Neural Computation for Robust and Holographic Face Detection

Mohsen Imani[1]*, Ali Zakeri[1], Hanning Chen[1], TaeHyun Kim[2], Prathyush Poduval[3], Hyunsei Lee[4], Yeseong Kim[4], Elaheh Sadredini[5] and Farhad Imani[6]

[1]University of California Irvine, [2]Pusan National University, [3]Indian Institute of Science, [4]Daegu Gyeongbuk Institute of Science and Technology, [5]University of California Riverside, [6]University of Connecticut

*Corresponding author: m.imani@uci.com

ABSTRACT

Face detection is an essential component of many tasks in computer vision with several applications. However, existing deep learning solutions are significantly slow and inefficient to enable face detection on embedded platforms. In this paper, we propose HDFace, a novel framework for highly efficient and robust face detection. HDFace exploits HyperDimensional Computing (HDC) as a neurally-inspired computational paradigm that mimics important brain functionalities towards high-efficiency and noise-tolerant computation. We first develop a novel technique that enables HDC to perform stochastic arithmetic computations over binary hypervectors. Next, we expand these arithmetic for efficient and robust processing of feature extraction algorithms in hyperspace. Finally, we develop an adaptive hyperdimensional classification algorithm for effective and robust face detection. We evaluate the effectiveness of HDFace on large-scale emotion detection and face detection applications. Our results indicate that HDFace provides, on average, 6.1× (4.6×) speedup and 3.0× (12.1×) energy efficiency as compared to neural networks running on CPU (FPGA), respectively.

ACM Reference Format:
Mohsen Imani[1]*, Ali Zakeri[1], Hanning Chen[1], TaeHyun Kim[2], Prathyush Poduval[3], Hyunsei Lee[4], Yeseong Kim[4], Elaheh Sadredini[5] and Farhad Imani[6]. 2022. Neural Computation for Robust and Holographic Face Detection. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC) (DAC '22), July 10–14, 2022, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3489517.3530653

1 INTRODUCTION

Face detection is an essential component of many computer vision applications, such as face tracking for surveillance [1], face recognition [2], emotion detection [3], face relighting and morphing [4], and facial shape reconstruction [5]. Face detection has also found several applications in human-computer and human-robot interaction systems. For example, recent digital cameras and commercial robots (e.g., Nao [6]) come with an embedded face detection module. In addition, the face detection is frequently used for image/person tagging.

Deep Neural Networks (DNNs) use costly convolutional layers as trainable feature extraction mechanisms. Due to its significant training cost, face detection algorithms often rely on static feature extraction methods. Popular feature extractions are pre-trained convolution layers, Histograms of Oriented Gradients (HOGs) [7, 8], Scale Invariant Feature Transform (SIFT) [9], Local Binary Patterns (LBPs) [10]. All these feature extractions are based on a set of arithmetic operations and aim to extract knowledge (e.g., shape and edge) from every image. Despite the success in the face detection domain, there are two major challenges with existing face detection algorithms: (1) they rely on costly feature extraction and learning algorithms. Running these algorithms on existing embedded platforms (e.g., devices limited power budget) results in slow and inefficient computation. (2) Today's face detection algorithms are getting noise from multiple sources. For example, with technology scaling, both hardware and network are becoming significantly unreliable. Unfortunately, machine learning algorithms have very low robustness to such noise and failure [11].

In this paper, we propose HDFace, a novel algorithm for highly efficient and robust face detection. To realize real-time performance and robustness, we redesign algorithms using strategies that more closely model *the human brain*. We exploit neurally-inspired HyperDimensional Computing (HDC) as an alternative paradigm that mimics important brain functionalities towards high-efficiency and noise-tolerant computation [12, 13, 14, 15]. HDC is motivated by the observation that the human brain operates on high-dimensional data representations. In HDC, objects are thereby encoded with high-dimensional vectors, called *hypervectors* [16]. HDC incorporates learning capability along with typical memory functions of storing/loading information. In this paper, we exploit HDC for two main purposes in our HDFace framework:

- As a pre-processing method for feature extraction. We have developed a novel technique that enables HDC to perform arithmetic computations. Inspired by stochastic computing, we develop a framework supporting hyperdimensional stochastic arithmetics. Instead of operating over original data, our framework represents each pixel value using a binary holographic hypervector, where the information is stored in a neural pattern. We accordingly enable arithmetic operations over these hypervectors using efficient and highly parallel operations. Finally, we expand HDC arithmetics to support feature extractions algorithms fully in hyperspace.
- We develop a hyperdimensional algorithm for effective and robust face detection. Our algorithm directly operates over hyperdimensional features exacted from feature exaction. Since feature extractors already generate hypervectors, there is no need for HDC encoding to map data points into high-dimension.

Our HDFace framework provides several advantages over competing learning: (1) being highly parallel and suitable for online on-device learning, (2) exposing hidden features, enabling single-pass learning with just a few samples, (3) being robust against noise and corrupted data. Our evaluation on large-scale datasets shows that HDFace provides better or comparable accuracy to state-of-the-art face detection algorithms, which provide significantly higher efficiency and robustness. Our results indicate that HDFace provides,

DAC '22, July 10–14, 2022, San Francisco, CA, USA
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9142-9/22/07.
https://doi.org/10.1145/3489517.3530653

on average, 6.1× and 3.0× (4.6× and 12.1×) speedup and energy efficiency compared to DNN running on CPU (FPGA), respectively. Moreover, our framework provides at least an order of magnitude higher computational robustness than DNN.

