Deep Convolutional Tables: Deep Learning Without Convolutions

Shay Dekel, Yosi Keller, and Aharon Bar-Hillel

***Abstract*— We propose a novel formulation of deep networks that do not use dot-product neurons and rely on a hierarchy of voting tables instead, denoted as convolutional tables (CTs), to enable accelerated CPU-based inference. Convolutional layers are the most time-consuming bottleneck in contemporary deep learning techniques, severely limiting their use in the Internet of Things and CPU-based devices. The proposed CT performs a fern operation at each image location: it encodes the location environment into a binary index and uses the index to retrieve the desired local output from a table. The results of multiple tables are combined to derive the final output. The computational complexity of a CT transformation is independent of the patch (filter) size and grows gracefully with the number of channels, outperforming comparable convolutional layers. It is shown to have a better capacity:compute ratio than dot-product neurons, and that deep CT networks exhibit a universal approximation property similar to neural networks. As the transformation involves computing discrete indices, we derive a soft relaxation and gradient-based approach for training the CT hierarchy. Deep CT networks have been experimentally shown to have accuracy comparable to that of CNNs of similar architectures. In the low-compute regime, they enable an error:speed tradeoff superior to alternative efficient CNN architectures.**

***Index Terms*— Convolutional tables (CTs), deep learning, efficient computation.**

I. Introduction

DEEP learning techniques in general, and convolutional neural networks (CNNs) in particular, have become the core computational approach in computer vision. Currently, "deep learning techniques" and "deep neural networks" are synonyms, despite relating to different concepts. By definition [9], deep learning is the "use of multiple layers to progressively extract higher level features from raw input," i.e., learning a useful feature hierarchy. Artificial neural networks, on the other hand, are graphs of dot-products/convolutions computing elements. In this article, we argue that it is possible to unravel this dualism and that this is a worthwhile endeavor.

It is the need for accelerated and more efficient CPU-based inference that has motivated us to depart from the dot-product/convolution-based paradigm. CNNs consist of millions of "neurons": computing elements applying dot products to their inputs. The resulting computational load is significant and is usually handled by GPUs and other specialized hardware. Due to these computational demands, deep learning applications are often restricted to the server-side, where powerful dedicated hardware carries the computational load. Alternatively, applications running on standard CPUs or low-computing platforms such as cellphones are limited to small, low-accuracy networks, or avoid using networks altogether. Such applications are natural user interfaces (NUIs), face recognition, mobile robotics, and the Internet of Things (IoT) devices, to name a few. Low-compute platforms are not equipped with a GPU and are expected to handle the inference of simple deep learning tasks. Such devices are unable to train CNNs or run ImageNet-scale CNNs. Other devices of interest, such as most laptops and desktop computers, are unequipped with GPUs but have gigabytes of memory.

In this article, we propose a novel deep learning paradigm based on a hierarchy of deep voting tables. This allows the embedding of input signals such as images, similar to backbone CNNs, which is shown to be preferable in low-compute and GPU-less domains. Our approach uses CPUs that are capable of efficiently accessing large memory domains (random access). For instance, for a CPU with ideally unbounded memory access, the fastest classifier is a single huge table: a series of simple binary queries is applied to the input, an index is built from the resulting bits, and an answer is retrieved from the table entry. Such an approach is impractical due to the huge size of the resulting table. A practical alternative is to replace the single table with an ensemble of smaller hierarchical tables. Rather than relying on dot products, we propose a two-step, computationally efficient alternative. First, a set of K simple binary queries is applied to the input patch, thus encoding it with a binary codeword of K-bit. Second, the computed codeword is used as an index into a table to retrieve the corresponding output representation. The proposed transformation is called a convolutional table (CT). A CT layer consists of multiple CTs whose outputs are summed. Similar to convolutional layers, CT layers are stacked to form a CT network and derive a hierarchy of features. We detail this construction in Section III.

A fern is a tree in which all split criteria at a fixed tree depth are identical, such that the leaf identity is determined by using fixed K split criteria termed "bit functions." The CT operation is the application of a single fern to a single input location, and the operation of a CT layer (sum of CTs) corresponds to applying a fern ensemble. In Section IV, we compare the computational complexity of the CNN and CT operations. By design, the complexity of a single CT layer is independent of the size of the patch used ("filter size" in a convolutional layer), and its dependence on quadratic depth

AQ:1 Manuscript received 1 April 2021; revised 17 October 2021, 15 July 2022, and 12 January 2023; accepted 22 April 2023. *(All authors contributed equally to this work.) (Corresponding author: Yosi Keller.)*

AQ:2 Shay Dekel and Yosi Keller are with Bar Ilan University, Ramat-Gan, Israel (e-mail: yosi.keller@gmail.com).

AQ:3 Aharon Bar-Hillel is with Ben-Gurion University, Negev, Israel.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2023.3270402.

Digital Object Identifier 10.1109/TNNLS.2023.3270402

terms is with significantly lower constants. This allows us to derive conditions for which a CT network can be considerably faster than a corresponding CNN.

In Section V, we derive rigorous results regarding the capacity of a single fern and the expressiveness of a two-layer CT network. Specifically, the VC dimension of a single fern with K bits is shown to be $\Theta(2^K)$. This implies that a fern has an advantage over a dot- product operation: it has a significantly higher capacity-to-computing-effort ratio. Our results show that a two-layer CT network has the same universal approximation capabilities as a two-layer neural network and that this also holds for ferns with a single bit function per fern.

A core challenge of the proposed deep-table approach is the need for a proper training process. The challenge arises because inference requires computing discrete indices that are unnameable for gradient-based learning. Thus, we introduce in Section VI a "soft formulation of the calculation of the K bit codeword," as a Cartesian product of the K soft bit functions. Following [23], [37], bit functions are computed by comparing only two pixels in the region of interest. However, in their soft version, the pixel indices parameters are noninteger, and their gradient-based optimization relies on the horizontal and vertical spatial gradients of the input maps. During training, soft indices are used for table voting using a weighted combination of possible votes. We use an annealing mechanism and an incremental learning schedule to gradually turn trainable but inefficient soft voting into efficient, hard single-word voting at the inference time.

We developed a CPU-based implementation of the proposed scheme and applied it to multiple standard datasets, consisting of up to 250-K images. The results, presented in Section VII, show that CT-networks achieve an accuracy comparable to CNNs with similar structural parameters while outperforming the corresponding CNNs trained using weights and activations binarization. More importantly, we consider the speed:error and speed:error:memory tradeoffs, where speed is estimated via operation count. CT networks have been shown to provide a better speed:error tradeoff than efficient CNN architectures such as MobileNet [35] and ShuffleNet [29] in the tested domain. The improved speed:error tradeoff requires a higher memory consumption, as CT networks essentially trade speed for memory. Overall, CT networks can provide 3.1–12× acceleration over CNNs with memory expansions of 2.3×–10.6×, which are applicable to laptops and IoT applications.

In summary, our contribution in this article is twofold: first, we present an alternative to convolution-based networks and show for the first time that useful deep learning (i.e., learning of a feature hierarchy) can be achieved by other means. Second, from an applicable viewpoint, we suggest a framework enabling accelerated CPU inference for low-compute domains, with moderate costs of accuracy degradation and memory consumption.

II. Related Work

Computationally efficient CNN schemes were extensively studied using a plethora of approaches, such as low-rank or tensor decomposition [18], weight quantization [2], [20], conditional computation [1], [3], [8], using FFT for efficient convolution computations [32], or pruning of networks' weights and filters [6], [13], [26], [40]. In the following, we focus on the methods most related to our work.

Binarization schemes aim to optimize CNNs by binarizing network weights [7], [28], [31] and internal network activations [27], [34]. Recent approaches apply weights–activations-based binarization [4], to reduce memory footprint and computational complexity by utilizing only binary operations. Methods that binarize the activations are related to the proposed scheme as they encode each local environment into a binary vector. However, their encoding is dense, and the output is computed using a (binary) dot product. The proposed scheme also utilizes binary activations, but differs significantly, as convolution/dot-product operations are not applied. In contrast, we apply binary comparisons between random samples in the activation maps.

Knowledge distillation approaches [15], [44] use a large "Teacher" CNN to train a smaller computationally efficient "Student" CNN. We show in our experiments that this technique can be combined with our approach to improve the accuracy. Regarding network design, several architectures were suggested for low-compute platforms [11], [29], [35], [41], utilizing group convolutions [35], efficient depthwise convolutions [29], [35], map shifts [41], dubbed Bi-Real net [28], and sparse convolutions [11]. Differentiable soft quantization (DSQ) was proposed by Gong et al. [12] to quantize the weights of a CNN to a given number of bits. An article by Huang et al. [16] proposed a method for efficient quantization using a multilevel binarization technique and a linear or logarithmic quantizer to simultaneously binarize weights and quantize activations to a low bit width. The authors of this article then compare the tradeoffs of this method to low-compute versions of MobileNetV2 [35], ShuffleNet [29], and Bi-Real Net [28]. The use of trees or ferns applied using convolutions for fast predictors was thoroughly studied in computer vision [23], [24], [36], [37]. A convolutional random forest enabled the estimation of human pose in real time in the Kinect console [37]. A flat CT ensemble classifier for hand poses, that was proposed in [23], enabled inference in less than 1 ms on a CPU. It was extended [24] to a full-hand pose estimation system. However, in all these works, flat predictors were used, whereas we extend the notion to a deep network with gradient-based training. Other works merged tree ensembles with MLPs or CNNs for classification [17], [19], or semantic segmentation [5]. Specifically, "conditional networks" were presented in [17] as CNNs with a tree structure, enabling conditional computation to improve the speed–accuracy ratio. Unlike our approach, where ferns replace the convolutions, these networks use standard convolutions and layers, and the tree/forest is a high-level routing mechanism. Yang et al. [42] proposed a probability distribution approach for quantization by treating the discrete weights in a quantized neural network as searchable variables and using a differential method to search for them accurately. They represent each weight as a probability distribution over the set of discrete values. Although this method was able to produce quantized neural networks with higher performance than other state-of-

