Analysis and Design of Convolutional Networks
via Hierarchical Tensor Decompositions

Nadav Cohen   COHENNADAV@CS.HUJI.AC.IL
Or Sharir   OR.SHARIR@CS.HUJI.AC.IL
Yoav Levine   YOAVLEVINE@CS.HUJI.AC.IL
Ronen Tamari   RONENT@CS.HUJI.AC.IL
David Yakira   DAVIDYAKIRA@CS.HUJI.AC.IL
Amnon Shashua   SHASHUA@CS.HUJI.AC.IL
The Hebrew University of Jerusalem

Abstract

The driving force behind convolutional networks – the most successful deep learning architecture to date, is their expressive power. Despite its wide acceptance and vast empirical evidence, formal analyses supporting this belief are scarce. The primary notions for formally reasoning about expressiveness are efficiency and inductive bias. Expressive efficiency refers to the ability of a network architecture to realize functions that require an alternative architecture to be much larger. Inductive bias refers to the prioritization of some functions over others given prior knowledge regarding a task at hand. In this paper we overview a series of works written by the authors, that through an equivalence to hierarchical tensor decompositions, analyze the expressive efficiency and inductive bias of various convolutional network architectural features (depth, width, strides and more). The results presented shed light on the demonstrated effectiveness of convolutional networks, and in addition, provide new tools for network design. [1]

Keywords: *Convolutional Networks*, *Expressiveness*, *Hierarchical Tensor Decompositions*

1. Introduction

Convolutional networks (LeCun and Bengio (1995)) are the cornerstone of modern deep learning. Since the work of Krizhevsky et al. (2012), they prevail in the domain of visual recognition, and recently, they have also been delivering state of the art results in speech and text processing tasks (see for example van den Oord et al. (2016); Kalchbrenner et al. (2016)). As opposed to classic deep network architectures, such as the multilayer perceptron (feed-forward fully-connected neural network – Rosenblatt (1961)), employing a modern convolutional network involves setting dozens or even hundreds of architectural parameters. Namely, besides the basic choices of network depth, width of each layer, and type of non-linear activations (*e.g.* sigmoid or ReLU – Nair and Hinton (2010)), one must decide on the type of pooling operator in each layer (*e.g.* max or average), the kernel sizes and strides in every convolution/pooling, the connectivity scheme to employ (*e.g.* skip connections – He et al. (2015)), and much more. To date, these architectural choices are typically made heuristically, based on past experience, conventional wisdom, and trial-and-error. This often leads to lengthy and inefficient development cycles, ultimately concluding in suboptimal results.
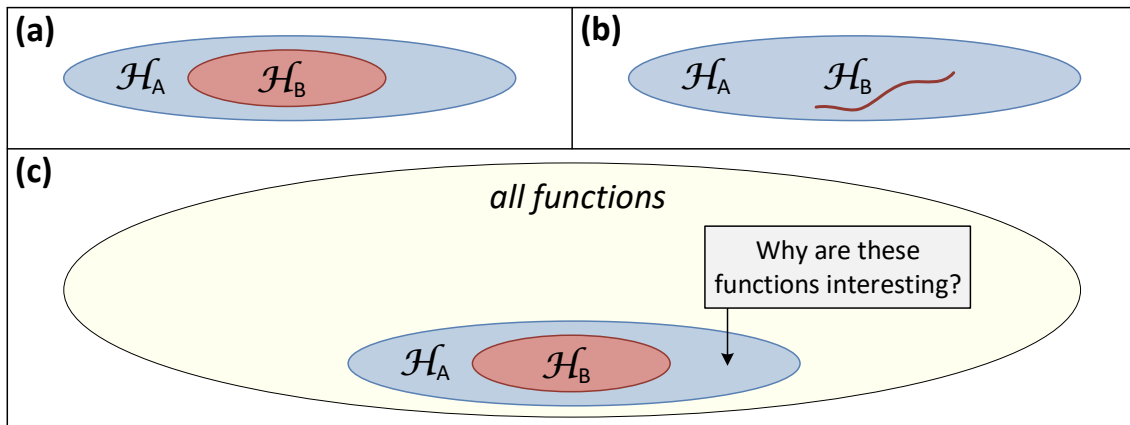

Figure 1: Illustrations of expressive efficiency and inductive bias. $\mathcal{H}_A$ and $\mathcal{H}_B$ are the sets of functions practically realizable by network architectures A and B (respectively). *(a)* Expressive efficiency – $\mathcal{H}_A$ is a strict superset of $\mathcal{H}_B$. *(b)* Complete expressive efficiency – $\mathcal{H}_B$ has zero "volume" inside $\mathcal{H}_A$. *(c)* Inductive bias – efficiency of A over B does not imply that $\mathcal{H}_A$ contains meaningful functions; this can only be explained through the requirements of real-world problems.

More principled design practices, which could be made available by a more formal understanding of modern convolutional network architectures, are thus of great interest.

It is widely accepted that the driving force behind convolutional networks, and deep networks in general, is their *expressiveness*, *i.e.* their ability to compactly represent rich and effective classes of functions. The primary notions for formally reasoning about expressiveness are *efficiency* and *inductive bias*. Efficiency refers to a situation where one network must grow unfeasibly large in order to realize (or approximate) functions of another. Inductive bias refers to real-world tasks requiring specific types of functions (*e.g.* translation invariant), not just arbitrary ones. Our interest lies on the expressiveness of convolutional networks, or more specifically, on the expressive efficiency and inductive bias brought forth by their various architectural features. We review in this paper a series of works written by the authors (Cohen et al. (2016b,a); Sharir et al. (2017); Cohen and Shashua (2016, 2017); Sharir and Shashua (2017); Cohen et al. (2017); Levine et al. (2017)), which address these topics through the mathematical notion of *hierarchical tensor decompositions* (see Hackbusch (2012) for a comprehensive introduction). Our presentation here is soft and oftentimes informal, with the objective of creating a manuscript accessible to a wide range of audience. For an exact and formal presentation, we refer the reader to the papers we review.

2. Expressive Efficiency and Inductive Bias

As stated in the introduction, *expressive efficiency* refers to a situation where one network must grow unfeasibly large in order to realize (or approximate) functions of another. More explicitly, consider network architectures A and B, with size parameters r_A and r_B (respectively). For example, A could be an instance of AlexNet (Krizhevsky et al. (2012)) with each layer (convolutional and fully-connected) comprising r_A channels, whereas B could be an instance of ResNet (He et al. (2015)) with r_B channels in each of its layers. Denote by $\mathcal{H}_A$ (respectively $\mathcal{H}_B$) the set of functions

that may be realized by A (respectively B) with size parameter r_A (respectively r_B) that is small enough for practical implementation. We say that architecture A is efficient with respect to architecture B if $\mathcal{H}_A$ is a strict superset of $\mathcal{H}_B$ (see illustration in fig. 1(a)), meaning that A can realize with practical size anything that B can, whereas the converse does not hold – there exist functions compactly realizable by A that cannot be practically replicated by B.

