Low-complexity Approximate Convolutional Neural Networks
Renato J. Cintra, Stefan Duffner, Christophe Garcia, André Leite

To cite this version:
Renato J. Cintra, Stefan Duffner, Christophe Garcia, André Leite. Low-complexity Approximate Convolutional Neural Networks. IEEE Transactions on Neural Networks and Learning Systems, 2018, 10.1109/TNNLS.2018.2815435. hal-01727219

HAL Id: hal-01727219
https://hal.science/hal-01727219
Submitted on 27 Aug 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Low-complexity Approximate Convolutional Neural Networks

Renato J. Cintra, Senior Member, IEEE, Stefan Duffner, Christophe Garcia and André Leite

Abstract—In this paper, we present an approach for minimizing the computational complexity of trained Convolutional Neural Networks (ConvNet). The idea is to approximate all elements of a given ConvNet and replace the original convolutional filters and parameters (pooling and bias coefficients; and activation function) with efficient approximations capable of extreme reductions in computational complexity. Low-complexity convolution filters are obtained through a binary (zero-one) linear programming scheme based on the Frobenius norm over sets of dyadic rationals. The resulting matrices allow for multiplication-free computations requiring only addition and bit-shifting operations. Such low-complexity structures pave the way for low-power, efficient hardware designs. We applied our approach on three use cases of different complexity: (i) a “light” but efficient ConvNet for face detection (with around 1 000 parameters); (ii) another one for handwritten digit classification (with more than 130 000 parameters); and (iii) a significantly larger ConvNet: AlexNet with ≈1.2 million matrices. We evaluated the overall performance on the respective tasks for different levels of approximations. In all considered applications, very low-complexity approximations have been derived maintaining an almost equal classification performance.

Index Terms—Convolutional Neural Networks, Approximation, Optimization, Numerical Computation

I. INTRODUCTION

Since their introduction in the 1990s by LeCun et al. [1], convolutional neural networks (ConvNets) have proven to be very powerful in many challenging computer vision tasks, such as hand-written character recognition [1], 2, embedded text detection and recognition [3], 4, automatic facial analysis [5], 7, traffic sign recognition [8], pedestrian detection [9], vision-based navigation [10], and house numbers recognition [11], just to cite a few. Although state-of-the-art results have been reached in many different fields, ConvNets have become very popular only recently with the impressive results obtained by Krizhevsky et al. [12] in the recognition task, followed by Simonyan and Zisserman who won the localization challenge at the Large-Scale Visual Recognition Challenge (ImageNet) 2014 [13].

The main property of ConvNets is their capability for automatic extraction of complex and application-suitable features from raw data (e.g., pixels in computer vision). To do so, they integrate a pipeline of convolution and pooling layers generally followed by a multi-layer perceptron, jointly performing local feature extraction and classification (or regression) in a single architecture where all parameters are learnt using the classical error back-propagation algorithm [14]. Traditionally, ConvNets are used for image processing tasks and, to this end, they are often applied on small regions of a bigger image in a sliding-window framework, for instance to detect an object. Because of weight sharing, each layer essentially performs convolution or pooling operations using small kernels inside a “retina”, and when applied to a large image, the replication of the ConvNet operation over all positions in the image can be significantly optimized by performing each convolution over the full image at once, efficiently implementing a full image transformation pipeline.

However, during training and when applying the trained ConvNet, there is still a significant number of floating-point computations (multiplications and additions). In order to accelerate these operations, most current approaches and available software rely on parallel computing using GPUs [12], [15] facilitated to some extent by the inherently parallel architectures of ConvNets. The recent trend in “deep learning” to use more and more complex models with millions of parameters requires enormous amounts of computational resources, in particular for training the models but also for applying the learnt ConvNets. Reducing the computational complexity of these models is thus of great interest to the research community as well as to industry. Moreover, such reduction in complexity of ConvNets is necessary to implement them on devices with limited resources (such as mobile devices) in order to operate at acceptable speed, for example for real-time applications, and reduce the overall power consumption.

In this paper, we focus on drastically reducing the computational cost itself by proposing a post-training approximation scheme aiming at replacing all parameters of a ConvNet with low-complexity versions. That is, for the convolution filters, only additions and bit-shifting operations are performed—no multiplication is necessary. Additionally, the activation function is sought to be replaced with low-complexity alternatives. Formulated as an optimization problem, we adopt matrix approximation techniques based on the Frobenius norm error and dyadic rational numbers represented in terms of Canonical Signed Digit (CSD) encoding.

Indeed, a sound approximation theory is a necessary step to facilitated hardware development. This is illustrated in the case of image compression where the most efficient coding schemes are based on approximate matrices realized in dedi-
cated hardware [16]–[19]. We aim at introducing an approach for approximating CNNs in order to pave the way for future efficient dedicated hardware design.

This work is organized as follows. Section II provides a literature review on the efficient numerical implementation of neural networks and, in particular, ConvNets. Section III details our approach for approximating the elements of a given ConvNet aiming at designing low-complexity structures. In Section IV we present the results of our experimental evaluation of the proposed approach for two typical ConvNet architectures. We assess the approximate ConvNets relative to their exact counterparts in terms of several figures of merit. We conclude the paper in Section V.

II. RELATED WORK

Although GPU implementations [20] allow for fast training and application of ConvNets on sufficiently equipped platforms, their integration on embedded systems for real-time applications may be more difficult due to the limited amount of available resources on these devices, which usually requires a good trade-off between performance and code size. Several previous works have tackled this problem. In early works, [21]–[23] weight parameters of neural networks have been represented as power-of-two integers. Thus, all multiplications can be operated as simple bit shifts. Direct training of these networks was also possible by keeping a floating-point version of each weight parameter, or otherwise use a technique called “weight dithering” [24]. Simard and Graf [25] extended this idea by encoding all other parameters as powers of two except the weights, i.e., neuron activations, gradients, and learning rates. Later, Draghici [26] conducted a broader analysis on the computational power of neural networks with reduced precision weights. Recently, Machado et al. [27] proposed a specific approximation scheme for sparse representation learning using only values of powers of two, and they integrated this quantization into the learning process. Finally, in the work of Courbariaux et al. [28], all weights are encoded as binary values (1-bit), and for the training a floating-point version is still needed. Similarly, Kim and Paris [29] proposes a completely binary neural network. However, an initial real-valued training phase is required.