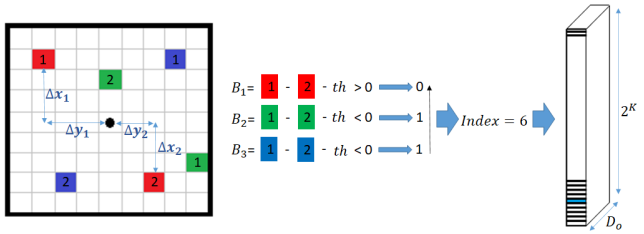


Fig. 1. CT with a word calculator of three bits operating on a 2-D input tensor. Each bit function is given by two pixels whose difference is compared to a threshold (colored in red, green, and blue). The word computed by the three bits, "110" = 6, is used to access the 6th entree in the voting table, thus computing an output vector $\in \mathbb{R}^{D_o}$.

the-art quantized methods in image classification, it is still reliant on standard convolution layers with dot products, which could be a bottleneck in low-power systems. In Section VII, we compare our method with that of the authors and report superior results in terms of accuracy. An effective channel-pruning method was suggested by Su et al. [38], using a bilaterally coupled network to determine the optimal width of a neural network, resulting in a more compact network. A bilaterally coupled network is a network that has been trained to approximate the optimal width of another network, to achieve a more compact network while maintaining or improving its performance. A new approach for dynamically removing redundant filters from a neural network was proposed by Tang et al. [39]. The channels are pruned by embedding the manifold information of all instances into the space of pruned networks. The performance of deep convolutional neural networks is improved by removing redundant filters while preserving the important features of the input data. The authors show that their method is an alternative to traditional channel-pruning methods, which may not be able to achieve optimal results in terms of both accuracy and efficiency. A scheme related to our approach was proposed by Zhou et al. [47], wherein the authors suggest a cascade of forests applied using convolutions. Each forest is trained to solve the classification task, and the class scores of a lower layer forest are used as features for the next cascade level. While the classifiers' structure is deep, they are not trained end to end, but the trees are trained independently using conventional random forest optimization. Hence, the classifier uses thousands of trees and is only able to reach the accuracy of a two-layer CNN.

III. CT Transform

A CT is a transformation accepting a representation tensor $T^{\text{in}} \in \mathbb{R}^{H_i \times W_i \times D_i}$ and outputting a representation tensor $T^{\text{out}} \in \mathbb{R}^{H_o \times W_o \times D_o}$, where H_i, H_o and W_i, W_o are the spatial dimensions, whereas D_i, D_o are the representation's dimensionality for the input and output tensors, respectively. Formally, it is the tuple (B, W), where B is a word calculator, and W is a voting table. A word calculator B is a feature extractor applied to a patch, returning a K-bit index, such that $B : \mathbb{R}^{l \times l \times D_i} \to \{0,1\}^K$, where l is the patch size. The voting table $W \in M^{2^K \times D_o}$ contains all possible 2^K outputs as rows, each being a vector $\in \mathbb{R}^{D_o}$.

The index is computed by sequentially applying K bit functions $\{B^k\}_{k=1}^K$, each computing a single bit of the index. Let $P = (p_x, p_y) \in \mathbb{R}^{H_i \times W_i}$ be a location in T^{in}. We denote by $B(P;\Theta) = (B^1(P;\Theta^1), \ldots, B^K(P,\Theta^K))$ the index computed for the patch centered at location P. Here, $\Theta = (\Theta^1, \ldots, \Theta^K)$ are the word calculator parameters, with Θ^k explicitly characterized below. The functions $B^k(P;\Theta^k) \in \{0,1\}$ (dependence on T^{in} is omitted for notation brevity) are computed by thresholding at 0 the simple smooth functions $b^k(P;\Theta^k)$ of a few input pixels (i.e., $B^k(P;\Theta) = Q(b^k(P;\Theta^k))$ with $Q(\cdot)$ being the Heaviside function). Bit functions b^k are the thresholded differences between two locations

$$b^k(P;\Theta_k) = T^{\text{in}}\big(p_x + \Delta x_1^k,\, p_y + \Delta y_1^k,\, c^k\big) - T^{\text{in}}\big(p_x + \Delta x_2^k,\, p_y + \Delta y_2^k,\, c^k\big) - \text{th}^k \quad (1)$$

where the learned parameters Θ^k define the compared locations $(\Delta x_1^k, \Delta y_1^k, c^k)$ and $(\Delta x_2^k, \Delta y_2^k, c^k)$, and the margin threshold th^k. th^k is the learnable scalar margin threshold for comparing two locations within the patch corresponding to the kth bit. The codeword computed by the word calculator is used as an index to the voting table to retrieve $W(B(P;\Theta),:) \in \mathbb{R}^{D_o}$ for location P. Fig. 1 depicts the CT operation at a single location. A CT (using padding = "same" and stride = 1) applies the operation defined above at each position $P \in \mathbb{R}^{H_i \times W_i}$ of the input tensor to obtain the corresponding output tensor column. The same as in convolutional layers, padding can be applied to the input for computations at border pixels, and strides may be used to reduce the spatial dimensions of the output tensor. A CT layer includes M CTs $\{(B_m, W_m)\}_{m=1}^M$ whose outputs are summed

$$T^{\text{out}}[P, \ : \ ] = \sum_{m=1}^{M} W_m\big[B_m(T^{\text{in}}_{N(P)}), \ : \ \big]$$
$$B_m = \{B^k\}_{k=1}^K. \quad (2)$$

The operation of a CT layer is summarized in Algorithm 1. Essentially, it applies a fern ensemble at each location, summing the output vectors from the ensemble's ferns. A CT network stacks multiple CT layers in a cascade or graph structure, similar to stacking convolutional layers in standard CNNs.

IV. Computational Complexity

Let the input and output tensors be $T^{\text{in}} \in \mathbb{R}^{H_i \times W_i \times D_i}$ and $T^{\text{out}} \in \mathbb{R}^{H_o \times W_o \times D_o}$, respectively, where the dimensions are the same as in Section III, and let $l \times l$ be the filter's support. For a convolution layer, the number of operations per location is $C_{\text{CNN}} = l^2 D_i D_o$, since it computes D_o inner products at the cost of $l^2 D_i$ each. For a CT layer with M CTs and K bits in each CT, the cost of bits computation is MKC_b, with C_b being the cost of computing a single bit function. The voting cost is MD_o, since we add the vectors $\in \mathbb{R}^{\text{MD}_\text{o}}$ to get the result. Thus, the complexity of a CT layer is $C_{\text{CT}} = M(C_b \cdot K + D_o)$.

It follows that the computational complexity of a CT is independent of the filter size l. Hence, it allows agglomerating evidence from large spatial regions with no additional

Algorithm 1 CT Layer

INPUT:
A tensor $T^{\text{in}} \in \mathbb{R}^{H_i \times W_i \times D_i}$,
transformation parameters $\{(\Theta^m, W^m)\}_{m=1}^{M}$
OUTPUT:
An Output tensor $T^{out} \in \mathbb{R}^{H_o \times W_o \times D_o}$
Initialization:
$T^{out} = 0$,
$\{(\Theta^m, W^m)\}_{m=1}^{M} \sim \mathcal{N}(0, \sigma^2 = 1)$
for $P \in \{1, .., H_i\} \times \{1, .., W_i\}$ **do** ▷ All pixel locations
  for $m = 1, .., M$ **do** ▷ All CT tables
    Compute $B_m(P; \Theta^{k,m}) \in \{0, 1\}^K$ ▷ Eq. 1
    $T^{out}[P, :] = T^{out}[P, :] + W^m[B_m(P; \Theta^{k,m}), :]$ ▷ Eq. 2
  end for
end for
Return T^{out}

computational cost. To compare CNN and CT complexities, assume that both use the same representation dimension $D_i = D_o = D$. The relationship between the total number of bit functions MK and D depends on whether each bit function uses a separate dimension, or the dimensions are reused by multiple bit functions. For a general analysis, assume that each dimension is reused by R bit functions, such that $D = MK/R$ (in our experiments $R \in [1, 3]$). Thus, the ratio of complexities between CNNs and CT layers with the same representation dimension D is

$$\frac{C_{\text{CNN}}}{C_{\text{CT}}} = \frac{l^2 D^2}{M(C_b \cdot K + D)} = \frac{l^2 D^2}{C_b DR + \frac{D^2 R}{K}} \approx \frac{Kl^2}{R} \quad (3)$$

where the last approximation is true for large D and the quadratic terms in D are dominant. The computational cost of computing a single bit function, C_b, is defined as the number of operations required to perform a single comparison operation between two locations within the patch size. This computation is described in (1) and involves four additions of the offsets of Δ_x, Δy, and a subtraction of the threshold, th^k. This results in a total of five operations. In addition, load operations are required for the two relevant memory locations and their two base addresses, bringing the total number of operations to nine. Although the most stringent count does not exceed ten operations, for simplicity, we set Cb to 10. The ratio of complexities between CNNs and CT layers in (3) suggests that there is the potential for acceleration by more than an order of magnitude for reasonable choices of parameter values. For example, using $l = 3$, $K = 8$, and $R = 2$ results in a 36× acceleration. Although the result of this analysis is promising, several lower level considerations are essential to achieve actual acceleration. First, operation counts translate into actual improved speed only if the operations can be efficiently parallelized and vectorized. In particular, vectorization requires contiguous memory access in most computations. The CT transformation adheres to these constraints. Contrary to trees, bit computations in ferns can be vectorized over adjacent input locations as all locations use the same bit functions, and the memory access is contiguous. The voting operation can be vectorized over the output dimensions. This implementation was already built and has been shown to be highly efficient for flat CT classifiers [24]. Although a CT transformation uses a "random memory access" operation when the table is accessed, this random access is rare: there is a single random access operation included in the $(C_b \cdot K + D_o)$ operations required for a single-fern computation. Another important issue is the need to keep the model's total storage size within reasonable bounds to enable its memory to be efficiently accessed in a typical L_2 cache. For example, a model consisting of 50 layers, wherein $M = 10$ ferns, $K = 6$, and $D = 60$, has a total size of 1.92 MB when the parameters are stored in 8-bit precision.

