Factorizers for distributed sparse block codes

Michael Hersche [a,b], Aleksandar Terzić [a,b], Geethan Karunaratne [a], Jovin Langenegger [a],
Angéline Pouget [b], Giovanni Cherubini [a], Luca Benini [b], Abu Sebastian [a] and Abbas Rahimi [a,*]

[a] *IBM Research – Zurich, Säumerstrasse 4, 8803 Rüschlikon, Switzerland*
E-mail: abr@zurich.ibm.com

[b] *Department of Information Technology and Electrical Engineering, ETH Zürich, Gloriastrasse 35, 8092 Zürich,*
Switzerland

Abstract. Distributed sparse block codes (SBCs) exhibit compact representations for encoding and manipulating symbolic data structures using fixed-width vectors. One major challenge however is to disentangle, or factorize, the distributed representation of data structures into their constituent elements without having to search through all possible combinations. This factorization becomes more challenging when SBCs vectors are noisy due to perceptual uncertainty and approximations made by modern neural networks to generate the query SBCs vectors. To address these challenges, we first propose a fast and highly accurate method for factorizing a more flexible and hence generalized form of SBCs, dubbed GSBCs. Our iterative factorizer introduces a threshold-based nonlinear activation, conditional random sampling, and an ℓ_∞-based similarity metric. Its random sampling mechanism, in combination with the search in superposition, allows us to analytically determine the expected number of decoding iterations, which matches the empirical observations up to the GSBC's bundling capacity. Secondly, the proposed factorizer maintains a high accuracy when queried by noisy product vectors generated using deep convolutional neural networks (CNNs). This facilitates its application in replacing the large fully connected layer (FCL) in CNNs, whereby C trainable class vectors, or attribute combinations, can be implicitly represented by our factorizer having F-factor codebooks, each with $\sqrt[F]{C}$ fixed codevectors. We provide a methodology to flexibly integrate our factorizer in the classification layer of CNNs with a novel loss function. With this integration, the convolutional layers can generate a noisy product vector that our factorizer can still decode, whereby the decoded factors can have different interpretations based on downstream tasks. We demonstrate the feasibility of our method on four deep CNN architectures over CIFAR-100, ImageNet-1K, and RAVEN datasets. In all use cases, the number of parameters and operations are notably reduced compared to the FCL.

Keywords: Vector-symbolic architectures, iterative factorizer, sparse block codes, convolutional neural networks, deep learning

1. Introduction

Vector-symbolic architectures (VSAs) [10,11,22,39,41] are a class of computational models that provide a formal framework for encoding, manipulating, and binding symbolic information using fixed-size distributed representations. VSAs feature compositionality and transparency, which enabled them to perform analogical mapping and

---

*Corresponding author. E-mail: abr@zurich.ibm.com.

retrieval [12,21,40], inductive reasoning [6,45], and probabilistic abductive reasoning [18,19]. Moreover, the VSA's distributed representations can mediate between rule-based symbolic reasoning and connectionist models that include neural networks. Recent work [18] has shown how VSA, as a common language between neural networks and symbolic AI, can overcome the binding problem in neural networks and the exhaustive search problem in symbolic AI.

In a VSA, all representations – from atoms to composites – are high-dimensional distributed vectors of the same fixed dimensionality. An atom in a VSA is a randomly drawn i.i.d. vector that is dissimilar (i.e., quasi-orthogonal) to other random vectors with very high probability, a phenomenon known as concentration of measure [32]. Composite structures are created by manipulating and combining atoms with well-defined dimensionality-preserving operations, including multiplicative binding, unbinding, additive bundling (superposition), and permutations. The binding operation can yield quasi-orthogonal results, which, counterintuitively, can still encode semantic information. For instance, we can describe a concept in a scene (e.g., a `black circle`) with two factors (color and shape) by binding quasi-orthogonal atomic vectors ($\mathbf{x}_{\text{black}} \circledast \mathbf{x}_{\text{circle}}$). The resulting product vector is quasi-orthogonal to all other possible vectors (atomic and composite). Yet, decomposing it into its factors ($\mathbf{x}_{\text{black}}$ and $\mathbf{x}_{\text{circle}}$) reveals the semantic relation between a `black circle` and a `black square` since both include $\mathbf{x}_{\text{black}}$.

However, decomposing, or disentangling, a bound product vector into its factors is computationally challenging, requiring checking all the possible combinations of factors. Extending the previous example from two to F factors, each factor f having a codebook of M_f codevectors, there are $\prod_{f=1}^{F} M_f$ possible combinations to be searched in the product space for factorizing a product vector. To alleviate this hard combinatorial search problem, rapid iterative approaches were proposed such as resonator networks [7,23] and follow-up stochastic factorizers with nonlinear activation [31]. However, these existing solutions can only infer the factors of dense bipolar product vectors (i.e., each vector element is a Rademacher random variable) and face challenges with other types of VSA representations such as sparse block codes (SBCs) [8,29,43]. SBCs exhibit compact memory footprint, ideal variable binding properties [8], high information capacity for associative memories [14,26,51], biological plausibility [2,36, 38,58], and amenability for implementation on emerging neuromorphic hardware [1,46]. Motivated by these key aspects of SBCs, there is a need to come up with a rapid iterative approach to accurately factorize SBC product vectors.

This paper provides the following contributions, which are divided into two main parts. In Part I, for the first time, we propose an iterative block code factorizer (BCF) that can reliably factorize blockwise distributed product vectors. The used codebooks in BCF are binary SBCs, which span the product space, while BCF can factorize product vectors from a more generalized sparse block code (GSBC). Hence, factorizing binary SBCs is a special case. BCF introduces a configurable threshold, a conditional sampling mechanism, and a new ℓ_∞-based similarity search operation. During the iterative decoding, the novel sampling mechanism induces a random search in superposition over the product space if no confident solution is present. BCF improves the convergence speed of the state-of-the-art stochastic factorizer [31] on a large problem size of 10^6 by up to $6\times$. To gain a deeper understanding of BCF's iterative search in superposition, we leverage its configurable threshold and sampling mechanism to configure it as an unconditional sampler that randomly searches over the product space. This allows us to determine the expected number of decoding iterations analytically, which matches the empirical observations when operating within the GSBC's bundling capacity.

In Part II, we present an application for BCF that reduces the number of parameters in fully connected layers (FCLs). FCLs are ubiquitous in modern deep learning architectures and play a major role by accounting for most of the parameters in various architectures, such as transformers [13,30], extreme classifiers [9,33], and CNNs for edge devices [42]. Given an FCL with respective input and output dimensions D_i and D_o, we can replace its trainable parameters $\mathbf{W} \in \mathbb{R}^{D_i \times D_o}$ by a BCF with F codebooks of fixed parameters, each $\mathbf{X}^f \in \{0, 1\}^{D_i \times \sqrt[F]{D_o}}$. The structure of the codebooks is naturally given when the product space is defined by semantic attributes (e.g., in RAVEN [59]), or can be arbitrarily defined when no semantic attributes are provided (e.g., for natural images in ImageNet-1K [3]). To map sensory inputs to GSBC product vectors, we train deep convolutional layers with a novel blockwise additive loss that can directly use BCF in place of an FCL classifier. Our BCF reduces the total number of parameters across a wide range of deep CNNs and datasets by 0.5–44.5% while maintaining a high accuracy within 0–4.46% compared to the baseline CNNs using the large FCL. Our BCF also lowers the computational cost of the classifier layer by 55.2–86.7% with respect to FCLs.

2. VSA preliminary

VSAs define operations over (pseudo)random vectors with independent and identically distributed (i.i.d.) components. Computing with VSAs begins by defining a basis in the form of a codebook $\mathbf{X} := \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M\} := \{\mathbf{x}_i\}_{i=1}^{M}$. If the dimension D_p of two randomly drawn vectors of the space is sufficiently large, they are highly likely to have an almost-zero similarity, i.e., they are quasi-orthogonal [22]. VSAs use three primary operations – bundling, binding, and permutation – that form an algebra over the space of vectors. The bundling represents a set of vectors via vector addition with a possible nonlinearity, resulting a vector that is similar to all vectors from the set. In contrast, binding and permutation yield result vectors that are dissimilar to their input vectors. Combined with a similarity metric, these operations support various cognitive data structures: variable binding, sequence, and hierarchy. See [24] for a review.

For example, consider a VSA model based on the bipolar vector space [11], i.e., $\mathbf{x} \in \{-1, +1\}^{D_p}$. One can define binding and unbinding in this vector space as the Hadamard (i.e., elementwise) product. The similarity between two vectors in the space is typically measured using the cosine similarity metric. A possible bundling operation is the elementwise sum followed by the sign function, setting all negative elements to -1 and the positive to $+1$. To keep representations bipolar, elements with a sum equal to zero are randomly set to -1 or $+1$.

As an alternative, binary sparse block codes (binary SBCs) [29] induce a local blockwise structure that exhibits ideal variable binding properties [8] and high information capacity when used in associative memories [14,26,51]. In binary SBCs, the D_p-dimensional vectors are divided into B blocks of equal length, $L = D_p/B$, where only one element per block is set to 1. The vectors can either be described with a D_p-dimensional binary SBC vector (denoted as $\mathbf{x}$), or with a B-dimensional offset vector where each element indicates the index of the nonzero element within each block (denoted as $\dot{\mathbf{x}}$). The vectors are initialized by randomly setting one element in each block to 1. The bundling of two or more vectors is defined as their elementwise addition, followed by a selection function that retains the sparsity by setting the largest element of each block to 1 and the remaining elements to 0. The binding of two vectors is the elementwise modulo-L sum of their offset representation. Similarly, unbinding is defined using the modulo-L difference. Both binding and unbinding preserve dimensionality and sparsity. A typical choice for the similarity metric is the normalized dot-product, which counts the number of overlapping elements of two vectors [25].

3. Related work

3.1. Factorizing distributed representations

The resonator network [7,23] avoids brute-force search through the combinatorial space of possible factorization solutions by exploiting the *search in superposition* capability of VSAs. The iterative search process converges empirically by finding correct factors under operational capacity constraints [23]. The resonator network can accurately factorize dense bipolar distributed vectors generated by a two-layer perceptron network trained to approximate the multiplicative binding for colored MNIST digits [7]. Alternatively, the resonator network can also factorize complex-valued product vectors representing a scene encoded via a template-based VSA encoding [47] or convolutional sparse coding [28]. However, the resonator network suffers from a relatively low operational capacity (i.e., the maximum factorizable problem size given a certain vector dimensionality), and the limit cycles that impact convergence. To overcome these two limitations, a stochastic in-memory factorizer [31] introduces new nonlinearities and leverages intrinsic noise of computational memristive devices. As a result, it increases the operational capacity by at least five orders of magnitude, while also avoiding the limit cycles and reducing the convergence time compared to the resonator network.

Nevertheless, we observed that the accuracy of both the resonator network and the stochastic factorizer notably drops (by as much as 16.22%) when they are queried with product vectors generated from deep CNNs processing natural images (see Table 4). This challenge motivated us to switch to alternative block code representations instead of dense bipolar, whereby we can retain high accuracy by using our BCF. Moreover, compared to the state-of-the-art stochastic factorizer, BCF requires fewer iterations irrespective of the number of factors F (see Table 2). Interestingly, it only requires two iterations to converge for problems with a search space as large as 10^4.

