Deep Convolutional Tables: Deep Learning without Neurons

Shay Dekel, Bar Ilan University Ramat-Gan, Israel
Yosi Keller, Bar Ilan University Ramat-Gan, Israel
Aharon Bar-Hillel Ben-Gurion University Negev, Israel

1 Introduction

During the last decade, deep learning techniques in general, and convolutional neural networks (CNNs) in particular have become the core computational approach in computer vision [17], [22]. Currently 'deep learning techniques' and 'deep neural networks' are synonyms, despite relating to different concepts. By definition [8], 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-product computing elements. Is the tight coupling between these concepts essential, or is it contingent and merely historical? Can deep learning be achieved without neurons? We claim here that this connection can be broken and that there is a good motivation for doing so.

The motivation to deviate from the dot-product neuron paradigm is the need for accelerated and more efficient CPU-based inference. CNNs consist of millions of 'neurons' - computing elements applying dot products to their inputs. The resulting computational burden is significant and is usually coped with by using GPUs or other specialized hardware. Due to these computational demands, deep learning applications are often restricted to server-side, where powerful dedicated hardware carry the computational load. Alternatively, applications running on standard CPU or low-computing platforms such as cellphones are limited to small, low-accuracy networks, or avoid using networks altogether. Examples of such applications are natural user interfaces (NUIs): face recognition and gesture estimation, mobile robotics including intelligent drones, or Internet of Things (IoT) devices. Low compute platforms are not equipped with a GPU and are expected to handle the inference of simple deep learning tasks. Such devices are not expected to train CNNs or run ImageNet-scale CNNs. For instance, most consumer laptops are not equipped with GPUs, but have gigabytes of memory, and can thus utilize the proposed scheme.

Why is deep learning currently implemented using neural networks? There are several arguments in favor of this connection. First, there is the biological inspiration: artificial neural network, at least in early days, attempted to mimic the brain operation, where millions of neurons operate in parallel. Second, there are theoretic arguments in favor of neural networks. It is known that they are universal approximators, able to approximate every continuous function on a compact domain in two layers only [13]. Moreover, there are theorems showing that they are computationally efficient: functions computed in T steps (on a Turing machine) can be implemented as network of depth $O(T)$ and size $O(T^2)$ [34]. Third and most important: the main argument in favor of neural networks is their empirical success in the last decade. However, in this work we claim that the arguments for neural networks are not conclusive, and that an alternative - a hierarchy of voting tables - may be preferable in low-compute domains.

Regarding the biological inspiration, its validity when engineering tasks are considered might be limited. Insofar as CPUs are not brains, they do not have the immense neuron parallelism of biological systems. Instead, CPUs are able to access large memory areas efficiently (random access). For a CPU with ideally unbounded memory access capabilities, the fastest classifier should be based on a single huge table: apply a sequence of simple binary queries to the input, build an index from the resulting bits and retrieve an answer from the corresponding table entry. Of course, this idea is hypothetical since the table would be huge. A practical alternative, however, is to replace the single table with an ensemble of smaller hierarchical tables.

In a convolutional layer, a dot product operation is densely applied to a patch around every input location. Instead of applying a dot product, we propose an alternative transformation comprising two steps. First, a set of K simple binary queries is applied to the input patch, thus encoding it with a K-bit binary codeword. Second, the computed codeword is used as an index into a table and retrieves the corresponding output representation. We denote the proposed transformation a Convolutional Table (CT). A CT layer includes multiple CTs whose outputs are summed, and same as convolutional layers, such layers are iteratively stacked in a CT network to derive a feature hierarchy. We detail this construction in Section 3.

Considering a single input location, the CT operation is the application of a single fern, and the operation of a CT layer (sum of CTs) corresponds to applying a fern ensemble. A fern is a tree in which all the 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.' In Sec-

All authors contributed equally.

tion 4, we compare the computational complexity of CNN and CT operations. By design, the complexity of a single CT layer does not depend on the size of the patch used ('filter size' in a convolutional layer), and its dependency on quadratic depth 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 5, we derive analytic 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. Regarding expressiveness, we show that a two-layered CT network has the same universal approximation capabilities of a two-layered neural network, and 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 procedure. The challenge arises since the inference requires computation of discrete indices, which is unamenable to gradient-based learning. Thus, we introduce in Section 6 a 'soft' formulation of K-bit codeword calculation, as a Cartesian product of K soft bit-functions. Following [24], [36], bit-functions are computed by comparing only two pixels in the region of interest. However, in their soft version, the pixel indices parameters are non-integer, 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 by a weighted combination of the possible votes. We use an annealing mechanism and an incremental learning schedule to gradually turn trainable, but inefficient, soft voting into efficient, single word voting at inference time.

We developed a CPU-based implementation of the proposed scheme and applied it to multiple standard datasets, consisting of up to 250K images. The results, presented in Section 7, show that CT-networks achieve an accuracy comparable to CNNs with similar structural parameters, while outperforming corresponding CNNs trained using weights and activations binarization. More importantly, We consider speed:accuracy and speed:accuracy:memory trade-offs, where speed is estimated via operation count. CT networks are shown to provide better speed:accuracy trade-off than efficient CNN architectures such as MobileNet [33] and ShuffleNet [28] in the tested domain. The improved speed:accuracy trade-off requires a higher memory consumption, as CT networks essentially trade speed for memory. Overall, CT networks are able to provide $3.1-12X$ acceleration over CNNs with memory expansions of $2.3X-10.6X$, which are often applicable for laptops and IoT applications.

In summary, our contribution in this paper is two-fold: First, we present an alternative to neuron-based networks and show for the first time that useful deep learning (i.e., learning of a feature hierarchy) can be achieved with other means. This is a significant departure from a tradition of dozens of years, and it enriches our understanding of what deep learning consists of. Second, from an applicative viewpoint, we suggest a framework enabling accelerated CPU inference for low-compute domains, with moderate costs of accuracy degradation and memory consumption.

2 Related work

Computationally efficient CNN schemes were extensively studied using a plethora of approaches, such as low-rank or tensor decomposition [16], [29], weight quantization [2], [19], [38], [45], conditional computation [1], [3], [7], using FFT for efficient convolution computations [30], or pruning of networks weights and filters [5], [12], [26], [37]. In the following, we focus on methods more related to our work.