V. CT Capacity and Expressiveness

In this section, we present two analytic insights that indicate the feasibility of deep CT networks and their potential advantages. In Section V-A, the capacity of a fern is shown to be $\Omega(2^K)$ with $O(K)$ computational effort, providing a much better capacity:compute ratio than a linear convolution. In Section V-B, a network of two fern layers is shown to be a universal approximation, similar to known results on neural networks. In the analysis, we consider a simplified nonconvolutional setting, with an input vector $X \in [0, 1]^d$, and a nonconvolutional fern ensemble classifier. Furthermore, simple bit functions of the form $B^k(X) = Q((-1)^{s^k}(X[I^k] - t^k))$ are considered, where $s^k \in \{0, 1\}$, $I^k \in 1, \ldots, d$, and $t^k \in [0, 1]$. These simple bit functions just compare a single entry to a threshold.

A. Capacity:Compute Ratio

The following lemma characterizes the capacity of a single fern in VC-dimension terms.

Lemma 1: Define the hypothesis family of binary classifiers based on a single K-bit fern $H = \{f(X) = \text{sign}(W(B(X))) : B = \{(s^k, I^k, t^k)\}_{k=1}^{K}, W \in R^{2^K}\}$. For $K \geq 4$ and $\log_2(d) < K \leq d$, its VC dimension is

$$2^K \leq \text{VC-dim}(H) \leq K2^K. \quad (4)$$

Proof: Since we only consider the sign of the chosen table entry $W(B(X))$, we may equivalently consider the family of classifiers $f(X) = W(B(X))$ for $W \in \{-1, 1\}^{2^K}$.

1) Lower Bound: Consider a sample containing points on the d-dimensional cube

$$S = \{P_c = (-1^{c_1}, \ldots, -1^{c_K}) : \ c = (c_1, \ldots, c_K) \in \{0, 1\}^K \ \}. \quad (5)$$

It follows that this sample can be shattered by the binary fern family. We may choose the bit functions $b^k(X) = Q(X[k])$. Assume an arbitrary label assignment l, i.e., for each example P_c, we have $l(P_c) = y_c$ with y_c arbitrarily chosen in $\{-1, 1\}$. By definition, for each c and k, we have $b^k(P_c) = \{-1\}^{c_k} = P_c[k]$. Since any two points P_{c_1}, P_{c_2} differ by at least a single dimension, they will have at least different bits in a single bit function. Hence, the 2^K points are mapped into the 2^K different cells of W. By choosing $W[c] = y_c$, we may get the desired labeling l. Therefore, S is shattered, showing $\text{VCdim}(H) \geq 2^K$.

2) Upper Bound: Assume that $K \geq 4$ and $\log_2(d) < K \leq d$. We compute an upper bound on the number of possible labels (label vectors) enabled by the binary fern family on a sample of size n and show that it is smaller than 2^n for $n = K2^K$.

For a K-bit fern, there are d^K ways to choose the input dimension of the K bit functions (one dimension per function). For each bit function, once the input dimension I is chosen, we may reorder the examples according to their value in dimension I, i.e., $X_1[I] \leq X_2[I], \ldots, \leq X_n[I]$. Nontrivial partitions of the sample, i.e., partitions into two nonempty sets, are introduced by choosing the threshold t in $(X_1[I], X_n[I])$, and there are at most $n-1$ different options. Including the trivial option of partitioning S into S and Φ, there are n options.

Consider the number of partitions of the examples into 2^K cells, which are made possible using K bit functions. Two different partitions must differ w.r.t. the induced partition by a single bit function choice, at least. Hence, their number is bounded by the number of ways to choose the bit function partitions, which is $d^K n^K$ at most, according to the above considerations. For each such partition into 2^K cells, 2^K different classifiers can be defined by choosing the table entries $\{W[c]\}_{c=0}^{2^K-1}$. Hence, the total number of possible different fern classifiers for n points is bounded by $(dn)^K 2^{2^K}$. When this bound is lower than 2^n, the sample cannot be shattered. Thus, we solve the problem

$$(dn)^K 2^{2^K} < 2^n \tag{6}$$

and

$$2^K + K\log_2 d + K\log_2 n < n. \tag{7}$$

By choosing $n = K2^K$, it follows that (7) is satisfied since

$$\begin{aligned} &2^K + K\log_2 d + K\log_2 K + K^2 \\ &\qquad < 2^K + 3K^2 \leq 2^K + (K-1)2^K \leq K2^K. \end{aligned} \tag{8}$$

For the second inequality, we used $\log_2 d < K$ (assumed), and for the third, we used the inequality $3K^2 \leq (K-1)2^K$, which holds for $K \geq 4$. ∎

Results similar to Lemma 1 were derived for trees [30], [45], and the lemma implies that the capacity of ferns is not significantly lower than that of trees. To understand its implications, compare such a single-fern classifier to a classifier based on a single dot-product neuron, i.e., a linear classifier. A binary dot-product neuron with $d = K$ input performs $O(K)$ operations to obtain a VC dimension of $O(K)$. The fern also performs $O(K)$ computations, but the response is chosen from a table of 2^K leaves, and the VC dimension is $\Omega(2^K)$. The higher (capacity):(computational-complexity) ratio of ferns indicates that they can compute significantly more complex functions than linear neurons using the same computational budget.

B. CT Networks Are Universal Approximators

The following theorem states the universal approximation capabilities of a two-layer CT network. It resembles the known result for two-stage neural networks [14].

Theorem 1: Any continuous function $f : [0,1]^d \to [0,1]$ can be approximated in L_∞ using a two-layer fern network with $K = 1$ in all ferns.

Proof: The core of the proof is to show that by using $2K$ 1-bit ferns at the first layer, a second layer fern can create a function with an arbitrary value inside a hyperrectangle of choice, and zero outside the rectangle. Since sums of such "step functions" are dense in $C[0,1]$, so are two-layer fern networks.

Let $R = [a_1, b_1] \times [a_2, b_2], \ldots, \times [a_d, b_d]$ be a hyperrectangle in R^d, and let $v \in R$ be a scalar value. A rectangle function $s(x; v, R) : [0,1]^d \to R$ is defined as $v \cdot 1_{x \in R}$, i.e., a function whose value is $f(x) = v$ for $x \in R$ and 0 otherwise. Define the family of step functions

$$G = \{g : [0,1]^d \to [0,1] : g(x) = \sum_{p=1}^{P} s(x; v_p, R_p)\}. \tag{9}$$

It is known that G is dense in $(C[0,1]^d)$ (from the Stone–Weierstrass theorem; see, e.g., [10]). We will show that the family of two-layer CT networks includes this set.

Let $s(v, R)$ be an arbitrary rectangle function. For each dimension $i = 1, \ldots, d$, define the following two bit functions: $L_i(x) = Q(x - a_k)$ and $R_i(x) = Q(-(x - b_k))$. Denote these bit functions by $B^j_{(R)}$ for $j = 1, \ldots, 2d$. By construction, $\{B^j_R\}_{j=1}^{2d}$ characterize the rectangle

$$x \in R \iff \forall j = 1, \ldots, 2d, \quad B^j_R(x) = 1. \tag{10}$$

Equivalently, we have $x \in R$ iff $\sum_{j=1}^{2d} B^j_R(x) > 2d - (1/2)$. Given a function $g(x) = \sum_{p=1}^{P} s(x; v_p, R_p)$ to implement, we define at the first layer P sets of ferns. Fern set p contains $2d$ ferns, denoted by F^j_p, with a single bit function each, and the bit function of F^j_p is $B^j_{R_p}$. The output dimension of the first layer is P. The weight table $W^j_p \in M_{2\times P}$ of F^j_p is defined by

$$W^j_p[i, l] = \begin{cases} 1, & i = 1, l = p, \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

The output vector of a layer is the sum of all ferns $Y = \sum_{p'=1}^{P} \sum_{j=1}^{2d} W^j_{p'}[B^j_{R_{p'}}(x), :]$. By construction, we have

$$\begin{aligned} Y[p] &= \sum_{p'=1}^{P} \sum_{j=1}^{2d} W^j_{p'}\left[B^j_{R'_p}(x), p\right] \\ &= \sum_{j=1}^{2d} W^j_p\left[B^j_{R_p}(x), p\right] = \sum_{j=1}^{2d} B^j_{R_p}(x). \end{aligned} \tag{12}$$

The second equality holds since $W^j_{p'}[i, p]$ only differs from zero for $p = p'$. The last equality is valid since $W^j_p[i, p] = i$ for $i = 0, 1$. We, hence, have $Y[p] = 2d > 2d - (1/2)$ for $x \in R_p$, which has lower values anywhere else.

The second layer of ferns contains P ferns, each with a single bit and a single output dimension. For fern p, we define the bit function $B^{(2)}_p(Y) = Q(Y[p] - (2d - (1/2)))$ which fires only for $x \in R_p$ and the table $W^{(2)}_p = [0, v(p)]$. The output

unit U computes

$$U = \sum_{p=1}^{P} W_p^{(2)} \left[B_p^{(2)}(x) \right] = \sum_{p=1}^{P} v(p) \cdot B_p^{(2)}(x) = \sum_{p=1}^{P} v(p) \cdot 1_{x \in R_p}. \quad (13)$$

Therefore, U implements the function $g(x)$, which completes the proof. ■

VI. Training With Soft CTs

Following (1), $B^k(P; \Theta_k) = Q(b^k(P; \Theta_k))$ does not enable gradient-based optimization, as the derivative of the Heaviside function $Q(\cdot)$ is zero almost anywhere. We suggest a soft version of the CT in Section VI-A that enables gradient-based optimization and discusses its gradient in Section VI-B.