2 BACKGROUND AND MOTIVATION

Hyperdimensional computing (HDC): is based on the understanding that brains compute with patterns of neural activity that are not readily associated with numbers. Due to the very size of the brain's circuits, neural patterns can be modeled with hypervectors [12]. HDC builds upon a well-defined set of operations with random hypervectors, is extremely robust in the presence of failures, and offers a complete computational paradigm that is easily applied to learning problems [17, 18, 19, 20, 21, 22]. There exist a huge number of different, nearly orthogonal hypervectors with dimensionality in the thousands. This lets us combine such hypervectors using well-defined vector space operations while keeping the information of the two with high probability [15, 23, 24]. A hypervector contains all the information combined and spread across all its components in a full holistic representation so that no component is more responsible for storing any piece of information than another.

Feature Extraction: There are multiple existing feature extraction mechanisms for face detection. One example of feature extractions are Histogram of oriented Gradients (HOG), HAAR-like feature extraction, and convolution. Despite the difference in the functionalities, these feature extraction methods operate over a similar set of arithmetic operations. As an example, here we explain the functionality of HOG as one of the popular feature extraction mechanisms [7, 8].

HDC & Feature Extraction: Although HDC is significantly powerful in reasoning and association of the abstract information, it is weak in feature extraction from complex data, e.g., image/video. As a result, most existing HDC solutions are operating over costly pre-processed data [25, 14, 26]. This feature extraction often takes a large portion of the total learning cost. More importantly, since feature extractions are running on original data representation, they have high sensitivity to noise and failure. For example, we evaluate HDC on a large-scale face detection [27]. We use HOG as a feature extraction mechanism. Our evaluation on the ARM A53 CPU shows that HoG takes above 85% of total training time. In addition, 2% random bit error on HoG feature extraction causes 12% quality loss, while the HDC model is significantly robust against noise.

3 HDFACE OVERVIEW

In this paper, we introduce HDFace, a robust and efficient brain-inspired framework for face detection in hyperdimensional space. Figure 1 shows an overview of HDFace framework consisting of three main blocks: (a) base hypervector generation, (b) HDC feature extraction, and (c) HDC classification. In the following, we explain the general functionality of each step.

Base Hypervector generation: Instead of operating over original binary representation, HDFace assigns correlative hypervectors to pixel values. Let us assume a black-white image. In traditional binary data, colors are represented using values between 0 to $2^n - 1$, where n is bit precision. For example, in $n = 8$ bit representation, the white and black get 0 and 255 values, respectively. In contrast, HDFace assigns each pixel to a hypervector depending on its color. We assign two random hypervectors representing two extreme colors (Figure 1a). For example, white and black colors are assigned to two random hypervector $\vec{\mathcal{H}}_w$ and $\vec{\mathcal{H}}_b$, where hypervectors belong to $\{0, 1\}^D$. These hypervectors have nearly orthogonal representation $\delta(\vec{\mathcal{H}}_w, \vec{\mathcal{H}}_b) \simeq 0$. We use vector quantization to generate correlative hypervectors representing intermediate color values. For example, a pixel with 2^{n-1} value (where n is the bit precision) will get half of dimensions from black and half from white hypervector to generate correlative hypervector, $\delta(\vec{\mathcal{H}}_{2^{n-1}}, \vec{\mathcal{H}}_w) \simeq \delta(\vec{\mathcal{H}}_{2^{n-1}}, \vec{\mathcal{H}}_b) \simeq 0.5$. Using the above vector quantization, we will assign a hypervector representing each pixel.

HDC Feature Extraction: We develop hyperdimensional arithmetic operations that can be used to process the entire feature extraction in hyperspace. Our method supports stochastic addition, subtraction, multiplication, division, and compare operations over long binary vectors (Figure 1b). Our stochastic operations are highly parallel and efficient with significant robustness to noise. In Section 4, we cover the details of our HDC arithmetic operation.

HDC Learning: We develop a hyperdimensional learning algorithm operating on extracted features to enable efficient and robust learning (Figure 1c). Unlike existing HDC learning methods that require costly encoding, our extracted features are already in high dimensional space. Therefore, they can be directly used for hyperdimensional learning. Our HDC learning is based on the memorization of "face" and "no-face" as two hypervectors. Our learning framework eliminates redundant information memorization in each hypervectors to eliminate overfitting. For a given query, we first map data into high-dimensional space. A similarity search between an encoded query and an HDC model hypervector will return a prediction result. Details are explained in Section 5.

Our framework provides: (1) an end-to-end framework for fully HDC-based face detection over raw image data, (2) high computational robustness, as the entire application (including feature extractor) can benefit from the holographic data representation, and (3) significant efficiency as HDC revisits the complex feature extraction with parallel bitwise operations.

4 HDFACE STOCHASTIC PRIMITIVES

4.1 HDC Supported Operations

HDC encoding works based on a set of defined primitives [12]. Our goal is to exploit the same primitives to define SC-based arithmetic operations over HDC vectors [15]. HDC is an algebraic structure; it uses search along with several key operations (and their inverses): *Bundling* (+) that acts as memorization during hypervector addition, *Binding* (*) that associates multiple hypervectors, and *Permutation* (ρ) which preserves the position by performing a single rotational shift.