3.2. Fixing the final FCL in CNNs

Typically, a learned affine transformation is placed at the end of deep CNNs, yielding a per-class value used for classification. In this FCL classifier, the number of parameters is proportional to the number of class categories. Therefore, FCLs constitute a large portion of the network's total parameters: for instance, in models for edge devices, FCLs constitute 44.5% of ShuffleNetV2 [35], or 37% of MobileNetV2 [50] for ImageNet-1K. This dominant parameter count is more prominent in lifelong continual learning models, where the number of classes quickly exceeds a thousand and increases over time [16].

To reduce the training complexity associated with FCLs, various techniques have been proposed to fix their weight matrix during training. Examples include replacing the FCL with a Hadamard matrix [20], or a cheaper Identity matrix [42], or vertices of a simplex equiangular tight frame [62]. Although partly effective, due to square-shaped structures, these methods are restricted to problems in which the number of classes is smaller than or equal to the feature dimensionality, i.e., $D_i = D_o$. Methods that simply draw class vectors randomly distributed over a hypersphere [37,54] were proposed to address this limitation. However, these methods still need to store the individual class vectors, which imposes the FCL's conventional cost of $\mathcal{O}(D_i \cdot D_o)$ for memory storage and compute complexity during training and inference.

Our BCF with two factors can reduce the memory and compute complexity to $\mathcal{O}(D_i \cdot \sqrt{D_o})$. This is done by using randomly-drawn distributed binary SBCs that form an intermediate product vector space whose dimensionality (D_p) is notably lower than the number of classes ($D_p \ll D_o$), but high enough such that a large number of classes can be expressed thanks to the supplied quasi-orthogonality. We show that the product vector space can be built either at the output of the last convolutional layer directly (i.e., its dimensionality is set by the feature dimension $D_p = D_i$), or at the output of a smaller FCL as a projection layer (i.e., its dimensionality can be chosen). This flexibly enables a trade-off between the number of removable parameters and obtainable accuracy.

4. Part I: Factorization of generalized sparse block codes

This section presents our first contribution: we propose a novel block code factorizer (BCF) that efficiently finds the factors of product vectors based on block codes. We first introduce GSBC, a generalization of the previously presented binary SBC. We present corresponding binding, unbinding, and bundling operations and a novel similarity metric based on the ℓ_∞-norm. We then continue to the exposition and experimental evaluation of our BCF, which is capable of fast and accurate factorization of GSBC product vectors.

4.1. Generalized sparse block codes (GSBCs)

Like binary SBCs, GSBCs divide the D_p-dimensional vectors into B blocks of equal length $L = D_p/B$. However, the individual blocks are not restricted to be binary or sparse. The requirements imposed upon the vectors are that their elements are in $\mathbb{R}^+$, and each block has a unit ℓ_1-norm. Binary SBCs satisfy both constraints and are valid GSBCs. Figure 1 illustrates an example of a binary SBC and a GSBC vector. The blockwise distribution of the GSBC representation can be interpreted as blockwise probabilistic mass functions, serving as a proxy for the binary SBC vector in this example. Neural networks can produce such GSBC product vectors. Besides their benefits in the integration with neural networks, the GSBC representations inside BCF enable more accurate and faster factorization.

The individual operations for the GSBCs are defined as follows:

Binding/unbinding We exploit general binding and unbinding operations in blockwise circular convolution and correlation to support arbitrary block representations. Specifically, if both operands have blockwise unit ℓ_1-norm, the result does as well.

Bundling The bundling of several vectors is defined as their elementwise sum followed by a normalization operation, ensuring that each result block has unit ℓ_1-norm.
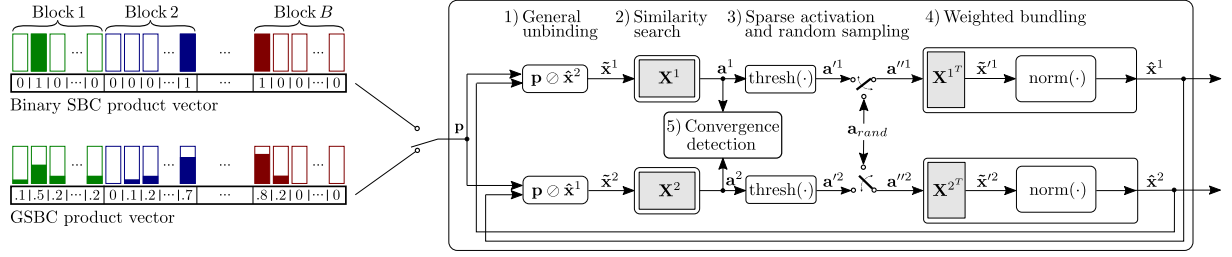
Fig. 1. Block code factorizer (BCF) for $F = 2$ factors. It can factorize both synthetic binary SBC product vectors and GSBC product vectors (**p**) which might result from a neural network mapping.

Table 1
Comparison of operations of binary SBCs and our GSBCs. All operations except for the similarity are applied blockwise

	Binary SBCs [29]	GSBCs (ours)
Binding ($\circledast$)	Modulo-L sum of nonzero indices	Blockwise circular convolution
Unbinding ($\oslash$)	Modulo-L difference between nonzero indices	Blockwise circular correlation
Bundling ($\oplus$)	Argmax of sum	Sum & normalization
Similarity	Dot-product	ℓ_∞-based sim. or dot-product

ℓ_∞-*Based similarity* We propose a novel similarity measure based on the ℓ_∞-norm of the elementwise difference between two GSBC vectors $\mathbf{x}_i$ and $\mathbf{x}_j$:

$$s_\infty(\mathbf{x}_i, \mathbf{x}_j) := 1 - \ell_\infty(\mathbf{x}_i - \mathbf{x}_j), \tag{1}$$

with $\ell_\infty(\mathbf{a}) = \max_i |\mathbf{a}[i]|$, where $\mathbf{a}[i]$ denotes the i-th element of a vector $\mathbf{a}$.

For any GSBC vectors $\mathbf{a}$ and $\mathbf{b}$, it holds that $0 \leqslant \ell_\infty(\mathbf{a} - \mathbf{b}) \leqslant 1$. Therefore, our novel similarity metric satisfies $0 \leqslant s_\infty(\mathbf{a}, \mathbf{b}) \leqslant 1$, whereby equality on the right-hand side holds if, and only if, $\mathbf{a} = \mathbf{b}$. Table 1 compares the operations of the GSBCs with respect to the binary SBCs.

4.2. Factorization problem

We define the factorization problem for GSBCs and our factorization approach for two factors. Applying our method to more than two factors is straightforward; corresponding experimental results will be presented in Section 4.6.

Given two codebooks,[1] $\mathbf{X}^1 := \{\mathbf{x}_i^1\}_{i=1}^{M_1}$ and $\mathbf{X}^2 := \{\mathbf{x}_i^2\}_{i=1}^{M_2}$, and a product vector $\mathbf{p} = \mathbf{x}_i^1 \circledast \mathbf{x}_j^2$ formed by binding two factors from the codebooks, we aim to find the estimate factors $\hat{\mathbf{x}}^1 \in \mathbf{X}^1$ and $\hat{\mathbf{x}}^2 \in \mathbf{X}^2$ that satisfy

$$\mathbf{p} = \hat{\mathbf{x}}^1 \circledast \hat{\mathbf{x}}^2. \tag{2}$$

A naive brute-force approach would compare the product vector (**p**) to all possible combinations spanned by the product space $\mathbf{P} = \mathbf{X}^1 \circledast \mathbf{X}^2 := \{\mathbf{x}_1^1 \circledast \mathbf{x}_1^2, \mathbf{x}_1^1 \circledast \mathbf{x}_2^2, \dots, \mathbf{x}_{M_1}^1 \circledast \mathbf{x}_{M_2}^2\}$. This results in a combinatorial search problem requiring $M_1 \cdot M_2$ similarity computations. BCF can notably reduce the computational complexity.

4.3. Block code factorizer (BCF)

Here, we introduce our novel BCF that efficiently finds the factors of product vectors based on GSBCs, shown in Fig. 1. The product vector decoding begins with initializing the estimate factors $\hat{\mathbf{x}}^1(0)$ and $\hat{\mathbf{x}}^2(0)$ by bundling all

---

[1]In our experiments, the codebooks consist of binary SBC codewords, but they could be GSBC vectors too.

vectors from the corresponding codebooks. Then, the estimate factors are iteratively updated through the following steps.

Step 1: Unbinding. At the start of iteration $t \geqslant 1$, the estimate factors from the previous iteration $t - 1$ are unbound from the product vector by using blockwise circular correlation:

$$\tilde{\mathbf{x}}^1(t) = \mathbf{p} \oslash \hat{\mathbf{x}}^2(t - 1) \tag{3}$$

$$\tilde{\mathbf{x}}^2(t) = \mathbf{p} \oslash \hat{\mathbf{x}}^1(t - 1). \tag{4}$$

Step 2: Similarity search. Next, we query the associative memory, containing the codebook $\mathbf{X}^f$ for factor f, with the unbound factor estimates. We deploy our novel ℓ_∞-based similarity as an effective associative search. At iteration t, this yields a vector of similarity scores $\mathbf{a}^f(t) \in \mathbb{R}^{M_f}$ for each factor f. The i-th element in $\mathbf{a}^f(t)$ is computed as:

$$\mathbf{a}^f(t)[i] = s_\infty\big(\tilde{\mathbf{x}}^f(t), \mathbf{x}_i^f\big). \tag{5}$$

We observe that a conventional dot-product similarity causes a notable performance drop, as shown in Fig. 3.

Step 3: Sparse activation and conditional random sampling. Recent work on stochastic factorizers [31] demonstrated that applying a threshold function to the elements of the similarity vector can improve convergence speed and operational capacity. We deploy a similar idea in our BCF. In this step, the previously computed similarities are compared against a fixed threshold $T \in \mathbb{R}^+$. Similarity values that are larger than the threshold propagate forward, whereas lower ones get zeroed out:

$$\mathbf{a}'^f(t) = \mathrm{thresh}\big(\mathbf{a}^f(t); T\big) \tag{6}$$

$$\mathrm{thresh}(\mathbf{a}; T)[i] = \begin{cases} \mathbf{a}[i], & \text{if } \mathbf{a}[i] \geqslant T \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

This nonlinearity allows us to focus on the most promising solutions by discarding the presumably incorrect low-similarity ones. However, thresholding entails the possibility of ending up with an all-zero similarity vector, effectively stopping the decoding procedure. To alleviate this issue, upon encountering an all-zero similarity vector, we randomly generate a subset of equally weighted similarity values:

$$\mathbf{a}''^f(t) = \begin{cases} \mathbf{a}'^f(t), & \text{if } \mathbf{a}'^f(t) \neq \mathbf{0} \\ \mathbf{a}_{\mathrm{rand}}, & \text{otherwise,} \end{cases} \tag{8}$$

where $\mathbf{a}_{\mathrm{rand}} \in \mathbb{R}^{M_f}$ is a vector in which A-many randomly selected elements which are set to $1/A$. In combination with step 4 (weighted bundling), the conditional random sampling given by Eq. (8) yields an equally weighted bundling of A randomly selected codewords, where A can be interpreted as the sampling width.

The novel threshold and conditional sampling mechanisms are simple and interpretable, and they lead to faster convergence. The stochastic factorizer [31] relied on various noise instantiations at every decoding iteration. The necessary stochasticity was supplied from intrinsic noise of phase-change memory devices and analog-to-digital converters of a computational analog memory tile. Instead, BCF remains deterministic in the decoding iterations unless all elements in the similarity vector are zero, in which case it activates only a single random source. This can be seen as a conditional *restart* of BCF using a new random initialization. The conditional random sampling could be implemented with a single random permutation of a seed vector in which A-many arbitrary values are set to $1/A$. The conditional random sampling mechanism is also interpretable, in the sense that it allows to analytically determine the expected number of decoding iterations, subject to the bundling capacity (i.e., the maximum number of vectors that can be bundled and reliably retrieved). Section 4.7 provides empirical insights.