A. Soft CT Version

To enable gradient-based learning, we suggest replacing the Heaviside function during optimization with a linear sigmoid $q(x; t)$ for $t > 0$

$$q(x; t) = \min(\max((t + x)/2t\ , 0), 1) \quad (14)$$

where $q(x; t)$ is identical to the Heaviside function for $x \gg 0$ and is a linear function in the vicinity of $x = 0$. It has a nonzero gradient in $[-t, t]$, where t is a hyperparameter controlling its smoothness. For low t values, we have $q(x; t) \underset{t \to 0}{\to} Q(x)$. Moreover, we have $q(x; t) + q(-x; t) = 1$. Thus, it can be interpreted as a pseudoprobability, with $q(x)$ estimating the probability of a bit being 1 and $q(-x)$ the probability of being 0.

Following Section III, a word calculator B maps a patch P to a single index or equivalently to a one-hot vector (row) in R^{2^K}. Denote the word calculator in the latter view (mapping into R^{2^K}) by $\vec{B}(P; \Theta)$. Given this notation, the CT output is given by the product $\vec{B} \cdot W$. Hence, extending the word calculator to a soft function in R^{2^K}, with dimension $b \in \{0, \ldots, 2^K - 1\}$ measuring the activity of the word b, provides a natural differentiable extension of the CT formulation. For an index b, define the sign of its kth bit by $s(b, k) \triangleq (-1)^{(1+u(b,k))}$, where $u(b, k)$ denotes the kth bit of the index b in its standard binary expansion, such that

$$s(b, k) = \begin{cases} 1, & \text{bit } k = 1 \\ -1, & \text{bit } k = 0. \end{cases} \quad (15)$$

The activity level of the word b in a soft-word calculator is defined by

$$\vec{B}^s(P; \Theta)[b] = \prod_{k=1}^{K} q(s(b, k) \cdot b_k(P; \Theta_k); t). \quad (16)$$

Intuitively, the activity level of each possible codeword index b is a product of the activity levels of the individual bit functions in the directions implied by the codeword bits. Marginalization implies that $\sum_{b=0}^{2^K-1} \vec{B}^s(P)[b] = 1$, and hence the soft-word calculator defines a probability distribution over the possible K-bit words. Soft CT is defined as a natural extension $\vec{B}^s W$. Note that as $t \to 0$, the sigmoid $q(x; t)$ becomes sharp, $\vec{B}^s$ becomes a one-hot vector, and the soft CT becomes a standard CT as defined in Section III, without any inner products.

Soft CT can be considered as a consecutive application of two layers: a soft indexing layer $\vec{B}^s$ followed by a plain linear layer (though sparse if most of the entries in $\vec{B}^s$ are zero). Since CT is applied to all spatial locations on the activation map, the linear layer W corresponds to a 1×1 convolution. We implemented a sparse 1×1 convolutional layer operating on a sparse tensor $\mathbf{x} \in R^{H \times W \times 2^K}$ and outputting a dense tensor $\mathbf{y} \in R^{H \times W \times D_o}$.

B. Gradient Computation and Training

Since applying the soft CT in a single position is $W \cdot B^s(P; \Theta)$, with W being a linear layer, it suffices to consider the gradient of $\vec{B}^s(P; \Theta)$. Denote the neighborhood $l \times l$ of the location P in T^{in} by $T^{\text{in}}_{N(P)}$. Considering a single output variable b at a time, the gradient with respect to the input patch $T^{\text{in}}_{N(P)}$ is given by

$$\frac{\partial \vec{B}^s(P)[b]}{\partial T^{\text{in}}_{N(P)}} = \sum_k \frac{\partial \vec{B}^s(P)[b]}{\partial b_k(P; \Theta_k)} \cdot \frac{\partial b_k(P; \Theta_k)}{\partial T^{\text{in}}_{N(P)}} \quad (17)$$

with a similar expression for the gradient with respect to Θ. For a fixed index k, the derivative $(\partial \vec{B}^s(P)[b])/(\partial b_k(P; \Theta_k))$ is given by

$$\frac{\partial \vec{B}^s(P)[b]}{\partial b_k(P; \Theta_k)} = \prod_{j \neq k} q(s(b, j) b_j(P; \Theta_j); t) \frac{\partial q(s(b, j) b_j(P; \Theta_j); t)}{\partial b_k(P; \Theta_k)} = \frac{\vec{B}^s(P)[b] \cdot t^{-1} \cdot 1_{|b_k| < t} \cdot s(b, k)}{2q(s(b, k) \cdot b_k(P; \Theta_k))}. \quad (18)$$

The derivative is nonzero when the word b is active (i.e., $\vec{B}^s(P)[b] > 0$), and the bit function $b_k(P)$ is in the dynamic range $-t < b_k(P) < t$.

To further compute the derivatives of $b_k(P)$ with respect to the input and parameters $[((\partial b_k(P; \Theta_k))/dT^{\text{in}}_{N(P)})$ and $((\partial b_k(P; \Theta_k))/d\Theta]$, respectively), we first note that the forward computation of $b_k(P)$ estimates the input tensor at fractional spatial coordinates. This is because the offset parameters $\Delta x_1, \Delta y_1, \Delta x_2, \Delta y_2$ are trained with gradient descent, which requires them to be continuous. We use bilinear interpolation to compute image values at fractional coordinates in the forward inference. Hence, such bilinear interpolation (of spatial gradient maps) is required for gradient computation. For instance, $((\partial b_k(P)/\partial \Delta x_1^k)$ is given by

$$\frac{\partial b_k(P)}{\partial \Delta x_1^k} = \frac{\partial T^{\text{in}}\left(p_x + \Delta x_1^k, p_y + \Delta y_1^k, c\right)}{\partial \Delta x_1^k} = \frac{\partial T^{\text{in}}(:, :, c)}{\partial x}\left[\left(p_x + \Delta x_1^k, p_y + \Delta y_1^k\right)\right] \quad (19)$$

where $((\partial T^{\text{in}}(:, :, c))/\partial x)$ is the partial x-derivative of the image channel $T^{\text{in}}(:, :, c)$, which is estimated numerically and sampled at $(x, y) = (p_x + \Delta x_1^k, p_y + \Delta y_1^k)$ to compute (19).

The derivatives with respect to other parameters are computed in a similar way. Note that the channel index parameters c^k are not learned—these are fixed during network construction for each bit function. As for $((\partial b_k(P;\Theta_k))/dT^{\text{in}}_{N(P)})$, by considering (1), it follows that $b_k(P;\Theta_k)$ only relates to two fractional pixel locations: $(p_x+\Delta x_1^k, p_y+\Delta y_1^k)$ and $(p_x+\Delta x_2^k, p_y+\Delta y_2^k)$. This implies that a derivative with respect to eight pixels is required, as each value of a fractional coordinate's pixel is bilinearly interpolated using its four neighboring pixels in integer coordinates.

The sparsity of the soft-word calculator (both forward and backward) is governed by the parameter t of the sigmoid in (14), which acts as a threshold. When t is large, most of the bit functions are ambiguous and soft, i.e., not strictly 0 or 1, and the output will be dense. We can control the output's sparsity level by adjusting t. In particular, as $t \to 0$, the bit functions become hard, and $B^s(P)$ converges toward a hard fern with a single active output word. While a dense flow of information and gradients is required during training, fast inference in test time requires a sparse output. Therefore, we introduce an annealing scheduling scheme, such that t is initiated as $t \gg 0$, set to allow a fraction f of the values of the bit functions to be in the "soft zone" $[-t, t]$. The value of t is then gradually lowered to achieve a sharp and sparse classifier toward the end of the training phase.

VII. Experimental Results

We experimentally examined and verified the accuracy and computational complexity of the proposed CT scheme, applying it to multiple small and medium image datasets, consisting of up to 250-K images. Such datasets are applicable to IoT and non-GPU low-compute devices. We explored the sensitivity of CT networks to their main hyperparameters in Section VII-A. To this end, we studied the number of ferns M and the number of bit functions per fern K, and the benefits of distillation as a CT training technique. We then compared our CT-based architectures to similar CNN classes. Such a comparison is nontrivial due to the inherent architectural differences between the network classes. Thus, in Section VII-B, we compared Deep CT to CNNs with similar depth and width, that is, the number of layers and maps per layer. The comparison focuses on efficient methods suggested for network quantization and binarization, since the CT framework uses similar notions. In Section VII-C, the CT is compared with CNN formulations with comparable computational complexity, i.e., similar number of multiply–accumulate (MAC) operations. We compared CT with CNN architectures specifically designed for inference efficiency, such as MobileNet [35] and ShuffleNet [29]. The comparison was applied by studying the speed:error and speed:error:memory tradeoffs of the different frameworks. Finally, to exemplify the applicability of Deep CT to low-power IoT applications and larger datasets (250-K images), it was applied to IoT-based face recognition in Section VII-D.

The proposed novel CT layer does not use standard CNN components such as activations, convolutions, or FCs. Thus, the Deep CT word calculator layer and the sparse voting table layer were implemented from scratch, in unoptimized C, compiled as MEX files, and applied within a MATLAB implementation. Our experiments were carried out on the same computer using CPU-only implementations for all schemes.

Similar to other inherently novel layers, such as Spatial Transformer Networks, it is challenging to implement a converging end-to-end training process. To facilitate the convergence of the proposed scheme, a three-phase training scheme was applied. First, the lower half of the CNN (layers #1–#6 in Table VIII) was trained, by adding the average pooling (AP), Softmax (SM), and cross-entropy (CE) layers on top of layer #6. Then, the full architecture was trained with layers #1–#6 frozen. Finally, following Hinton et al. [15], we unfroze all layers and refined the entire network using a distillation-based approach. The CT was trained to optimize a convex combination of CE loss and Kullback–Leibler divergence with respect to the probabilities induced by a teacher CNN. The temperature parameter was initially set at $t = 4$ and gradually annealed to $t = 1$ toward the end of the training. All Deep CT networks were trained from scratch using random Gaussian initialization of the parameters.