Given that A is efficient with respect to B, a natural question arises: How many of the functions realizable by A reflect its efficiency over B? Is it just one function that A can realize compactly and B cannot, or are there many? This question amounts to reasoning about the "volume" of $\mathcal{H}_B$ inside $\mathcal{H}_A$ – if the volume is small, a significant portion of the functions compactly realizable by A lay outside the reach of B, whereas on the other hand, a large volume implies that B comes close to A in terms of the functions it supports. The strongest form of efficiency, referred to as *complete*, takes place when the volume of $\mathcal{H}_B$ in $\mathcal{H}_A$ is essentially zero (see illustration in fig. 1(b)). In this case, almost all functions realizable by A cannot be replicated by B unless that is unfeasibly large.

For concreteness, we provide below definitions of expressive efficiency and completeness that are slightly more formal than the above:

Definition 1 *Let A and B be network architectures with size parameters r_A and r_B (respectively). We say that A is* expressively efficient *with respect to B if the following two conditions hold:*

  (i) Any function realized by B with size r_B can be replicated by A with size r_A that is no more than linear in r_B (i.e. $r_A \in \mathcal{O}(r_B)$).

  (ii) There exist functions realized by A with size r_A that cannot be replicated by B unless its size r_B is super-linear in r_A (i.e. $r_B \in \Omega(f(r_A))$ for a super-linear function $f(\cdot)$)

The expressive efficiency of A over B is complete *if randomizing the weights of A by any continuous distribution leads, with probability 1, to a function satisfying condition (ii).*

Expressive efficiency alone does not convey the entire story behind the effectiveness of functions realized by deep networks. Under any network architecture, the set of functions realizable with practical size is merely a small fraction of all possible functions. Accordingly, even if architecture A is expressively efficient with respect to architecture B, we have no information indicating that functions compactly realizable by A would be effective in practice. In other words, even if $\mathcal{H}_A$ is a strict superset of $\mathcal{H}_B$, it is still a mere corner in the space of all functions, and a-priori, may not include any meaningful function (see illustration in fig. 1(c)). To understand why for certain architectures, *e.g.* convolutional networks, $\mathcal{H}_A$ is so effective in practice, one must consider the *inductive bias*, *i.e.* the actual needs of real-world problems. Functions required for successful execution of tasks such as image classification or speech-to-text annotation, are not arbitrary – there are certain task-dependent properties, for example smoothness or translation invariance, that must be met. Accordingly, a given network architecture need not realize all functions, only those possessing certain properties. Empirical evidence suggests that by properly designing a convolutional network, functions fulfilling requirements of various tasks become available. Formal understanding of this phenomenon is lacking.

2.1. Questions on the Expressive Efficiency and Inductive Bias of Convolutional Networks

Our interest lies on the expressive efficiency and inductive bias brought forth by the various architectural features of modern convolutional networks. Below are the specific questions we address.

Question 1 (Efficiency of Depth – addressed in sec. 4) Perhaps the most prominent empirical finding of deep learning, which in some sense identifies the field, is that deep networks, when operated appropriately, greatly outperform shallow ones (see LeCun et al. (2015) for a survey of such results). The conventional argument for explaining this phenomenon is that depth brings forth a representational power that is otherwise unattainable. Formally, it amounts to saying that deep networks are expressively efficient with respect to shallow ones. This proposition, which traces back to classical questions from the world of circuit complexity, has recently been proven for various network architectures (see for example Delalleau and Bengio (2011); Pascanu et al. (2013); Montufar et al. (2014); Telgarsky (2015); Eldan and Shamir (2015); Poggio et al. (2015); Mhaskar et al. (2016)). For convolutional networks however, the proposition has not been proven, and the question of whether or not depth brings forth expressive efficiency remains open. Moreover, even if one makes the reasonable assumption by which convolutional networks, similarly to other architectures, admit depth efficiency, it still is unclear how frequent the latter is, and in particular, whether or not it is complete. Completeness of depth efficiency has never been established, for any network architecture of a practical nature.

Question 2 (Inductive Bias of Convolution/Pooling Geometry – addressed in sec. 5) A key ingredient of convolutional networks is the locality of their convolution and pooling (decimation) operations. Traditionally, convolution and pooling windows are chosen to be contiguous blocks (squares in 2D networks, intervals in 1D), reflecting an intuitive assumption by which such geometries are suitable for data of a continuous nature (*e.g.* images or audio). Recently however, several works have demonstrated that different geometries, such as non-contiguous windows with internal dilations (*cf.* Yu and Koltun (2015); van den Oord et al. (2016); Kalchbrenner et al. (2016)), or even windows with dynamically learned shapes (*cf.* Li et al. (2017)), can lead to improved performance. We would like to understand the relations between a network's convolution/pooling geometry, the set of functions it can model, and the suitability of this set for different tasks. That is to say, we would like to understand the inductive bias governing geometries of convolution and pooling windows in convolutional networks. Formal results on this line are not only of theoretical interest – they may potentially provide practical guidelines for tailoring a network's convolution/pooling geometry in accordance with a task at hand.

Question 3 (Efficiency of Overlapping Operations – addressed in sec. 6) Modern convolutional networks, *e.g.* VGG (Simonyan and Zisserman (2014)) or GoogLeNet (Szegedy et al. (2015)), include a mix of convolution and pooling operations, some of which are overlapping (stride smaller than window size), while others are not (stride and window size equal). Empirical evidence suggests that non-overlapping operations are beneficial, but nonetheless, must be accompanied by overlapping operations in order to produce competitive performance. We would like to understand whether this need for overlaps can be attributed to expressiveness, or more specifically, whether convolutional networks with overlapping operations are expressively efficient with respect to ones without.

Question 4 (Inductive Bias of Layer Widths – addressed in sec. 7) A fundamental architectural choice to be made when designing a convolutional network is the width of (number of channels in) each layer. At present, there are no firm principles for making this decision – in the majority of cases layer widths are either set uniformly across a network, or such that deeper layers are wider, so as to avoid "representational bottlenecks" (Szegedy et al. (2016)). Given a fixed amount of computational resources, it is unclear what would be an effective distribution of layer widths across a network, and

how this depends on the particular task at hand. From a representational perspective, this boils down to reasoning about the inductive bias of layer widths, *i.e.*, about the implication of widening one layer versus another in terms of the functions a network can realize. A formal treatment of this question could pave the way to more principled convolutional network designs, in which layer widths are tailored to the nature of a given task.

Question 5 (Efficiency of Connectivity – addressed in sec. 8) The classic convolutional network architecture, oftentimes referred to as LeNet (see LeCun and Bengio (1995)), consists of layers concatenated one after the other in a feed-forward (chain) scheme. Until recently, networks adhering to this architectural paradigm provided state of the art visual recognition performance. In 2014, with the rise of GoogLeNet (Szegedy et al. (2015)), a new type of convolutional networks has emerged. These networks no longer follow the simple feed-forward approach, but rather run layers in parallel, employing various connectivity (split/merge) schemes. In 2015, connectivity schemes in convolutional networks took one step further, with the introduction of ResNet (He et al. (2015)), whose layers are linked through "skip connections". Nowadays, nearly all state of the art convolutional networks (*e.g.* Huang et al. (2016); van den Oord et al. (2016); Kalchbrenner et al. (2016)), for visual recognition as well as audio and text processing tasks, employ elaborate connectivity schemes. The question we ask is whether this can be understood in terms of expressive efficiency, *i.e.*, whether connectivity schemes bear the potential to create networks that are expressively efficient with respect to the standard feed-forward architecture.