HDC Hypervector Generation: We generate a random hypervector with elements ±1 such that +1 appears with probability p. This will allow us to construct HDC representations of arbitrary numbers via a D dimensional vector. In our HDC system, information is stored with components ±1. We fix a random HDC vector $\vec{\mathcal{V}}_1$ to be a Basis vector. A random HDC vector $\vec{\mathcal{V}}_h$ ($h \in [-1, 1]$) is said to represent the number h if $\delta(\vec{\mathcal{V}}_h, \vec{\mathcal{V}}_1) = h$. This is consistent with our notation for $\vec{\mathcal{V}}_1$ which means that $\vec{\mathcal{V}}_1$ represents the number 1. Note that based on our representation, $\vec{\mathcal{V}}_{-a} = -\vec{\mathcal{V}}_a$.

Similarity Measurement: Between two HD vectors $\vec{\mathcal{V}}_1$ and $\vec{\mathcal{V}}_2$, the similarity defines as $\delta(\vec{\mathcal{V}}_1, \vec{\mathcal{V}}_2) = \frac{\vec{\mathcal{V}}_1 \cdot \vec{\mathcal{V}}_2}{D}$ where D is the number

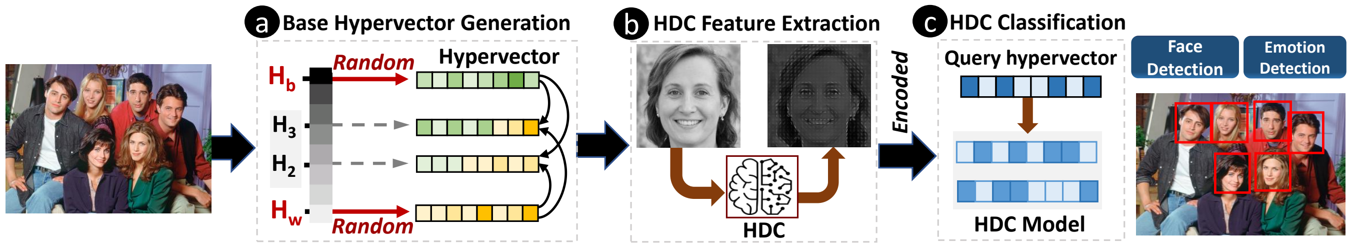

Figure 1: Overview of HDFace **framework: (a) base hypervector generation, (c) HDC-based feature extraction, (c) HDC learning**

of dimensions and $(\cdot)$ is the vector dot product operator. HDC supports other similarity metrics, such as Hamming similarity, that measures dimensions at which two HDC vectors differ.

4.2 HDC Arithmetic Operations

Weighted Average: This operation constructs the weighted average of two numbers if we are given two random HDC vectors $\vec{\mathcal{V}}_a$ and $\vec{\mathcal{V}}_b$ with the corresponding weights $\{p,\ q\} \in [0,1]$ $(p+q=1)$ respectively. We construct $C = p\vec{\mathcal{V}}_a \oplus q\vec{\mathcal{V}}_b$ by choosing each component from $\vec{\mathcal{V}}_a$ or $\vec{\mathcal{V}}_b$ randomly with probability p and q, respectively.

Construction: In HDFace framework, we are given two random vectors $\vec{\mathcal{V}}_0$ and $\vec{\mathcal{V}}_1$ which represent numbers 0 and 1 respectively. Using the weighted average operation, we can construct the representation of an arbitrary number $a \in [-1,1]$ as $\vec{\mathcal{V}}_a = \frac{a+1}{2}\vec{\mathcal{V}}_1 \oplus \frac{1-a}{2}(-\vec{\mathcal{V}}_1)$. Because $a \in [0,1]$,we have that $\frac{1\pm a}{2} \in [0,1]$ and so the corresponding weights for averaging are positive.

Multiplication: We can multiply two HDC representations $\vec{\mathcal{V}}_a$ and $\vec{\mathcal{V}}_b$ to construct $\vec{\mathcal{V}}_{ab}$. If the i^{th} dimension of both $\vec{\mathcal{V}}_a$ and $\vec{\mathcal{V}}_b$ are equal, then set the corresponding dimension of $\vec{\mathcal{V}}_{ab}$ to be that of $\vec{\mathcal{V}}_1$. Otherwise, set the corresponding dimension of $\vec{\mathcal{V}}_{ab}$ to be that of $\vec{\mathcal{V}}_{-1}$. We denote the multiplication operation by $\vec{\mathcal{V}}_a \otimes \vec{\mathcal{V}}_b = \vec{\mathcal{V}}_{ab}$.

Square Root: In the HOG process, we would need to calculate the magnitude of vectors. For this, we need one last operation to perform square roots of positive numbers. That is, given $\vec{\mathcal{V}}_a$ we would like to construct $\vec{\mathcal{V}}_{\sqrt{a}}$. To do this, we perform a similar binary search method as division.

- Define $\vec{\mathcal{V}}_{low} = \vec{\mathcal{V}}_0$ and $\vec{\mathcal{V}}_{high} = \vec{\mathcal{V}}_1$
- Define $\vec{\mathcal{V}}_m = 0.5\vec{\mathcal{V}}_{low} \oplus 0.5\vec{\mathcal{V}}_{high}$ and $\vec{\mathcal{V}}_{m^2} = \vec{\mathcal{V}}_m \otimes \vec{\mathcal{V}}_m$
- If $\vec{\mathcal{V}}_{m^2} > \vec{\mathcal{V}}_a$, then do $\vec{\mathcal{V}}_{high} \rightarrow \vec{\mathcal{V}}_m$
- If $\vec{\mathcal{V}}_{m^2} < \vec{\mathcal{V}}_a$, then do $\vec{\mathcal{V}}_{low} \rightarrow \vec{\mathcal{V}}_m$
- Repeat from step 2 until $\vec{\mathcal{V}}_{m^2} = \vec{\mathcal{V}}_a$ upto statistical margins of error