A. Hyperparameters Sensitivity

The expressive power of the Deep CT transformation is related to the number of CTs per layer M, and the number of bit functions K used in each CT. We tested the sensitivity of the Deep CT network to these parameters and the suggested annealing schedule. These experiments applied CT networks with four layers to the SVHN, CIFAR-10, and CIFAR-100 datasets, and two layers to the MNIST dataset. The networks are similar to those shown in the Appendix, in Tables VI and VII. We used a fixed patch size $l = 5$ and input and output dimensions of $D_i = D_o = 32$, respectively. It is best to apply CT networks with a patch size larger than the kernel size of CNN-based networks, since CT patches are sparsely represented. Comparatively, CNN complexity is quadratic with respect to patch size, whereas CT networks' computational complexity is independent of patch size.

The classification errors with respect to the bit functions K (for a fixed M), and the number of CTs M (for a fixed K), are shown in Fig. 2(a) and (b), respectively. For most datasets, there is a range in which the error exponentially decreases as the number of network parameters increases, whereas the accuracy improvement saturates for $M > 8$ and $K > 8$ for the deeper networks (datasets other than MNIST).

Fig. 2(c) shows the average number of active words in a CT as a function of the training time for CIFAR-10. The fraction f of ambiguous bits (bits whose value is not set to either 0 or 1) was started at 0.2 and was gradually reduced exponentially, whereas the sigmoid threshold t was adjusted periodically according to the required f. In our experiments, the sigmoid threshold was set to $t = 0.5$. The number of active words per CT is thus gradually reduced and converged to a single active word in the final classifier. The CNN with four layers obtained a test error of 12.55%, compared with 12.49% with the same architecture without annealing. Therefore, annealing-based training led to a sparser classifier without significantly degrading the classification accuracy.

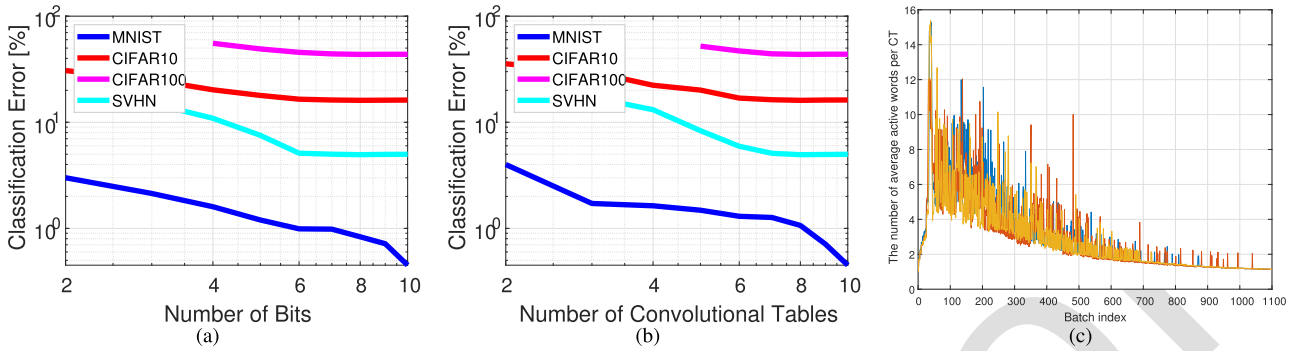


Fig. 2. Hyperparameters study on MNIST, CIFAR10, CIFAR100, and SVHN datasets. (a) Deep CT network error versus the number of bits K in a log–log plot. In all layers, the number of CTs M was fixed to 10. (b) Deep CT network error versus the number of CTs M in a log–log plot, for a fixed number of bits $K = 10$. (c) Average number of active words per CT was evaluated using the CIFAR-10 dataset and plotted against the training batch index. Each convolution table is represented by a different color: yellow, blue, and red. As expected, the number of active words decreases during the training process due to the annealing mechanism and eventually converges to a single active word.

TABLE I
JOINT HYPERPARAMETERS TUNING FOR THE FOUR- AND SIX-LAYER CT NETWORKS DESCRIBED IN TABLES VII AND II. M AND K DENOTE THE CONVOLUTION TABLES' SIZE AND THE NUMBER OF BITS, RESPECTIVELY

K	M	Layers#	Distil	Error [%]
9	8	6	No	11.31
8	9	6	No	12.31
8	**8**	6	No	**11.05**
7	8	6	No	13.93
8	7	6	No	13.34
8	**8**	6	Yes	**10.25**
9	8	4	No	12.88
8	9	4	No	13.05
8	**8**	4	No	**12.49**
7	8	4	No	13.25
8	7	4	No	12.90
8	**8**	4	Yes	**10.91**

To establish the best hyperparameters, additional experiments were carried out for multiple M and K combinations around $M = 8, K = 8$. The experiments were conducted using the CIFAR-10 dataset, and the CT networks having four and six layers, detailed in Tables II and VII, respectively. In addition, we examined training with and without distillation from a teacher. The $M = 8, K = 8$ configuration was found to be the most accurate. Distillation was found to be beneficial, providing accuracy improvements of $1\% - 2\%$.

B. Accuracy Comparison of Low-Complexity Architectures

Although deep CT and CNN networks use very different computational elements, both use a series of intermediate representations to perform classification tasks. Hence, we consider CT and CNN backbone networks similar if they have the same number of intermediate representations (depth) and the same number of maps in each representation (width). In this section, we report experiments comparing CNN and CT networks that have identical configurations in these respects. Although the intermediate spatial tensors were chosen to be similar, we allowed for different top inference layers. Those were chosen to optimize the classification accuracy for each network class. Thus, the CNN-based networks used max-pooling and an FC layer, whereas the CT only used an average pooling layer.

We compared the classification accuracy of CT networks with standard CNNs and contemporary low-complexity CNN architectures based on CNN binarization schemes. The methods we compared against include the BinaryConnect (BC) [7],[1] XNOR-Net (XN) [34],[2] XNOR-Net++ (XN++) [4], Bi-Real Net (Bi-Real) [28], SLB [42][3], and TABCNN [27],[4] using their publicly available implementations. These methods, [34] and [27] in particular, also encode the input activity using a set of binary variables. In contrast to our CT table-based approach, they use dense encodings and apply a binary dot product to their encoded input. Finally, we applied the state-of-the-art DSQ scheme [12][5] to the conventional CNN to quantize it to 2, 4, and 6 bits.

All networks were applied to the MNIST [25], CIFAR-10 [21], CIFAR-100 [22], and SVHN [33] datasets. For the MNIST dataset, we applied CT networks and CNNs consisting of two layers, as this relatively small dataset did not require additional depth. For the other datasets, networks consisting of four and six layers were applied.

The architectures of the six-layer networks are detailed in Table II, and the architectures of the two and four layers used are detailed in the Appendix in Tables VI and VII, respectively. The classification accuracy results are reported in Table III. On the left-hand side of the table, all methods are compared using the cross-entropy loss. The proposed CT networks outperformed the binarization and discretization schemes in all cases and were outperformed by conventional CNNs. In the right-hand column of the table, we show the results of a CT network using a distillation loss and a VGG16 [46] teacher network. These CT networks improve the accuracy significantly further, and the margin of CNNs over them is also significantly smaller. Several key findings can be drawn from these results: First, the good accuracy of CT networks indicates that they can be trained using standard SGD

[1]https://github.com/MatthieuCourbariaux/BinaryConnect
[2]https://github.com/rarilurelo/XNOR-Net
[3]https://github.com/zhaohui-yang/Binary-Neural-Networks/tree/main/SLB
[4]https://github.com/layog/Accurate-Binary-Convolution-Network
[5]https://github.com/ricky40403/DSQ

TABLE II

SIX-LAYER DEEP CT AND CNN NETWORKS APPLIED TO THE CIFAR10, CIFAR100, AND SVHN DATASETS. THE CNN BINARIZATION SCHEMES WERE APPLIED TO THE CNN ARCHITECTURE, WHERE l AND K DENOTE THE PATCH SIZE AND THE NUMBER OF BITS, M IS THE NUMBER OF CTS AND D_o IS THE NUMBER OF OUTPUT MAPS

	Deep CT Layer	**Parameters** l, K, M, D_o	**Output**	**Convolution Layers**	**Parameters**	**Output**
1	CT Layer 1	$7, 8, 8, 32$	$26 \times 26 \times 32$	Conv,ReLU	$l = 3, D_o = 32$	$30 \times 30 \times 32$
2	Average Pooling	$l = 3$	$24 \times 24 \times 32$	Max Pooling	$l = 3, Stride = 2$	$14 \times 14 \times 32$
3	CT Layer 2	$5, 8, 8, 32$	$20 \times 20 \times 32$	Conv,ReLU	$l = 3, D_o = 32$	$12 \times 12 \times 32$
4	Average Pooling	$l = 3$	$18 \times 18 \times 32$	Max Pooling	$l = 2$	$11 \times 11 \times 32$
5	CT Layer 3	$5, 8, 8, 64$	$14 \times 14 \times 64$	Conv,ReLU	$l = 3, D_o = 64$	$9 \times 9 \times 64$
6	Average Pooling	$l = 3$	$12 \times 12 \times 64$	Max Pooling	$l = 2$	$8 \times 8 \times 64$
7	CT Layer 4	$5, 8, 8, 128$	$8 \times 8 \times 128$	Conv,ReLU	$l = 3, D_o = 128$	$6 \times 6 \times 128$
8	Average Pooling	$l = 2$	$7 \times 7 \times 128$	Max Pooling	$l = 2$	$5 \times 5 \times 128$
7	CT Layer 5	$3, 8, 8, 64$	$5 \times 5 \times 64$	Conv,ReLU	$l = 3, D_o = 64$	$4 \times 4 \times 64$
8	Average Pooling	$l = 2$	$4 \times 4 \times 64$	Max Pooling	$l = 2$	$3 \times 3 \times 64$
9	CT Layer 6	$3, 8, 8, 10$	$2 \times 2 \times 10$	Conv,ReLU	$l = 3, D_o = 10$	$1 \times 1 \times 10$
10	Average Pooling	$l = 2$	$1 \times 1 \times 10$	FC	$D_o = 10$	$1 \times 1 \times 10$
11	SoftMax			SoftMax		1