Step 4: Weighted bundling. Finally, we generate the next factor estimate $\hat{\mathbf{x}}^f(t)$ as the normalized weighted bundling of the factor's codevectors:

$$\hat{\mathbf{x}}^f(t) = \frac{(\mathbf{X}^f)^T \mathbf{a}''^f(t)}{\sum_{i=1}^{M_f} \mathbf{a}''^f(t)[i]}. \tag{9}$$

$\mathbf{X}^f$ is a row-matrix of codevectors for factor f. The codevectors $\{\mathbf{x}_i^1\}_{i=1}^{M_1}$ are GSBCs with unit ℓ_1-norm blocks; hence, dividing the weighted bundling by the sum of the weights yields valid GSBCs.

Step 5: Convergence detection. The iterative decoding is repeated until BCF converges or a predefined maximum number of iterations (N) is reached. We define the maximum number of iterations such that BCF does at most as many similarity searches as the brute-force approach [31]:

$$N := \frac{\prod_{f=1}^F M_f}{\sum_{f=1}^F M_f}. \tag{10}$$

The convergence detection mechanism is based on an additional, fixed threshold. Decoding is stopped as soon as both similarity vectors ($\mathbf{a}^1(t)$ and $\mathbf{a}^2(t)$) contain an element that exceeds a predefined detection threshold value (T_c) [31]. We set it to $T_c = 0.8$ for synthetic product vectors and $T_c = 0.5$ for noisy product vectors from deep neural networks.

4.4. Hyperparameter optimization

This section explains the methodology for finding optimal BCF hyperparameters to achieve high accuracy and fast convergence. The optimal configuration is denoted by $\mathbf{c}^\star = (T^\star, A^\star)$, corresponding to the optimal threshold and sampling width. As an automatic hyperparameter search method, we employ Bayesian optimization [55].

The loss function is defined as the error rate given by the percentage of incorrect factorizations out of 512 randomly selected product vectors. To put a strong emphasis on fast convergence, we reduced the maximal number of the iterations to $N' = 0.05N$ for all Bayesian optimization runs. The error rate is an unknown function of the hyperparameters, modeled as a Gaussian process with a radial basis function kernel. Hyperparameter sampling is done using the expected improvement acquisition function.

For each problem (F, D_p, $\prod_{f=1}^F M_f$, B), we run a separate hyperparameter search, each of which tests 200 different hyperparameter combinations restricted to the domains $A \in [0, M_f]$ and $T \in [0, 1]$. Finally, we select the hyperparamters with the lowest error rate at the default maximum number of iterations (N).

Figure 2 shows the resulting threshold (T) and sampling width (A) over various problem sizes for $D_p = 512$, $B = 4$, and $F = 2$. For a range of problem sizes (10^3–10^5), BCF does not require the threshold and sampling

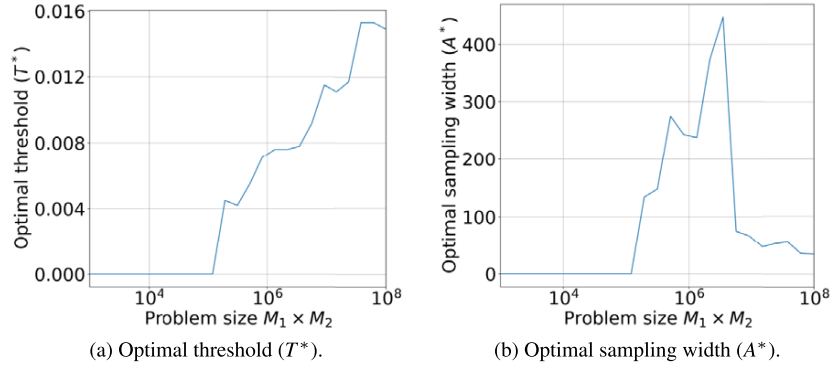(a) Optimal threshold (T^*).   (b) Optimal sampling width (A^*).

Fig. 2. Optimal threshold and sampling width found using Bayesian optimization with $D_p = 512$, $B = 4$, and $F = 2$.

dynamics: it sets the threshold and the sampling width to 0. For larger problem sizes ($> 10^5$), the threshold T grows with the problem size. This can be explained by the fact that querying larger codebooks is likely to activate more codevectors, which will be bundled. As such, we expect a higher interference between likely incorrect low-similarity solutions and promising high-similarity solutions. A higher threshold effectively reduces the number of bundled vectors, reducing interference. Similarly, the sampling width (A) grows until a problem size of 4,000,000, where it sharply declines. The sharp decline was observed for all investigated problem settings (F, D_p, B) and might stem from the limited bundling capacity.

4.5. Experimental setup

We evaluate the performance of our novel BCF on randomly selected synthetic product vectors. For each problem $(F, D_p, \prod_{f=1}^{F} M_f, B)$, we assess the factorization accuracy and the number of iterations by averaging over 5000 experiments. In each experiment, we randomly select one vector from each of the F-many codebooks, bind the selected vectors together to form a product vector, then use the product vector as the input into BCF. In this section, the queries are always binary SBCs. In contrast, the representations inside BCF (i.e., factor estimates at any step of the decoding loop) do not have this restriction imposed upon them unless specified otherwise. The operational capacity is defined as the largest problem size for which BCF achieves an accuracy higher than 99% [23].

4.6. Comparative results

Figure 3 compares the accuracy (left) and the number of iterations (right) of various BCF configurations with $D_p = 512$ and $F = 2$. Dotted lines indicate a less than 99% factorization accuracy. Starting with binary SBC vectors and the dot-product similarity metric, we can see that this BCF configuration fails to solve any problem of size larger than 10^3 accurately. The operational capacity increases to $4.2 \cdot 10^3$ when relaxing the sparsity constraint of binary SBCs by allowing for GSBC representations inside BCF. However, the required iterations are still high, requiring almost as many searches as the brute-force approach. The introduction of the ℓ_∞-based similarity increases the operational capacity by more than an order of magnitude. We can also notice a drastic reduction in the number of iterations necessary for converging to the correct solution. For problem sizes up to 10^4, BCF needs only 2 iterations to converge, the minimum possible number of iterations to detect convergence reliably. However, as the problem size goes beyond $1.2 \cdot 10^5$, BCF encounters limit cycles and spurious fixed points, hindering its convergence to the correct solution [23]. To this end, we introduce the threshold nonlinearity coupled with conditional random sampling. With these new dynamics, BCF further increases the operational capacity by over an order of magnitude to $5 \cdot 10^6$.

Next, we analyze BCF's decoding performance for a varying number of blocks (B), vector dimensions (D_p), and numbers of factors (F). Figure 4 shows the number of iterations of BCF for $F = 2$ (left) and $F = 3$ (right) factors when varying the vector dimension. For both two and three factors, the operational capacity and convergence speed increase with growing vector dimensionality. As we move from two to three factors, the operational capacity
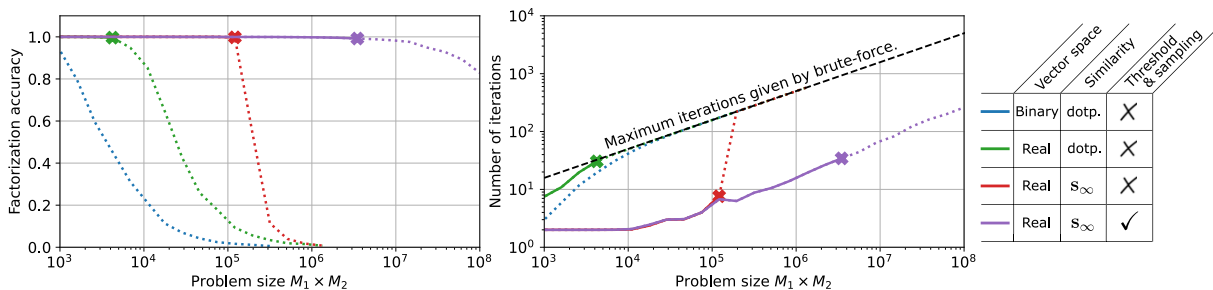

Fig. 3. Factorization accuracy (left) and number of iterations (right) of various BCF configurations on synthetic (i.e., exact) product vectors for different problem sizes ($\prod_{f=1}^{F} M_f$). We set $D_p = 512$, $F = 2$, and $B = 4$. The maximum operational capacity is marked with an x. Problem sizes exceeding the operational capacity are marked with dashed lines which face an accuracy lower than 99%. BCF configured with binary SBC operations (in blue) cannot solve any of the displayed problem sizes at the required accuracy.
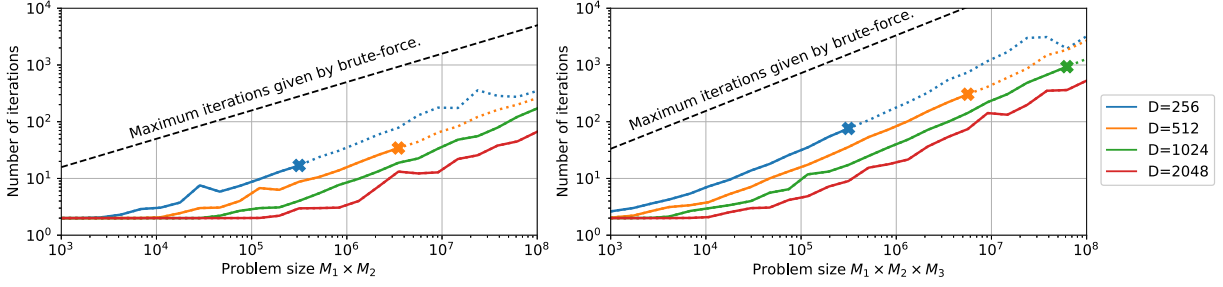
Fig. 4. Effect of the dimension D_p on the number of iterations for BCF with $B = 4$, $F = 2$ (left) and $F = 3$ (right).
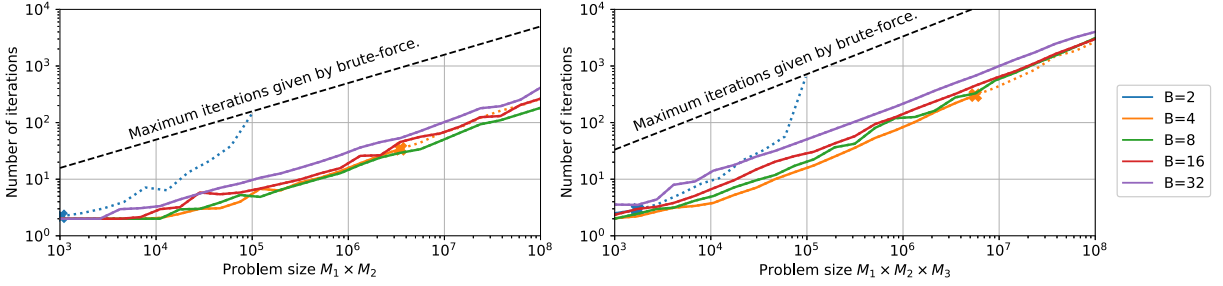

Fig. 5. Effect of the number of blocks B on the number of iterations for BCF with $D = 512$, $F = 2$ (left) and $F = 3$ (right).