Thus, we have constructed $\vec{\mathcal{V}}_m = \vec{\mathcal{V}}_{\sqrt{a}}$

4.3 HOG Pre-Processing Using HDC

To perform HOG processing on the image, we first normalize the image feature vector so that each value is between 0 and 1. This is because the HD vectors in our design can only store values between -1 and 1. Then, we consider a 3×3 cell of pixels where the i^{th} row and j^{th} column has value $C_{i,j}$ $(i,j = 0,1,2)$. First, we produce an HD representation for all the pixels, and we label this as

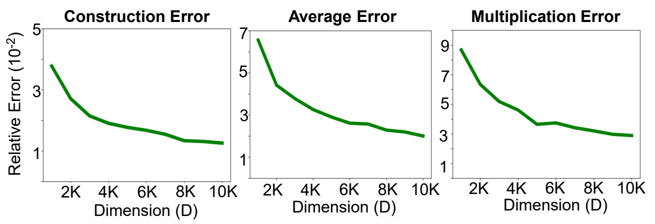

Figure 2: Error for (a) Construction, (b) Average, and (c) Multiplication between two numbers.

$\vec{\mathcal{V}}_{C_{i,j}}$. To calculate the contribution to the histogram by this cell, we need to follow a three-step process.

Calculating Gradient: We need to calculate the gradient vector $\vec{G} = (G_x, G_y)$ of the pixels at location $(1,1)$.The components are given by $G_x = \left((C_{2,1} - C_{0,1}\right)/2$ ad $G_y = \left(C_{1,2} - C_{1,0}\right)/2$. The two components can be found in HD space using $\vec{\mathcal{V}}_{G_x} = \vec{\mathcal{V}}_{(C_{2,1}-C_{0,1})/2} = \vec{\mathcal{V}}_{C_{2,1}} \oplus \left(-\vec{\mathcal{V}}_{C_{0,1}}\right)$ and $\vec{\mathcal{V}}_{G_y} = \vec{\mathcal{V}}_{(C_{1,2}-C_{1,0})/2} = \vec{\mathcal{V}}_{C_{1,2}} \oplus \left(-\vec{\mathcal{V}}_{C_{1,0}}\right)$

Calculating Magnitude: To find the magnitude, we first calculate $\vec{\mathcal{V}}_{\frac{G_x^2+G_y^2}{2}} = (\vec{\mathcal{V}}_{G_x} \otimes \vec{\mathcal{V}}_{G_x}) \oplus (\vec{\mathcal{V}}_{G_y} \otimes \vec{\mathcal{V}}_{G_y})$. Next, we perform the square root of this vector to find $\vec{\mathcal{V}}_{\sqrt{\frac{G_x^2+G_y^2}{2}}}$. Note that this differs from the true magnitude of the gradient by a factor of $\frac{1}{\sqrt{2}}$, however it will not affect the final HOG features since all values will be scaled evenly.

Calculating Angle Bin: The final step of HOG pre-processing is to find the angle bin where we need to place the magnitude of the gradient vector. However, this is the most complicated step if we perform it in terms of HD operations. The original way is to find the angle of the gradient vector using $\theta = \arctan \frac{G_y}{G_x}$. However, the arctan function cannot be implemented easily within the HD framework. As a workaround, we use the fact that the tan function is a monotonic function within specific ranges. Suppose θ_i are the boundaries of the bins, with $i = 1,2,3,4,..,7,8$. We first construct the hypervectors $\vec{\mathcal{V}}_{\tan\theta_i}$ and $\vec{\mathcal{V}}_{\cot\theta_i}$. The reason for considering $\cot\theta_i$ will become clear soon. Next, we evaluate $\vec{\mathcal{V}}_{G_x}$ and $\vec{\mathcal{V}}_{G_y}$ to localize on the specific quadrant where the gradient vector lies (Based on their signs).

Finally, we consider all the boundaries θ_i that lie in the quadrant found. Our goal is to find the appropriate angle bin we need to add the magnitude into to construct the histogram. To prevent computational difficulties, we add the point $\pi/2$ and $3\pi/2$ as boundaries too because the tan function blows up and changes sign at these point, and is no longer monotonic. If θ lies in bin i, and in the I or IV quadrant, then $\theta_i < \theta < \theta_{i+1}$ is satisfied. This is equivalent to

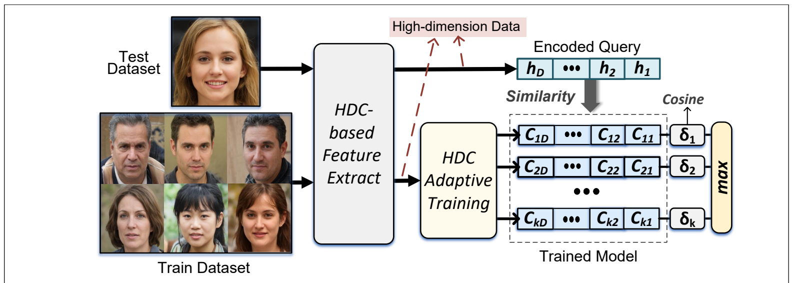
High-dimension Data
Test Dataset
Encoded Query
h_D ••• h_2 h_1
Similarity
Cosine
HDC-based Feature Extract
HDC Adaptive Training
C_{1D} ••• C_{12} C_{11} δ_1
C_{2D} ••• C_{22} C_{21} δ_2
C_{kD} ••• C_{k2} C_{k1} δ_k
max
Train Dataset
Trained Model

Figure 3: HDFace training for accurate classification.