Binarization schemes binarize network weights [6] and internal network activations [27], [32], thus reducing memory footprint and computational complexity by utilizing only binary operations. Methods which 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. Knowledge distillation approaches [14], [42] 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 improving accuracy. Regarding network design, several architectures were suggested for low-compute platforms [10], [28], [33], [39], utilizing group convolutions [33], efficient depth-wise convolution [28], [33], map shifts [39] and sparse convolutions [10]. Differentiable soft quantization was proposed by Gong et al., to quantize the weights of a CNN to a given number of bits. We compare the trade-offs of our scheme to low-compute versions of [28], [33].

The use of trees or ferns applied convolutionally for fast predictors was thoroughly studied in computer vision [23], [24], [35], [36]. A convolutional random forest enabled real-time human pose estimation in the Kinect console [36]. In [24] a flat CT ensemble classifier for hand pose enabled inference in less than 1 CPU millisecond, and in [23] it was extended to a full hand pose estimation system. However, in all these works, flat predictors were used, while we extend the notion to a deep network with gradient-based training. Other works merged tree ensembles with MLPs or CNNs for classification [15], [18] or semantic segmentation [4]. Specifically, 'conditional networks' were presented in [15] as CNNs with a tree structure enabling conditional computation to improve the speed-accuracy ratio. Unlike our approach, where ferns replace the dot-product neurons, these networks use standard dot-product neurons and layers, and the tree/forest is a high-level routing mechanism.

A scheme related to our approach was proposed by Zhou et al. [46], wherein the authors suggest a cascade of forests applied convolutionally. 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 classifier's structure is deep, there is no attempt for global end-to-end optimization - 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 CNNs.

3 Convolutional Table Transform

A single convolutional table is a transformation accepting a representation tensor $T^{in} \in \mathbb{R}^{H_i \times W_i \times D_i}$ and creating an output representation tensor $T^{out} \in \mathbb{R}^{H_o \times W_o \times D_o}$. Formally

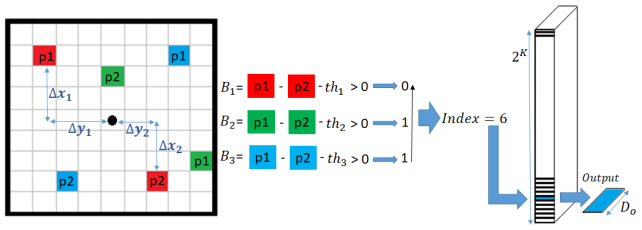

Fig. 1. A Convolutional Table (CT) with a word calculator of three bits operating on a 2D input tensor. Each bit-function is given by two pixels whose difference is compared to a threshold (colored in red, green, and blue). The word computed by the three bits, '110' = 6, is used to access the 6^{th}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

it is a pair (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 and returning a K-bit index, i.e., a function $B : \mathbb{R}^{l\times l\times D_i} \rightarrow \{0,1\}^K$, where l is the patch size. The voting table $W \in M_{2^K\times D_o}$ contains the possible 2^K outputs as rows, each a D_o-dimensional vector.

The index is computed by a sequential application of K bit-functions $\{B^k\}_{k=1}^K$, each of which computes a single bit of the index. Let $P = (p_x, p_y) \in \{1, .., H_o\} \times \{1, .., W_o\}$ be a location in T^{in}. We denote by $B(P;\Theta) = (B^1(P;\Theta^1), ..., B^K(P,\Theta^K))$ the computed index for the patch centered at location P. $\Theta = (\Theta^1, ..., \Theta^K)$ are the word calculator parameters, with Θ^k explicitly characterized below. The functions $B^k(P;\Theta^k)$ (dependence on T^{in} is omitted for notation brevity) produce their output in $\{0,1\}$ by thresholding at 0 simple smooth functions $b^k(P;\Theta^k)$ of a few input pixels. (i.e., $B^k(P;\Theta) = Q\left(b^k(P;\Theta^k)\right)$ with $Q(\cdot)$ the Heaviside function). Following previous work with flat classifiers [24], [36], the bit-functions b^k that we suggest to start with are thresholded differences of two input values:

$$\begin{aligned} b^k\left(P;\Theta_k\right) &= T^{in}\left(p_x+\Delta x_1^k, p_y+\Delta y_1^k, c^k\right) \\ &\quad -T^{in}\left(p_x+\Delta x_2^k, p_y+\Delta y_2^k, c^k\right) - th^k \end{aligned} \tag{1}$$

where $\Theta^k = (\Delta x_1^k, \Delta y_1^k, \Delta x_2^k, \Delta y_2^k, th^k, c^k)$ are the learned parameters. The codeword computed by the word calculator is used as an index to the voting table to obtain the D_o-dimensional output $W(B(P;\Theta),:)$ for location P. Figure 1 depicts the convolutional table operation at a single location.

A CT (with 'same' padding, stride=1) applies the operation defined above at each position $P \in \{1, .., H_i\} \times \{1, .., W_i\}$ of the input tensor to obtain the corresponding output tensor column. Same as in convolutional layers, padding can be applied to the input for computations at border pixels, and a stride may be included to reduce the spatial dimensions of the output tensor. A CT layer includes M convolutional tables $\{(B_m, W_m)\}_{m=1}^M$ whose output is summed:

$$T^{out}[p_x, p_y, :] = \sum_{m=1}^{M} W_m[B_m(T^{in}_{N(P)}), \ : \ ]$$

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

Algorithm 1 Convolutional Table Layer

INPUT: A tensor $T^{in} \in \mathbb{R}^{H_i\times W_i\times D_i}$, 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: $T^{out} = 0$
for $P \in \{1, .., H_o\} \times \{1, .., W_o\}$ **do** ▷ All locations
  for $m = 1, .., M$ **do** ▷ All tables
    Compute $b = B_m(P;\Theta^{k,m}) \in \{0,1\}^K$
    $T^{out}[p_x, p_y, :] = T^{out}[p_x, p_y, :] + W(b, :)$
  end for
end for
Return T^{out}

CT network stacks multiple CT layers one over the other in a chain or graph structure, similar to convolutional layers usage in standard CNNs.