More recently, special attention has been paid to hardware implementations of ConvNets, especially on FPGAs. In [30], for example, a high-level optimization methodology is applied to the implementation of the CFF face detector [6]. They propose algorithmic optimizations and advanced memory management and transform the floating-point computation into fixed-point arithmetic. Interestingly such coarse approximation could furnish very similar detection rates and low false-alarm rates on referenced datasets, for a roughly sevenfold gain of speed. Later, this work has been extended in [31], where the authors present for the first time several implementations of the CFF algorithm on FPGA, with a parallel architecture composed of a Processing Element ring and a FIFO memory, which constitutes a generic architecture capable of processing images of different sizes. Farabet and LeCun [32] also propose a scalable hardware architecture to implement large-scale ConvNets, with a modular vision engine for large image processing, with FPGA and ASIC implementations. Chakradhar et al. [33] present a dynamically configurable FPGA co-processor that adapts to complex ConvNet architectures exploiting different types of parallelism. A very low-complexity ASIC design of ConvNets has been developed by Chen et al. [34], allowing for very high execution speeds and power consumption of state-of-the-art ConvNets. Finally, Zhang et al. [35] propose a FPGA design strategy and algorithmic enhancements to optimize the computational throughput and memory bandwidth for any given ConvNet architecture.

Other recent works have focused on the algorithmic and memory optimizations of large-scale ConvNets. For example, Mamanet et al. [36] proposed different strategies for simplifying the convolutional filters (fusion of convolutional and pooling layers, 1D separable filters), in order to modify the hypothesis space, and to speed-up learning and processing times. These convolutions can also effectively be performed by simple multiplications of the filters with the respective input images in the frequency domain [37]. However, due to the overhead of the FFT, there is only a computational gain with larger filter sizes, and if a given filter can be reused consecutively for many input images. Vanhoucke et al. [38] presented a set of different techniques to accelerate the computation of ConvNets on CPU, mostly for Intel and AMD CPUs, exploiting for example SIMD instructions, memory locality, and fixed-point representations. Also, many recent works [39]–[46] have focused on reducing the complexity of convolution or fully-connected layers of large-scale ConvNets by replacing the high-dimensional matrix or tensor multiplications with several low-rank matrix multiplications using different low-rank factorization methods, either at test-time or both for training and testing. Although, large gains in computational and memory resources can be obtained on complex ConvNets, these optimizations do not focus on hardware implementation and low-power constraints.

As opposed to many previous works that integrate the approximation process into the learning [47]–[51], our approach operates on existing fully-trained models, that originally may have been aimed for standard PCs or more powerful architectures. Thus, our approximation scheme allows to integrate these models into hardware with much fewer resources.

III. APPROXIMATION APPROACH

A. General Goal

Our goal is to derive low-complexity structures capable of reducing the computational costs of a given ConvNet. Ideally, the following two conditions are simultaneously expected to be satisfied:

(i) the computational elements of the ConvNet (convolutional filters, sub-sampling coefficients, bias values, and sigmoid function calls) are replaced by corresponding low-complexity structures;

(ii) the performance of the ConvNet is not significantly degraded.

However, addressing both above conditions proves to be a hard task. In particular, the large number of variables, the
non-linearities, and extremely long simulation times prevents such approach. Also, to the best of our knowledge, literature furnishes no mathematical result linking the approximation of individual ConvNet elements and the final ConvNet performance. Thus, we adopt a greedy-like heuristic which consists in individually simplifying each computational structure of a ConvNet in the hope of finding a resulting structure capable of good performance [52].

In a ConvNet two main types of mathematical elements are found: (i) matrix structures and (ii) activation functions. The matrix structures are represented by convolution filter weights, sub-sampling operations, and bias values; whereas the activation function is usually a non-linear function, such as the threshold, piecewise-linear, and sigmoid functions [53].

To approximate these two classes of elements, different tools are required. For the matrix-based structures, we selected matrix approximation methods as a venue to derive low-complexity computational elements [54]–[57]. For the activation, we separate methods capable of approximating functions with efficient digital implementation [58]–[60].

B. Low-complexity Matrix Structures

In [54], [57], [61]–[64], several methods for deriving approximations of discrete transform matrices—such as the discrete cosine transform [65]—were proposed. Let \( M \) be an \( N \times N \) given matrix. For instance, \( M \) can be a convolutional filter. In this case, a computational instantiation of \( M \) applied to evaluate a single output pixel requires in principle \( N^2 \) floating point multiplications. A typical ConvNet may contain thousands of convolutional filters. For example, the classical architecture described in [12] contains 244,760 filters, which is nowadays considered a relatively small network. Therefore, to minimize such a significant computational cost, we aim at obtaining a low-complexity matrix \( \hat{M} \) capable of satisfying the following relation in an optimal sense: \( M \approx \hat{M} \). The matrix \( \hat{M} \) is said to be an approximation for \( M \). Such approximate convolutional filters would allow the realization of computationally intensive ConvNets in limited resources architectures.

In this paper, a low-complexity matrix is a matrix of dyadic rational entries. Dyadic rational numbers are fractions of the form \( m/2^n \), where \( n \) is a positive integer and \( m \) is an odd integer. Such numbers are suitable for binary arithmetic. Indeed, a multiplication by a dyadic rational consists of a multiplication by \( m \) followed by a right shift of \( n \) bits. Because \( m \) is an integer, we can take full advantage of fixed-point arithmetic. Indeed \( m \) can be given a binary representation with minimum number of adders, aiming at multiplicative irreducibility. Multiplicative irreducibility is attained whenever the minimum number of additions to implement a multiplication by \( m \) is equal to the number of ones in the binary representation of \( m \) [64]. Multiplicative irreducibility is often obtained when the CSD representation is considered [66]. Therefore, a multiplication by \( m \) can be converted into a sequence of additions and bit-shifting operations. As a consequence, low-complexity matrices are multiplierless, a very desirable property as floating-point operations are much more costly than additions and bit-shifting operations.

Standard methods for matrix approximation include: inspection [67], matrix parametrization [61], and matrix factorization [68]. However, since a typical ConvNet may contain from thousands to millions of filters, inspection-based approaches are not feasible. Methods based on the parametrization of the matrix elements are also ineffective because (i) the elements of convolutional filters are usually not clearly related, i.e., they do not satisfy identifiable mathematical relationships and (ii) the elements are not repeated. Additionally, ConvNet filters are not expected to satisfy properties, such as symmetry and orthogonality, which favors the derivation of approximations. Thus, methods based on matrix factorizations are less adequate.

C. Matrix Approximation by Linear Programming

We adopted a general approach to the problem of obtaining \( \hat{M} \) according to an optimization problem as described below:

\[
\hat{M} = \arg \min_{T} \|M - \alpha \cdot T\|^2. \tag{1}
\]

The above optimization problem can yield better approximate matrices if an expansion factor \( \alpha \) is introduced [63]. By adopting the usual Frobenius norm [50] as an error measure, (1) can be recast according to the following mixed integer nonlinear programming (INLP) setup [69]:

\[
(\alpha^*, T^*) = \arg \min_{\alpha, T} \|M - \alpha \cdot T\|^2, \tag{2}
\]

where \( \alpha > 0 \) is the real-valued expansion factor and \( \| \cdot \| \) is the Frobenius norm [50]. The choice of the Frobenius norm is justified by the following argument. An approximate CNN must have its elements numerically ‘close’ to elements from the exact CNN. Therefore, a measure that takes into consideration distance in a energy-based manner (euclidean distance sense) emerges naturally as a means to guarantee that the approximate filtering structures (e.g., convolution kernels) are close to the exact counterpart. The Frobenius norm satisfies the above rationale. This analysis is confirmed in [39].