$\tan\theta_i < \tan\theta < \tan\theta_{i+1}$. If it is in the II or III quadrant, then the corresponding relationship is $\tan\theta_{i+1} < \tan\theta < \tan\theta_i$. Note that the values of $\tan\theta_i$ and $\cot\theta_i$ are known to us.

In the end, we need some way to compare $\tan\theta = G_y/G_x$ with a real number r given only the vectors $\vec{\mathcal{V}}_{G_x}, \vec{\mathcal{V}}_{G_y}, \vec{\mathcal{V}}_r$ and $\vec{\mathcal{V}}_{1/r}$. Let us write $G_y/G_x = \sigma|G_y|/|G_x|$ and $r = \eta|r|$ where σ and η denote the corresponding signs. If $|r| < 1$, we find the quantity $\alpha = \frac{\sigma|G_y|-r|G_x|}{2}$. If $\alpha > 0$, then $\tan\theta > r$, if $\alpha < 0$ then $\tan\theta < r$ and $\tan\theta = r$ otherwise. But, the hypervector for α can be calculated as $\vec{\mathcal{V}}_\alpha = 0.5\left(\sigma\vec{\mathcal{V}}_{G_y}\right) \oplus 0.5\left(-\vec{\mathcal{V}}_{r\times|G_x|}\right)$. By evaluating $\vec{\mathcal{V}}_\alpha$, we can find the value of α and use this for comparisons.

Above, we chose $\alpha = \frac{\sigma|G_y|-r|G_x|}{2}$ because $r|G_x|$ is a number between -1 and 1. However, this need not be true if $|r| > 1$. In this case we define $\alpha = \frac{\sigma\frac{1}{|r|}|G_y|-\eta|G_x|}{2}$ and $\vec{\mathcal{V}}_\alpha = 0.5\left(\sigma\vec{\mathcal{V}}_{\frac{1}{|r|}\times G_y}\right) \oplus 0.5\left(-\eta\vec{\mathcal{V}}_{|G_x|}\right)$

Figure 2 shows the HDFace relative error during different operations. The results are reported for construction, average, and multiplication operation. Our evaluation shows that the relative error rate decreases with the hypervector dimensionality. These error rates can be easily expanded to analyze the error rate of different feature extraction methods using stochastic arithmetic operations. For example, the HOG error rate can be estimated in each dimensionality. This, along with robustness analysis at the application level, determines a suitable dimensionality that ensures accuracy.

5 HYPERDIMENSIONAL FACE DETECTION

We exploit hyperdimensional learning to directly operate over encoded data. Figure 3 shows an overview of HDC classification. HDC receives raw image data. After processing the images using our HDC-based feature extractor, the pre-processed data is already in high-dimensional space. This data will be given to our adaptive training module to generate a single hypervector for each class. During the inference, each given test image will pass through the same feature extractor. Then, the generated hypervector, called query hypervector, will be compared with all class hypervectors. A class with the highest similarity will be selected as a prediction result. In the following, we provide more details about HDC classification steps.

HDC identifies common patterns during learning and eliminates the saturation of the class hypervectors during single-pass training. In inference, HDC checks the similarity of each encoded test data with the class hypervector in two steps. The first step encodes the input to produce a query hypervector $\vec{\mathcal{H}}$. Then we compute the similarity (δ) of $\vec{\mathcal{H}}$ and all class hypervectors. Query data gets the label of the class with the highest similarity.

Table 1: Datasets (n: image size, k: number of classes)

	n	k	Train Size	Description
EMOTION	48×48	7	36,685	Facial Emotion Detection [31]
FACE1	1024×1024	2	40,172	HD Face Detection [32]
FACE2	512×512	2	522,441	Face Detection [27]

6 EVALUATION

6.1 Experimental Setup

We developed a PyTorch-based library of Hyperdimensional face detection, supporting all stochastic arithmetic and learning operations. We also design a cycle-accurate simulator based on PyTorch [28] that emulates HDFace functionality during classification. We implement HDFace training and testing on two embedded platforms: Cortex A53 CPU and Kintex-7 FPGA. For FPGA, we design the HDFace functionality using Verilog and synthesize it using Xilinx Vivado Design Suite [29]. The synthesis code has been implemented on the Kintex-7 FPGA KC705 Evaluation Kit. We ensure our efficiency is higher than the automated FPGA implementation at [30]. For CPU, the HDFace code has been written in C++ and optimized for performance. The code has been implemented on Raspberry Pi (RPi) 3B+ using ARM Cortex A53 CPU. The simulator collects the execution time and measures power for each connected platform while running the learning procedures. The power consumption is collected by Hioki 3337 meter.

Table 1 summarizes the evaluated datasets. The tested benchmarks range from emotion detection to two large-scale face detection datasets, which include hundreds of thousands of images of facial and non-facial data. We use HoG as a feature extraction mechanism for all image data.

6.2 HDFace Learning Accuracy

State-of-the-art Learning Algorithms: We compare HDFace classification accuracy with state-of-the-art learning algorithms, including Deep Neural Networks (DNN) and Support Vector Machine (SVM). All learning modules use the same HOG feature extraction, and they are optimized to provide their maximum accuracy. Our HDFace results are reported in three different configurations: (1) HOG feature extraction running on original space. In this configuration, HDC exploits non-linear encoder to map extracted features into high dimension. (2) HOG running based on our stochastic hyperdimensional representation. For HDC, we use the same dimensionality ($D = 4k$) for both feature exaction and learning. In this configuration, HDC learning directly happens on extracted features (high-dimension) with no encoding module.