4 Computational Complexity

Let the input and output tensors be $T^{in} \in R^{H_i\times W_i\times D_i}$ and $T^{out} \in R^{H_i\times W_i\times D_o}$ respectively, and let $l \times l$ be the filter dimensions. For a convolution layer, the number of operations per location is $C_{CNN} = l^2 D_i D_o$, as it computes D_o internal 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 the cost of computing a single bit-function. The voting cost is MD_o, as we add M D_o-dimensional vectors to get the output. In summary, the complexity of a CT layer is $C_{CT} = M(C_b \cdot K + D_o)$.

An immediate observation is that the computational complexity of a CT does not depend on the filter size l. Hence, it allows combining evidence from large spatial regions with no additional computational cost. To compare CNN and CT complexities, assume that both use the same representation dimension $D_i = D_o = D$. The relation between the total number of bit-functions MK and D depends on whether each bit-function uses a separate dimension, or dimensions are re-used by multiple bit-functions. For a general analysis, assume each dimension is reused by R bit-functions, such that $D = MK/R$ (in our experiments $R \in [1, 3]$). The complexity ratio between CNNs and CT layers with the same representation dimension D is

$$\frac{C_{CNN}}{C_{CT}} = \frac{l^2D^2}{M(C_b\cdot K + D)} = \frac{l^2D^2}{C_bDR + \frac{D^2R}{K}} \approx \frac{Kl^2}{R} \tag{2}$$

where the last approximation is valid for large D, and the quadratic terms in D are dominant (C_b is a small constant,with $C_b = 10$ in our implementation). This ratio indicates an acceleration potential larger than an order of magnitude for a reasonable choice of parameter values. For instance, using $l = 3$, $K = 8$, $R = 2$ results in $36X$ acceleration.

While the outcome of this basic analysis is promising, several lower-level considerations are important for 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 the vast majority of the computation. The CT transformation, however, obeys these constraints. Contrary to trees, bit computation in ferns

can be vectorized over adjacent input locations since all locations use the same bit-functions, and memory access is contiguous. The voting operation can be vectorized over output dimensions. Such an implementation was already built and shown to be highly efficient for flat CT classifiers in [23]. 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 instance, a model of 50 layers, wherein $M = 10$ ferns, $K = 6$, and $D = 60$, has a total size of $1.92MB$ if each parameter is stored in 8-bit precision.

5 CT CAPACITY AND EXPRESSIVENESS

In this section we present two analytic insights indicating the feasibility of deep CT networks and their possible advantage. In Section 5.1 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 neuron. In Section 5.2 a network of two fern layers is shown be a universal approximator, similarly to known results on neural networks.

In the analysis we consider a simplified non-convolutional setting, with an input vector $X \in [0,1]^d$, and a non-convolutional fern ensemble classifier. Furthermore, simple bit-functions of the form $B^k(X) = Q\left((-1)^{s^k}(X[I^k] - t^k)\right)$ are considered, where $s^k \in \{-1, 1\},\ I^k \in 1, .., d\ , t^k \in [0, 1]$. These simple bit-functions just compare a single input dimension to a threshold.

5.1 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 $H = \{f(X) = sign(W(B(X))) : B = \{(s^k, I^k, t^k)\}_{k=1}^{K}, W \in R^{2^K}\}$. For $K \geq 4$ and $log_2(d) < K \leq d$ its VC-dimension is*

$$2^K \leq VC - dim(H) \leq K2^K \tag{3}$$

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

Lower bound: Consider a sample containing the points on the d-dimensional cube, i.e.

$$S = \{P_c = (-1^{c_1}, ... - 1^{c_K}) :\ c = (c_1, .., c_K) \in \{0, 1\}^K\ \}. \tag{4}$$

It is easy to see 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 at least with respect to one dimension, they will have different bits in at least one bit functions. 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. Hence S is shattered, showing $VCdim(H) \geq 2^K$.

Upper Bound: Assume $K \geq 4$ and $log_2(d) < K \leq d$. We compute an upper bound on the number of possible labelings (label vectors) enabled by the binary fern family on a sample of size n, and show 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 for the K bit-functions (one dimension per function). For each bit-function, once the input dimension I is chosen, we may re-order the examples according to their value in dimension I, i.e., $X_1[I] \leq X_2[I] ... \leq X_n[I]$. Non trivial partitions of the sample, i.e. partitions into two non-empty 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 one 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 may 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. We hence solve

$$(dn)^K 2^{2^K} < 2^n \tag{5}$$

and after taking log of base 2

$$2^K + Klog_2 d + Klog_2 n < n \tag{6}$$

By choosing $n = K2^K$ we see that Eq. 6 is satisfied since

$$\begin{aligned} &2^K + Klog_2 d + Klog_2 K + K^2 \\ &\qquad < 2^K + 3K^2 \leq 2^K + (K-1)2^K \leq K2^K \end{aligned} \tag{7}$$

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 are known for trees [43], [44], and the lemma implies that the capacity of ferns is not significantly lower than 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 $d = K$ inputs performs $O(K)$ operations to get 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.

5.2 CT networks are universal approximators

The following theorem states the universal approximation capabilities of a 2-layer CT network. It resembles the known result regarding two-staged neural networks [13].

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

Proof. The proof idea is simple: we shows 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 hyper rectangle 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 $R = [a_1, b_1] \times [a_2, b_2] ... \times [a_d, b_d]$ be a hyper rectangle in R^d and $v \in R$ a scalar value. A rectangle function $s(x; v, R) : [0,1]^d \rightarrow R$ 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

$$G = \{g : [0,1]^d \rightarrow [0,1] : g(x) = \sum_{p=1}^{P} s(x; v_p, R_p)\} \quad (8)$$

It is known that G is dense in $(C[0,1]^d)$ (from the Stone-Weierstrass theorem, see e.g. [9]). We will show that the family of 2-layer CT networks includes this set.