To ensure that the candidate matrices \( T \) have low complexity, we limited the search space of the above problem to the matrices whose elements are defined over a sets of dyadic rationals \( \mathcal{D} \). Some particular sets are [57], [67]:

\[
\mathcal{D}_1 = \{-1, 0, 1\}, \\
\mathcal{D}_2 = \{-2, -1, 0, 1, 2\}, \\
\mathcal{D}_3 = \{-4, -3, -2, -1, 0, 1, 2, 3, 4\}, \\
\mathcal{D}_4 = \{-4, -3, -2, -1, \frac{3}{4}, \frac{1}{2}, \frac{1}{4}, 0, \frac{1}{2}, \frac{3}{4}, 1, 2, 3, 4\}, \\
\mathcal{D}_5 = \{-7, -6, -5, -4, -3, -2, -1, -\frac{3}{4}, -\frac{1}{2}, -\frac{1}{4}, 0, \\
\frac{1}{2}, \frac{3}{4}, 1, 2, 3, 4, 5, 6, 7\}, \\
\mathcal{D}_6 = \{-4, -\frac{15}{4}, -\frac{7}{2}, -\frac{13}{4}, \ldots, \frac{13}{4}, \frac{7}{2}, \frac{15}{4}\}, \\
\mathcal{D}_7 = \{-5, -\frac{19}{4}, -\frac{9}{2}, -\frac{17}{4}, \ldots, \frac{17}{4}, \frac{9}{2}, \frac{19}{4}\}, \\
\mathcal{D}_8 = \{-7, -\frac{27}{4}, -\frac{13}{2}, -\frac{25}{4}, \ldots, \frac{25}{4}, \frac{13}{2}, \frac{27}{4}\}.
\]
Sets $D_6$, $P_9$, and $D_9$ possess uniformly spaced rationals.

A straightforward way of addressing (2) is as follows. Considering a given set of dyadic rationals $D$, for each element of $\alpha \cdot T$, we simply find the closest neighbour of such element in $D$. Such approach can be efficiently implemented by means of binary search. However, this approach is only effective as long as (2) remains an unconstrained optimization problem. Alternatively, we can consider a more flexible approach based on integer linear programming (ILP).

For fixed values of $\alpha$, the mixed INLP problem posed in (2) can be efficiently solved by means of binary (zero-one) linear programming. In other words, we aim at converting a nonlinear problem into a linear one. Indeed, let $m_{ij}$, $i, j = 1, 2, \ldots, N$, denote the entries of $M$ and $r \in D$ be a dyadic rational. We adopt the following binary decision variables:

$$x_{i,j}(r) = \begin{cases} 1, & \text{if } t_{ij} = r, \\ 0, & \text{otherwise.} \end{cases}$$

For binary (zero-one) variables we have $y^2 = y$, where $y$ is a dummy variable. This fact paves the way for the linearization of the above-mentioned optimization problem. Therefore, (2) can be re-written according to the following binary linear programming problem [70]–[72]:

$$\min_{x_{i,j}(r)} \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{r \in D} (m_{ij} - \alpha \cdot r)^2 \cdot x_{i,j}(r),$$

subject to

$$\sum_{r \in D} x_{i,j}(r) = 1, \quad i, j = 1, 2, \ldots, N.$$  

The above constraint is to ensure that each element $m_{ij}$ is approximated by a unique dyadic rational in $D$. The solution of the above problem is denoted as $x_{i,j}^{(\alpha)}$, $i, j = 1, 2, \ldots, N$, $r \in D$, being linked to the choice of $\alpha$. Such binary (zero-one) solution can be employed to compute the actual entries $t_{ij}^{(\alpha)}$ of the low-complexity matrix associated to the considered $\alpha$ according to:

$$t_{ij}^{(\alpha)} = \sum_{r \in D} r \cdot x_{i,j}^{(\alpha)}(r).$$

The resulting low-complexity matrix is denoted by $T_\alpha$. The approximation error is implied by (2) and can be computed according to:

$$\text{Error}(\alpha) = \| M - \alpha \cdot T_\alpha \|^2.$$  

Because a sequence of values for $\alpha$ is selected, the above problem is solved for each instantiation; furnishing the sequence of errors indexed by $\alpha$: Error$(\alpha)$. Being a linear programming problem, each instantiation can be solved efficiently and very quickly by contemporary computational packages [73], [74]. State-of-the-art solvers can obtain solutions for ILP problems at an average computation complexity in $O(N)$ [75] or $O(N \log N)$ [70]–[72]. Finally, we determine the global optimum value $\alpha^*$ according to:

$$\alpha^* = \arg \min_{\alpha} \text{Error}(\alpha),$$

which can be solved by simple minimization over a vector of values. Associated to $\alpha^*$, we also obtain $T^* = T_{\alpha^*}$, which is the global optimal low-complexity matrix. Therefore, the sought approximation $M$ is given by:

$$M = \alpha^* \cdot T^*.$$  

The above ILP approach allows the user to easily include constraints to the optimization problem. This is relevant for further investigation in this topic; in particular when specific mathematical properties are expected to be enforced on the resulting low-complexity matrices (for instance, 2D filter normalization [76], p. 115).

We emphasize that the solving method for (2) is only required to be efficient enough to cope with the time constraints at the design phase of the approximate neural network. In other words, solvers available in contemporary optimization packages are suitable; and the choice of the particular method for solving (2) is not a critical for our approach. Additionally, we note that the optimization solver is simply a step for obtaining the final neural network. Once the approximate structures are found, optimization solver are clearly not required anymore.