Table 2

Comparison between stochastic factorizer [31] and our BCF at problem size 10^6

F	D	Number of iterations	
		Dense bipolar [31]	BCF ($B = 4$)
2	1024	68.47	11.16
3	1536	72.48	52.91
4	2048	157.17	89.05

remains approximately the same while the number of decoding iterations increases. However, an increase in the number of iterations does not directly lead to higher computational cost as each iteration requires fewer similarity computations ($F \cdot \sum_f M_f$) for larger F due to the F-root dependence of M_f. For example, at $D_p = 512$ and $\prod_{f=1}^{F} M_f = 10^6$, BCF at $F = 2$ requires, on average, 15.73 iterations corresponding to a total of 31,460 searches, whereas at $F = 3$ it requires, on average, 85.17 iterations and 25,551 searches.

Figure 5 shows BCF's performance for a fixed D_p while varying the number of the blocks (B) for two and three factors. For a very small number of blocks ($B = 2$), the operational capacity lies at approximately 10^3 for both $F = 2$ and $F = 3$. The operational capacity increases to around $5 \cdot 10^6$ when $B = 4$. Further increasing the number of blocks ($B \geqslant 8$) exhibits an operational capacity beyond 10^8, the largest problem size we measured. The convergence speed peaks at $B = 4$ and $B = 8$ blocks, gradually decreasing as the vectors get denser. Overall, the experimental results shown in Fig. 4 and Fig. 5 demonstrate the broad applicability of our BCF: it accurately solves factorization problems within the computational constraints for a wide range of problem sizes, block sizes ($B \geqslant 4$), and number of factors ($F \in \{2, 3\}$).

Finally, we compare our BCF with the state-of-the-art stochastic factorizer [31] operating with dense bipolar vectors. We fix the problem size to 10^6 and compare the number of iterations for three configurations according to those featured in [31], namely $F = 2$ with $D_p = 1024$, $F = 3$ with $D_p = 1536$, and $F = 4$ with $D_p = 2048$. We configure our BCF with $B = 4$ blocks. Table 2 summarizes the results. Both factorizers achieve $> 99\%$ accuracy across all configurations, whereby our BCF requires up to $6\times$ fewer iterations.

4.7. Ablation study

This section provides more insights into BCF's two main hyperparameters: the threshold (T) and the sampling width (A).

Effect of sampling width in an unconditional random sampler The sampling width (A) determines how many codevectors will be randomly sampled and bundled in case the thresholded similarity is an all-zero vector. Intuitively, we expect too low sampling widths to result in a slow walk over the space of possible solutions. Alternatively, suppose the sampling width is too large (e.g., larger than the bundling capacity). In that case, we expect high interference between the randomly sampled codevectors to hinder the accuracy and convergence speed due to the limited bundling capacity.

To experimentally demonstrate this effect, we run BCF with $F = 2$ in an operational mode that corresponds to unconditional random vector sampling. Concretely, we fix the sampling width (A), generate the initial guess as a bundling of A-many randomly selected codevectors, and set both the sparsification threshold (T) and the convergence detection threshold (T_c) to the inverse sampling width ($1/A$). With this configuration, we expect BCF to execute a random walk over the solution space with sampling width determining the number of solutions that are simultaneously evaluated. If the procedure samples the correct solution and the interference from sampled incorrect solutions is low, the correct factor triggers the threshold, and the factorization stops. As a result, at every iteration, we test $M_f \cdot A^{F-1}$ combinations per factor f. The expected number of iterations for this procedure is:

$$\mathbb{E}[t] = \frac{\prod_{f=1}^{F} M_f}{\left(\sum_{f=1}^{F} M_f\right) \cdot A^{F-1}}, \tag{11}$$

where the numerator reflects the overall problem size, and the denominator is the number of combinations the random sampler tests per iteration. With $F = 2$ factors and codebooks of equal size $M_1 = M_2 = M$, the expected number of iterations equals $M/(2A)$.

Figure 6 shows experimental results with this factorizer mode for $F = 2$, $M = 1000$, varying D_p between 512 and 1024. The expected number of iterations corresponds to the expression $M/(2A)$. All presented configurations reach $> 99\%$ accuracy, but in a different number of iterations. For small sampling widths, the empirical results match the expectation. As the sampling width increases, the discrepancy between the empirical and theoretical results grows, more evidently at the smaller dimensions. The discrepancy could stem from the bundling capacity, i.e., the number of retrievable elements, which decreases with a shrinking dimension.

Effect of sampling width (A) in BCF In this set of experiments, we do not restrict the threshold to be $1/A$. Instead, we run a grid search for each sampling width over threshold values (T) in [0, 1] and use the optimal value in our benchmarks.

Table 3 shows how the accuracy and the number of iterations change as we vary the sampling width (A) in {10, 50, 100, 200, 500, 1000}. As expected, there is a sweet spot for sampling width, which lies at around 100 in
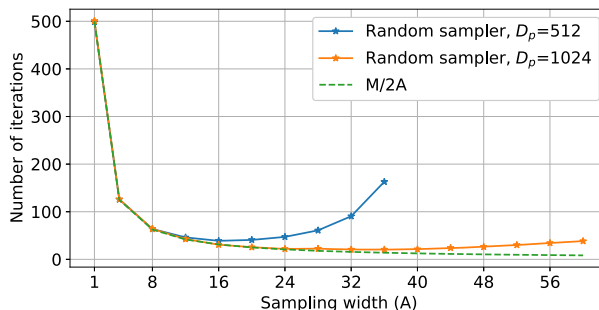

Fig. 6. Number of iterations when BCF is configured as an unconditional random sampler with varying sampling width (A). We set $B = 4$, $F = 2$, and $M_1 = M_2 = M = 1000$.

Table 3

BCF performance when varying the sampling width (A). $D_p = 512$, $F = 2$, $M_1 = M_2 = 1000$

A	T^*	Accuracy	Num. iters.
10	0.00602	99.4%	39.16
50	0.00641	99.4%	17.86
100	0.00641	99.4%	15.73
500	0.00722	98.6%	24.25
1000	0.00441	46.9%	276.78

(a) ℓ_∞-based similarity.
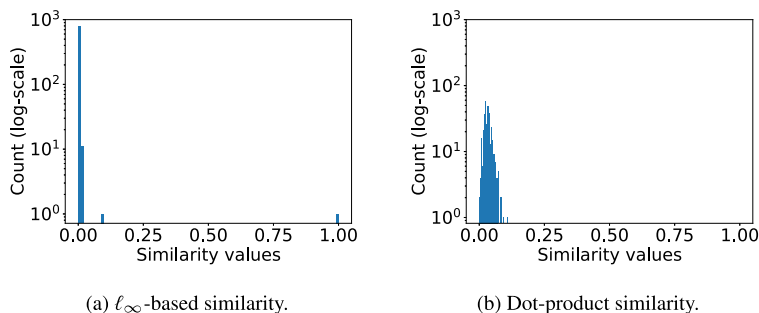

(b) Dot-product similarity.

Fig. 7. Log-scale histograms of ℓ_∞-based and dot-product similarities. BCF with $D_p = 512$, $B = 4$, $F = 2$, $M_1 = M_2 = 200$, and $T = 0$.

this setting. Smaller values of sampling widths do not negatively impact the accuracy, but convergence speed does decrease. As we increase the sampling width beyond 100, the accuracy drops. Moreover, with a fine-tuned threshold value, we can factorize product vectors notably faster (15.73 iterations) than when BCF is run in the unconditional random sampling mode (38.87 iterations).

Similarity metric   Here, we compare the ℓ_∞-based similarity with the dot-product similarity by considering the similarity distributions of the associative memory search inside BCF. We select a problem size that cannot be solved by the dot-product similarity, but can be solved by the ℓ_∞-based similarity. The threshold nonlinearity and conditional random sampling are disabled. For $F = 2$, $B = 4$, and $D_p = 512$, one such problem size is $4 \cdot 10^4$. We execute the decoding for two iterations and show the resulting histogram in Fig. 7. The ℓ_∞-based similarity tends to induce sparse activations: most similarities have a value of 0 and will have no effect on the weighted bundling of codevectors. Conversely, the dot-product similarity exhibits a wider distribution with almost no zero-valued similarities. Finally, the ℓ_∞-based similarity found the correct solution (similarity value of 1), whereas the dot-product similarity did not.

5. Part II: Effective replacement of large FCLs with block code factorizers

So far, we have applied our BCF on synthetic (i.e., exact) product vectors. In this section, we present our second contribution, expanding the application of our BCF to classification tasks in deep CNNs. This is done by replacing the large final FCL in CNNs with our BCF, as shown in Fig. 8. Instead of training C hyperplanes for C classes, embodied in the trainable weights $\mathbf{W} \in \mathbb{R}^{D_i \times D_o}$ of the FCL, where $C = D_o$, we represent the classes with a fixed binary SBC product space $\mathbf{P} \in \{0, 1\}^{D_p \times D_o}$. The product space requires only $B \cdot \sum_{f=1}^{F} M_f$ fixed integer values to be stored with the binary SBC offset notation accounting for 256 values on ImageNet-1K with $B = 4$ and $M_1 = M_2 = 32$. We provide two variants to interface the D_i-dimensional output features of the CNN's final convolutional layer with our BCF, depicted in Fig. 8b and Fig. 8c. In the first variant, BCF is directly interfaced with the CNN's output features; hence, the dimensionality becomes $D_p = D_i$. The second variant uses an intermediate, trainable projection $\mathbf{W}' \in \mathbb{R}^{D_i \times D_p}$, where $D_p \ll D_o$, The number of parameters is notably reduced in both variants, by $D_o \cdot D_i$ without the projection, and by $(D_o - D_p) \cdot D_i$ with the projection.
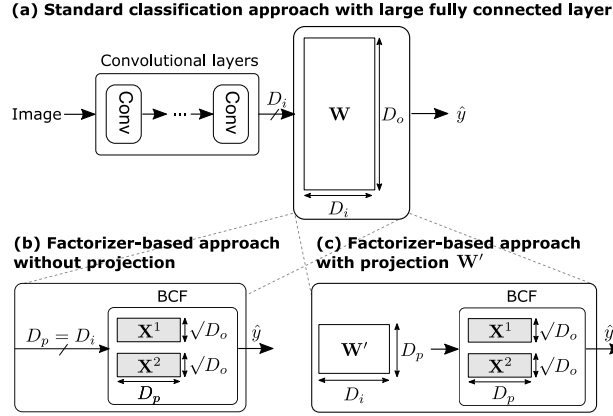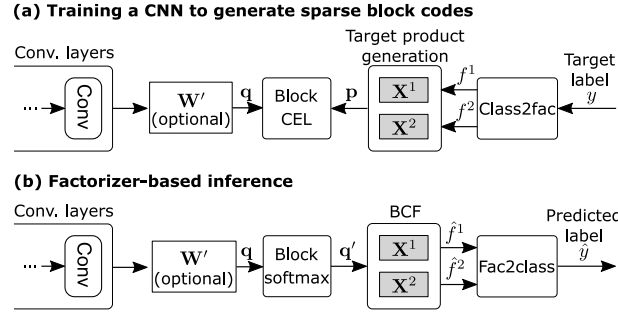
(a) Standard classification approach with large fully connected layer

(b) Factorizer-based approach without projection

(c) Factorizer-based approach with projection $\mathbf{W}'$

Fig. 8. Replacement of a large FCL with our BCF (b) without or (c) with a projection $\mathbf{W}'$.