Our evaluation shows that HDC, regardless of feature extraction, provides better accuracy than state-of-the-art algorithms. For example, HDC accuracy is, on average, 3.9% and 10.4% higher than DNN and SVM, respectively. Our results also indicate that our stochastic hyperdimensional feature extraction provides the same quality of detection as feature extraction in original space.

6.3 HDFace in Different Configurations

Figure 5a shows the impact of dimensionality on HDFace's classification accuracy. The results are reported when the hypervector dimensionality for pre-processing and learning scales from $D = 1k$ to $D = 10k$. In general, HDC accuracy increases with hypervector dimensionality. This accuracy improvement comes from: (1) increasing the capacity of each hypervector to learn and memorize

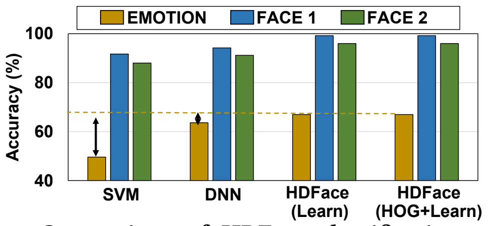

Figure 4: Comparison of HDFace classification accuracy with state-of-the-art learning techniques.

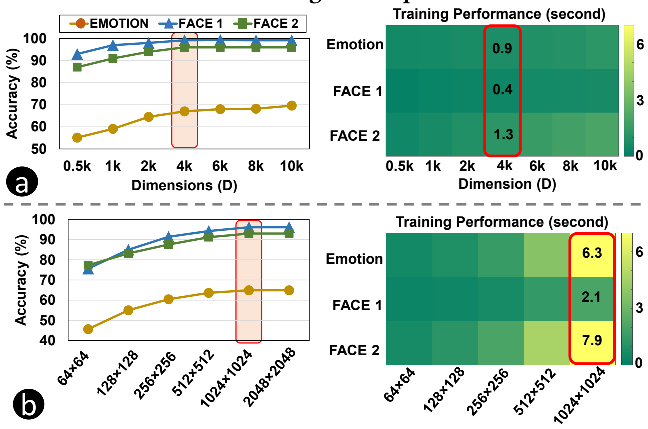

Figure 5: (a) Impact of dimensionality on HDFace accuracy & training performance. (b) Impact of DNN configuration on classification accuracy and training performance.

information, (2) enabling more accurate feature extraction, and (3) reducing the chance of overfitting in high-dimension. Since HDC operates over redundant representation, it has natural robustness to dimensionality reduction. Our results in Figure 5a indicate that HDFace provides maximum accuracy using $D = 4k$ dimensions. Using longer dimensionality results in accuracy saturation.

Figure 5b shows the DNN accuracy using different network configurations. We use four layers neural network where two hidden layers can get different sizes (as shown in the x-axis of Figure 5b). Our results indicate that DNN gets maximum accuracy using 1024×1024 neurons in hidden layers. However, this accuracy is still slightly lower than the maximum accuracy that HDFace provides in its best configuration ($D = 4k$).

The heatmaps in Figure 5 compare HDFace and DNN training performance in different configurations. While HDFace ensures maximum accuracy with $D = 4k$ dimensions, DNN achieves its maximum accuracy using hidden layers with 1024×1024 sizes. We compare HDFace and DNN efficiency in these configurations. Our results indicate that HDFace provides significantly faster training than DNN. For example, training a single epoch in HDFace takes 0.9 seconds, while the DNN requires 5.4 seconds to be trained on the embedded CPU.

6.4 HDFace Quality & Dimensionality

Figure 6a visually compares HDFace's face detection accuracy using hypervector with different dimensions. The results are reported when HoG feature extraction's window moves across an image in an overlapping manner. Each window gets blue color if HDFace detects a face on that window. Our results show that HDFace can accurately detect faces when the dimensionality is large. In our example, HDFace with D=1k dimensions incorrectly predicts a few

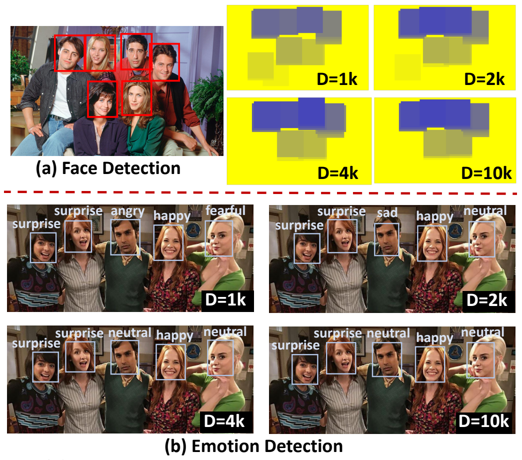

Figure 6: (a) visuallizing the impact of HDFace dimensionality on (a) face detection (b) emotion detection.

places of an image as a face. This misprediction does not occur when the dimensionality is over $D = 4k$. Figure 6b also visualizes HDFace's capability for emotion detection. Our results show that HDFace can correctly identify the emotion using hypervectors with large dimensions ($D = 4k$ and larger). However, the prediction will have an error when dimensionality reduces to $D = 1k$.