D. Example

To illustrate the procedure, we selected $D_9$ and considered the search space of the expansion factor to be the interval $[0.25, 1]$ with a step of $10^{-3}$. Additionally, we consider the following particular convolutional filter employed in the ConvNet described in Section IV-B:

$$M_0 = \begin{bmatrix} 1.5200701 & 1.0317051 & 0.7906240 & -0.2153791 & -0.2340538 \\ 1.3982610 & 2.1860176 & 2.0152923 & 1.5620477 & 0.8270900 \\ -0.6848867 & 0.7470516 & 1.6923728 & 1.2537112 & 1.1946758 \\ -1.2387477 & -0.5483563 & 0.1261987 & 0.8677999 & 0.7742613 \\ -1.4691808 & -1.2178997 & -0.2924347 & 0.2172496 & 0.1325074 \end{bmatrix}. $$

Solving (2) for the above matrix, we obtain:

$$\alpha^* = 0.30931,$$

$$T^* = \begin{bmatrix} 5 & 3.25 & 2.5 & -0.75 & -0.75 \\ 4.5 & 7 & 6.5 & 5 & 2.75 \\ -2.25 & 2.5 & 5.5 & 4 & 3.75 \\ -4 & -1.75 & 0.5 & 2.75 & 2.5 \\ -4.75 & -4 & -1 & 0.75 & 0.5 \end{bmatrix},$$

$$= \frac{1}{4} \begin{bmatrix} 20 & 13 & 10 & -3 & -3 \\ 18 & 28 & 26 & 20 & 11 \\ -9 & 10 & 22 & 16 & 15 \\ -16 & -7 & 2 & 11 & 10 \\ -19 & -16 & -4 & 3 & 2 \end{bmatrix}. $$

Fig. 1(a) depicts the Frobenius norm error for varying values of $\alpha$ (cf. 2). For very small $\alpha$, the values of $\alpha \cdot M$ are close to zero. So the discrete entries of the candidate matrices $T$ are unable to provide a good approximation. As $\alpha$ increases, a similar effect happens. However, for intermediate values the minimum can be found. Fig. 1(b) shows details in the vicinity of the optimum. The curves shown in Fig. 1 are piecewise concatenations of parabolae. This is due to the quadratic nature of the coefficients $(r - \alpha \cdot m_{ij})^2$ of the linear programming problem in (3). Each parabola is linked to a particular approximate candidate $T$. 

\[ \]
Notice that the low-complexity matrix \( T^* \) is expressed in terms of small integers, which can be given simple binary expansions (e.g., \( 22 = 2^5 - 2^3 - 2^2 \)). Similarly, we have that \( \alpha^* = 0.30931 \approx 2^{-2} + 2^{-4} - 2^{-8} = 0.30859375 \).

Therefore, considering \( \phi \), the actual fully multiplierless approximation is furnished by:

\[
\hat{M} = (2^{-4} + 2^{-6} - 2^{-10}) \cdot \begin{bmatrix}
20 & 13 & 10 & -3 & -3 \\
18 & 28 & 26 & 20 & 11 \\
-9 & 10 & 22 & 16 & 15 \\
-16 & -7 & 2 & 11 & 10 \\
-19 & -16 & -4 & 3 & 2
\end{bmatrix}.
\]

E. Activation Function Approximations

Although there are several types of activation functions, we focus our analyses on the continuous tanh-sigmoid function, which is defined according to the hyperbolic tangent function \([53]\). As indicated in \([53]\), \([77]\), the mathematical expression for the tanh-sigmoid function is given by:

\[
\phi(x) = a \cdot \tanh(b \cdot x),
\]

where \( a = 1.7159 \) and \( b = 2/3 \). This particular activation function has been originally proposed by LeCun \([78]\) and adopted in several working models as the Convolutional Face Finder (CFF) \([6]\).

In \([58]\), \([60]\), \([79]\)–\([82]\), approximations for the related sigmoid function given by \( y = 1/(1 + e^{-x}) \) were examined, including the Alippi and Storti-Gajani (ASG) approximation \([83]\), the piecewise linear approximation of a non-linear function (PLAN) \([84]\), and simple linear \([58]\) and quadratic \([79]\) approximations. Based on these approximations, we derived expressions for the tanh-sigmoid approximations as shown in Table I. We adopted an 8-bit representation for \( a \) resulting in the following approximate value: \( \hat{a} = 7/4 \).

F. Complexity

As a consequence of the above approximations, we have substantial savings in computation costs. Indeed, a single call of the original \( N \times N \) matrix \( M \) requires \( N^2 \) multiplications of floating-point entries per pixel. On the other hand, the proposed approximation \( \hat{M} \) contains only small integers that can be very efficiently encoded with minimal number of adders \([66]\). Similarly, the expansion factor \( \alpha^* \) can be given a truncated rational approximation in the form of dyadic rationals. The same rationale also applies to the remaining computational structures of the original ConvNet. Thus, the final resulting structure is fully multiplierless—only additions and bit-shifting operations are required. In terms of hardware realization, the number of arithmetic operations translate into chip area and power consumption \([55]\), \([56]\). Thus, in limited resource scenarios (e.g., embedded systems and wireless sensors), approximations may provide an effective way of porting large ConvNets into physical realization.

To summarize, the proposed approximation approach consists of:

(i) finding approximate convolutional filter by solving (1) for each exact convolutional filter from a given ConvNet;
(ii) converting scaling factors, sub-sampling coefficients, and bias values into CSD representation aiming at the minimization of computation costs and multiplicative irreducibility;
(iii) approximating the activation function to a simple function.

IV. EXPERIMENTS

We studied the effectiveness of the proposed approximation approach on two classical computer vision problems: (i) a binary and (ii) a multi-class classification problem. The first application is face detection, where the ConvNet classifies image regions as face or non-face. The second one is handwritten digit recognition, where the trained model is used to classify a given image patch into one of the ten digits “0” to “9”. For each of the two applications, we trained a ConvNet in a classical way and evaluated its performance in terms of precision and recall, for the given application.

The first ConvNet is relatively small, whereas the second model (for digit recognition) contains much more parameters. We aim at demonstrating that our proposed approach is able to effectively process larger networks.

After approximating the parameters of the models, we compared their performance with their respective original, exact versions. Note that we do not aim at improving the state-of-the-art in face detection or hand-written digit recognition. Indeed, current literature presents concrete example of
approximating its parameters while maintaining a very similar
given trained ConvNet model can be reduced significantly by
digit recognition [88]. Our goal is to demonstrate—based
for face detection [87] and huge ensemble classifiers for
complex models, such as multi-view or part-based detectors
for face detection [87] and huge ensemble classifiers for
digit recognition [88]. Our goal is to demonstrate—based
on common representative models—that the complexity of
a given trained ConvNet model can be reduced significantly by
approximating its parameters while maintaining a very similar
performance.

Hereafter an approximate network based on dyadic set \( \mathcal{D}_i \)
is referred to as \( A_i \).

### A. Binary Classification

Our first set of experiments employs a ConvNet that was
trained for face detection in grey-scale images. Thus, such
network is a binary classifier that decides whether the given
input image is a face or not. As a working model, we selected
the classical face detector called Convolutional Face Finder
(CFF) proposed by Garcia and Delakis [6]. This model is a
relatively “light” ConvNet with an input size of \( 32 \times 36 \) and six
layers: four layers alternating convolution and average pooling
operations, with 4, 4, 14, and 14 maps, respectively, followed
by 14 neurons and one single final output neuron. The first
convolution layer contains four filters of size \( 5 \times 5 \), the second
one contains 20 filters of size \( 3 \times 3 \), and the 14 neurons
of the first neuron layer are treated as convolutions of size \( 6 \times 7 \),
each neuron being connected to only one map. Pooling
maps contain a single coefficient, and all maps and neurons
have an additional bias. The entire ConvNet has 951 trainable
parameters in total. A thorough description of this particular
ConvNet is supplied in [6]. The employed activation function
is the exact continuous tanh-sigmoidal function as detailed in [7].