TABLE III

COMPARISON OF THE CLASSIFICATION ERROR PERCENTAGE. WE COMPARE THE RESULTS FOR CNNS CONSISTING OF TWO LAYERS (MNIST) OR FOUR AND SIX LAYERS (OTHER DATASETS). ALL QUANTIZED/BINARIZED CNNS WERE DERIVED BY APPLYING THE CORRESPONDING QUANTIZATION/BINARIZATION SCHEMES TO THE BASELINE CNN HAVING THE SAME NUMBER OF LAYERS (TABLES VI, VII AND II). THE MOST ACCURATE RESULTS ARE MARKED IN **BOLD**

Dataset	**DeepCT**	**BC [7]**	**XN [34]**	**XN++ [4]**	**TABCNN [27]**	**DSQ [12] 2b**	**DSQ [12] 4b**	**DSQ [12] 6b**	**Bi-Real [28]**	**SLB [42]**	**DeepCT + distil.**	**CNN**
Two layers [%]												
MNIST	**0.35**	0.61	0.51	0.39	0.49	0.41	0.38	0.36	0.36	0.37	0.39	0.334
Four layers [%]												
CIFAR10	**12.49**	14.47	14.77	12.97	13.97	12.87	12.71	12.68	12.72	12.66	10.91	10.86
CIFAR100	**39.17**	50.27	47.32	41.03	45.32	40.14	40.05	39.96	39.97	39.55	37.12	36.81
SVHN	**4.75**	6.21	5.88	4.92	5.11	4.85	4.79	4.77	4.79	4.80	4.46	4.27
Six layers [%]												
CIFAR10	**11.05**	12.56	12.91	11.88	12.13	11.35	11.27	11.14	11.19	11.16	10.25	9.88
CIFAR100	**37.95**	42.43	40.14	39	39.37	38.31	38.25	38.12	38.15	38.05	36.05	35.58
SVHN	**4.12**	4.95	4.54	4.25	4.28	4.21	4.18	4.15	4.17	4.25	3.96	3.77

optimization, such as CNNs, without significant overfitting. Second, the similar precision of the CT and CNN networks with a similar intermediate representation structure indicates that the accuracy is more dependent on the representation hierarchy structure than the particular computing elements used.

C. Tradeoff Comparison: Accuracy, Speed, and Memory

We studied the Error:Speed and Error:Speed:Memory tradeoffs enabled by the CT networks and compared them with those of contemporary state-of-the-art schemes. The experiments were conducted using the CIFAR10 and CIFAR100 datasets. Seven models of the CT network were trained, with operation count budgets $3.1\times$–$12\times$ lower than those of the baseline CNN in Table VII ($7.54 \cdot 10^6$). Similar to the baseline CNN, the CT models had six layers. The variations were derived by changing the hyperparameters M and K and the number of channels at intermediate layers.

We compare the tradeoffs of the CT networks with those of the MobileNet V2 [35] and ShuffleNet V2 [29] efficient CNN architectures. The first is based on inverted bottleneck residual blocks, where depthwise convolutions are performed on high-depth representations, and the second used blocks in which half of the channels are processed, followed by a channel

TABLE IV

SPEED AND MEMORY TRADEOFFS COMPARING THE PROPOSED CT, MOBILENET2 [35], AND SHUFFLENET2 [29], USING THE CIFAR-10 AND CIFAR-100 DATASETS. EACH ROW RELATES TO A PARTICULAR CLASSIFICATION PERCENTAGE ERROR AND SHOWS THE CORRESPONDING SPEED AND MEMORY TRADEOFFS. SPEEDUP AND MEMORY RATIOS ARE COMPUTED BY $(R_{\text{CNN}}/R_{\text{Net}})$, WHERE R_{CNN} IS THE RESOURCES REQUIRED BY THE BASELINE CNN, AS IN TABLE III, AND R_{Net} IS THE NETWORK'S REQUIREMENTS

Error [%]	**Speedup** CT	[35]	[29]	**Memory ratio** CT	[35]	[29]
CIFAR-10						
10.6	4.91	0.72	1.47	0.09	2.67	2.86
11.6	5.69	1.59	3.51	0.14	5.53	8.12
12.6	14.29	3.42	6.74	0.45	6.76	26.46
CIFAR-100						
36.6	4.91	0.72	2.80	0.09	2.67	6.77
38.1	5.69	1.59	3.51	0.14	5.53	8.12
39.6	13.74	3.42	6.74	0.28	6.76	26.46

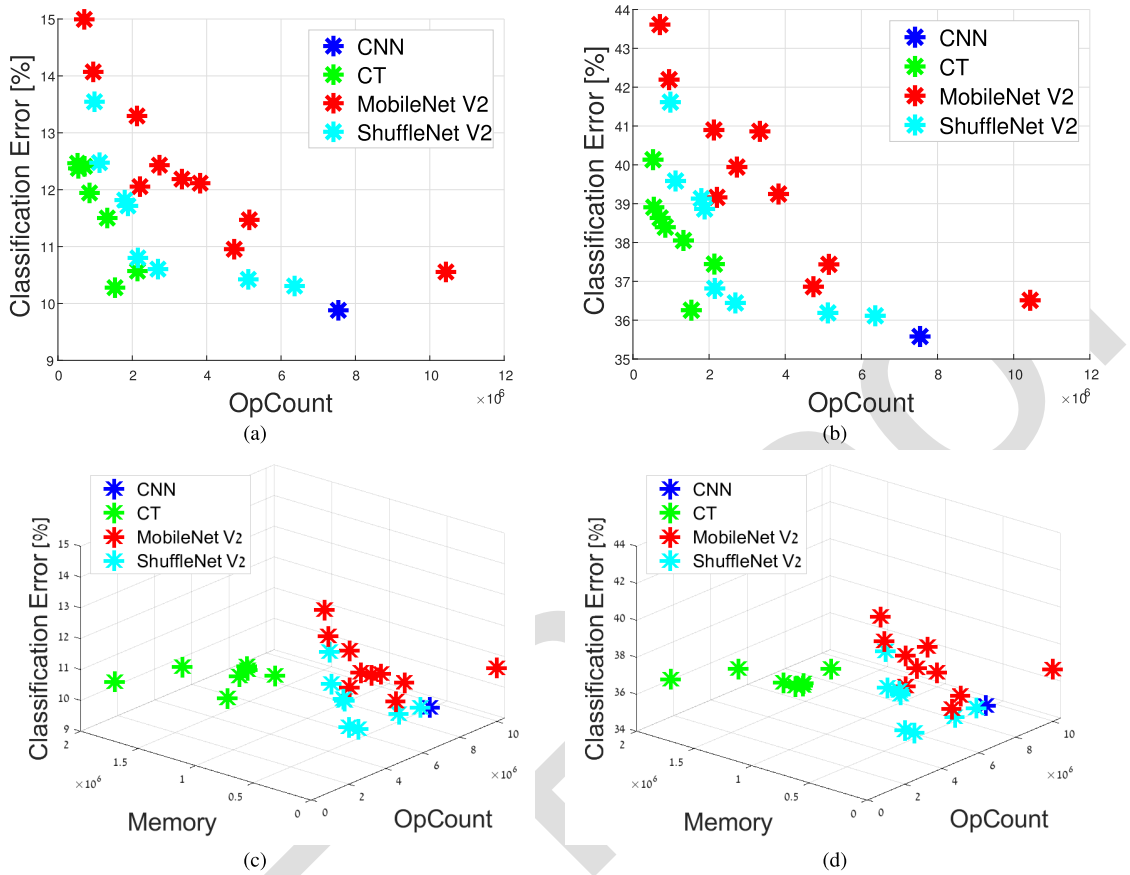


Fig. 3. Performance tradeoffs of the deep CT using an architecture of six layers. (a) Speed:Error tradeoff for several families of models applied to the CIFAR-10 dataset. The models are based on the CNN in Table VII, CT networks, MobileNet V2 [35], and ShuffleNet V2 [29]. We considered architectures using up to 10^7 operations. (b) Speed:Error tradeoff on the CIFAR-100 dataset. (c) Speed:Error:Memory tradeoff on CIFAR-10. (d) Speed:Error:Memory tradeoff on CIFAR-100. CT networks provide better Speed:Error tradeoffs in this domain while requiring additional memory. The speed is measured by the MACC operation count. The memory is given by the number of parameters, whereas the error is the classification percentage error.

TABLE V
ACCURACY PERCENTAGE OF DEEP CT, CNN, MOBILENET V2 [35], AND SHUFFLENET V2 [29] WHEN APPLIED TO SUBSETS OF THE CASIA-WEBFACE DATASET

Dataset	Deep CT	BC	XN	TABCNN	DSQ 2 bits	DSQ 4 bits	DSQ 6 bits	MobileNet	ShuffleNet	CNN
CASIA 25K	**87.21**	85.36	86.286	86.95	87.05	87.15	87.19	87.20	87.186	89.96
CASIA 250K	**90.66**	87.95	88.35	88.89	90.06	90.27	90.35	90.63	90.54	93.25

shuffle operation. We used the code provided by the corresponding authors[6,7] and adapted it to the relevant low-compute regime by adjusting their hyperparameters. For MobileNet, the networks were generated by adjusting the expansion parameter t (Table I in [35]), the number of channels in intermediate representations, and the number of bottleneck modules in a block (between 1 and 2). For ShuffleNet, the networks were generated by changing the number of channels in the intermediate layers and the number of bottleneck modules in a block (between 1 and 4).

