: a fast, robust and interpretable paradigm for biological data,” 2024.
[6] A. Rahimi, P. Kanerva, and J. M. Rabaey, “A robust and energy-efficient classifier using brain-inspired hyperdimensional computing,” in *ISLPED*, ser. ISLPED ’16, p. 64–69.
[7] A. G. Ayar, S. Aygun, M. H. Najafi, and M. Margala, “Word2HyperVec: From word embeddings to hypervectors for hyperdimensional computing,” in *Great Lakes Symposium on VLSI 2024*, 2024.
[8] D. Kleyko, D. A. Rachkovskij, E. Osipov, and A. Rahimi, “A survey on hyperdimensional computing aka vector symbolic architectures, part i,” *ACM Comput. Surv.*, vol. 55, no. 6, 2022.
[9] T. Basaklar, Y. Tuncel, S. Y. Narayana, S. Gumussoy, and U. Y. Ogras, “Hypervector design for efficient hyperdimensional computing on edge devices,” 2021, arXiv:2103.06709.
[10] A. Joshi, J. Halseth, and P. Kanerva, “Language geometry using random indexing,” vol. 10106, 01 2017, pp. 265–274, lecture Notes in CS.
[11] A. Hernández-Cano, N. Matsumoto, E. Ping, and M. Imani, “Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system,” in *DATE ’21*, 2021, pp. 56–61.
[12] A. van den Oord, O. Vinyals, and K. Kavukcuoglu, “Neural discrete representation learning,” 2018.
[13] O. Keleş, M. A. Yılmaz, A. M. Tekalp, C. Korkmaz, and Z. Dogan, “On the computation of psnr for a set of images or video,” 2021.
[14] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, “The unreasonable effectiveness of deep features as a perceptual metric,” 2018.
[15] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, “Language models are unsupervised multitask learners,” *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.
[16] A. Coates, A. Ng, and H. Lee, “An analysis of single-layer networks in unsupervised feature learning,” in *International Conference on Artificial Intelligence and Statistics*, 2011.