After training the ConvNet as described in [6], we approx-
imated all the convolution filter matrices with low-complexity
versions. We created several approximations using the different
sets of dyadic rationals described in the previous section:
\( \mathcal{D}_1, \mathcal{D}_4, \ldots, \mathcal{D}_8 \). We also replaced all average pooling coeffi-

cients and bias terms with their closest CSD representation
using 8 bits, being 7 bits for the fractional part.

Table I lists the arithmetic costs of the exact ConvNet
model compared to its approximations. Floating-point multiplica-
tions, direct additions, additions due to the CSD expansion,
and bit-shifting operations were counted. The exact structure
requires both floating-point multiplications and additions. In
contrast, the approximate methods completely eliminates the
need for multiplications at the expense of much simpler
operations: additions and bit-shifting operations. Because the
approximate quantities can be easily represented in fixed-
point arithmetic representation, it is suitable for hardware
implementation. Additionally, the hardware implementation of
bit-shifting operations require virtually no cost, because it can
be implemented by simple physical wiring. As a result, we
have a very favorable trade-off: multiplications are exchanged
for additions.

In order to analyse the effect of the approximation on
the actual performance of the ConvNet, we evaluated the
different versions on three standard face detection benchmarks:
FDDB [89] (2845 images), AFW [90] (205 images), and

<table>
<thead>
<tr>
<th>( \mathcal{D}_i )</th>
<th>( \sigma_1(x) = \hat{a} \cdot \begin{cases} -1 &amp; x &lt; 0, \ 1 &amp; x \geq 0. \end{cases} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear I</td>
<td>( \sigma_3(x) = \hat{a} \cdot \begin{cases} -1 &amp; x &lt; -4, \ \frac{x}{4} &amp; -4 \leq x &lt; 4, \ 1 &amp; x \geq 4. \end{cases} )</td>
</tr>
<tr>
<td>Quadratic I</td>
<td>( \sigma_6(x) = \hat{a} \cdot \begin{cases} -1 &amp; x &lt; -4, \ \left(\frac{x}{4} + 1\right)^2 - 1 &amp; -4 \leq x &lt; 0, \ 1 - \left(\frac{x}{4} + 1\right)^2 &amp; 0 \leq x &lt; 4, \ 1 &amp; x \geq 4. \end{cases} )</td>
</tr>
</tbody>
</table>

| Plan-based | \( \sigma_2(x) = \hat{a} \cdot \begin{cases} -1 & x < -5, \\ \frac{-19}{8} & -5 \leq x < -\frac{19}{8}, \\ \frac{x}{16} - \frac{89}{128} & -\frac{19}{8} \leq x < -\frac{1}{4}, \\ -\frac{1}{4} & -\frac{1}{4} \leq x < 1, \\ \frac{x}{4} + \frac{1}{4} & 1 \leq x < \frac{19}{8}, \\ \frac{-19}{16} & \frac{19}{8} \leq x < 5, \\ 1 & x \geq 5. \end{cases} \) |

| Quadratic II | \( \sigma_6(x) = \hat{a} \cdot \begin{cases} -1 & x < -2, \\ \frac{x}{2} & -2 \leq x < 2, \\ 1 & x \geq 2. \end{cases} \) |

| Linear II | \( \sigma_4(x) = \hat{a} \cdot \begin{cases} -1 & x < -2, \\ \frac{x}{2} & -2 \leq x < 2, \\ 1 & x \geq 2. \end{cases} \) |

<table>
<thead>
<tr>
<th>( \mathcal{D}_i )</th>
<th>( \sigma_1(x) = \hat{a} \cdot \begin{cases} -1 &amp; x &lt; 0, \ 1 &amp; x \geq 0. \end{cases} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear I</td>
<td>( \sigma_3(x) = \hat{a} \cdot \begin{cases} -1 &amp; x &lt; -4, \ \frac{x}{4} &amp; -4 \leq x &lt; 4, \ 1 &amp; x \geq 4. \end{cases} )</td>
</tr>
<tr>
<td>Quadratic I</td>
<td>( \sigma_6(x) = \hat{a} \cdot \begin{cases} -1 &amp; x &lt; -4, \ \left(\frac{x}{4} + 1\right)^2 - 1 &amp; -4 \leq x &lt; 0, \ 1 - \left(\frac{x}{4} + 1\right)^2 &amp; 0 \leq x &lt; 4, \ 1 &amp; x \geq 4. \end{cases} )</td>
</tr>
</tbody>
</table>

| Plan-based | \( \sigma_2(x) = \hat{a} \cdot \begin{cases} -1 & x < -5, \\ \frac{-19}{8} & -5 \leq x < -\frac{19}{8}, \\ \frac{x}{16} - \frac{89}{128} & -\frac{19}{8} \leq x < -\frac{1}{4}, \\ -\frac{1}{4} & -\frac{1}{4} \leq x < 1, \\ \frac{x}{4} + \frac{1}{4} & 1 \leq x < \frac{19}{8}, \\ \frac{-19}{16} & \frac{19}{8} \leq x < 5, \end{cases} \) |

| Quadratic II | \( \sigma_6(x) = \hat{a} \cdot \begin{cases} -1 & x < -2, \\ \frac{x}{2} & -2 \leq x < 2, \\ 1 & x \geq 2. \end{cases} \) |
Pascal Faces dataset [91] (851 images); and we used the improved annotation and evaluation protocol proposed by Mathias et al. [87].

Tables III-V show the average precision rates for different combinations of sigmoid and weight matrix approximations relative to the exact model for the three datasets.

Overall, the “Linear II” sigmoid approximation provides the best results, followed by “ASG”, “Quadratic I”, and “PLAN”. In terms of weight matrix approximations, $A_1$–$A_5$ generally give unsatisfactory results, and $A_7$ performs best. Also, it is interesting to note that the finer approximation $A_8$ gave worse results than $A_7$. We further evaluated some variants, where different layers of the ConvNet have been approximated with different sets of dyadic rationals. For such mixed approximate structures, we have denoted them by $A_{i,j,k,l}$, where the subscripts indicate the selected dyadic set for each layer. In other words, indices $i, j$ indicate that the dyadic sets $D_i$ and $D_j$, respectively, are employed in the first two convolution layers; and similarly, indices $k, l$ correspond to the adoption of the dyadic sets $D_k$ and $D_l$, respectively, for the two final fully-connected layers. We found that the first layer requires a finer approximation than the other layers. This allowed us to maintain a good performance with very low-complexity approximations (e.g., $A_3, A_4$) for these later layers. This can be explained according to the following: (i) by its own very nature, the layers have different degrees of importance; (ii) errors in initial layers tend to propagate through the succeeding layers; and (iii) error propagation can potentially be amplified along the layers. Fig. 2 shows the receiver operating characteristic (ROC) curves of the best performing approximations for the three datasets.