(a) Training a CNN to generate sparse block codes

(b) Factorizer-based inference

Fig. 9. Training and inference with BCF.

5.1. Casting classification as a factorization problem

First, we describe how the classification problem can be transformed into a factorization problem. The codebooks and product space are naturally provided if a class is a combination of multiple attribute values. For example, the RAVEN dataset contains different objects formed by a combination of shape, position, color, and size. Hence, we define four codebooks ($\mathbf{X}^1$, $\mathbf{X}^2$, $\mathbf{X}^3$, and $\mathbf{X}^4$) where the size of each codebook (M_f) corresponds to the number of values the individual attribute can have [18] (e.g., the codebook $\mathbf{X}^1$ representing five shapes has $M_1 = 5$ elements). The resulting product space is $\mathbf{P} = \mathbf{X}^1 \circledast \mathbf{X}^2 \circledast \mathbf{X}^3 \circledast \mathbf{X}^4$.

If no such semantic information is available, the codebooks and product space are chosen arbitrarily. When targeting two factors, we first define a product space $\mathbf{P} = \mathbf{X}^1 \circledast \mathbf{X}^2$ that contains $M_1 \cdot M_2$ unique quasi-orthogonal product vectors. The size of the product space is set to the number of classes C, such that each product vector in $\mathbf{P}$ can be assigned to a unique class. For example, for representing the $C = 100$ classes in the CIFAR-100 dataset, we define a product space with size 100 using two codebooks of size $M_1 = M_2 = \sqrt{C} = 10$. Then, the product vector $\mathbf{p}_1 := \mathbf{x}_1^1 \circledast \mathbf{x}_1^2$ belongs to "*class 1*" and $\mathbf{p}_{100} := \mathbf{x}_{10}^1 \circledast \mathbf{x}_{10}^2$ to "*class 100*".[2]

5.2. Training CNNs with blockwise cross-entropy loss

After defining the product space, we train a function f_θ (e.g., a CNN) with trainable parameters θ to map the input data (e.g., images) to the target product vectors of the corresponding classes. Figure 9a illustrates the training procedure. Given a labeled training sample ($\mathbf{I}$, y) containing the input image $\mathbf{I} \in \mathbb{R}^{c \times h \times w}$ and the target label y,

---

[2]If $\sqrt{C}$ is not an integer, we take the ceiling of $\sqrt{C}$.

we first pass the image through the function f_θ, yielding $\mathbf{q} = f_\theta(\mathbf{I})$. Next, we generate the target product vector by mapping the target label y to the factor indices (f^1 and f^2) and forming the corresponding product $\mathbf{p} = \mathbf{x}^1_{f^1} \circledast \mathbf{x}^2_{f^2}$.

A typical loss function for binary sparse target vectors is the binary cross-entropy loss in connection with the sigmoid nonlinearity. However, we experienced a notable classification drop when using the binary cross-entropy loss; e.g., the accuracy of MobileNetV2 on the ImageNet-1K dataset dropped below 1% when using this loss function. To this end, we propose a novel blockwise loss that computes the sum of per-block categorical cross-entropy loss (CEL). For each block b, we extract the L-dimensional block $\mathbf{q}_b := \mathbf{q}[(b-1)L + 1 : bL]$ from the output features $\mathbf{q}$ and the target index $\dot{\mathbf{p}}[b]$. Then, the blockwise CEL is defined as:

$$\mathcal{L}(\mathbf{q}, \dot{\mathbf{p}}, s) = \frac{1}{B} \sum_{b=1}^{B} \mathcal{L}_{\text{CEL}}(s \cdot \mathbf{q}_b, \dot{\mathbf{p}}[b]), \tag{12}$$

where $\mathcal{L}_{\text{CEL}}$ is the categorical CEL, which combines the softmax activation with the negative log-likelihood loss, and s a trainable inverse softmax temperature for improved training [20,53,54]. The loss function $\mathcal{L}$ is minimized by batched stochastic gradient descent (SGD).

5.3. BCF-based inference

The BCF-based inference is illustrated in Fig. 9b. We pass a query image ($\mathbf{I}$) through the CNN, yielding the query product ($\mathbf{q}$), which can be interpreted as a "noisy" version of the ground-truth product vector $\mathbf{p}$. We then search for the product vector $\hat{\mathbf{p}} \in \mathbf{P}$ with the highest similarity to $\mathbf{q}$. One baseline is to compute the similarity between $\mathbf{q}$ and each product vector in $\mathbf{P}$ in a brute-force manner; however, this requires many similarity computations and the storage of each product vector in $\mathbf{P}$. Instead, we search for the closest product vector by factorizing the query product vector using BCF, shown in Fig. 9b.

We pass the output of the CNN through a blockwise softmax function with an inverse softmax temperature s_F, which shapes and normalizes the blockwise distribution of the query vector. The optimal inverse softmax temperature was found to be $s_F = 1.5$, based on a grid search on the ImageNet-1K training set with MobileNetV2, and applied for all architectures.

5.4. Experimental setup

Datasets   We evaluate our new method on three image classification benchmarks.

ImageNet-1K. The ImageNet-1K dataset [3] is a large-scale image classification benchmark with colored images from $C = 1000$ different classes. The training set contains over 1.2 M samples, and the validation set includes 50,000 samples (50 per class), all with a resolution of 224×224.

CIFAR-100. The CIFAR-100 dataset [27] contains colored images with a resolution of 32×32 from $C = 100$ classes. The dataset provides 500 samples for training and 100 for testing per class.

RAVEN. The RAVEN dataset [59] contains Raven's progressive matrices tests with gray-scale images with a resolution of 160×160. A test provides 8 context and 8 answer panels, each containing objects with the following attributes: position, type, size, and color. In this work, we exclusively target the recognition of single objects inside the panels. Therefore, we extract all panels with single objects from the center, 2×2 grid, and 3×3 grid constellation. The extraction gives us a dataset with $136,321$ panels for training, $45,144$ for validation, and $45,144$ for testing. We combine the positions from the different constellations, yielding 14 unique positions: 1 for the center, 4 for the 2×2 grid, and 9 for the 3×3 grid. Overall, this dataset contains $C = 4200$ attribute combinations (14 positions $\times$ 5 types $\times$ 6 sizes $\times$ 10 colors).

Architectures   ShuffleNetV2 [35], MobileNetV2 [50], ResNet-18, and ResNet-50 [15] serve as baseline architectures. In addition to our BCF-based replacement approach, we evaluate each architecture with the bipolar dense resonator [7,23], the Hadamard readout [20], and the Identity readout [42]. For the FCL replacement strategies without the intermediate projection layer, we removed the nonlinearity and batch norm of the last convolutional layer of all CNN architectures. This notably improved the accuracy of all replacement strategies; e.g., the accuracy

of MobileNetV2 with the Identity replacement improved from 60.28% to 70.72% when removing the batch norm and the ReLU6 of the last convolutional layer. All other architectural blocks remained the same, including shortcuts in the last block of the ResNet-18 and ResNet-50. To apply the Identity approach where $D_i \neq D_o$, we used an identity matrix that reads out the first D_o elements of the output feature vector and ignores the remaining $D_i - D_o$ elements. We could not adjust the dimension D_i since it would have required additional adaptations in the CNNs, e.g., a downsampling layer in the shortcut connection of ResNet-50.

Training setup   The CNN models are implemented in PyTorch (version 1.11.0) and trained and validated on a Linux machine using up to 4 NVIDIA Tesla V100 GPUs with 32 GB memory. We train all CNN architectures with SGD with architecture-specific hyperparameters, described in Appendix A.1. For each architecture, we use the same training configuration for the baseline and all replacement strategies (i.e., Hadamard, Identity, resonator networks, and our BCF). We repeat the main experiments five times with a different random seed and report the average results and standard deviation to account for training variability.

5.5. Comparative results

Table 4 compares the classification accuracy of the baseline with various replacement approaches without projection, namely Hadamard [20], Identity [42], bipolar dense [7], and our BCF. On ImageNet-1K, BCF reduces the total number of parameters of deep CNNs by 4.4%–44.5%,[3] while maintaining a high accuracy within 2.39% across the majority of architectures, with the only exception of ShuffleNetV2 showing 4.46% accuracy drop due to its very large FCL accounting for 44.5% of total parameters. Our BCF matches the brute-force accuracy within 0.44% in all architectures while requiring only 5–7 iterations on average, despite the query product vectors from the CNNs being "noisy." Inspired by extremely fast convergence on the synthetic experiments (see Fig. 3), we could show that BCF can match the brute-force accuracy within 0.54% when only allowing up to $N = 3$ iterations (see Appendix A.4). This does not hold for the bipolar dense resonators showing notable accuracy drop (up to 16.22%), compared to the

Table 4

Comparison of approaches which replace the final FCL without any projection layer ($D_i = D_p$). We report the average accuracy $\pm$ the standard deviation over five runs with different seeds for the baseline and our GSBCs

Dataset/architecture	(D_i, D_o)	Baseline acc.	FCL replacement approach									
			Had. acc.	Id. acc.	Bipolar dense			Our BCF ($B = 4$)				
					BF acc.	Res. acc.*	Avg. iter.*	BF acc.	Fac. acc.	Avg. iter.	Param. saving↑	FCL comp. saving↑
ImageNet-1K												
ShuffleNetV2	(1024, 1k)	$69.22^{\pm0.20}$	68.02	67.62	66.17	54.54	150	$65.09^{\pm0.10}$	$64.76^{\pm0.13}$	7	44.5%	55.2%
MobileNetV2	(1280, 1k)	$71.57^{\pm0.13}$	71.30	70.72	70.64	60.83	150	$70.00^{\pm0.07}$	$69.76^{\pm0.13}$	6	37.6%	61.6%
ResNet-18	(512, 1k)	$70.39^{\pm0.11}$	N/A	N/A	68.65	54.17	150	$68.44^{\pm0.08}$	$68.00^{\pm0.07}$	7	4.4%	55.2%
ResNet-50	(2048, 1k)	$76.21^{\pm0.28}$	75.30	74.65	75.80	67.98	150	$76.34^{\pm0.04}$	$76.25^{\pm0.07}$	5	8.0%	68.0%
CIFAR-100												
ResNet-18	(512, 100)	$78.10^{\pm0.31}$	77.21	76.56	76.63	71.15	150	$77.31^{\pm0.15}$	$77.19^{\pm0.17}$	2	0.5%	60.0%
RAVEN												
ResNet-18	(512, 4.2k)	$99.88^{\pm0.01}$	N/A	N/A	99.89	94.92	45	$99.87^{\pm0.01}$	$99.82^{\pm0.02}$	16	16.2%	86.7%
p-value	–	–	0.125	0.125	0.063	0.031	–	0.094	0.063	–	–	–

Acc. = Accuracy (%); N/A = Not applicable; Had. = Reproduced Hadamard [20]; Id. = Reproduced Identity [42]; BF = Brute-force; Res. = Resonator nets [7]; Fac. = Block code factorizer whereby it sets $F = 2$ for ImageNet-1K and CIFAR-100, and $F = 4$ for RAVEN. *p*-value is determined by signed Wilcoxon test with respect to baseline accuracy.
* Maximum number of iterations was increased to $N = 150$ for better performance in the resonator nets that leads to $10\times$ more operations than the brute-force search.

---

[3]The relative parameter reduction depends on the size of the overall network and the final FCL, which we replace with our method.

brute-force search, despite allowing a high number of iterations ($N = 150$) and conducting extensive hyperparameter tuning across various loss functions, including arcface [4] (see Appendix A.2).