Error:Speed results are shown in Fig. 3(a) and (b) for CT networks and competing lean network approaches. These plots show that the proposed CT networks significantly outperform the baseline CNN and competing architectures. To enable a more explicit assessment, the left-hand side of Table IV shows the speedup achieved by the most efficient networks in each framework, for several error values on the CIFAR-10 and CIFAR-100 datasets. In the high-accuracy end, while losing at most 1% error w.r.t. to the baseline CNN, the CT network enables 4.91× acceleration over the CNN and an acceleration of 1.75–6.8× over competing methods. When more significant errors are allowed (2.7% for CIFAR-10 and 4% for CIFAR-100), the CT networks provide the accelerations of 13×–14× over the baseline CNN, significantly outperforming the 6.7× acceleration obtained by the best competitor [29]. The baseline CNN uses $7.54 \cdot 10^7$ MACC operations, $181 \cdot 10^6$ parameters, and obtains the errors of 9.88 and 35.58 on CIFAR-10 and CIFAR-100, respectively. The tables for CIFAR-10 and CIFAR-100 are similar, as the same architectures are preferable for both datasets in each category of error and network type.

[6]https://github.com/xiaochus/MobileNetV2
[7]https://github.com/TropComplique/shufflenet-v2-tensorflow

TABLE VI

TWO-LAYER DEEP CT AND CNN NETWORKS APPLIED TO THE MNIST DATASET. l AND K DENOTE THE SIZE OF THE PATCH AND THE NUMBER OF BITS, M IS THE NUMBER OF CTS AND D_o IS THE NUMBER OF OUTPUT MAPS

	Deep CT Layer	Parameters l, K, M, D_o	Output	Convolution Layer	Parameters l, D_o	Output
1	CT Layer 1	$9, 10, 10, 100$	$20 \times 20 \times 100$	Conv,ReLU	$l = 5, D_o = 100$	$24 \times 24 \times 100$
2	Average Pooling	$l = 7$	$14 \times 14 \times 100$	Max Pooling	$l = 5, Stride = 2$	$10 \times 10 \times 100$
3	CT Layer 2	$9, 10, 10, 10$	$6 \times 6 \times 10$	Conv,ReLU	$l = 5, D_o = 10$	$6 \times 6 \times 10$
4	Average Pooling	$l = 6$	$1 \times 1 \times 10$	Max Pooling+FC	$l = 6$	$1 \times 1 \times 10$
5	SoftMax		1	SoftMax		1

TABLE VII

FOUR-LAYER DEEP CT AND CNN NETWORKS APPLIED TO THE CIFAR10, CIFAR100, AND SVHN DATASETS. THE CNN BINARIZATION SCHEMES WERE APPLIED TO THE CNN ARCHITECTURE, WHERE l AND K DENOTE THE PATCH SIZE AND THE NUMBER OF BITS, M IS THE NUMBER OF CTS, AND D_o IS THE NUMBER OF OUTPUT MAPS

	Deep CT Layer	Parameters l, K, M, D_o	Output	Convolution Layers	Parameters	Output
1	CT Layer 1	$7, 8, 8, 32$	$26 \times 26 \times 32$	Conv,ReLU	$l = 5, D_o = 32$	$28 \times 28 \times 32$
2	Average Pooling	$l = 3$	$24 \times 24 \times 32$	Max Pooling	$l = 3, Stride = 2$	$13 \times 13 \times 32$
3	CT Layer 2	$7, 8, 8, 32$	$18 \times 18 \times 32$	Conv,ReLU	$l = 3, D_o = 32$	$11 \times 11 \times 32$
4	Average Pooling	$l = 3$	$16 \times 16 \times 32$	Max Pooling	$l = 3$	$9 \times 9 \times 32$
5	CT Layer 3	$7, 8, 8, 64$	$10 \times 10 \times 64$	Conv,ReLU	$l = 3, D_o = 64$	$7 \times 7 \times 64$
6	Average Pooling	$l = 3$	$8 \times 8 \times 64$	Max Pooling	$l = 3$	$5 \times 5 \times 64$
7	CT Layer 4	$7, 8, 8, 10$	$2 \times 2 \times 10$	Conv,ReLU	$l = 3, D_o = 10$	$3 \times 3 \times 10$
8	Average Pooling	$l = 2$	$1 \times 1 \times 10$	Max Pooling + FC	$l = 3$	$1 \times 1 \times 10$
9	SoftMax			SoftMax		1

TABLE VIII

DEEP CT AND CNN ARCHITECTURES APPLIED TO THE SUBSETS OF THE CASIA-WEBFACE DATASET, WHERE l AND K DENOTE THE PATCH SIZE AND THE NUMBER OF BITS, M IS THE NUMBER OF CTS, AND D_o IS THE NUMBER OF OUTPUT MAPS

	Deep CT Layer	Parameters l, K, M, D_o	Output	Convolution Layers	Parameters	Output
1	CT Layer 1	$9, 8, 8, 32$	$56 \times 56 \times 32$	Conv,ReLU	$l = 3, D_o = 32$	$62 \times 62 \times 32$
2	Average Pooling	$l = 5$	$52 \times 52 \times 32$	Max Pooling	$l = 3, Stride = 2$	$30 \times 30 \times 32$
3	CT Layer 2	$9, 8, 8, 32$	$44 \times 44 \times 32$	Conv,ReLU	$l = 3, D_o = 32$	$28 \times 28 \times 32$
4	Average Pooling	$l = 5$	$40 \times 40 \times 32$	Max Pooling	$l = 2, Stride = 2$	$14 \times 14 \times 32$
5	CT Layer 3	$9, 8, 8, 64$	$32 \times 32 \times 64$	Conv,ReLU	$l = 3, D_o = 64$	$12 \times 12 \times 64$
6	Average Pooling	$l = 3$	$30 \times 30 \times 64$	Max Pooling	$l = 3$	$10 \times 10 \times 64$
7	CT Layer 4	$9, 8, 8, 128$	$22 \times 22 \times 128$	Conv,ReLU	$l = 3, D_o = 128$	$8 \times 8 \times 128$
8	Average Pooling	$l = 3$	$20 \times 20 \times 128$	Max Pooling	$l = 3$	$6 \times 6 \times 128$
9	CT Layer 5	$9, 8, 8, 256$	$12 \times 12 \times 256$	Conv,ReLU	$l = 3, D_o = 256$	$5 \times 5 \times 256$
10	Average Pooling	$l = 3$	$10 \times 10 \times 256$	Max Pooling	$l = 3$	$3 \times 3 \times 256$
11	CT Layer 6	$9, 8, 8, 50$	$2 \times 2 \times 50$	Conv,ReLU	$l = 3, D_o = 50$	$1 \times 1 \times 50$
12	Average Pooling	$l = 2$	$1 \times 1 \times 50$	Fully Connected		$1 \times 1 \times 50$
13	SoftMax			SoftMax		1

Fig. 3(c) and (d) depicts the Error:Speed:Memory tradeoff for CT networks and other networks, using the CIFAR-10 and CIFAR-100 datasets, respectively. The memory requirements of the most efficient networks for each accuracy level are listed on the right-hand side of Table IV, as compression ratios with respect to the baseline CNN. Clearly, the better Error:Speed tradeoff of the CT networks comes at the expense of a larger memory footprint (additional parameters). This is expected, as in CT transformations with memory-based retrieval replacing dot-product operations, the compute:memory ratio is much lower. For instance, in the high-accuracy regime, using the CIFAR-10 dataset allows a 4.9× speedup over the baseline CNN while requiring 10.7× more parameters. However, the size of excess memory can be controlled by limiting K (for which the memory demand is exponential) to moderate values. Table IV shows that the fastest CT network enabling the CIFAR-10 error of 12.6% provides a 14.3× acceleration while requiring only 2.2× additional parameters over the baseline CNN.

D. Face Recognition Test Case

We experimented with CT networks for face recognition using the CASIA-WebFace dataset [43]. Two datasets of 64 × 64 extracted face images were formed. First, we randomly drew a subset of 50 classes (each with 5000 images), 250-K images overall, to evaluate the proposed CT on a *large-scale* dataset. We also randomly drew 50 identities and a class of negatives (not belonging to the 50 identities), each consisting of 500 images. This exemplifies a typical IoT application, where face recognition is an enabler of personalized IoT services in smart homes, as well as security cameras. For both

datasets, we trained a six-layer CT network and a CNN comparable in terms of the number of layers and intermediate tensor width. The architectures used are detailed in Table VIII in the Appendix. The CT accuracy was compared with the baseline CNN, with the CNN versions produced by the compression techniques used in Section VII-B, and with the efficient CNN frameworks [29], [35] used in Section VII-C. To compare with the six-layer CNN and CT networks, the MobileNet2 and ShuffleNet2 networks were constructed with six blocks: an initial convolution layer and a bottleneck block in the high resolution, followed by two bottleneck blocks in the second and third resolutions. The number of maps at each intermediate representation was identical to the widths used by the CT and CNN networks in Table VIII. The results are presented in Table V. With 25 and 250-K data samples, the CT network outperformed all accelerated methods, with results comparable with MobileNet V2, and was only outperformed by conventional CNN. In particular, the 250-K set results show that the CT can be effectively applied to large datasets.