These results are quite impressive given the fact that we considerably reduced the precision of each parameter of the ConvNet, and given the highly non-linear classification problem where the frontier between the face and non-face classes can be very thin and complex.

Fig. 3 shows some face detection results from the exact model (top) and the approximation $A_{7,3,3,3}$ (bottom), i.e. a finer approximation for the first layer and a very coarse one for the rest of the layers. The results are almost identical.

**B. Multi-class Classification**

We studied a second case where a ConvNet has been trained for a classical multi-class classification problem: the MNIST hand-written digit recognition dataset [92]. To show that the proposed approximations can also be applied to larger networks we trained a ConvNet with a different architecture containing again six layers but much more maps and around 180 000 parameters and more than 5 300 matrices in total. The input is a $32 \times 32$ grey-scale image, and the network is composed of five convolution maps ($5 \times 5$ kernels) followed by five average pooling maps (connected one-to-one), 50 convolution maps ($3 \times 3$, fully connected), 50 average pooling maps (connected one-to-one), 100 neurons ($6 \times 6$ matrices, fully connected), and the 10 final output neurons corresponding to the 10 digits to classify.

After having trained this ConvNet model on the MNIST dataset, we approximated all the convolution filters, the fully-
### TABLE III: Average precision for CFF with the FDDB test set and different approximations relative to the exact model

<table>
<thead>
<tr>
<th></th>
<th>Exact</th>
<th>ASG</th>
<th>PLAN</th>
<th>Linear I</th>
<th>Linear II</th>
<th>Quadratic I</th>
<th>Quadratic II</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact</td>
<td>1.000</td>
<td>0.953</td>
<td>0.894</td>
<td>0.002</td>
<td>0.988</td>
<td>0.887</td>
<td>0.938</td>
</tr>
<tr>
<td>A1</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>A2</td>
<td>0.001</td>
<td>0.001</td>
<td>0.000</td>
<td>0.000</td>
<td>0.001</td>
<td>0.000</td>
<td>0.002</td>
</tr>
<tr>
<td>A3</td>
<td>0.549</td>
<td>0.311</td>
<td>0.432</td>
<td>0.005</td>
<td>0.510</td>
<td>0.426</td>
<td>0.180</td>
</tr>
<tr>
<td>A4</td>
<td>0.523</td>
<td>0.602</td>
<td>0.365</td>
<td>0.001</td>
<td>0.558</td>
<td>0.383</td>
<td>0.602</td>
</tr>
<tr>
<td>A5</td>
<td>0.490</td>
<td>0.098</td>
<td>0.517</td>
<td>0.000</td>
<td>0.657</td>
<td>0.510</td>
<td>0.073</td>
</tr>
<tr>
<td>A6</td>
<td>0.938</td>
<td>0.914</td>
<td>0.817</td>
<td>0.002</td>
<td>0.888</td>
<td>0.797</td>
<td>0.949</td>
</tr>
<tr>
<td>A7</td>
<td>0.960</td>
<td>0.943</td>
<td>0.847</td>
<td>0.002</td>
<td>0.963</td>
<td>0.848</td>
<td>0.935</td>
</tr>
<tr>
<td>A8</td>
<td>0.821</td>
<td>0.792</td>
<td>0.648</td>
<td>0.001</td>
<td>0.810</td>
<td>0.650</td>
<td>0.793</td>
</tr>
<tr>
<td>A7,3,3,3</td>
<td>0.917</td>
<td>0.902</td>
<td>0.886</td>
<td>0.003</td>
<td>0.947</td>
<td>0.888</td>
<td>0.851</td>
</tr>
<tr>
<td>A7,4,4,4</td>
<td>0.967</td>
<td>0.931</td>
<td>0.855</td>
<td>0.000</td>
<td>0.976</td>
<td>0.861</td>
<td>0.928</td>
</tr>
<tr>
<td>A7,6,6,6</td>
<td>0.959</td>
<td>0.907</td>
<td>0.862</td>
<td>0.004</td>
<td>0.959</td>
<td>0.863</td>
<td>0.933</td>
</tr>
</tbody>
</table>

### TABLE IV: Average precision for CFF with the AFW test set and different approximations relative to the exact model

<table>
<thead>
<tr>
<th></th>
<th>Exact</th>
<th>ASG</th>
<th>PLAN</th>
<th>Linear I</th>
<th>Linear II</th>
<th>Quadratic I</th>
<th>Quadratic II</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact</td>
<td>1.000</td>
<td>0.839</td>
<td>0.829</td>
<td>0.000</td>
<td>1.041</td>
<td>0.797</td>
<td>0.383</td>
</tr>
<tr>
<td>A1</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>A2</td>
<td>0.001</td>
<td>0.001</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>A3</td>
<td>0.220</td>
<td>0.260</td>
<td>0.250</td>
<td>0.004</td>
<td>0.199</td>
<td>0.248</td>
<td>0.014</td>
</tr>
<tr>
<td>A4</td>
<td>0.422</td>
<td>0.356</td>
<td>0.261</td>
<td>0.000</td>
<td>0.457</td>
<td>0.336</td>
<td>0.275</td>
</tr>
<tr>
<td>A5</td>
<td>0.220</td>
<td>0.015</td>
<td>0.314</td>
<td>0.000</td>
<td>0.217</td>
<td>0.251</td>
<td>0.002</td>
</tr>
<tr>
<td>A6</td>
<td>0.864</td>
<td>0.794</td>
<td>0.715</td>
<td>0.000</td>
<td>0.893</td>
<td>0.694</td>
<td>0.700</td>
</tr>
<tr>
<td>A7</td>
<td>0.978</td>
<td>0.844</td>
<td>0.753</td>
<td>0.000</td>
<td>0.985</td>
<td>0.755</td>
<td>0.614</td>
</tr>
<tr>
<td>A8</td>
<td>0.698</td>
<td>0.553</td>
<td>0.576</td>
<td>0.000</td>
<td>0.565</td>
<td>0.544</td>
<td>0.169</td>
</tr>
<tr>
<td>A7,3,3,3</td>
<td>0.955</td>
<td>0.525</td>
<td>0.835</td>
<td>0.004</td>
<td>0.914</td>
<td>0.816</td>
<td>0.184</td>
</tr>
<tr>
<td>A7,4,4,4</td>
<td>1.020</td>
<td>0.827</td>
<td>0.755</td>
<td>0.000</td>
<td>0.954</td>
<td>0.761</td>
<td>0.576</td>
</tr>
<tr>
<td>A7,6,6,6</td>
<td>0.967</td>
<td>0.881</td>
<td>0.787</td>
<td>0.000</td>
<td>0.979</td>
<td>0.788</td>
<td>0.827</td>
</tr>
</tbody>
</table>