On CIFAR-100, BCF matches the baseline within 0.91% with only 2 average iterations; while on RAVEN, it requires a slightly higher number of iterations (16) due to the larger number of factors ($F = 4$) and the asymmetric codebook sizes. Across all datasets and architectures, our BCF reduces the large FCL's computational cost by 55.2–86.7%.

Considering the other FCL replacement approaches, Hadamard consistently outperforms Identity. However, both the memory and computation requirements of Hadamard are $\mathcal{O}(D_i \cdot D_o)$, while our BCF reduces both to $\mathcal{O}(D_i \cdot \sqrt{D_o})$, as $F = 2$ and $N = 3$ are constant. Hence, Hadamard is ineffective for a large value of D_o. Moreover, Identity is only competitive when D_i is within the range of D_o (MobileNetV2 and ShuffleNetV2); for other combinations, either it is not applicable (ResNet-18 where $D_i < D_o$), or ineffective (ResNet-50 where $D_i > D_o$).

We compare our approach to weight pruning techniques, which usually sparsify the weights in all layers, whereas we focus on the final FCL due to its dominance in compact networks. Such pruning can be similarly applied to earlier layers in addition to our method. Pruning the final FCL of a pretrained MobileNetV2 with iterative magnitude-based pruning [61] yields notable accuracy degradation as soon as more than 95% of the weights are set to zero. In contrast, our method remains accurate (69.76%) in high sparsity regimes (i.e., 99.98% zero elements). See Appendix A.6 for more details.

Furthermore, we compare our results with [54], which randomly initialize the final FCL and keep it fixed during training. On CIFAR-100 with ResNet-18, they could show that fixing the final FCL layer even slightly improves the accuracy compared to the trainable FCL (75.9% vs. 74.9%; see their Table 1). However, our BCF-based approach outperforms their fixed FCL approach (77.19%) while reducing the memory requirements and the FCL compute cost.

Table 5 shows the performance of our BCF when using the projection layer ($D_p = 512$). The projection layer improves the BCF-based accuracy in all benchmarks, especially in cases where the BCF replacement approach faced challenges (i.e., ShuffleNetV2). Overall, we reduce the total number of parameters by 2.2%–21.7% while maintaining the accuracy within 1.2%, compared to the baseline with trainable FCL.

5.6. Ablation study

We give further insights into the BCF-based classification by analyzing the effect of the number of blocks, the projection dimension, the number of factors, and the initialization of the CNN weights.

Number of blocks B Table 6 shows the brute-force and BCF classification accuracy for block codes with different numbers of blocks. The brute-force accuracy degrades as the number of blocks (B) increases, particularly in networks where the final FCL is dominant (e.g., ShuffleNetV2). These experiments demonstrate that deep CNNs are well-matched with very sparse vectors (e.g., $B = 4$), and motivated us to devise a BCF that can factorize product vectors with such a low number of blocks. Our BCF achieves an accuracy within 0.43% of the brute-force accuracy

Table 5

Classification accuracy when interfacing the last convolution layer with BCF using a projection layer with $D_p = 512$

Dataset/architecture	Baseline			Our BCF ($D_p = 512$, $B = 4$)				
	(D_i, D_o)	# Param.	Acc.	BF acc.	Fac. acc.	Avg. iter.	Param. saving↑	FCL comp. saving↑
ImageNet-1K								
ShuffleNetV2	(1024, 1k)	2.3 M	$69.22^{\pm0.20}$	$68.67^{\pm0.11}$	$68.41^{\pm0.14}$	6	21.7%	29.6%
MobileNetV2	(1280, 1k)	3.4 M	$71.57^{\pm0.13}$	$71.69^{\pm0.11}$	$71.49^{\pm0.10}$	6	18.4%	33.4%
ResNet-18	(512, 1k)	11.5 M	$70.39^{\pm0.11}$	$69.57^{\pm0.17}$	$69.19^{\pm0.13}$	6	2.2%	10.4%
ResNet-50	(2048, 1k)	25.5 M	$76.21^{\pm0.28}$	$76.72^{\pm0.06}$	$76.56^{\pm0.08}$	5	3.9%	40.8%
RAVEN								
ResNet-18	(512, 4.2k)	13.3 M	$99.88^{\pm0.01}$	$99.88^{\pm0.00}$	$99.85^{\pm0.02}$	16	14.2%	78.6%
p-value	–	–	–	0.465	0.313	–	–	–

Table 6

Classification accuracy (%) on ImageNet-1K for the baseline and the block code-based replacement approaches ($F = 2$) with different number of blocks (B). Lower number of blocks (B) results in higher accuracy

Classification approach	B	ShuffleNetV2		MobileNetV2		ResNet-18		ResNet-50	
		BF	Fac.	BF	Fac.	BF	Fac.	BF	Fac.
Baseline	–	$69.22^{\pm0.20}$	–	$71.57^{\pm0.13}$	–	$70.39^{\pm0.11}$	–	$76.21^{\pm0.28}$	–
GSBCs	4	$65.09^{\pm0.10}$	$64.76^{\pm0.13}$	$70.00^{\pm0.07}$	$69.76^{\pm0.13}$	$68.43^{\pm0.08}$	$68.00^{\pm0.07}$	$76.34^{\pm0.04}$	$76.25^{\pm0.07}$
	8	64.30	63.65	69.53	69.20	67.90	67.76	76.69	76.48
	16	63.22	62.04	69.34	68.83	67.02	64.98	76.52	76.27
	32	61.96	59.74	69.12	68.35	65.09	61.39	76.08	75.62

Table 7

Loading a pretrained ResNet-18 model improves accuracy and training time. Classification accuracy (%) on ImageNet-1K using BCF with ResNet-18 (with projection $D_p = 512$, $B = 4$, $F = 2$)

	Pretrained model	Training epochs	BF acc.	Fac. acc.
Baseline	✗	100	$70.39^{\pm0.11}$	–
GSBCs	✗	100	$69.57^{\pm0.17}$	$69.10^{\pm0.14}$
GSBCs	✓	50	$68.50^{\pm0.14}$	$68.05^{\pm0.08}$
	✓	75	$69.21^{\pm0.09}$	$68.83^{\pm0.07}$
	✓	100	$70.08^{\pm0.16}$	$69.72^{\pm0.15}$

for product vectors with $B = 4$. For a larger number of blocks ($B \geqslant 8$), our BCF matches the brute-force accuracy within 3.7%. Note that BCF's hyperparameters were exclusively tuned for $B = 4$, and then applied for other blocks. Hence, BCF for $B = \{8, 16, 32\}$ could be further improved by hyperparameter tuning.

Projection dimension D_p   We varied the projection dimension (D_p) from 128 (high reduction) to 1000 (no reduction since $D_p = D_o$) for MobileNetV2 on ImageNet-1K. With an extremely low dimension ($D_p = 128$), BCF shows a 2.86% accuracy drop compared to the baseline with trainable FCL while saving 32.8% of the parameters. When going to higher dimensions (e.g., $D_p = 768$), BCF even surpasses the baseline accuracy while saving 8.7% of the parameters. See Appendix A.3.

Number of factors F   So far, we have evaluated BCF with two factors, each having codebooks of size $M_1 = M_2 = 32$ on the ImageNet-1K dataset. We demonstrate BCF's capability with $F = 3$ codebooks of size $M_f = 10$, which achieves similar accuracy while requiring a higher number of iterations (11 vs. 6) than $F = 2$. However, since each iteration requires fewer search operations for $F = 3$ (30 vs. 64 searches), the overall saving in computational complexity of the FCL remains similar. See Appendix A.5.

Initialize ResNet-18 with pretrained weights   Finally, we show that the training of BCF-based CNNs can be improved by initializing their weights from a model that was pretrained on ImageNet-1K. Table 7 shows the positive impact of pretraining of ResNet-18 (with projection) on ImageNet-1K. The pretraining improves the accuracy of BCF by 0.62%, compared to the random initialization, when keeping the number of epochs the same. Moreover, when reducing the number of training epochs to 75 and 50, BCF still yielded accurate predictions (68.83% and 68.05%). This experiment shows that our BCF-based method can be applied with reduced training cost if a pretrained model is available.

6. Discussion

BCF is a powerful tool to iteratively decode both synthetic and noisy product vectors by efficiently exploring the exponential search space using computation in superposition. As one viable application, this allowed us to effectively replace the final large FCL in CNNs, reducing the memory footprint and the computational complexity of the

model while maintaining high accuracy. If the classes were a natural combination of multiple attribute values (e.g., the objects in RAVEN), we cast the classification as a factorization problem by defining codebooks per attribute, and their combination as vector products. In contrast, the codebooks and product space were chosen arbitrarily if the dataset did not provide such semantic information about the classes (e.g., ImageNet-1K or CIFAR-100). Instead of this random fixed assignment, one could use an optimized dynamic label-to-prototype vector assignment [49]. It would be also interesting if a product space could be *learned*, e.g., by gradient descent, revealing the inherent structure and composition of the individual classes. Besides, other applications may benefit from a structured product representation, e.g., representing birds as a product of attributes in the CUB dataset [57]. Indeed, high-dimensional distributed representations have already been proven to be helpful when representing classes as a superposition of attribute vectors in the zero-shot setting [48]. Representing the combination of attributes in a product space may further improve the decoding efficiency.

This work focuses on decoding single vector products; however, efficiently *decoding superpositions* of vector products with our BCF would be highly beneficial. First, it would allow us to decode images with multiple objects (e.g., multiple shapes in an RPM panel on RAVEN). Second, it enables the replacement of arbitrary FCL in neural networks, which usually involve activating multiple neurons. This limitation has been addressed in [17] albeit for dense codes, where a mixed decoding method efficiently extracts a set of vector products from their fixed-width superposition. The mixed decoding combines sequential and parallel decoding methods to mitigate the risk of noise amplification, and increases the number of vector products that can be successfully decoded. However, the number of retrievable vector products in the superposition still needs to be higher to be able to replace arbitrary FCLs in neural networks. Therefore, future work into advanced decoding techniques could improve this aspect of BCF.

Finally, our BCF could enhance Transformer models [56] on different fronts. First, large embedding tables are a bottleneck in Transformer-based recommendation systems, consuming up to 99.9% of the memory [5]. Replacing the embedding tables with our fixed-width product space would reduce the memory footprint as well as the computational complexity in inference when leveraging our BCF. Second, the internal feedforward layers in Transformer models could be replaced by BCF, specifically the first of the two FCLs which can be viewed as key retrieval in a key-value memory [13]. As elaborated in the previous paragraph, the number of decodable vector products in superposition is still limited. Hence, sparsely activated keys would be beneficial. It has been shown that these sparse activations can be found in the middle layers of Transformer models [44].

7. Conclusion

We proposed an iterative factorizer for generalized sparse block codes. Its codebooks are randomly distributed high-dimensional binary sparse block codes, whose number of blocks can be as low as four. The multiplicative binding among the codebooks forms a quasi-orthogonal product space that represents a large number of class categories, or combinations of attributes. As a use-case for our factorizer, we also proposed a novel neural network architecture that replaces trainable parameters in an FCL (aka classifier) with our factorizer, whose reliable operation is verified by accurately classifying/disentangling noisy query vectors generated from various CNN architectures. This quasi-orthogonal product space not only reduces the memory footprint and computational complexity of the networks working with it, but also can reserve a huge representation space to prevent future classes/combinations from coming into conflict with already assigned ones, thereby further promoting interoperability in a continual learning setting.