6.5 Efficiency in Different Platforms

Figure 7 shows the computational efficiency of HDFace compared to the DNN on an ARM Cortex A53 embedded CPU. Our results indicate that HDFace provides significantly higher computational efficiency than DNN, especially during the training phase. Our evaluation shows that HDFace's training is, on average, 6.1× faster and 3.0× more efficient than DNN. Figure 7a also compares HDFace's computational efficiency with DNN on Kintex 7 FPGA platform. Our evaluation shows that in all tested applications HDFace outperforms DNN's computational efficiency. This efficiency comes from HDFace's capability in simplifying feature exaction and learning process such that end-to-end classification task can perform using highly parallel bitwise or low-precision computations. The data representation along with HDC's natural parallelism makes FPGA an ideal platform for hardware acceleration. FPGA consists of a huge number of lightweight lookup tables (LUTs) resources that can be used to accelerate HDC feature extraction and learning. In contrast, processing HOG and DNN on original data representation relies on high-precision arithmetic that needs to be processed using costly DSP resources of FPGA. Our results show that HDFace achieves, on average, 4.6× speedup and 12.1× higher energy efficiency compare to DNN while running on the same FPGA platform.

Figure 7b compares HDFace and DNN efficiency during the inference. Unlike training, HDFace's inference efficiency has a closer margin to DNN. Our evaluation shows that HDFace's inference is, on average, 2.9× and 2.6× (1.4× and 1.7×) faster and more energy-efficient than DNN running on FPGA (ARM CPU), respectively.

6.6 HDFace Robustness

Table 2 compares HDFace and DNN robustness to random bit error. For DNN, the results are reported when using a network with 16-bits, 8-bits, and 4-bits model precision. Our results indicate that there is a tradeoff between accuracy and robustness in DNN. On

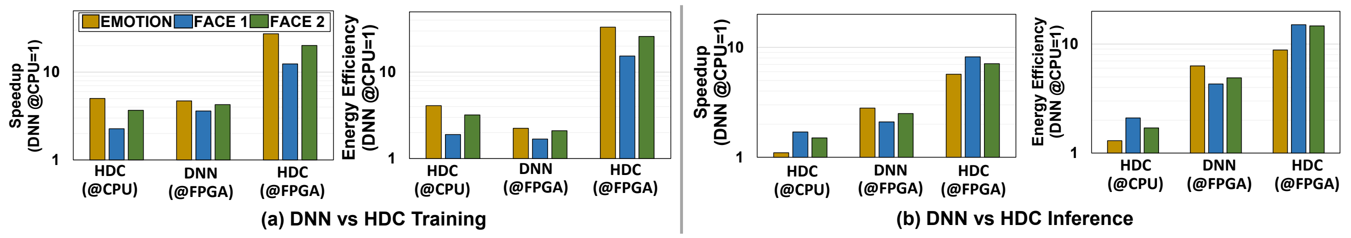

Figure 7: Speedup and energy efficiency of HDFace and DNN on CPU and FPGA during (a) training and (b) inference.

Table 2: DNN and HDFace robustness to random noise.

Error Rate		*0%*	*1%*	*2%*	*4%*	*8%*	*12%*	*14%*
DNN	**16-bit**	0%	2.8%	4.8%	8.1%	13.8%	23.4%	39.8%
	8-bit	1.6%	3.8%	5.1%	7.2%	10.6%	16.0%	24.7%
	4-bit	2.7%	4.1%	4.7%	5.4%	6.5%	8.1%	10.2%
HDFace +HoG +Learn	**D=10k**	0%	0%	0%	0%	0.6%	0.9%	1.2%
	D=4k	0%	0%	0%	0.2%	1.2%	1.8%	2.0%
	D=1k	2.8%	2.8%	3.4%	3.9%	5.3%	6.0%	6.9%
HDFace +Learn	**D=10k**	0%	1.8%	3.2%	5.5%	9.1%	17.3%	29.2%
	D=4k	0%	2.3%	3.5%	5.2%	7.8%	12.1%	19.0%
	D=1k	2.8%	3.5%	4.1%	4.9%	6.1%	7.5%	9.3%

the one hand, DNN requires high precision weights to ensure maximum classification accuracy. For example, DNN with a 4-bit precision model provides 1.1% and 2.7% lower accuracy than 8-bit and 16-bit models. On the other hand, DNN robustness reduces with model precision. Errors in high-precision representation significantly change DNN weight values, thus resulting in high sensitivity to even slight noise. For example, a 12% random bit error rate could result in 8.1% and 23.4% quality loss in DNN with 4-bits and 16-bits, respectively.

In contrast, HDFace has significant robustness to random noise. In HDFace, hypervectors have redundant holographic representation, where all hypervector element equality contribute on the computation. This provides natural robustness to noise and failure on hypervector elements. There are three directional tradeoffs in selecting suitable HDFace dimensions: accuracy, efficiency, and robustness. A low dimensional HDFace model is desired for efficient computation. However, this model results in lower classification accuracy and reduces the robustness of the model to possible noise. For example, HDFace in $D = 1k$ dimension provides 2.8% lower accuracy than $D = 4k$ dimensions. Although HDFace with $D = 4k$ dimensions provides the best tradeoff between accuracy and efficiency, the model's robustness can still be enhanced by increasing the dimension size. Table 2 also shows HDFace robustness to bit error rate when processing HoG feature extraction on original data representation. Our results show that processing feature extraction on original data representation entirely removes the advantage of our hyperdimensional model.