3. Convolutional Arithmetic Circuits and Hierarchical Tensor Decompositions

To analyze the expressive efficiency and inductive bias (see sec. 2) of convolutional networks, and in particular, to address the questions laid out in sec. 2.1, we focus on a family of models named *convolutional arithmetic circuits*. Convolutional arithmetic circuits are convolutional networks with a particular choice of non-linearities. Namely, they arise by setting point-wise activations to be linear (as opposed to sigmoid or ReLU), and pooling operators to be based on product (as opposed to max or average).[2] The reason we focus on convolutional arithmetic circuits is their intimate relation to various mathematical fields (tensor analysis, measure theory, functional analysis, theoretical physics, graph theory and more), rendering them especially amendable to theoretical analyses. We will see in sec. 4.1 how mathematical machinery developed for the analysis of convolutional arithmetic circuits can be adapted to account for other types of convolutional networks as well, for example ones with ReLU activation and max or average pooling. We note that besides their theoretical merits, convolutional arithmetic circuits also deliver promising results in practice. Specifically, they excel in computationally constrained settings (Cohen et al. (2016a)), and give state of the art results in classification under missing data (Sharir et al. (2017)).[3]

The convolutional arithmetic circuit architecture we consider as baseline is the one depicted in fig. 2. It is a 2D convolutional network comprising an initial convolutional layer (referred to as *representation*) followed by L hidden layers, which in turn are followed by a dense (linear) output layer. Each hidden layer consists of a convolution and spatial pooling. Besides the fact that the

---

2. As an alternative viewpoint, convolutional arithmetic circuits can be seen as sum-product networks (Poon and Domingos (2011)) whose structure is convolutional.

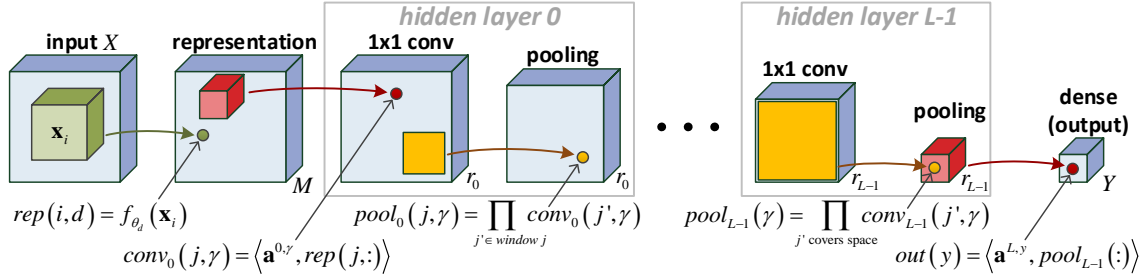3. An implementation of convolutional arithmetic circuits (also known as *SimNets*) for Caffe toolbox (Jia et al. (2014)) can be found on-line at https://github.com/HUJI-Deep/caffe-simnets.

input X representation   **hidden layer 0**   **1x1 conv**   pooling   **hidden layer L-1**   **1x1 conv**   pooling   dense (output)

$rep(i,d) = f_{\theta_d}(\mathbf{x}_i)$

$conv_0(j,\gamma) = \langle \mathbf{a}^{0,\gamma}, rep(j,:) \rangle$

$pool_0(j,\gamma) = \prod_{j' \in window\ j} conv_0(j',\gamma)$

$pool_{L-1}(\gamma) = \prod_{j'\ covers\ space} conv'_{L-1}(j',\gamma)$

$out(y) = \langle \mathbf{a}^{L,y}, pool_{L-1}(:) \rangle$

Figure 2: Baseline convolutional arithmetic circuit architecture – the starting point of our analyses.

convolution's activations are linear ($\sigma(z){=}z$), and that the pooling operator is based on products ($P\{c_j\}{=}\prod_j c_j$), further restrictions of the architecture are that receptive fields of the convolution are $1{\times}1$, and pooling windows do not overlap. All of these limitations will be relieved as we move forward (sec. 4.1 and 6). At the starting point however, we remain with the baseline architecture (fig. 2), as it holds strong connections to well-established mathematical constructions, in particular ones from the field of tensor analysis.

There are different ways to formulate a connection between convolutional networks and tensors (multi-dimensional arrays), the simplest being through the notion of grid tensors. Let $f(\cdot)$ be a real-valued function defined over the unit square in the plane, *i.e.* $f : [0,1]^2 \to \mathbb{R}$. For a positive integer $M \in \mathbb{N}$, we may discretize the interval $[0,1]$ into the M points $\{1/M, 2/M, \ldots, M/M\}$, and define an $M{\times}M$ matrix holding function values over discretized inputs:

$$\mathcal{A} := \begin{bmatrix} f(1/M, 1/M) & f(1/M, 2/M) & \cdots & f(1/M, M/M) \\ f(2/M, 1/M) & f(2/M, 2/M) & \cdots & f(2/M, M/M) \\ \vdots & \vdots & \ddots & \vdots \\ f(M/M, 1/M) & f(M/M, 2/M) & \cdots & f(M/M, M/M) \end{bmatrix}$$

This matrix is in fact a lookup table representing $f(\cdot)$, that becomes larger and more fine-grained as M grows. Suppose now that $f(\cdot)$ was defined over the N-dimensional unit hypercube, *i.e.* $f : [0,1]^N \to \mathbb{R}$. In this case the lookup table $\mathcal{A}$ would transform from a matrix into a tensor (multi-dimensional array), having N modes (axes) of length M each. We refer to $\mathcal{A}$ as the *grid tensor* of the function $f(\cdot)$. A convolutional network with fixed weights can be viewed (without loss of generality) as a function over the unit hypercube, where the dimension N is equal to the number of input elements (*e.g.* pixel values or audio samples). We will study such functions through their corresponding grid tensors.