VIII. Conclusion

In this work, we introduced a novel deep learning framework that utilizes indices computation and table access instead of dot products. The experimental successful validation of our framework implies that both optimization and generalization are not uniquely related to the properties and dynamics of dot-product neuron interaction and that having such neuronal ingredients is an unnecessary condition for successful deep learning. Our analysis and experimental results show that the suggested framework outperforms conventional CNNs in terms of computational complexity:capacity ratio. Empirically, we showed that it enables improved speed:accuracy tradeoff for the low-compute inference regime at the cost of additional processing memory. Such an approach is applicable to a gamut of applications that are either low-compute or are to be deployed on laptops/tablets that are equipped with large memory, but lack dedicated GPU hardware. Future work will extend the applicability of CT networks using an improved GPU-based training procedure. In this context, it should be noted that only basic CT networks were introduced and tested in this work, whereas competing CNN architectures enjoy a decade of intensive empirical research. The fact that CT networks were found competitive in this context is therefore highly encouraging. Combining CT networks with CNN optimization techniques (batch/layer normalization), regularization techniques (dropout), and architectural benefits (Residuals connections) is yet to be tested. Finally, to cope with the memory costs, methods for discretization/binarization of the model's parameters should be developed, similar to CNN discretization algorithms.

Appendix

The architectures used in Section VII-B for networks consisting of two and four layers are described in Tables VI and VII, respectively. The architectures used for CASIA-WebFace data in Section VII-C are detailed in Table VIII.

References

[1] A. Waissman and A. Bar-Hillel, "Input-dependent feature-map pruning," in *Proc. Int. Conf. Artif. Neural Netw. (ICANN)*, 2018, pp. 706–713.

[2] M. Al-Hami, M. Pietron, R. Casas, S. Hijazi, and P. Kaul, "Towards a stable quantized convolutional neural networks: An embedded perspective," in *Proc. 10th Int. Conf. Agents Artif. Intell.*, 2018, pp. 573–580.

[3] E. Bengio, P.-L. Bacon, J. Pineau, and D. Precup, "Conditional computation in neural networks for faster models," 2015, *arXiv:1511.06297*.

[4] A. Bulat and G. Tzimiropoulos, "XNOR-Net++: Improved binary neural networks," in *Proc. Brit. Mach. Vis. Conf. (BMVC)*, 2019, p. 62.

[5] S. R. Bulo and P. Kontschieder, "Neural decision forests for semantic image labelling," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 81–88.

[6] W. Chen, T. J. Wilson, S. Tyree, Q. K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2015, pp. 2285–2294.

[7] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, Cambridge, MA, USA, 2015, pp. 3123–3131.

[8] S. A. Davis and I. Arel, "Low-rank approximations for conditional feedforward computation in deep neural networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Y. Bengio and Y. LeCun, Eds., 2014, pp. 1–10.

[9] L. Deng and D. Yu, "Deep learning: Methods and applications," *Found. Trends Signal Process.*, vol. 7, nos. 3–4, pp. 197–387, Jun. 2014.

[10] Y. Eidelman, V. D. Milman, and A. Tsolomitis, *Functional Analysis: An Introduction*, vol. 66. 2004. AQ:9

[11] J. Ephrath, L. Ruthotto, E. Haber, and E. Treister, "LeanResNet: A low-cost yet effective convolutional residual networks," in *Proc. ICML Workshop On-Device Mach. Learn. Compact Deep Neural Netw.*, 2019, pp. 1–11.

[12] R. Gong et al., "Differentiable soft quantization: Bridging full-precision and low-bit neural networks," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 4851–4860.

[13] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2016, pp. 1–14.

[14] M. Hassoun, *Fundamentals of Artificial Neural Networks*. 1995.

[15] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*.

[16] K. Huang, B. Ni, and X. Yang, "Efficient quantization for neural networks with binary weights and low bitwidth activations," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 3854–3861.

[17] Y. Ioannou et al., "Decision forests, convolutional networks and the models in-between," 2016, *arXiv:1603.01250*.

[18] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," in *Proc. Brit. Mach. Vis. Conf.*, 2014, pp. 1–12.

[19] P. Kontschieder, M. Fiterau, A. Criminisi, and S. R. Bulo, "Deep neural decision forests," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1467–1475.

[20] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," 2018, *arXiv:1806.08342*.

[21] A. Krizhevsky, V. Nair, and G. Hinton, "CIFAR-10 (Canadian Institute for Advanced Research)," Tech. Rep., AQ:10

[22] A. Krizhevsky, V. Nair, and G. Hinton, "CIFAR-100 (Canadian Institute for Advanced Research)," Tech. Rep.,

[23] E. Krupka, A. Vinnikov, B. Klein, A. B. Hillel, D. Freedman, and S. Stachniak, "Discriminative ferns ensemble for hand pose recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 3670–3677.

[24] E. Krupka et al., "Toward realistic hands gesture interface: Keeping it simple for developers and machines," in *Proc. CHI Conf. Human Factors Comput. Syst.*, New York, NY, USA, May 2017, pp. 1887–1898.

[25] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *ATT Labs Online*, vol. 2, 2010. AQ:11

[26] Y. LeCun, S. John Denker, and A. Sara Solla, "Optimal brain damage," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, D. S. Touretzky, Ed., 1990, pp. 598–605.

[27] X. Lin, C. Zhao, and W. Pan, "Towards accurate binary convolutional neural network," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2017, pp. 344–352.

[28] Z. Liu, B. Wu, W. Luo, X. Yang, W. Liu, and K. Cheng, "Bi-Real Net: Enhancing the performance of 1-bit CNNs with improved representational capability and advanced training algorithm," in *Proc. 15th Eur. Conf.*, Sep. 2018, pp. 747–763.

[29] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet V2: Practical guidelines for efficient CNN architecture design," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 116–131.

[30] Y. Mansour, "Pessimistic decision tree pruning based on tree size," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 1997, pp. 195–201.

[31] B. Martinez, J. Yang, A. Bulat, and G. Tzimiropoulos, "Training binary neural networks with real-to-binary convolutions," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2020, pp. 1–11.

[32] M. Mathieu, M. Hena, and Y. LeCun, "Fast training of convolutional networks through FFTs," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2014, pp. 1–11.

[33] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2011, pp. 1–9.

[34] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, B. Leibe, J. Matas, N. Sebe, M. Welling, Eds. Cham, Switzerland: Springer, 2016, pp. 525–542.

[35] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.

[36] S. Ren, X. Cao, Y. Wei, and J. Sun, "Global refinement of random forest," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 723–730.

[37] J. Shotton et al., "Real-time human pose recognition in parts from single depth images," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2011, pp. 1297–1304.

[38] X. Su et al., "Searching for network width with bilaterally coupled network," *IEEE Trans. Pattern Anal. Mach. Intell.*, early access, Dec. 5, 2022, doi: 10.1109/TPAMI.2022.3226777.

[39] Y. Tang et al., "Manifold regularized dynamic network pruning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Los Alamitos, CA, USA, Jun. 2021, pp. 5016–5026.

[40] M. Tu, V. Berisha, M. Woolf, J.-S. Seo, and Y. Cao, "Ranking the parameters of deep neural networks using the Fisher information," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2016, pp. 2647–2651.

[41] B. Wu et al., "Shift: A zero FLOP, zero parameter alternative to spatial convolutions," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 9127–9135.

[42] Z. Yang et al., "Searching for low-bit weights in quantized neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, Red Hook, NY, USA, 2020, pp. 1–12.

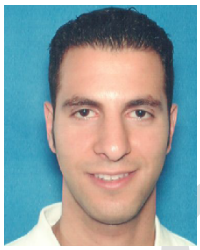
[43] D. Yi, Z. Lei, S. Liao, and S. Z. Li, "Learning face representation from scratch," 2014, *arXiv:1411.7923*.

[44] J. Yim, D. Joo, J. Bae, and J. Kim, "A gift from knowledge distillation: Fast optimization, network minimization and transfer learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 7130–7138.

[45] O. T. Yıldız, "VC-dimension of univariate decision trees," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 2, pp. 378–387, Feb. 2015.

[46] X. Zhang, J. Zou, K. He, and J. Sun, "Accelerating very deep convolutional networks for classification and detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 10, pp. 1943–1955, Oct. 2015.

[47] Z.-H. Zhou and J. Feng, "Deep forest: Towards an alternative to deep neural networks," in *Proc. 26th Int. Joint Conf. Artif. Intell.*, Aug. 2017, pp. 3553–3559.



Shay Dekel received the B.Sc. degree (summa cum laude) in electrical engineering from Bar Ilan University, Ramat-Gan, Israel, in 2005, and the M.Sc. degree (summa cum laude) in electrical engineering from Tel Aviv University, Tel Aviv, Israel, in 2011. He is currently pursuing the Ph.D. degree with the Faculty of Electrical Engineering, Bar Ilan University.

His research interests include neural 3-D reconstruction from sparse views, neuromorphic computing, and cognitive computing.

Yosi Keller received the B.Sc. degree in electrical engineering from the Technion-Israel Institute of Technology, Haifa, Israel, in 1994, and the M.Sc. and Ph.D. degrees (summa cum laude) in electrical engineering from Tel Aviv University, Tel Aviv, Israel, in 1998 and 2003, respectively.

He served as a Gibbs Assistant Professor for Yale University, New Haven, CT, USA, from 2003 to 2006. He is currently an Associate Professor with the Faculty of Engineering, Bar Ilan University, Ramat-Gan, Israel. His research interests include computer vision and deep learning.

Aharon Bar-Hillel was born in Beer-Sheva, Israel, in 1971. He received the B.A. degree in math and philosophy from Tel-Aviv University, Tel Aviv, Israel, in 1999, and the Ph.D. degree in neural computation from The Hebrew University of Jerusalem, Jerusalem, Israel, in 2006.

He was a Machine Learning and Computer Vision Researcher with Intel Research, from 2006 to 2008, GM Research, from 2009 to 2012, and a Microsoft Research, from 2013 to 2016. From 2016 to 2022, he served as a Senior Lecturer with the Department of Industrial Engineering and Management, Ben-Gurion University, Negev, Israel. He is currently a Researcher with Saips, a Ford Company. His research interests include interpretation and acceleration of deep networks, as well as applications of computer vision and machine learning for ADAS, agricultural phenotyping, and visual tracking.

AQ:12 AQ:13 AQ:14