7 CONCLUSION

In this paper, we exploit hyperdimensional computing for high-efficiency and noise-tolerant face detection. We first develop a novel technique that enables HDC to perform stochastic arithmetic computations over binary hypervectors. Next, we expand these arithmetic for efficient processing of feature extractions in hyperspace. Finally, we develop an adaptive classification algorithm for effective and robust face detection. Our evaluation shows that HDFace: (1) is highly parallel and suitable for online on-device learning, (2) exposes hidden features in a few iterations, (3) is robust against noise and corrupted data.

ACKNOWLEDGEMENTS

This work was supported in part by National Science Foundation #2127780, Semiconductor Research Corporation (SRC) Task #2988.001, Office of Naval Research (ONR) grant #N00014-21-1-2225, Air Force Office of Scientific Research, and a generous gift from Cisco.

REFERENCES

[1] P. Zhou *et al.*, "Two-stream neural networks for tampered face detection," in *CVPRW*. IEEE, 2017, pp. 1831–1839.
[2] X. He *et al.*, "Face recognition using laplacianfaces," *IEEE transactions on pattern analysis and machine intelligence*, vol. 27, no. 3, pp. 328–340, 2005.
[3] D. Reney and N. Tripathi, "An efficient method to face and emotion detection," in *CSNT*. IEEE, 2015, pp. 493–497.
[4] Z. Wen *et al.*, "Face relighting with radiance environment maps," in *CVPR*, vol. 2. IEEE, 2003, pp. II–158.
[5] P. Dou *et al.*, "End-to-end 3d face reconstruction with deep neural networks," in *CVPR*, 2017, pp. 5908–5917.
[6] S. R. Reyes *et al.*, "Face detection and recognition of the seven emotions via facial expression: Integration of machine learning algorithm into the nao robot," in *ICCRE*. IEEE, 2020, pp. 25–29.
[7] O. Déniz *et al.*, "Face recognition using histograms of oriented gradients," *Pattern recognition letters*, vol. 32, no. 12, pp. 1598–1603, 2011.
[8] A. Adouani *et al.*, "Comparison of haar-like, hog and lbp approaches for face detection in video sequences," in *SSD*. IEEE, 2019, pp. 266–271.
[9] T. Lindeberg, "Scale invariant feature transform," 2012.
[10] C. Rahmad *et al.*, "Comparison of viola-jones haar cascade classifier and histogram of oriented gradients (hog) for face detection." IOP, 2020.
[11] A. S. a. o. Rakin, "Bit-flip attack: Crushing neural network with progressive bit search," in *ICCV*, 2019.
[12] 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, 2009.
[13] A. Hernández-Cano *et al.*, "Reghd: Robust and efficient regression in hyper-dimensional learning system," in *DAC*. IEEE, 2021, pp. 7–12.
[14] M. Imani *et al.*, "A framework for collaborative learning in secure high-dimensional space," in *CLOUD*. IEEE, 2019, pp. 435–446.
[15] P. Poduval *et al.*, "Stochd: Stochastic hyperdimensional system for efficient and robust learning from raw data," in *ACM/IEEE DAC*. IEEE, 2021, pp. 1195–1200.
[16] A. Hernandez-Cane *et al.*, "Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system," in *DATE*. IEEE, 2021, pp. 56–61.
[17] Z. Zou *et al.*, "Scalable edge-based hyperdimensional learning system with brain-like neural adaptation," in *SC*, 2021, pp. 1–15.
[18] P. Poduval *et al.*, "Cognitive correlative encoding for genome sequence matching in hyperdimensional system," in *DAC*. IEEE, 2021, pp. 781–786.
[19] R. Chen *et al.*, "Joint active search and neuromorphic computing for efficient data exploitation and monitoring in additive manufacturing," *Journal of Manufacturing Processes*, vol. 71, pp. 743–752, 2021.
[20] D. Kleyko *et al.*, "Vector symbolic architectures as a computing framework for nanoscale hardware," *arXiv preprint arXiv:2106.05268*, 2021.
[21] Z. zou *et al.*, "Biohd: An efficient genome sequence search platform using hyper-dimensional memorization," in *ISCA*. IEEE, 2022.
[22] P. Poduval *et al.*, "Graphd: Graph-based hyperdimensional memorization for brain-like cognitive learning," *Frontiers in Neuroscience*, 2022.
[23] Z. Zou *et al.*, "Spiking hyperdimensional network: Neuromorphic models integrated with memory-inspired framework," *arXiv preprint arXiv:2110.00214*, 2021.
[24] A. Hérnandez-Cano *et al.*, "Prid: Model inversion privacy attacks in hyperdimensional learning systems," in *DAC*. IEEE, 2021, pp. 553–558.
[25] M. Imani *et al.*, "Revisiting hyperdimensional learning for fpga and low-power architectures," in *HPCA*. IEEE, 2021, pp. 221–234.
[26] M. Nazemi *et al.*, "Synergiclearning: Neural network-based feature extraction for highly-accurate hyperdimensional learning," *arXiv preprint arXiv*, 2020.
[27] A. Angelova *et al.*, "Pruning training sets for learning of object categories," in *CVPR*. IEEE, 2005.
[28] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *NIPS*, 2019, pp. 8026–8037.
[29] T. Feist, "Vivado design suite," *White Paper*, vol. 5, 2012.
[30] S. Salamat *et al.*, "F5-hd: Fast flexible fpga-based framework for refreshing hyperdimensional computing," in *FPGA*, 2019, pp. 53–62.
[31] "Emotion Detection Dataset," https://www.kaggle.com/ananthu017/emotion-detection-fer.
[32] P. Kottarathil, "Face mask lite dataset," 2020.