Grid tensors of convolutional networks are typically of very high order, *i.e.* have many modes. For example, if a network processes gray-scale images of size $100{\times}100$, N – the order of (number of modes in) its grid tensor, will be 10^4, meaning there are M^{10^4} entries in the tensor. Such exponentially large tensors are obviously impractical to manipulate or store directly. They may however be represented efficiently through algebraic constructions named *tensor decompositions*. Tensor decompositions are essentially parameterizations that allow representation of large tensors with a relatively small number of parameters. In the special case of order-2 tensors, *i.e.* matrices, most types of tensor decompositions boil down to simply a low-rank matrix decomposition. As an example of the latter, consider the space $\mathbb{R}^{10^6 \times 10^6}$, *i.e.* the space of matrices with size $10^6{\times}10^6$. Elements of this space are too large to be stored directly in a typical personal computer. However, if
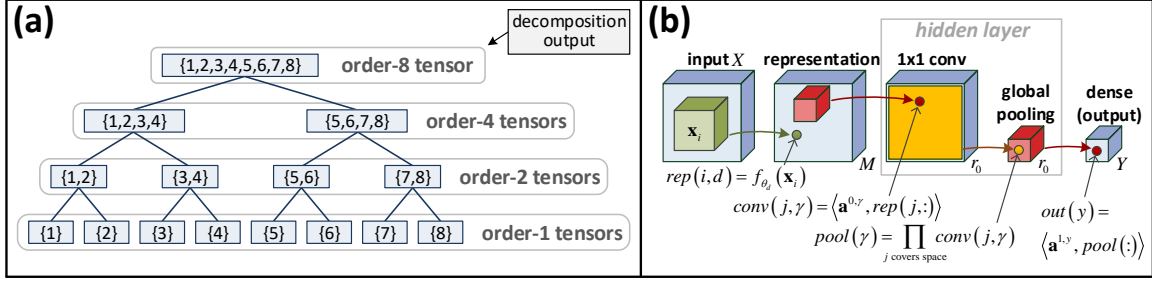
Figure 3: **(a)** Illustration of a tree over tensor modes (axes). Different mode trees give rise to different hierarchical tensor decompositions. The decomposition corresponding to this tree operates in three stages: the first assembles vectors (order-1 tensors) into matrices (order-2 tensors), the second uses these matrices to create order-4 tensors, and the third (final stage) combines the latter into an order-8 tensor – decomposition output. **(b)** Shallow (single hidden layer) convolutional arithmetic circuit.

we are willing to limit ourselves to a subset of this space comprising matrices of low rank, a compact parameterization immediately emerges. For instance, if matrices of rank 5 or less are sufficient, any element in our subset can be represented as a product of two matrices – one of size $10^6 \times 5$, and the other of size 5×10^6. We thus obtain a representation of tensors (matrices) with 10^{12} entries using only 10^7 parameters – a manageable number even for a low-end handheld device.

As opposed to the special case of matrices (order-2 tensors), decomposing tensors of a general order can be done in numerous ways. A rich family of decompositions, which allows representing tensors of extremely high order, is the so-called hierarchical format, also known as *hierarchical tensor decompositions*. Introduced in Hackbusch and Kühn (2009) (and later generalized in Hackbusch (2012)), these decompositions represent tensors by incrementally constructing intermediate tensors of increasing order. For example, suppose we are to decompose an order-8 tensor. A hierarchical decomposition could operate in three stages: the first assembles vectors (order-1 tensors) into matrices (order-2 tensors), the second uses these matrices to construct order-4 tensors, and the third (final stage) combines the latter tensors into the final order-8 output. This process can be described by a full binary tree[4] over tensor modes, as illustrated in fig. 3(a). In general, when the order of a tensor is high, there are many possible trees over its modes, and each tree corresponds to a different hierarchical decomposition.

A key observation we make is that ***convolutional arithmetic circuits are equivalent to hierarchical tensor decompositions***. More precisely, grid tensors of functions realized by the baseline convolutional arithmetic circuit architecture (fig. 2) can be represented via hierarchical tensor decompositions. The correspondence between networks and decompositions is bijective (one-to-one) – for every network structure (depth, width of each layer, geometry of pooling windows *etc.*) there exists a unique decomposition that represents its grid tensors, and vice versa. Under this correspondence, network weights (in convolution and output layers) are directly mapped to the parameters of the respective decomposition. We present below two canonical examples of networks and their corresponding decompositions. These examples will accompany us throughout the paper.

---

4. A full binary tree is a tree in which all nodes but the leaves have exactly two children.

Example 1 (Shallow Network $\longleftrightarrow$ CP Decomposition) Consider a shallow convolutional arithmetic circuit with a single hidden layer of width r_0, as illustrated in fig. 3(b). The convolutional filters of this network are the vectors $\{\mathbf{a}^{0,\gamma}\}_{\gamma=1}^{r_0}$, and the linear weights of output y are held in vector $\mathbf{a}^{1,y}$. Denote by $\mathcal{A}^y$ the grid tensor of the function realized by output y. This tensor is given by the following formula:

$$\mathcal{A}^y = \sum\nolimits_{\gamma=1}^{r_0} a_\gamma^{1,y} \cdot \underbrace{\mathbf{a}^{0,\gamma} \otimes \mathbf{a}^{0,\gamma} \otimes \cdots \otimes \mathbf{a}^{0,\gamma}}_{N \text{ times}} \tag{1}$$

where $a_\gamma^{1,y}$ stands for coordinate γ of the vector $\mathbf{a}^{1,y}$, and $\otimes$ is simply the outer product operator.[5] The formula in eq. 1 is an instance of the CANDECOMP/PARAFAC tensor decomposition, or *CP decomposition* for short. CP decomposition can be viewed as a special case of a hierarchical decomposition, and is perhaps the most classic tensor decomposition recorded in the literature, dating back to the early 20th century (see Kolda and Bader (2009) for a historic survey).

Example 2 (Deep Network $\longleftrightarrow$ HT Decomposition) Consider now the deep convolutional arithmetic circuit obtained by setting each pooling window in the baseline architecture (fig. 2) to have size 2. Grid tensors of functions realized by this network are given by the Hierarchical Tucker decomposition, whose name we abbreviate as *HT decomposition*. HT decomposition was the first tensor decomposition to explicitly incorporate a hierarchical structure, based on a perfect binary mode tree[6] as illustrated in fig. 3(a). Its introduction in Hackbusch and Kühn (2009) marks the dawn of hierarchical tensor decompositions as they are known today.

To summarize this section, we presented an algebraic (sum-product) variant of convolutional networks named convolutional arithmetic circuits, and discussed its equivalence to hierarchical tensor decompositions. There is a one-to-one correspondence between the structure of a network (depth, width of each layer *etc.*) and the type of its respective decomposition, with network weights mapped to decomposition parameters. This allows analyzing networks through their corresponding tensor decompositions, opening the door to a plurality of mathematical tools. Hereafter, we make use of these tools to analyze the expressive efficiency and inductive bias (see sec. 2) of convolutional arithmetic circuits, as well as other types of convolutional networks (ones with ReLU activation and max or average pooling).

4. Efficiency of Depth *(Cohen et al. (2016b); Cohen and Shashua (2016))*

In this section we address question 1 in sec. 2.1, dealing with the expressive efficiency brought forth by deepening convolutional networks. As a first step in this direction, we compare the shallow and deep convolutional arithmetic circuits presented in sec. 3 (examples 1 and 2), which correspond to CP and HT decompositions respectively. Recall from def. 1 that in order to establish expressive efficiency of the deep network with respect to the shallow one, two propositions are to be proven:

  (i) Any function realized by the shallow network can be replicated by the deep network with no more than linear growth in size

---

5. For example, if $\mathcal{B}$ and $\mathcal{C}$ are order-3 and order-4 tensors (3- and 4-dimensional arrays) respectively, their outer product $\mathcal{B} \otimes \mathcal{C}$ is the order-7 tensor defined by: $(\mathcal{B} \otimes \mathcal{C})_{d_1 \ldots d_7} = \mathcal{B}_{d_1 \ldots d_3} \cdot \mathcal{C}_{d_4 \ldots d_7}$.