### TABLE V: Average precision for CFF with the “Pascal faces” test set and different approximations relative to the exact model

<table>
<thead>
<tr>
<th></th>
<th>Exact</th>
<th>ASG</th>
<th>PLAN</th>
<th>Linear I</th>
<th>Linear II</th>
<th>Quadratic I</th>
<th>Quadratic II</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact</td>
<td>1.000</td>
<td>0.933</td>
<td>0.774</td>
<td>0.001</td>
<td>1.039</td>
<td>0.747</td>
<td>0.893</td>
</tr>
<tr>
<td>A1</td>
<td>0.001</td>
<td>0.001</td>
<td>0.000</td>
<td>0.000</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>A2</td>
<td>0.006</td>
<td>0.004</td>
<td>0.000</td>
<td>0.000</td>
<td>0.006</td>
<td>0.003</td>
<td>0.009</td>
</tr>
<tr>
<td>A3</td>
<td>0.234</td>
<td>0.128</td>
<td>0.239</td>
<td>0.000</td>
<td>0.234</td>
<td>0.230</td>
<td>0.098</td>
</tr>
<tr>
<td>A4</td>
<td>0.357</td>
<td>0.375</td>
<td>0.259</td>
<td>0.001</td>
<td>0.361</td>
<td>0.258</td>
<td>0.370</td>
</tr>
<tr>
<td>A5</td>
<td>0.433</td>
<td>0.143</td>
<td>0.386</td>
<td>0.000</td>
<td>0.449</td>
<td>0.345</td>
<td>0.101</td>
</tr>
<tr>
<td>A6</td>
<td>0.862</td>
<td>0.844</td>
<td>0.656</td>
<td>0.000</td>
<td>0.894</td>
<td>0.638</td>
<td>0.847</td>
</tr>
<tr>
<td>A7</td>
<td>0.917</td>
<td>0.879</td>
<td>0.707</td>
<td>0.001</td>
<td>0.971</td>
<td>0.702</td>
<td>0.889</td>
</tr>
<tr>
<td>A8</td>
<td>0.725</td>
<td>0.700</td>
<td>0.520</td>
<td>0.001</td>
<td>0.697</td>
<td>0.517</td>
<td>0.617</td>
</tr>
<tr>
<td>A7,3,3,3</td>
<td>0.899</td>
<td>0.752</td>
<td>0.755</td>
<td>0.004</td>
<td>0.899</td>
<td>0.752</td>
<td>0.645</td>
</tr>
<tr>
<td>A7,4,4,4</td>
<td>0.930</td>
<td>0.889</td>
<td>0.710</td>
<td>0.000</td>
<td>0.967</td>
<td>0.708</td>
<td>0.906</td>
</tr>
<tr>
<td>A7,6,6,6</td>
<td>0.931</td>
<td>0.906</td>
<td>0.725</td>
<td>0.004</td>
<td>0.970</td>
<td>0.722</td>
<td>0.916</td>
</tr>
</tbody>
</table>
Fig. 3: Some CFF face detection results on the AFW dataset. Top: exact model; bottom: approximation $A_{7,3,3}$. Despite the very coarse approximation, the results are very close. In the second last image, the approximate model even detects an additional face, missed by the original CFF. However, in the last example a false detection is produced.

**TABLE VI: Arithmetic cost for MNIST-based models**

<table>
<thead>
<tr>
<th>Model</th>
<th>Operation</th>
<th>Mult.</th>
<th>Add.</th>
<th>CSD Add.</th>
<th>Bit-shifting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact</td>
<td></td>
<td>183375</td>
<td>178110</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$A_1$</td>
<td></td>
<td>0</td>
<td>178110</td>
<td>12740</td>
<td>23325</td>
</tr>
<tr>
<td>$A_2$</td>
<td></td>
<td>0</td>
<td>178110</td>
<td>12722</td>
<td>23307</td>
</tr>
<tr>
<td>$A_3$</td>
<td></td>
<td>0</td>
<td>178110</td>
<td>49127</td>
<td>59712</td>
</tr>
<tr>
<td>$A_4$</td>
<td></td>
<td>0</td>
<td>178110</td>
<td>61228</td>
<td>71813</td>
</tr>
<tr>
<td>$A_5$</td>
<td></td>
<td>0</td>
<td>178110</td>
<td>65211</td>
<td>75796</td>
</tr>
<tr>
<td>$A_6$</td>
<td></td>
<td>0</td>
<td>178110</td>
<td>141401</td>
<td>151986</td>
</tr>
<tr>
<td>$A_7$</td>
<td></td>
<td>0</td>
<td>178110</td>
<td>188417</td>
<td>199002</td>
</tr>
<tr>
<td>$A_8$</td>
<td></td>
<td>0</td>
<td>178110</td>
<td>158595</td>
<td>169180</td>
</tr>
</tbody>
</table>

connected layer matrices and the activation functions which is based on the tanh-sigmoid function. Again, all pooling coefficients and bias terms were replaced by their closest CSD representation using 8 bits. The computation cost of the exact and approximate structures is shown in Table VI. Similarly to the previous experiment, the approximate models have totally eliminated the multiplicative costs. Floating-point arithmetic is not required; being fixed-point arithmetic adequate. The cost of the extra additions due to the CSD representation is very low compared to the multiplicative cost required by the exact model. The cost of bit-shifting operations is negligible.

Table VII shows the relative classification rates on the MNIST test set for the different approximations, and Fig. 4 depicts the respective ROC curves of the best-performing approximations (combined for the 10 classes).

The results show that the approximations, although very coarse, have a very small effect on the overall performance of the ConvNet. In a multi-class setting, the trained ConvNet, at least in this particular case, is much more robust to the loss in precision of the weights induced by our approximation scheme compared to the binary classifier. For example, a very coarse approximation like $A_{3,3,1,1}$ leads to a relative performance decrease of less than 1%.

C. Large-Scale Deep Neural Networks

Finally, we applied our approximation approach to a deeper and more complex network architecture, the well-known AlexNet proposed by Krizhevsky et al. [12], and the ImageNet dataset [13] for image classification with 1000 classes. This model contains more than $1.2 \times 10^8$ million matrices and 5096 vectors. We approximated all convolution filter matrices of the fully-trained 8-layer ConvNet using two different sets of dyadic rationals for different layers, a very coarse set $D_9$ and a slightly finer set $D_{10}$:

$$D_9 = \left\{-2, -1, -\frac{1}{2}, -\frac{1}{4}, -\frac{1}{8}, 0, \frac{1}{8}, \frac{1}{2}, 1, 2\right\},$$

$$D_{10} = \left\{-2, -1, -\frac{1}{2}, -\frac{1}{4}, -\frac{1}{8}, 0, \frac{1}{8}, \frac{1}{4}, \frac{1}{2}, 1, 2\right\}.$$  

Again, all other coefficients are approximated by their closest 8-bit CSD representation. The pooling layers do not have any coefficient here, and only linear and Rectified Linear
TABLE VII: Mean classification rates for the MNIST test set and different approximations relative to the exact model.