Let $s(v, R)$ be an arbitrary rectangle function. for each dimension $i = 1, .., d$ 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 $j = 1, .., 2d$. By construction, $\{B^j_R\}_{j=1}^{2d}$ characterize the rectangle

$$x \in R \iff \forall j = 1, .., 2d \quad B^j_R(x) = 1. \quad (9)$$

Equivalently, we have $x \in R$ iff $\sum_{j=1}^{2d} B^j_R(x) > 2d - \frac{1}{2}$. 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. 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

$$W^j_p[i, l] = \begin{cases} 1 & i = 1, l = p \\ 0 & otherwise \end{cases} \quad (10)$$

The output vector of 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

$$\begin{aligned} Y[p] &= \sum_{p'=1}^{P} \sum_{j=1}^{2d} W^j_{p'}[B^j_{R'_p}(x), p] \quad (11) \\ &= \sum_{j=1}^{2d} W^j_p[B^j_{R_p}(x), p] \\ &= \sum_{j=1}^{2d} B^j_{R_p}(x) \end{aligned}$$

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

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

$$\begin{aligned} U &= \sum_{p=1}^{P} W^{(2)}_p[B^{(2)}_p(x)] \quad (12) \\ &= \sum_{p=1}^{P} v(p) \cdot B^{(2)}_p(x) = \sum_{p=1}^{P} v(p) \cdot 1_{x \in R_p} \end{aligned}$$

U hence implements the function $g(x)$, which completes the proof. □

6 Training with soft Convolutional Tables

Following Equation 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 6.1 and discuss its gradient in 6.2

6.1 A soft CT version

In order to enable gradient-based learning, we suggest to replace the Heaviside function during optimization with a linear sigmoid $q(x; t)$ for $t > 0$:

$$q(x; t) = min(max(\ (t + x)/2t\ , 0), 1) \quad (13)$$

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

Following Section 3, a word calculator B maps a patch P to a single index, or equivalently into a (row) one-hot vector in R^{2^K}. Denote the word calculator under 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 $b \in \{0, .., 2^K - 1\}$ measuring the activity of word b, provides a natural differentiable extension of the CT formulation. For an index b define the sign of its k^{th} bit by $s(b, k) \triangleq (-1)^{(1+u(b,k))}$, where $u(b, k)$ denotes the k^{th} bit of the index b in its standard binary expansion. $s(b, k)$ is 1 if bit k is 1, -1 if it is 0. The activity level of word b in a soft word calculator is defined by

$$\vec{B}^s(P; \Theta)[b] = \prod_{k=1}^{K} q(\ s(b, k) \cdot b_k(P; \Theta_k); t\ ) \quad (14)$$

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 shows that $\sum_{b=0}^{2^K-1} \vec{B}^s(P)[b] = 1$, hence the soft word calculator defines a probability distribution over the possible K-bit words. The soft CT is defined as the

natural extension $\vec{B}^s W$. Note that for $t \to 0$, 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 3.

The 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 the CT is applied to all of the spatial locations in the activation map, the linear layer W corresponds to a 1×1 convolution, and we implemented a sparse 1×1 convolutional layer operating on a sparse tensor $\mathbf{x} \in H \times W \times 2^K$ and outputting a dense tensor $\mathbf{y} \in R^{H \times W \times D_o}$.

6.2 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 $l \times l$ neighborhood of location P in T^{in} by $T^{in}_{N(P)}$. Considering a single output variable b at a time, the gradient with respect to the input patch $T^{in}_{N(P)}$ is given by

$$\frac{\partial \vec{B}^s(P)[b]}{\partial T^{in}_{N(P)}} = \sum_k \frac{\partial \vec{B}^s(P)[b]}{\partial b_k(P;\Theta_k)} \cdot \frac{\partial b_k(P;\Theta_k)}{\partial T^{in}_{N(P)}}, \quad (15)$$

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

$$\frac{\partial \vec{B}^s(P)[b]}{\partial b_k(P;\Theta_k)} = \prod_{j \neq k} q(s(b,j)b_j(P;\Theta_j);t) \frac{\partial q(s(b,j)b_j(P;\Theta_j);t)}{\partial b_k(P;\Theta_k)} = \frac{\vec{B}^s(P)[b] \cdot t^{-1} \cdot 1_{|b_k|<t} \cdot s(b,k)}{2q(\, s(b,k) \cdot b_k(P;\Theta_k))} \quad (16)$$

The derivative is non-zero 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 $-t < b_k(P) < t$.

In order to further compute the derivatives of $b_k(P)$ with respect to the input and parameters ($\frac{\partial b_k(P;\Theta_k)}{dT^{in}_{N(P)}}$ and $\frac{\partial b_k(P;\Theta_k)}{d\Theta}$ respectively), note first that the forward computation of $b_k(P)$ requires in practice estimation of the input tensor at fractional spatial coordinates. This is the case since the offset parameters $\Delta x_1, \Delta y_1, \Delta x_2, \Delta y_2$ are now trained with gradient descent, requiring 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 example, $\frac{\partial b_k(P)}{\partial \Delta x^k_1}$ is given by

$$\frac{\partial b_k(P)}{\partial \Delta x^k_1} = \frac{\partial T^{in}\left(p_x + \Delta x^k_1, p_y + \Delta y^k_1, c\right)}{\partial \Delta x^k_1} = \frac{\partial T^{in}(:,:,c)}{\partial x}[(p_x + \Delta x^k_1, p_y + \Delta y^k_1)] \quad (17)$$

where $\frac{\partial T^{in}(:,:,c)}{\partial x}$ is the partial x-derivative of the image channel $T^{in}(:,:,c)$, that is estimated numerically and sampled at $(x,y) = (p_x + \Delta x^k_1, p_y + \Delta y^k_1)$ to compute Eq. 17. The derivatives with respect to other parameters are computed similarly. Note that the channel index parameters c^k are not learnt - these are fixed during network construction for each bit-function. As for $\frac{\partial b_k(P;\Theta_k)}{dT^{in}_{N(P)}}$, by considering Eq. 1, it can be seen that $b_k(P;\Theta_k)$ only relates to two fractional pixel locations: $\left(p_x + \Delta x^k_1, p_y + \Delta y^k_1\right)$ and $\left(p_x + \Delta x^k_2, p_y + \Delta y^k_2\right)$. This implies that a derivative with respect to eight pixels is required, as each fractional-coordinates pixel value is bilinearly interpolated from its four neighboring pixels in integer coordinates.

The sparsity of the soft word calculator (for both forward and backward) is governed by the parameter t of the sigmoid in Eq. 13, that acts as a threshold. When t is large, most of the bit-functions are 'ambiguous,' 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. Hence, we introduce an annealing scheduling scheme, such that t is initiated as $t \gg 0$, set to allow a fraction f of the bit-functions values to be in the 'soft zone' $[-t,t]$. The value of t is then gradually lowered to achieve a sharp and sparse classifier towards the end of the training phase.