6. A perfect binary tree is a tree in which all interior (non-leaf) nodes have exactly two children and all leaves have exactly the same depth.

(ii) There exist functions realized by the deep network that cannot be replicated by the shallow network unless that is allowed to grow super-linearly

Proposition (i) is trivial – it follows from the fact that the deep network reduces to the shallow one if we set all of its hidden convolutions but the first to be identity mappings. Proposition (ii) is much less obvious – we prove it by showing that under a matrix arrangement, ranks of grid tensors realized by the deep network are far greater than those brought forth by the shallow one.

The process of arranging a tensor as a matrix is called matricization. Let $\mathcal{A}$ be a tensor with N modes (order-N), each of length M. Let (I, J) be a partition of these modes, *i.e.* I and J are disjoint subsets of $\{1, 2, \ldots, N\}$ whose union covers the entire set. The *matricization of $\mathcal{A}$ with respect to (I, J)*, denoted $[\![\mathcal{A}]\!]_{I,J}$, is an arrangement of $\mathcal{A}$ as a matrix, with rows corresponding to modes indexed by I, and columns corresponding to modes indexed by J. For example, suppose that $N = 5$, $I = \{2, 3, 5\}$ and $J = \{1, 4\}$. In this case $[\![\mathcal{A}]\!]_{I,J}$ is obtained by reordering the modes of $\mathcal{A}$ via $(2, 3, 5, 1, 4)$ (*e.g.* using NumPy's `transpose()` function or MATLAB's `permute()`), and then reshaping the resulting array to a $M^3 \times M^2$ matrix (*e.g.* with NumPy or MATLAB's `reshape()` functions).

The following claim, proven in Cohen and Shashua (2017), characterizes tensors generated by CP decomposition in terms of their ranks when subject to matricization:

Claim 2 *Tensors generated by CP decomposition, when matricized with respect to any partition (I, J), have rank that does not exceed the number of terms (summands) in the decomposition.*

Recall from example 1 that in the CP decomposition corresponding to a shallow convolutional arithmetic circuit (eq. 1), r_0 – the number of terms, is precisely equal to the number of hidden channels in the network (see fig. 3(b)). Claim 2 thus implies that grid tensors of functions realized by the shallow convolutional arithmetic circuit have matricization ranks that do not exceed the number of hidden channels.

In stark contrast to CP decomposition, HT decomposition generates tensors with exponentially high matricization ranks. This is formulated in the theorem below, proven in Cohen et al. (2016b):

Theorem 3 *Almost every tensor generated by HT decomposition, when matricized with respect to an even-odd partition ($I = \{1, 3, \ldots\}$, $J = \{2, 4, \ldots\}$), has an exponentially high rank.*

From a network perspective, theorem 3 implies that grid tensors of functions realized by the deep convolutional arithmetic circuit (example 2), when matricized with respect to a particular (even-odd) partition, have ranks that are exponentially high. Moreover, this holds for almost every grid tensor, meaning that if we randomize the weights of the deep network by some continuous distribution, with probability 1, we obtain a function whose grid tensor has an exponential matricization rank.

Taken together, claim 2 and theorem 3 lead to the following corollary:

Corollary 4 *Suppose we randomize the weights of a deep convolutional arithmetic circuit by some continuous distribution. Then, with probability 1, we obtain functions that may only be replicated by a shallow convolutional arithmetic circuit if that has an exponential number of hidden channels.*

Corollary 4 can be phrased succinctly by saying that **with convolutional arithmetic circuits, the expressive efficiency of depth is exponential and complete**. We have treated here the specific shallow and deep networks presented in sec. 3 (examples 1 and 2 respectively), but the methodology employed is readily applicable to arbitrary structures (instances of the baseline convolutional arithmetic

circuit architecture – fig. 2). A less immediate step is the adaptation of our analysis to convolutional networks that are not arithmetic circuits, for example ones with ReLU activation and max or average pooling. This is the topic of the subsection that follows.

4.1. Convolutional Rectifier Networks

Convolutional rectifier networks are convolutional networks with ReLU (Rectified Linear Unit – Nair and Hinton (2010)) activation and max or average pooling. They are the most commonly used type of convolutional networks these days, and thus are of particular interest. We demonstrate below how mathematical machinery developed for the analysis of convolutional arithmetic circuits can be adapted to account for convolutional rectifier networks as well. Our use case will be the study of expressive efficiency brought forth by depth.

Our analysis of convolutional arithmetic circuits is facilitated by their equivalence to hierarchical tensor decompositions (sec. 3). The central operator in hierarchical tensor decompositions is the outer product $\otimes$, also known as *tensor product*. Given two tensors $\mathcal{A}$ and $\mathcal{B}$ of orders P and Q respectively, the tensor (outer) product $\mathcal{A} \otimes \mathcal{B}$ is the tensor of order $P+Q$ defined by:

$$(\mathcal{A} \otimes \mathcal{B})_{d_1...d_{P+Q}} = \mathcal{A}_{d_1...d_P} \cdot \mathcal{B}_{d_{P+1}...d_{P+Q}} \tag{2}$$

The multiplication in the definition of the tensor product is suitable for convolutional arithmetic circuits (linear activation, product pooling), but not for other models. However, by replacing multiplication with a different operator $g : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, we may extend the equivalence to other types of convolutional networks, and in particular, to convolutional rectifier networks. The next paragraph provides details.

Consider a convolutional arithmetic circuit, *i.e.* an instance of the architecture depicted in fig. 2. As we have seen in sec. 3, this network corresponds to some hierarchical tensor decomposition $\mathbf{D}$. Suppose now that we modify the network by adding point-wise activations $\sigma(\cdot)$ after each convolution, and replacing product pooling with a different pooling operator $P\{\cdot\}$.[7] Define the *activation-pooling operator*:

$$g : \mathbb{R} \times \mathbb{R} \to \mathbb{R} \ , \ g(a,b) = P\{\sigma(a), \sigma(b)\} \tag{3}$$

and consider the *generalized tensor product* $\otimes_g$ obtained by placing $g(\cdot)$ instead of multiplication in the tensor product $\otimes$ (eq. 2):

$$(\mathcal{A} \otimes_g \mathcal{B})_{d_1...d_{P+Q}} = g(\mathcal{A}_{d_1...d_P}, \mathcal{B}_{d_{P+1}...d_{P+Q}})$$

If we replace all instances of the tensor product $\otimes$ in the decomposition $\mathbf{D}$ by the generalized tensor product $\otimes_g$, we obtain what is called a *generalized hierarchical tensor decomposition*, naturally denoted by $\mathbf{D}_g$. As it turns out, grid tensors of functions realized by the convolutional network with activation $\sigma(\cdot)$ and pooling $P\{\cdot\}$, are precisely the tensors represented by the generalized decomposition $\mathbf{D}_g$ (which is based on the activation-pooling operator – eq. 3). We thus have a framework for analyzing general convolutional networks, not just convolutional arithmetic circuits.