Acknowledgements

Appendix. Effective replacement of large FCLs

This appendix provides more details on the effective replacement of large FCLs using the proposed BCF.

Table 8

Hyperparamters for CNN training

Dataset/architecture	Ep.	Bs.	Mmt.	Wd.	Learning rate schedule			
					Init. value	Type	Decay rate	Step size
ImageNet-1K								
ShuffleNetV2	400	2048	4e-5	0.9	0.5	lin	–	–
MobileNetV2	150	256	4e-5	0.9	0.04	cos	–	–
ResNet-18	100	256	1e-4	0.9	0.1	exp	0.1	30
ResNet-50	100	256	1e-4	0.9	0.1	exp	0.1	30
CIFAR-100								
ResNet-18	200	128	5e-4	0.9	0.1	exp	0.2	60
RAVEN								
ResNet-18	100	256	1e-4	0.9	0.1	exp	0.1	50

Ep. = Epochs; Bs. = Batch size; Mmt. = Momentum; Wd. = Weight decay.

A.1. Overall CNN training setup

We train all CNN architectures with SGD with architecture-specific hyperparameters, summarized in Table 8. The training setups for the different networks mainly differ in the chosen learning rate schedule. ShuffleNetV2 was trained for 400 epochs using a learning rate initially set to 0.5 and linearly decreased towards 0 at every epoch. MobileNetV2 was trained with a cosine learning rate schedule [34], where the learning rate is decreased based on a cosine function (single cycle) from 0.04 to 0 within 150 epochs. An additional warmup period of 5 epochs was used. ResNet-18 and ResNet-50 were trained with an exponential decaying learning rate schedule. For training the ResNet-18 on the RAVEN dataset, we started with a pre-trained model from the ImageNet-1k dataset [63].

A.2. Loss functions for the bipolar dense resonator network integration

We evaluated different loss functions for replacing FCL with the bipolar dense resonator networks. A standard cross-entropy loss operating on the cosine similarities between the query vector and the fixed bipolar vectors, scaled with a trainable inverse softmax temperature s [20], yielded good brute-force accuracies but notable drops when integrating the resonator networks. For example, when replacing the final FCL of ShuffleNetV2 on ImageNet-1K, we achieved a brute-force accuracy of 66.14% but a much lower resonator network accuracy (44.88%). Alternative adaptive cosine-based loss functions, such as AdaCos [60], were ineffective too.

Instead, we found that the fixed but configurable arcface [4] loss function is the most suitable for the resonator network integration. Arcface computes the angle of the target logit, adds an additive angular margin (m) to the target angle, and gets the target logit back again by the cosine function. Finally, the logits are rescaled by a fixed scaling factor (s) before applying the cross-entropy loss. To find the optimal hyperparameters (s, m), we conducted a grid search across a wide range of configurations ($s \in \{1, 10, 30, 50, 70\}$, $m \in \{0.0, 0.05, 0.1, 0.15\}$). On ImageNet-1K, the search yielded the parameters (70, 0.1) for ShuffleNetV2, (50, 0.1) for MobileNetV2, (70, 0.1) for ResNet-18, and (70, 0.1) for ResNet-50. The optimal parameters for the remaining datasets with ResNet-18 were (30, 0.1) on CIFAR-100 and (10, 0.1) on RAVEN. The large margin separation notably increased the resonator network-based accuracy, e.g., by 9.66% for ShuffleNetV2 on ImageNet-1K.

A.3. Projection dimension D_p

The main results of BCF using the projection layer are reported for GSBC vectors with a dimension of $D_p = 512$. Here, we show that the dimension can be flexibly varied, providing a trade-off between parameter (and therefore computation) saving and accuracy. We varied the dimension D_p from 128 (high reduction) to 1000 (no reduction since $D_p = D_o$) for MobileNetV2 on ImageNet-1K. The hyperparameters of BCF were kept the same for all D_p. Table 9 shows the results. With an extremely low $D_p = 128$, BCF shows 2.86% accuracy drop compared to the

Table 9

Classification accuracy (%) on ImageNet-1K when replacing the FCL in MobileNetV2 with BCF using a projection layer with variable D_p. The remaining configurations ($B = 4$, $F = 2$) are kept constant

	D_p	BF	Fac.	Param. saving↑
Baseline	–	$71.57^{\pm 0.13}$	–	0.0%
GSBCs	128	$70.55^{\pm 0.10}$	$68.71^{\pm 0.12}$	32.8%
	256	$71.15^{\pm 0.07}$	$70.64^{\pm 0.13}$	28.0%
	512	$71.69^{\pm 0.11}$	$71.41^{\pm 0.10}$	18.4%
	768	$71.80^{\pm 0.09}$	$71.64^{\pm 0.11}$	8.7%
	1000	$71.86^{\pm 0.16}$	$71.56^{\pm 0.18}$	0.0%

Table 10

BCF-based replacement approach without and with projection ($D_p = 512$) when allowing BCF a maximum of $N = 15$ (standard) iterations, or $N = 3$

	Baseline acc	BF acc.	GSBCs (B = 4)			
			BCF ($N = 15$)		BCF ($N = 3$)	
			Acc.	Avg. iter.	Acc.	Avg. iter.
No projection						
ShuffleNetV2	$69.22^{\pm 0.20}$	$65.09^{\pm 0.10}$	$64.76^{\pm 0.13}$	7.3	$64.68^{\pm 0.15}$	2.4
MobileNetV2	$71.57^{\pm 0.13}$	$70.00^{\pm 0.07}$	$69.76^{\pm 0.13}$	6.2	$69.72^{\pm 0.13}$	2.3
ResNet-18	$70.39^{\pm 0.11}$	$68.44^{\pm 0.08}$	$68.00^{\pm 0.07}$	6.7	$67.90^{\pm 0.07}$	2.4
ResNet-50	$76.21^{\pm 0.28}$	$76.34^{\pm 0.04}$	$76.25^{\pm 0.07}$	4.8	$76.23^{\pm 0.07}$	2.2
Projection						
ShuffleNetV2	$69.22^{\pm 0.20}$	$68.67^{\pm 0.11}$	$68.41^{\pm 0.14}$	6.0	$68.33^{\pm 0.13}$	2.4
MobileNetV2	$71.57^{\pm 0.13}$	$71.69^{\pm 0.11}$	$71.49^{\pm 0.10}$	5.5	$71.44^{\pm 0.11}$	2.3
ResNet-18	$70.39^{\pm 0.11}$	$69.57^{\pm 0.17}$	$69.19^{\pm 0.13}$	6.2	$69.10^{\pm 0.14}$	2.4
ResNet-50	$76.21^{\pm 0.28}$	$76.72^{\pm 0.06}$	$76.56^{\pm 0.08}$	4.6	$76.54^{\pm 0.06}$	2.2

baseline with trainable FCL while saving 32.8% of the parameters. With a larger $D_p \geqslant 512$, it yields iso-accuracy with the baseline while saving up to 18.4% of the parameters. With $D_p = 768$, BCF eventually surpasses the accuracy of the baseline while saving 8.7% of the parameters.

A.4. Maximum number of iterations N

On ImageNet-1K, the maximum number of iterations of BCF with 2 codebooks is set to $N = \lfloor C/(M_1 + M_2) \rfloor = 15$. However, motivated by the extremely fast convergence on the synthetic product vectors (see Fig. 3 in main text), we show that the maximum number of iterations can be further limited to only 3 with negligible accuracy loss while reducing the computational cost. Table 10 compares the performance on ImageNet-1K between $N = 15$ and $N = 3$. Across all architectures, our BCF matches the brute-force accuracy within 0.44% with $N = 15$ iterations, and within 0.54% with $N = 3$ iterations at maximum. In both cases, the average number of iterations is lower than the maximum N; thus, the factorizer converges on average before the maximum N is reached.

A.5. Number of factors F

So far, we have evaluated BCF with two factors, each having codebooks of size $M_1 = M_2 = 32$ on the ImageNet-1K dataset. We demonstrate its capability with $F = 3$ codebooks of size $M_f = 10$ each. Consequently, the maximum number of iterations becomes $N = \lfloor 1000/30 \rfloor = 33$. Table 11 compares the classification accuracy of the factorizer for $F = 2$ with $F = 3$. The factorizer with the higher number of factors achieves similar accuracy while requiring more iterations (11 vs. 6) compared to $F = 2$. However, since each iteration requires fewer search operations for $F = 3$ (30 vs. 64 searches), the overall compute saving of the FCL remains similar. Our BCF provides

Table 11

Classification accuracy (%) on ImageNet-1K when replacing the FCL in MobileNetV2 with BCF using a projection with $D_p = 512$ and variable F

	F	BF	Fac.	Avg. iter.	FCL comp. saving↑
Baseline	–	$71.57^{\pm 0.13}$	–	–	0.0%
GSBCs	2	$71.69^{\pm 0.12}$	$71.49^{\pm 0.10}$	6	33.4%
	3	$71.61^{\pm 0.07}$	$71.27^{\pm 0.10}$	11	35.6%

Table 12

Comparison of BCF with pruning of final FCL weights of a pretrained MobileNetV2 on ImageNet-1K

Approach	Zero elements (%)	Accuracy (%)
Baseline	–	$71.57^{\pm 0.13}$
BCF	99.98	$69.76^{\pm 0.13}$
FCL pruning	80.00	71.13
	90.00	70.44
	95.00	69.48
	99.00	65.82
	99.98	18.38

a time-space trade-off between the number of factors while keeping the accuracy and the computational cost constant: a low number of factors ($F = 2$) requires more space to store the codebooks ($2 \cdot 32 = 64$ codevectors) but features lower average iterations (6), whereas a higher number of factors ($F = 3$) requires less space ($3 \cdot 10 = 30$ codevectors) but more iterations on average (11).

A.6. Comparison with weight pruning

We compare our approach to weight pruning techniques, which mostly sparsify the weights in all layers, whereas we focus on the final FCL due to its dominance in compact networks. Such pruning can be similarly applied to earlier layers in addition to our method. For a one-to-one comparison, we prune the final FCL weights of a pretrained MobileNetV2 using iterative magnitude-based pruning [61]. As shown in Table 12, the accuracy of the magnitude-based pruning method quickly degrades as soon as more than 95% of the weights are set to zero. In fact, this susceptibility forced related works to prune the final FCL only to 90% [52]. In contrast, our method remains accurate in high sparsity regimes: the codebooks store only $B \cdot F \cdot M_f = 4 \cdot 2 \cdot 32 = 256$ indices instead of $D_p \cdot C$ values (i.e., 99.98% zero elements) with an achieved an accuracy of 69.76%.

References

[1] G. Bent, C. Simpkin, Y. Li and A. Preece, Hyperdimensional computing using time-to-spike neuromorphic circuits, in: *International Joint Conference on Neural Networks (IJCNN)*, 2022.

[2] Y. Cui, S. Ahmad and J. Hawkins, The HTM spatial pooler – a neocortical algorithm for online sparse distributed coding, *Frontiers in Computational Neuroscience* **11** (2017). doi:10.3389/fncom.2017.00111.

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, Imagenet: A large-scale hierarchical image database, in: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 248–255.

[4] J. Deng, J. Guo, N. Xue and S. Zafeiriou, ArcFace: Additive angular margin loss for deep face recognition, in: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 4690–4699.

[5] A. Desai, L. Chou and A. Shrivastava, Random Offset Block Embedding (ROBE) for compressed embedding tables in deep learning recommendation systems, *Proceedings of Machine Learning and Systems* **4** (2022), 762–778.