7 Experimental Results

In section 7.1 we explore the sensitivity of CT networks to their main hyper parameters - the number of ferns M and number of bit functions per fern K, and show the benefits of distillation as a CT training technique. We then turn to comparison between CT-based architectures and similar CNNs. Such a comparison is non trivial, since due to the structural differences it is not clear what 'similar' means, and similarity can be defined in multiple planes. In section 7.2 we compare Deep CT to CNNs with similar depth and width, i.e. number of layers and maps per layer. The comparison focuses on efficient methods suggested for network quantization and binarization, as the CT framework uses similar notions. In Section 7.3 a comparison is done on a more practical basis: CT is compared to CNN versions with comparable computational complexity, i.e. similar number of MAC (Multiply–ACcumulate) operations. We compare the CT to CNN architectures specifically designed for inference efficiency, such as MobileNet [33] and ShuffleNet [28]. The comparison is done by studying speed:accuracy and speed:accuracy:memory trade-offs enabled by the compared frameworks. Finally, to exemplify the applicability of Deep CT to low-power Internet-of-Things (IoT) applications and larger datasets, it was applied to an IoT-based face recognition task with results reported in 7.4.

The Deep CT word calculator layer and the sparse voting table layer were implemented in un-optimized C. All Deep CT networks were trained from scratch using random Gaussian initialization of the parameters. To facilitate the convergence of the proposed scheme, a training scheme consisting of three phases was applied. First the lower half of the CNN (layers #1-#6 in Table 8) 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 the lower layers #1-#6 kept fixed. Last,

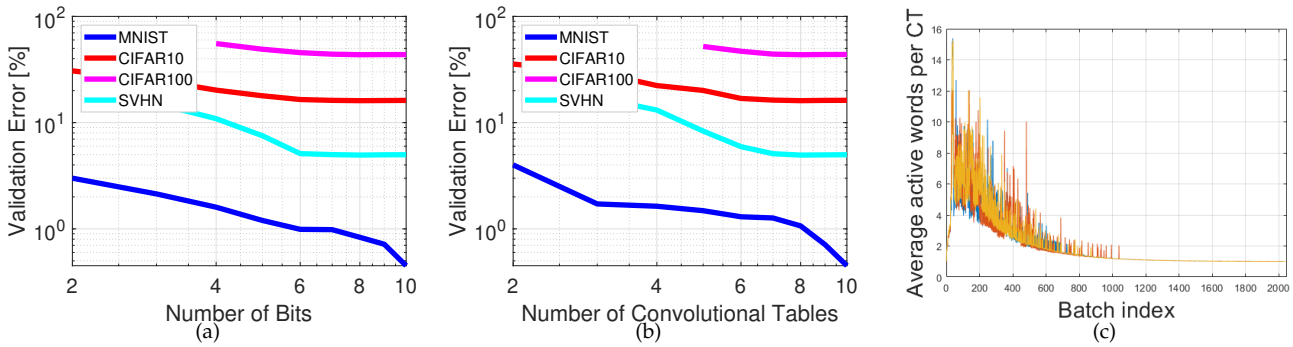

Fig. 2. Hyperparameters study on MNIST, CIFAR10, CIFAR100, and SVHN datasets. **(a)** Deep CT network error Vs. the number of bits K in a log-log plot. In all layers, the number of convolutional tables M was fixed to 10. **(b)** Deep CT network error Vs. the number of convolutional tables M in a log-log plot, for a fixed number of bits $K = 10$. **(c)** The average number of active words per CT Vs. the training batch index, evaluated using the CIFAR-10 dataset. Each convolution table is depicted in a different color. Due to the annealing mechanism, the number of active words is reduced during the training and converges to a single active word as expected.

we unfreezed all layers and refined the entire network using a distillation-based approach, following Hinton et al. [14]. In this approach the CT was trained to optimize a convex combination of the CE loss and a Kullback-Leibler divergence from probabilities induced by a teacher CNN. The temperature parameter of the induced probabilities was initiated at 4 and was gradually annealed to 1 towards the end of training.

7.1 Hyper parameter sensitivity

The expressive power of the Deep CT transformation depends on the number of CTs per layer M, and the number of bit-functions K used in each CT. We evaluated the sensitivity of the Deep CT network to these parameters and the suggested annealing schedule. In these experiments, CT networks with four layers were applied to the SVHN, CIFAR-10, CIFAR-100 datasets, and a two layers networks was used for MNIST. The Networks are similar to those shown in the appendix, in tables 6 and 7. We used a fixed patch size $l = 5$ and input and output dimensions of $D_i = D_o = 32$, respectively.

The classification errors as a function of K (for a fixed M) and as a function of M (for fixed K) are shown in Figs. 2a and 2b, respectively. For both parameters, and for most datasets, there is a range in which the error exponentially decrease as the parameters increases. For the deeper networks (datasets other than MNIST), the accuracy improvement saturates for M, K values higher than 8. Figure 2c shows the average number of active words in a convolutional table 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 initialized to 0.2 and was gradually reduced exponentially, where the sigmoid threshold t was set periodically according to the required f. The number of active words per CT is thus gradually reduced and converged to a single active word in the final classifier. The four-layer CNN obtained a test error of 13.904%, compared to 13.895% obtained with the same architecture without using annealing. The annealing-based training process hence resulted in a sparser classifier, without significantly degrading the classification accuracy.

To establish the best hyper parameter configuration more detailed experiments were carried for several M and K combinations around $M = 8, k = 8$. Experiments were conducted on the Cifar-10 dataset, with the four and six layers CT networks detailed in Tables 7 and 2, respectively. In addition, training with and without distillation from a teacher CNN were tested. The $M = 8, K = 8$ configuration was found the most accurate. Distillation was found beneficial, providing accuracy improvements of $1\% - 1.5\%$.

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	**10.57**
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**

TABLE 1
Joint hyper parameter tuning for the four and six layers CT networks described in Tables 7 and 2. M and K denote the Convolutions tables size and the number of bits, respectively.

7.2 Accuracy comparison of low-complexity architectures

While deep CT and CNN networks use very different computational elements, they both use a sequence of intermediate representations developed toward the classification task. Hence, CT and CNN networks can be considered similar if they have the same number of intermediate representations used (depth) and the same number of maps in each representation (width). In the experiments reported in this section, we compare CT networks to CNNs with identical configurations in these respects. While intermediate spatial tensors were designed to be similar, we allowed differences in top decision layers, which were chosen to optimize classification accuracy for each network type separately. Thus, the CNN-based networks use max pooling and an FC layer, while the CT CNN uses only an average pooling layer (while increasing accuracy for the CNNs, note that the FC layer utilizes many additional weights).