Focusing on the particular case of convolutional rectifier networks (corresponding to the choices $\sigma(z) = \max\{z, 0\}$ and $P\{c_j\} = \max\{c_j\}$ or $P\{c_j\} = \text{mean}\{c_j\}$), we follow a path similar to that taken in the analysis of convolutional arithmetic circuits (studying ranks of matricized grid tensors), and derive the following claim (see Cohen and Shashua (2016) for proof):

---

7. For example, $\sigma(\cdot)$ can be chosen as ReLU ($\sigma(z) = \max\{z, 0\}$), while $P\{\cdot\}$ could be set to max ($P\{c_j\} = \max\{c_j\}$).

Claim 5 *There exist functions realizable by a deep convolutional rectifier network that can only be replicated by a shallow network if that has an exponential number of hidden channels.*

Taking into account the fact that a deep network easily replicates functions of a shallow one (its second to last hidden convolutions can realize the identity mapping), we conclude from claim 5 that with convolutional rectifier networks exponential expressive efficiency of depth takes place. However, ***unlike in the case of convolutional arithmetic circuits, where the efficiency of depth is complete, with convolutional rectifier networks it is not***. This is stated in the following claim (proven in Cohen and Shashua (2016)):

Claim 6 *A non-negligible (positive measure) set of the functions realizable by a deep convolutional rectifier network can be replicated by a shallow network with just a few hidden channels.*

The expressive efficiency of depth is believed to be the key factor behind the success of convolutional networks (and deep learning in general). Our analyses indicate that from this perspective, the widely used convolutional rectifier networks are inferior to convolutional arithmetic circuits.[8] This leads us to believe that convolutional arithmetic circuits bear the potential to improve the performance of convolutional networks beyond what is witnessed today. Of course, a practical machine learning model is measured not only by its expressiveness, but also by our ability to train it. Over the years, massive amounts of research have been devoted to training convolutional rectifier networks. Convolutional arithmetic circuits on the other hand received far less attention, although they have been successfully trained in recent works, showing promising results in different settings (*cf.* Cohen et al. (2016a); Sharir et al. (2017)). We believe that developing effective methods for training convolutional arithmetic circuits, thereby fulfilling their expressive potential, may give rise to a deep learning architecture that is provably superior to convolutional rectifier networks but has so far been largely overlooked.

5. Inductive Bias of Pooling Geometry *(Cohen and Shashua (2017))*

In this section we focus on question 2 from sec. 2.1. Specifically, we study the effect of a convolutional network's pooling geometry on its ability to model interactions among regions of its input.

Let $f(x_1, x_2, \ldots, x_N)$ be a function realized by a convolutional network, where $x_1 \ldots x_N$ are input elements, for example pixel intensities of a gray-scale image. Interactions modeled by $f(\cdot)$ between regions of its input are formalized through the notion of separation rank – a commonly used measure in numerical analysis (*cf.* Beylkin and Mohlenkamp (2002)), which is also deeply rooted in the world of quantum physics (see sec. 7). Let (I, J) be a partition of input elements, *i.e.* I and J are disjoint subsets of $\{1, 2, \ldots, N\}$ whose union covers the entire set. The *separation rank of* $f(\cdot)$ *with respect to* (I, J), denoted $sep(f; I, J)$, measures the strength of interaction $f(\cdot)$

---

8. One may argue that this does not carry any information allowing a comparison between the two architectures. Indeed, we have seen that the expressive efficiency of deep convolutional arithmetic circuits with respect to shallow convolutional arithmetic circuits is complete, whereas that of deep convolutional rectifier networks with respect to shallow convolutional rectifier networks is incomplete. A-priori, it may be that depth is more beneficial with convolutional arithmetic circuits, while overall, convolutional rectifier networks are strictly superior in terms of expressiveness. Apparently, this is not the case – it is shown in Cohen and Shashua (2016) that the expressive efficiency of deep convolutional arithmetic circuits is complete not only with respect to shallow convolutional arithmetic circuits, but also with respect to shallow convolutional rectifier networks. Analogously, it is shown that the expressive efficiency of deep convolutional rectifier networks is *incomplete* with respect to shallow models of both architectures.

models between the input elements corresponding to $I - \{x_i\}_{i \in I}$, and those corresponding to $J - \{x_j\}_{j \in J}$. Assume for simplicity of notation, and without loss of generality, that $I = \{1, 2, \ldots, K\}$ and $J = \{K{+}1, K{+}2, \ldots, N\}$. If $f(\cdot)$ is separable with respect to (I, J), meaning there exist functions $g(\cdot)$ and $h(\cdot)$ such that:

$$f(x_1, \ldots, x_N) = g(x_1, \ldots, x_K) \cdot h(x_{K+1}, \ldots, x_N)$$

then under $f(\cdot)$, there is absolutely no interaction between $\{x_i\}_{i \in I}$ and $\{x_j\}_{j \in J}$.[9] In this case, by definition, $sep(f; I, J) = 1$. If the function $f(\cdot)$ itself is not separable, but can be written as a sum of two separable functions, then $sep(f; I, J) = 2$. If $f(\cdot)$ cannot be written as a sum of two separable functions but can be expressed as a sum of three separable functions then $sep(f; I, J) = 3$, and so forth. In general, the higher $sep(f; I, J)$ is, the farther $f(\cdot)$ is from separability with respect to (I, J), *i.e.* the stronger the interaction it models between $\{x_i\}_{i \in I}$ and $\{x_j\}_{j \in J}$.

We will analyze the separation ranks brought forth by convolutional arithmetic circuits. In particular, we focus on the deep network presented in sec. 3 (example 2), and study the dependence of its separation ranks on the input partition (I, J). The following claim (proven in Cohen and Shashua (2017)) links the network's separation ranks to its grid tensors (defined in sec. 3):

Claim 7 *Let $f(\cdot)$ be a function realized by a convolutional arithmetic circuit, and let $\mathcal{A}$ be its corresponding grid tensor. For any input partition (I, J), $sep(f; I, J)$ – the separation rank of $f(\cdot)$ with respect to (I, J), is equal to $rank [\![\mathcal{A}]\!]_{I,J}$ – the rank of $\mathcal{A}$ when matricized with respect to (I, J).*

Claim 7 opens the door to an analysis of separation ranks brought forth by convolutional arithmetic circuits through the hierarchical decompositions that represent their grid tensors (see sec. 3). In the case of the deep network under consideration, the corresponding hierarchical tensor decomposition is HT (see example 2). The matricization ranks it gives rise to are characterized in the theorem below (see Cohen and Shashua (2017) for proof):

Theorem 8 *The maximal rank of tensors generated by HT decomposition, when matricized with respect to a partition (I, J), is exponentially high if (I, J) meets certain conditions, and polynomial (or even linear) otherwise.*

Given claim 7, theorem 8 immediately leads to the following corollary:

Corollary 9 *A deep convolutional arithmetic circuit can realize exponentially high separation ranks for certain input partitions, whereas for others, it supports separation ranks that are no more than polynomial (or even linear) in network size.*