<table>
<thead>
<tr>
<th>Exact</th>
<th>ASG Plan</th>
<th>Linear I</th>
<th>Linear II</th>
<th>Quadratic I</th>
<th>Quadratic II</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0000</td>
<td>1.0000</td>
<td>0.9847</td>
<td>0.9680</td>
<td>0.9978</td>
<td>1.0000</td>
</tr>
<tr>
<td>A1</td>
<td>0.9684</td>
<td>0.9684</td>
<td>0.9588</td>
<td>0.9260</td>
<td>0.9615</td>
</tr>
<tr>
<td>A2</td>
<td>0.9643</td>
<td>0.9643</td>
<td>0.9627</td>
<td>0.8805</td>
<td>0.9573</td>
</tr>
<tr>
<td>A3</td>
<td>0.9961</td>
<td><strong>0.9961</strong></td>
<td>0.9848</td>
<td>0.9655</td>
<td><strong>0.9944</strong></td>
</tr>
<tr>
<td>A4</td>
<td>0.9973</td>
<td><strong>0.9973</strong></td>
<td>0.9863</td>
<td>0.9700</td>
<td><strong>0.9969</strong></td>
</tr>
<tr>
<td>A5</td>
<td>0.9976</td>
<td><strong>0.9976</strong></td>
<td>0.9866</td>
<td>0.9666</td>
<td><strong>0.9969</strong></td>
</tr>
<tr>
<td>A6</td>
<td>0.9991</td>
<td><strong>0.9991</strong></td>
<td>0.9868</td>
<td>0.9701</td>
<td><strong>0.9973</strong></td>
</tr>
<tr>
<td>A7</td>
<td>0.9992</td>
<td><strong>0.9992</strong></td>
<td>0.9846</td>
<td>0.9680</td>
<td><strong>0.9977</strong></td>
</tr>
<tr>
<td>A8</td>
<td>0.9994</td>
<td><strong>0.9994</strong></td>
<td>0.9848</td>
<td>0.9675</td>
<td><strong>0.9981</strong></td>
</tr>
<tr>
<td>A3,3,1,1</td>
<td>0.9931</td>
<td><strong>0.9931</strong></td>
<td>0.9749</td>
<td>0.9625</td>
<td><strong>0.9924</strong></td>
</tr>
<tr>
<td>A3,1,1,1</td>
<td>0.9891</td>
<td>0.9891</td>
<td>0.9684</td>
<td>0.9580</td>
<td>0.9866</td>
</tr>
<tr>
<td>A4,4,1,1</td>
<td>0.9937</td>
<td><strong>0.9937</strong></td>
<td>0.9780</td>
<td>0.9618</td>
<td><strong>0.9943</strong></td>
</tr>
<tr>
<td>A4,1,1,1</td>
<td>0.9885</td>
<td>0.9885</td>
<td>0.9655</td>
<td>0.9572</td>
<td>0.9872</td>
</tr>
</tbody>
</table>

TABLE VIII: Classification accuracy and top-5 accuracy for ImageNet and different approximations relative to the exact AlexNet model

<table>
<thead>
<tr>
<th></th>
<th>Absolute Accuracy</th>
<th>Top-5 Accuracy</th>
<th>Relative Accuracy</th>
<th>Top-5 Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact</td>
<td>0.5682</td>
<td>0.7995</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>A9</td>
<td>0.4862</td>
<td>0.7288</td>
<td>0.8558</td>
<td>0.9117</td>
</tr>
<tr>
<td>A10</td>
<td>0.5463</td>
<td><strong>0.7820</strong></td>
<td><strong>0.9616</strong></td>
<td><strong>0.9782</strong></td>
</tr>
<tr>
<td>A10,9,9,9,9</td>
<td>0.5423</td>
<td>0.7794</td>
<td>0.9544</td>
<td>0.9750</td>
</tr>
<tr>
<td>A10,10,9,9,9,9</td>
<td>0.5442</td>
<td>0.7796</td>
<td>0.9578</td>
<td>0.9751</td>
</tr>
</tbody>
</table>

Units (ReLU) are used as activation function, which are already of very low complexity and thus do not require any approximation.

We used the ImageNet 2012 validation set to evaluate our different approximations. And, as usual in the literature, we compute the classification accuracy as well as the top-5 accuracy for the 50000 test images. Table VIII shows the results. The approximation A10 with the set \( \mathcal{D}_0 \) gives the best performance, with a relative decrease in accuracy of only 3.84% and 2.18% on the top-5 accuracy. However, as the following line shows, we can achieve almost the same performance using the coarser set \( \mathcal{D}_9 \) for all convolution layers except the first one. This again suggests that a finer approximation of the first layer is required to prevent a drastic performance drop.

V. CONCLUSION

We presented a novel scheme for approximating the parameters of a trained ConvNet, notably the convolution filters, neuron weights, as well as pooling and bias coefficients. Activation functions were also approximated. The particularity of the matrix approximations is that they allow for an extremely efficient implementation—software or hardware—using only additions and bit-shifts, and no multiplication. We thoroughly evaluated the impact of this parameter approximation measuring the overall performance of ConvNets on three different use cases: one smaller ConvNet for face detection, a larger ConvNet for hand-written digit classification, and a much more complex, deep ConvNet for large-scale image classification.

For all three models, our proposed scheme was able to produce low-complexity approximations without a significant loss in performance.

These results suggest that huge reductions in computational complexity of trained ConvNet models can be obtained, and extremely efficient hardware implementations can be realized. Further studies need to be undertaken to analyse the impact of this type of approximations for more use cases and different architectures.

REFERENCES


Christophe Garcia is Full Professor at INSA Lyon. From 2011 to 2015, he has been head of the IMAGINE research team of the LIRIS laboratory. He is now Deputy Director of the LIRIS laboratory and Vice-President for Research in Information and Digital Society at Insa Lyon. His current technical and research activities are in the areas of ConvNets, neural networks, pattern recognition and computer vision. He holds 17 industrial patents and has published more than 140 articles in international conferences and journals. He has served as an associate editor of Int. Journal of Visual Communication and Image Representation (Elsevier), Image and Video Processing (Hindawi) and Pattern Analysis and Application (Springer-Verlag).

André Leite received the B.Sc., M.Sc., and D.Sc. degrees in electrical engineering from Universidade Federal de Pernambuco (UFPE), Brazil, in 2003, 2005, and 2009, respectively. Since 2010, he has been with the Department of Statistics, UFPE. His areas of interest include dynamical systems, control theory and optimization, fundamentals of probability and decision theory.