	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 2
The 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 convolutional tables, and D_o is the number of output maps.

Dataset	DeepCT	BC [6]	XN [32]	TABCNN [27]	DSQ [11] 2b	DSQ [11] 4b	DSQ [11] 6b	DeepCT + distil.	CNN
Two layers									
MNIST	**0.35**	0.61	0.51	0.49	0.41	0.38	0.36	-	0.334
Four layers									
CIFAR10	**12.49**	14.47	14.77	13.97	12.87	12.71	12.68	10.91	10.86
CIFAR100	**39.17**	50.27	47.32	45.32	40.14	40.05	39.96	37.12	36.81
SVHN	4.75	6.21	5.88	5.11	4.65	4.58	**4.51**	4.46	4.27
Six layers									
CIFAR10	**11.05**	12.56	12.91	12.13	11.35	11.27	11.14	10.25	9.88
CIFAR100	**37.95**	42.43	40.14	39.37	38.31	38.25	38.12	36.05	35.58
SVHN	4.12	4.95	4.54	4.28	4.1	4.08	**4.0**	3.96	3.77

TABLE 3
Classification accuracy comparison. We compare the results for CNNs consisting of 2 layers (MNist) or 4 and 6 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 6, 7 and 2). The most accurate results are marked in **bold**.

We compare the classification accuracy of CT networks to standard CNNs and to contemporary low complexity CNN architectures based on CNN binarization schemes. Compared methods include the BinaryConnect (BC) [6] [1], XNOR-Net XN [32] [2], and TABCNN [27][3], using their publicly available implementations. These methods, [32] and [27] in particular, also encode the input activity using a set of binary variables, but use dense encoding and apply a binary dot-product to their encoded input, unlike our table-based approach. Last, we applied the state-of-the-art Differentiable Soft Quantization (DSQ) scheme [11][4] to the conventional continuous CNN to quantize it to 2, 4 and 6 bits.

All networks were applied to the MNIST [25], CIFAR-10 [20], CIFAR-100 [21], and SVHN [31] datasets. For the MNIST dataset we applied CT networks and CNNs consisting of two layers, as this data does not required additional depth. For the other datasets networks of four and six layers were applied. The exact architectures for the six-layer networks are described in Table 2, and the architectures of the two and four layers used are detailed in the appendix in tables 6, 7 respectively. Classification accuracy results are reported in Table 3. On the left side of the table all methods are compared with the same loss, based on the standard cross entropy. In most cases, CT networks outperformed the binarization and discretization schemes. The closest competitors are DSQ-quantized CNNs which outperformed CT networks for the SVHN data set. All schemes were outperformed by the conventional CNN. On the right side of the table CT network results are presented when a distillation loss is used, with VGG16 [40] as the teacher network. In this case CT network results improve significantly further, and the margin of CNNs over them is remarkably small.

Based on these results, two fundamental points can be noted: first, the good accuracy of CT networks implies that in spite of their excess capacity they can be trained without significant overfit using standard SGD optimization, same as CNNs. While the 'implicit prior' preventing overfit is not clear for neither of them, it seems to operate for CT networks too. Second, the comparable accuracy of CT and CNN networks with similar intermediate representation structure is notable, and hints that accuracy is more related to representation hierarchy structure then to the nature of the computing elements.

7.3 Trade-offs comparison: Accuracy, speed and memory

A set of experiments was conducted to study the Accuracy:Speed and Accuracy:Speed:Memory trade-offs enabled by CT networks, and compare it to alternative schemes. The experiments were conducted on the CIFAR10 and CIFAR100

1. https://github.com/MatthieuCourbariaux/BinaryConnect
2. https://github.com/rarilurelo/XNOR-Net
3. https://github.com/layog/Accurate-Binary-Convolution-Network
4. https://github.com/ricky40403/DSQ

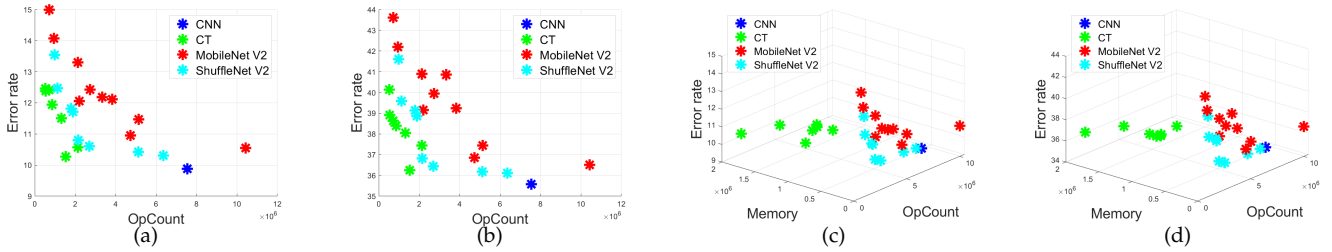

Fig. 3. Performance trade-offs of the deep CT six-layers architectures. **(a)** speed:Accuracy trade-off of several model families on Cifar-10. Models are based on a baseline CNN (Table 7), CT networks, MobileNet V2 [33], and ShuffleNet V2 [28] architectures. Speed is measured in terms of MACC operation count. Architectures with up to 10^7 operations are considered. **(b)** Speed:Accuracy trade-off on the Cifar-100 dataset. **(c)** Speed:Accuracy:Memory trade-off on Cifar-10. Memory is measured in terms of the number of parameters. **(d)** Speed:Accuracy:Memory trade-off on Cifar-100. CT-networks provide better accuracy:speed trade-off in this domain while requiring additional memory.