Corollary 9 directly relates to inductive bias (see sec. 2) – it states that ***a deep network can effectively model strong interactions between some input regions, whereas between others it cannot***. Put differently, there are certain, favored interactions that a deep network can model with reasonable size, and on the hand, unfavored interactions that can only be modeled if the network is unfeasibly large. Apparently, ***what determines which interactions are favored is the geometry of the network's pooling windows***. Standard contiguous windows favor interactions between regions that are highly intertwined (see illustration in fig. 4(a)), reflecting an assumption by which nearby

---

9. In a statistical setting, where $f(\cdot)$ is a probability density function, separability with respect to (I, J) corresponds to statistical independence between $\{x_i\}_{i \in I}$ and $\{x_j\}_{j \in J}$.
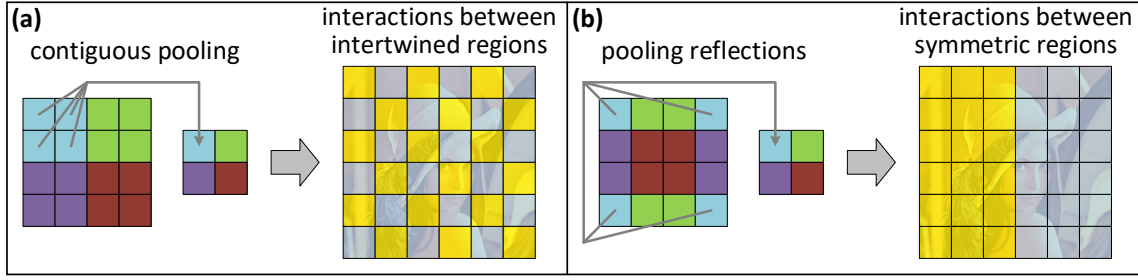
Figure 4: Best viewed in color. Illustrations of different pooling geometries and the type of interactions they favor (see sec. 5). *(a)* Standard contiguous windows – favor interactions between regions that are highly intertwined, complying with our intuition regarding the statistics of natural images (nearby pixels more correlated than ones that are far apart). *(b)* Windows that pool together reflecting elements – favor interactions between symmetric regions; may be useful for example when detecting symmetry in medical images.

input elements (*e.g.* image pixels) are more correlated than ones that are far apart. This explains why the type of pooling geometry most commonly employed in practice is in fact suitable for the kind of data convolutional networks are most frequently applied to (natural images). More importantly, ***by modifying pooling geometry one is able to control the type of interactions a network favors, and thereby tailor it to data that departs from the usual domain of natural imagery*** (see illustration in fig. 4(b)). This is demonstrated empirically in Cohen and Shashua (2017), with both convolutional arithmetic circuits and convolutional rectifier networks.

6. Efficiency of Overlapping Operations *(Sharir and Shashua (2017))*

In this section we treat question 3 in sec. 2.1. Specifically, focusing on the deep network presented in example 2, whose hidden convolution and pooling windows do not overlap, we ask whether introduction of overlaps into the latter can lead to expressive efficiency.

Recall from sec. 3 that each hidden layer in the deep network consists of convolution followed by pooling, where the convolution has receptive fields $1{\times}1$, and the pooling decimates feature maps via non-overlapping windows. We may view the convolution-pooling pair as a single unified operation whose receptive fields and strides match those of the pooling windows (see fig. 5(a,b)). This unified operation is referred to as a *generalized convolution*, signifying the fact that it would have been a standard convolution if pooling was based on summations instead of products. Given the generalized convolution viewpoint, a natural way to introduce overlaps into the network is by reducing strides. We consider the case of stride 1 across all generalized convolutions (see fig. 5(c)), and refer to the resulting model as the *overlapping network*.

We would like to show that the overlapping network is expressively efficient with respect to the original (non-overlapping) one. In accordance with the definition of expressive efficiency (sec. 2), this calls for establishing two propositions:

(i) Any function realized by the original network can be replicated by the overlapping one with no more than a linear growth in size

(ii) There exist functions realized by the overlapping network that cannot be replicated by the original one unless the latter's size is allowed to grow super-linearly
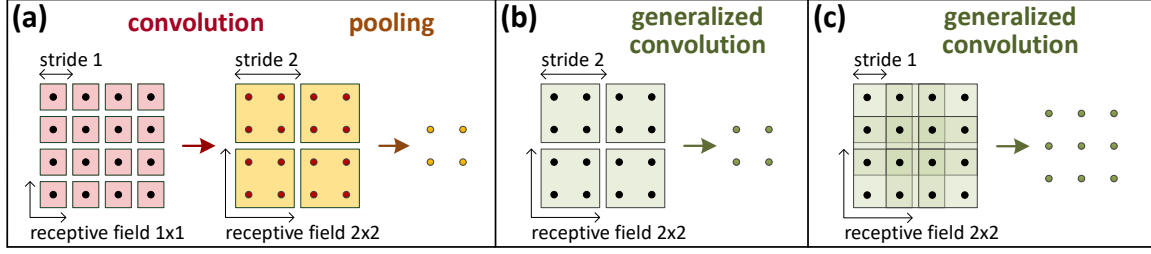
Figure 5: Best viewed in color. Illustrations of receptive fields and strides. *(a)* $1{\times}1$ convolution followed by non-overlapping pooling – a hidden layer as defined by the baseline convolutional arithmetic circuit architecture (fig. 2). *(b)* Unification of the convolution-pooling pair in (a) into a single operation named generalized convolution. Since the convolution in (a) is $1{\times}1$, the receptive fields and strides of the generalized convolution match those of the pooling windows. *(c)* Generalized convolution with overlaps – obtained by reducing the strides in (b) to 1.

Proposition (i) follows from the fact that the overlapping network reduces to the original one if we zero out an appropriately chosen subset of its weights. For proposition (ii), we recall that grid tensors of functions realized by the original network are given by HT decomposition (see example 2), and that by theorem 8, there exist partitions (I, J) under which matricizations of tensors generated by HT decomposition have ranks that are no more than polynomial. Taken together, these two findings imply that there exist partitions under which matricization ranks of grid tensors realized by the original network are no more than polynomial (in network size). Denote by $\mathbf{P}$ the set of such partitions, and consider the following theorem (proven in Sharir and Shashua (2017)):

Theorem 10 *There exist partitions in $\mathbf{P}$ under which matricizations of grid tensors realized by the overlapping network have exponentially high ranks.*

Theorem 10 implies that there exist partitions under which grid tensor matricization ranks are much higher with the overlapping network than they are with the original one. More precisely, the original network would have to be exponentially large in order to replicate ranks brought forth by the overlapping one. By this we establish proposition (ii) above, and prove that *a deep convolutional network with overlapping operations can be exponentially expressively efficient with respect to the same network without overlaps*.

7. Inductive Bias of Layer Widths *(Levine et al. (2017))*

In this section we address question 4 in sec. 2.1. Namely, we study the relation between the width of (number of channels in) each layer in a convolutional network, and the network's ability to model interactions among regions of its input. Our analysis is based on concepts and tools from the world of quantum physics.