[6] B. Emruli, R.W. Gayler and F. Sandin, Analogical mapping and inference with binary spatter codes and sparse distributed memory, in: *International Joint Conference on Neural Networks (IJCNN)*, 2013.

[7] E.P. Frady, S.J. Kent, B.A. Olshausen and F.T. Sommer, Resonator networks, 1: An efficient solution for factoring high-dimensional, distributed representations of data structures, *Neural Computation* **32**(12) (2020), 2311–2331. doi:10.1162/neco_a_01331.

[8] E.P. Frady, D. Kleyko and F.T. Sommer, Variable binding for sparse distributed representations: Theory and applications, *IEEE Transactions on Neural Networks and Learning Systems* (2021).

[9] A. Ganesan, H. Gao, S. Gandhi, E. Raff, T. Oates, J. Holt and M. McLean, Learning with holographic reduced representations, in: *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[10] R.W. Gayler, Multiplicative binding, representation operators & analogy, in: *Advances in Analogy Research: Integration of Theory and Data from the Cognitive, Computational, and Neural Sciences*, 1998.

[11] R.W. Gayler, Vector symbolic architectures answer Jackendoff's challenges for cognitive neuroscience, in: *Joint International Conference on Cognitive Science (ICCS/ASCS)*, 2003, pp. 133–138.

[12] R.W. Gayler and S.D. Levy, A distributed basis for analogical mapping, in: *New Frontiers in Analogy Research: Proceedings of the Second International Analogy Conference-Analogy*, 2009.

[13] M. Geva, R. Schuster, J. Berant and O. Levy, Transformer feed-forward layers are key-value memories, in: *Conference on Empirical Methods in Natural Language Processing*, 2021.

[14] V. Gripon and C. Berrou, Sparse neural networks with large learning diversity, *IEEE Transactions on Neural Networks* **22**(7) (2011), 1087–1096. doi:10.1109/TNN.2011.2146789.

[15] K. He, X. Zhang, S. Ren and J. Sun, Deep residual learning for image recognition, in: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[16] M. Hersche, G. Karunaratne, G. Cherubini, L. Benini, A. Sebastian and A. Rahimi, Constrained few-shot class-incremental learning, in: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 9057–9067.

[17] M. Hersche, Z. Opala, G. Karunaratne, A. Sebastian and A. Rahimi, Decoding superpositions of bound symbols represented by distributed representations, in: *Proceedings of the 17th International Workshop on Neural-Symbolic Learning and Reasoning (NeSy)*, 2023.

[18] M. Hersche, M. Zeqiri, L. Benini, A. Sebastian and A. Rahimi, A neuro-vector-symbolic architecture for solving Raven's progressive matrices, *Nature Machine Intelligence* (2023), 1–13.

[19] M. Hersche, M. Zeqiri, L. Benini, A. Sebastian and A. Rahimi, Solving Raven's progressive matrices via a neuro-vector-symbolic architecture, in: *Proceedings of the 17th International Workshop on Neural-Symbolic Learning and Reasoning (NeSy)*, 2023.

[20] E. Hoffer, I. Hubara and D. Soudry, Fix your classifier: The marginal value of training the last weight layer, in: *International Conference on Learning Representations (ICLR)*, 2018.

[21] P. Kanerva, Large patterns make great symbols: An example of learning from example, in: *Hybrid Neural Systems*, S. Wermter and R. Sun, eds, Springer, Berlin Heidelberg, 2000, pp. 194–203. ISBN 978-3-540-46417-4. doi:10.1007/10719871_13.

[22] P. Kanerva, Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors, *Cognitive Computation* **1**(2) (2009), 139–159. doi:10.1007/s12559-009-9009-8.

[23] S.J. Kent, E.P. Frady, F.T. Sommer and B.A. Olshausen, Resonator networks, 2: Factorization performance and capacity compared to optimization-based methods, *Neural Computation* **32**(12) (2020), 2332–2388. doi:10.1162/neco_a_01329.

[24] D. Kleyko, D.A. Rachkovskij, E. Osipov and A. Rahimi, A survey on hyperdimensional computing aka vector symbolic architectures, part I: Models and data transformations, *ACM Comput. Surv.* (2022).

[25] D. Kleyko, A. Rahimi, D.A. Rachkovskij, E. Osipov and J.M. Rabaey, Classification and recall with binary hyperdimensional computing: Tradeoffs in choice of density and mapping characteristics, *IEEE Transactions on Neural Networks and Learning Systems* **29**(12) (2018).

[26] A. Knoblauch and G. Palm, Iterative retrieval and block coding in autoassociative and heteroassociative memory, *Neural Computation* **32**(1) (2020), 205–260. doi:10.1162/neco_a_01247.

[27] A. Krizhevsky, Learning multiple layers of features from tiny images, 2009.

[28] C.J. Kymn, S. Mazelet, A. Ng, D. Kleyko and B.A. Olshausen, Compositional factorization of visual scenes with convolutional sparse coding and resonator networks, 2024, arXiv preprint arXiv:2404.19126.

[29] M. Laiho, J.H. Poikonen, P. Kanerva and E. Lehtonen, High-dimensional computing with sparse vectors, in: *2015 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, 2015.

[30] G. Lample, A. Sablayrolles, M.A. Ranzato, L. Denoyer and H. Jegou, Large memory layers with product keys, in: *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[31] J. Langenegger, G. Karunaratne, M. Hersche, L. Benini, A. Sebastian and A. Rahimi, In-memory factorization of holographic perceptual representations, *Nature Nanotechnology* (2023).

[32] M. Ledoux, *The Concentration of Measure Phenomenon*, American Mathematical Society, 2001.

[33] J. Liu, W.-C. Chang, Y. Wu and Y. Yang, Deep learning for extreme multi-label text classification, in: *International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2017.

[34] I. Loshchilov and F. Hutter, SGDR: Stochastic gradient descent with warm restarts, 2016, arXiv preprint arXiv:1608.03983.

[35] N. Ma, X. Zhang, H.-T. Zheng and J. Sun, ShuffleNet V2: Practical guidelines for efficient CNN architecture design, in: *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[36] N.Y. Masse, G.C. Turner and G.S.X.E. Jefferis, Olfactory information processing in drosophila, *Current Biology* **19**(16) (2009).

[37] P. Mettes, E. van der Pol and C. Snoek, Hyperspherical prototype networks, *Advances in Neural Information Processing Systems (NeurIPS)* **32** (2019).

[38] B.A. Olshausen and D.J. Field, Natural image statistics and efficient coding, *Network: Computation in Neural Systems* **7**(2) (1996), 333–339. doi:10.1088/0954-898X_7_2_014.

[39] T.A. Plate, Holographic reduced representations, *IEEE Transactions on Neural Networks* **6**(3) (1995), 623–641. doi:10.1109/72.377968.

[40] T.A. Plate, *Analogy Retrieval and Processing with Distributed Vector Representations, Expert Systems*, 2000.

[41] T.A. Plate, *Holographic Reduced Representations: Distributed Representation for Cognitive Structures*, Center for the Study of Language and Information, Stanford, 2003.

[42] Z. Qian, T.L. Hayes, K. Kafle and C. Kanan, Do we need fully connected output layers in convolutional networks? 2020, arXiv preprint arXiv:2004.13587.

[43] D.A. Rachkovskij and E.M. Kussul, Binding and normalization of binary sparse distributed representations by context-dependent thinning, *Neural Computation* **13**(2) (2001), 411–452. doi:10.1162/089976601300014592.

[44] H. Ramsauer, B. Schäfl, J. Lehner, P. Seidl, M. Widrich, L. Gruber, M. Holzleitner, T. Adler, D. Kreil, M.K. Kopp, G. Klambauer, J. Brandstetter and S. Hochreiter, Hopfield networks is all you need, in: *International Conference on Learning Representations (ICLR)*, 2021.

[45] D. Rasmussen and C. Eliasmith, A neural model of rule generation in inductive reasoning, *Topics in Cognitive Science* **3**(1) (2011), 140–153. doi:10.1111/j.1756-8765.2010.01127.x.

[46] A. Renner, Y. Sandamirskaya, F.T. Sommer and E.P. Frady, Sparse vector binding on spiking neuromorphic hardware using synaptic delays, in: *International Conference on Neuromorphic Systems (ICONS)*, 2022.

[47] A. Renner, L. Supic, A. Danielescu, G. Indiveri, B.A. Olshausen, Y. Sandamirskaya, F.T. Sommer and E.P. Frady, Neuromorphic visual scene understanding with resonator networks, 2024, *Nature Machine Intelligence* 641–652.

[48] S. Ruffino, G. Karunaratne, M. Hersche, L. Benini, A. Sebastian and A. Rahimi, Zero-shot classification using hyperdimensional computing, in: *2024 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, IEEE, 2024.

[49] M.S.E. Saadabadi, A. Dabouei, S.R. Malakshan and N.M. Nasrabad, Hyperspherical Classification with Dynamic Label-to-Prototype Assignment, 2024, arXiv preprint arXiv:2403.16937.

[50] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L.-C. Chen, MobileNetV2: Inverted residuals and linear bottlenecks, in: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[51] K. Schlegel, P. Neubert and P. Protzel, A comparison of vector symbolic architectures, *Artificial Intelligence Review* **55**(6) (2022), 4523–4555. doi:10.1007/s10462-021-10110-3.

[52] J. Schwarz, S. Jayakumar, R. Pascanu, P.E. Latham and Y. Teh, Powerpropagation: A sparsity inducing weight reparameterisation, *Advances in Neural Information Processing Systems (NeurIPS)* **34** (2021), 28889–28903.

[53] T.R. Scott, A.C. Gallagher and M.C. Mozer, von Mises-Fisher loss: An exploration of embedding geometries for supervised learning, in: *Proceedings of the IEEE/CVF International Conference on Computer Vision (CVPR)*, 2021.

[54] G. Shalev, G.L. Shalev and Y. Keshet, Redesigning the Classification Layer by Randomizing the Class Representation Vectors, 2021, arXiv preprint arXiv:2011.08704.

[55] J. Snoek, H. Larochelle and R.P. Adams, Practical Bayesian optimization of machine learning algorithms, *Advances in Neural Information Processing systems (NeurIPS)* **25** (2012).

[56] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser and I. Polosukhin, Attention is all you need, *Advances in Neural Information Processing Systems (NeurIPS)* **30** (2017).

[57] C. Wah, S. Branson, P. Welinder, P. Perona and S. Belongie, The Caltech-UCSD Birds-200-2011 Dataset, 2011.

[58] D.J. Willshaw, O.P. Buneman and H.C. Longuet-Higgins, Non-holographic associative memory, *Nature* **222**(5197) (1969), 960–962. doi:10.1038/222960a0.

[59] C. Zhang, F. Gao, B. Jia, Y. Zhu and S.-C. Zhu, RAVEN: A dataset for relational and analogical visual REasoNing, in: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[60] X. Zhang, R. Zhao, Y. Qiao, X. Wang and H. Li, AdaCos: Adaptively scaling cosine logits for effectively learning deep face representations, in: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[61] M. Zhu and S. Gupta, To prune, or not to prune: exploring the efficacy of pruning for model compression, 2017, arXiv preprint arXiv:1710.01878.

[62] Z. Zhu, T. Ding, J. Zhou, X. Li, C. You, J. Sulam and Q. Qu, A geometric analysis of neural collapse with unconstrained features, in: *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[63] T. Zhuo and M. Kankanhalli, Solving Raven's Progressive Matrices with Neural Networks, 2020, arXiv preprint arXiv:2002.01646.