Accuracy	Speedup			memory ratio		
	CT	[33]	[28]	CT	[33]	[28]
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

TABLE 4
Speed and memory trade-off with CT, MobileNet2 [33] and ShuffleNet2 [28], for cifar-10 and cifar-100. Each table line considers an accuracy goal, and shows the speed and memory trade-off enabled by the three frameworks, based on operation and parameter count respectively. Speedup and memory ratios are computed by $\frac{R_{CNN}}{R_{Net}}$ where R_{CNN} is the resource requirement of the baseline CNN from Table 3, and R_{Net} is the considered network requirements (so higher is better). The baseline CNN uses $7.54 \cdot 10^7$ MACC operations, $181 \cdot 10^6$ parameters, and obtains accuracy of 9.88 and 35.58 on Cifar-10 and Cifar-100 respectively. The tables for cifar-10 and cifar-100 are very similar since the same architectures are usually preferable for both datasets in each accuracy and network type category.

datasets. Seven CT network models were trained, with an operation count budget that was $3.1X - 12X$ lower than the operation count of the baseline CNN from Table 7 ($7.54 \cdot 10^6$). Like the baseline CNN, the CT models had six layers. The seven variations were obtained by varying the hyper parameters M, K, and the number of channels at intermediate layers. ′

We compare the trade-off obtained by CT networks to the trade-off enabled by the MobileNet V2 [33] and ShuffleNet V2 [28] efficient CNN architectures. The first is based on inverted bottleneck residual blocks, where depth-wise convolutions are performed on high-depth representations, and the latter utilized blocks where half of the channels are processed, followed by a channel shuffle operation. We used the code provided by the authors [5] [6], and adapted it to the relevant low-compute regime by controlling its hyperparameters. For MobileNet, 10 network versions were generated by controlling the expansion parameter t (Table 1 in [33]), the number of channels in intermediate representations, and the number of bottleneck modules in a block (between 1 and 2). For ShuffleNet, 8 networks were generated by changing the number of channels in intermediate layers and the number of bottleneck modules in a block (between 1 and 4).

The accuracy:speed results are shown in figure 3a-b for CT networks and the competing lean network approaches. The plots show a significant advantage of CT networks over the baseline CNN and competing architectures. To enable more explicit assessment, Table 4 left side shows the speedup enabled by the most efficient networks in each fraemwork, for several accuracy goals on Cifar-10 and Cifar-100. At the high accuracy end, of losing at most 1% accuracy w.r.t the baseline CNN, a CT network enables $4.91X$ acceleration over the CNN, and accleration of $1.75 - 6.8X$ over competing methods. When more significant accuracy drops can be tolerated (2.7% for Cifar-10, 4% for Cifar-100), CT networks enable accelerations of $13 - 14X$ over the baseline CNN, significantly better than the $6.7X$ acceleration obtained by the best competitor [28].

Figures 3c-d depicts the triple trade-off of Accuracy:Speed:Memory for CT networks and the competition, over Cifar-10 and Cifar-100 respectively. The memory requirements of the most efficient networks for each accuracy goal are listed on the right side of Table 4, as compression ratios w.r.t the baseline CNN. Clearly, the better Accuracy:Speed trade-off of CT networks comes at the expense of using larger memory (more parameters). This is not surprising, as in CT transformations, the compute:memory ratio is much lower, with memory-based retrieval replacing dot-product operations. For example, in the high accuracy regime for Cifar-10, obtaining the $4.9X$ speedup over baseline CNN has the cost of using $10.7X$ more parameters. Nevertheless, the excess memory can be controlled by limiting K (for which the memory demand is exponential) to moderate values. In can seen in the Table 4 that the fastest CT network enabling Cifar-10 12.6 accuracy provides an $14.3X$ acceleration with the cost of only $2.2X$ more parameters than the baseline CNN.

7.4 A face recognition application

We experimented with CT networks for face recognition using the CASIA-WebFace dataset [41]. Two datasets of 64×64 extracted face images were formed. First, we randomly drew a subset consisting of 50 classes (each with 5000 images), 250K 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

5. https://github.com/xiaochus/MobileNetV2
6. https://github.com/TropComplique/shufflenet-v2-tensorflow

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	**87.65**	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

TABLE 5
The accuracy of Deep CT, CNN, MobileNet V2 [33] and ShuffleNet V2 [28] on subsets of the CASIA-WebFace dataset.

identities), each consisting of 5000 images, to exemplify 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 CT network with six-layers, and a CNN comparable in the number of layers and intermediate tensor width. The architectures used are detailed in Table 8 in the appendix. CT accuracy was compared to the baseline CNN, to the CNN versions produced by the compression techniques used in Section 7.2, and to the efficient CNN frameworks [33], [28] used in Section 7.3. In order to obtained architectures comparable to the six-layers CNN and CT networks, 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 of Table 8.

The results are presented in Table 5. With $25K$ examples, the CT network was ranked second, only outperformed by the 6-bit DSQ method. With $250K$ examples it outperformed all accelerated methods, with results comparable to MobileNet V2, and was only outperformed by the conventional CNN. In particular, the $250K$ set results show that the CT can be effectively applied to large datasets.

8 Summary and Future Work

In this work we introduced a deep learning framework which does not use dot product neurons, and relies instead on indices computation and table access. To the best of our knowledge, this is the first time a successful deep learning framework is presented which does not use neurons at all. The fact that such a framework is possible has significant implications when theory explanations for the success of deep learning are sought for. Specifically, it entails that both successful optimization, as well as successful generalization, are not tightly related to the properties and dynamics of dot-product neuron interaction, or at least: that having such neuronal ingredients is not a necessary condition for successful deep learning.

Our analysis and experimental results show that the suggested framework has advantages over conventional CNNs in terms of computational complexity:capacity ratio. Empirically, we showed it enables improved speed:accuracy trade-off for the low compute inference regime at some cost in memory. Such an approach is applicable to a gamut of applications that are either low-compute, or similar to laptops/tablets that are equipped with large memories, but lack dedicated hardware as a GPU.

Due to implementation considerations - the lack of a highly parallelized implementation - experiments were currently limited to medium size datasets and network complexities. Future work will include extending the applicability range of CT networks using a GPU-based training implementation and improved training procedures. In this context, it should be noted that only basic CT networks were introduced and tested in this work, while competing CNN architectures enjoy from a decade of intensive empirical research. The fact that CT networks were found competitive in this context is hence highly encouraging. Combining CT networks with CNN optimization techniques (like batch/layer normalization), regularization techniques (like dropout), and architectural benefits (like Residuals connections) is yet to be tested. Finally, in order to cope with the memory costs, methods for discretization/binarization of the model's parameters should be developed, similarly to CNN discretization algorithms.

Appendix

The architectures used in section 7.2 for networks of 2 and 4 layers are described in tables 6,7 respectively. The architectures used for CASIA-Webface data in section 7.3 are described in table 8.

References

[1] Waissman A. and Bar-Hillel A. Input-dependent feature-map pruning. In *International Conference on Artificial Neural Networks (ICANN)*, pages 706–713, 2018.

[2] Motaz Al-Hami, Marcin Pietron, Raúl A. Casas, Samer L. Hijazi, and Piyush Kaul. Towards a stable quantized convolutional neural networks: An embedded perspective. In *ICAART*, 2018.

[3] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *CoRR*, abs/1511.06297, 2015.

[4] Samuel Rota Bulo and Peter Kontschieder. Neural decision forests for semantic image labelling. In *CVPR*, 2014.

[5] Wenlin Chen, James T. Wilson, Stephen Tyree, Kilian Q. Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 2285–2294. JMLR.org, 2015.

[6] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 3123–3131, Cambridge, MA, USA, 2015. MIT Press.

[7] Andrew S. Davis and Itamar Arel. Low-rank approximations for conditional feedforward computation in deep neural networks. *CoRR*, abs/1312.4461, 2013.

[8] L. Deng and D. Yu. *Deep Learning: Methods and Applications*, volume 7. Foundations and Trends in Signal Processing, 2014.

[9] Yuli Eidelman, Vitali D Milman, and Antonis Tsolomitis. *Functional analysis: an introduction*, volume 66. American Mathematical Soc., 2004.

[10] Jonathan Ephrath, Lars Ruthotto, Eldad Haber, and Eran Treister. LeanResNet: A low-cost yet effective convolutional residual networks. *ICML Workshop on On-Device Machine Learning and Compact Deep Neural Network*, 2019.

[11] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. pages 4851–4860, 10 2019.

[12] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations (ICLR)*, 2016.

	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 6
The two-layers Deep CT and CNN networks applied to the MNIST dataset. l and K denote the patch size and the number of bits, M is the number of convolutional tables 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 7
The four-layers 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 convolutional tables, 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

TABLE 8
The 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 convolutional tables, and D_o is the number of output maps.

[13] M. Hassoun. *Fundamentals of Artificial Neural Networks*. MIT Press, 1995.

[14] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.

[15] Yani Ioannou, Duncan Robertson, Darko Zikic, Peter Kontschieder, Jamie Shotton, Matthew Brown, and Antonio Criminisi. Decision forests, convolutional networks and the models in-between. *CoRR*, abs/1603.01250, 2016.

[16] M Jaderberg, A Vedaldi, and A Zisserman. Speeding up convolutional neural networks with low rank expansions. In *BMVC*, 2014.

[17] He Kaiming, Zhang Xiangyu, Ren Shaoqing, and Sun Jian. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[18] Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulo. Deep neural decision forests. In *ICCV*, 2015.

[19] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *CoRR*, abs/1806.08342, 2018.

[20] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[21] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[22] Alex Krizhevsky, Ilya Sutskever, and Hinton Geoffrey E. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[23] Eyal Krupka, Kfir Karmon, Noam Bloom, Daniel Freedman, Ilya Gurvich, Aviv Hurvitz, Ido Leichter, Yoni Smolin, Yuval Tzairi, Alon Vinnikov, and Aharon Bar Hillel. Toward realistic hands gesture interface: Keeping it simple for developers and machines. In *Computer Human Interaction (CHI)*, 2017.

[24] Eyal Krupka, Alon Vinnikov, Ben Klein, Aharon Bar Hillel, Daniel Freedman, and Simon Stachniak. Discriminative ferns ensemble for hand pose recognition. In *CVPR*, 2014.

[25] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[26] Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. Morgan-Kaufmann, 1990.

[27] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 344–352, 2017.

[28] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *ECCV*, pages 116–131, 2018.

[29] Franck Mamalet and Christophe Garcia. Simplifying convnets for fast learning. In Alessandro E. P. Villa, Włodzisław Duch,

Péter Érdi, Francesco Masulli, and Günther Palm, editors, *Artificial Neural Networks and Machine Learning – ICANN 2012*, pages 58–65, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[30] M. Mathieu, M.Hena, and Y LeCun. Fast training of convolutional networks through ffts. In *International Conference on Learning Representations (ICLR)*, 2014.

[31] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[32] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 525–542, Cham, 2016. Springer International Publishing.

[33] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, pages 4510–4520, 2018.

[34] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[35] Shaoqing Ren, X. Cao, Yichen Wei, and J. Sun. Global refinement of random forest. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 723–730, June 2015.

[36] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *CVPR 2011*, pages 1297–1304, June 2011.

[37] M. Tu, V. Berisha, M. Woolf, J. Seo, and Y. Cao. Ranking the parameters of deep neural networks using the Fisher information. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2647–2651, March 2016.

[38] Peisong Wang, Qinghao Hu, Yifan Zhang, Chunjie Zhang, Yang Liu, and Jian Cheng. Two-step quantization for low-bit neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[39] Bichen Wu, Alvin Wan, Xiangyu Yue, Peter Jin, Sicheng Zhao, Noah Golmant, Amir Gholaminejad, Joseph Gonzalez, and Kurt Keutzer. Shift: A zero flop, zero parameter alternative to spatial convolutions. In *CVPR*, pages 9127–9135, 2018.

[40] Zhang Xiangyu, Zou Jianhua, Hey Kaiming, and Jian Sun. Accelerating very deep convolutional networks for classification and detection. *CoRR*, abs/1505.06798, 2015.

[41] Dong Yi, Zhen Lei, Shengcai Liao, and Stan Z. Li. Learning face representation from scratch. *ArXiv*, abs/1411.7923, 2014.

[42] Junho Yim, Donggyu Joo, Jihoon Bae, and Junmo Kim. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[43] Mansour Yishay. Pessimistic decision tree pruning based on tree size. In *ICML*, 1997.

[44] Olcay Taner Yıldız. VC-dimension of univariate decision trees. *IEEE Transactions On Neural Networks and Learning Systems*, 26, 2015.

[45] Aojun Zhou, Anbang Yao, Kuan Wang, and Yurong Chen. Explicit loss-error-aware quantization for low-bit deep neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[46] Zhi-Hua Zhou and Ji Feng. Deep forest: Towards an alternative to deep neural networks. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.