A quantum system comprising N particles is typically represented by a *quantum many-body wave function*, which for our purposes may simply be thought of as a function over N variables. A key property of the system, with broad physical implications, is the amount of interaction between different sets of particles. Interactions are quantified via *quantum entanglement measures* (see Plenio and Virmani (2007) for an introduction) – quantities computed from the many-body
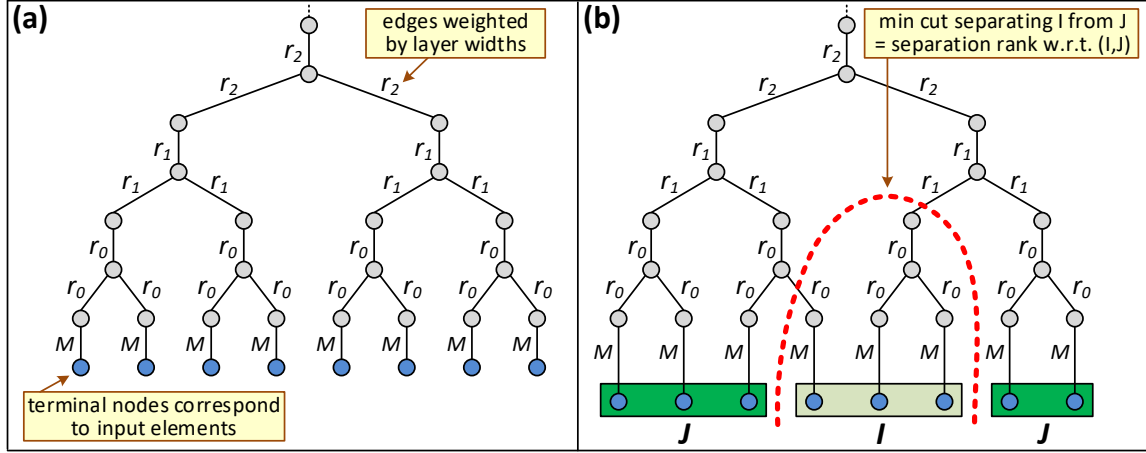
Figure 6: Best viewed in color. *(a)* Illustration of tensor network (graph) corresponding to the deep convolutional arithmetic circuit presented in example 2. Edges in the graph are weighted by the circuit's layer widths, and a set of terminal (degree-1) nodes corresponds to its input elements. *(b)* Illustration of the result in theorem 11 – for any partition of terminal nodes into groups I and J, the minimal cut separating the two groups is equal to the separation rank with respect to (I, J) supported by the convolutional arithmetic circuit.

wave function. There are different types of entanglement measures, for example entanglement entropy, geometric measure and *Schmidt number*. The latter was shown in Cohen and Shashua (2017) to be exactly equivalent to separation rank, as defined in sec. 5.

For simulative purposes, quantum many-body wave functions are usually realized via computational constructs named *tensor networks*. While an introduction to tensor networks is beyond our scope (the interested reader is referred to Orús (2014)), we note here that these can be viewed as graphs in which nodes correspond to tensors (multi-dimensional arrays), edges correspond to tensor modes (axes), and each edge is weighted by the length of its respective mode. An important class of results relates minimal cuts in the graph underlying a tensor network, to the quantum entanglement measures of many-body wave functions it can realize. Such results are used by physicists to design tensor networks in accordance with the needs of quantum systems to be modeled.

Returning to the realm of convolutional networks, a clear analogy arises – we are also concerned with functions over many local elements (*e.g.* image pixels or audio samples), and are interested in being able to model the required interactions between them (thereby adhering to the inductive bias – see sec. 2). What opens the door to utilization of tools from quantum physics is the fact that convolutional arithmetic circuits (fig. 2) can be cast as tensor networks. In the tensor network corresponding to a convolutional arithmetic circuit, edges are weighted by layer widths, and there exists a set of terminal (degree-1) nodes corresponding to the network's input elements (see fig. 6(a)).

With the connection to quantum physics in place, we rely on the analysis of Cui et al. (2016), and derive a result characterizing separation ranks (Schmidt entanglements) of a convolutional arithmetic circuit in terms of minimal cuts in its corresponding tensor network (see Levine et al. (2017) for proof):

Theorem 11 *Let $\mathcal{C}$ be a convolutional arithmetic circuit, and let Θ be its corresponding tensor network. For any input partition (I, J), the highest separation rank that may be realized by $\mathcal{C}$ with*
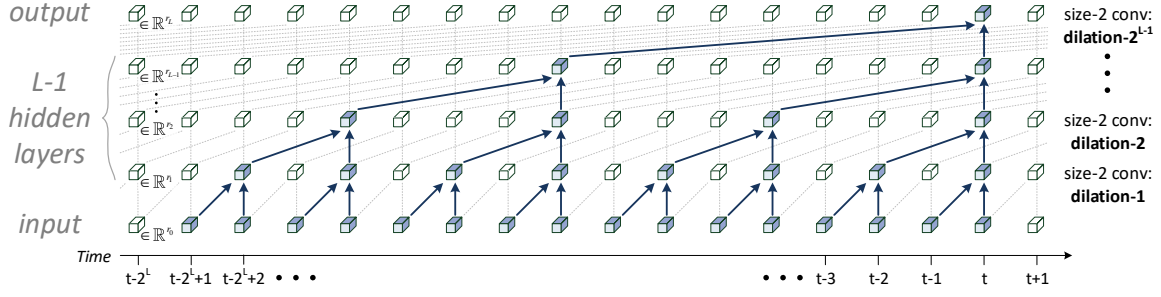
Figure 7: Baseline dilated convolutional network – underlies Google's WaveNet

respect to (I, J), is equal to the minimal (multiplicative) cut in Θ separating terminal nodes of I from those of J (see illustration in fig. 6(b)).

Taking into account the fact that edges in Θ are weighted by widths of (number of channels in) layers in $\mathcal{C}$, theorem 11 can be used to tailor layer widths so as to optimize separation ranks (interactions) of interest. Namely, given a convolutional arithmetic circuit with a fixed computational budget, an effective approach for distributing layer widths across the network is to maximize minimal cuts of input partitions for which we would like to model strong interactions. We head on in Levine et al. (2017) and focus on the deep network presented in example 2, showing that **widths of deep layers are important for modeling long-range interactions, whereas for short-range interactions, widths of early layers are those that matter**. This is demonstrated empirically with convolutional rectifier networks, exemplifying once again that analyses carried out with convolutional arithmetic circuits produce practical conclusions that are applicable to other types of convolutional networks as well.

8. Efficiency of Interconnectivity *(Cohen et al. (2017))*

In this section we treat question 5 in sec. 2.1, which concerns the ability of connectivity schemes to introduce expressive efficiency over the classic feed-forward (chain) approach. As opposed to our previous analyses, in which the general tendency was to consider 2D convolutional networks operating on images, we focus here on 1D networks. Specifically, we treat *dilated convolutional networks* operating on sequences. Dilated convolutional networks are a family of models gaining increased attention in the deep learning community. In particular, they form the basis of Google's WaveNet (van den Oord et al. (2016)) and ByteNet (Kalchbrenner et al. (2016)) models, which provide state of the art performance in audio and text processing tasks.

The dilated convolutional network we consider as baseline is the one underlying WaveNet, depicted in fig. 7. It is a 1D convolutional network without pooling, whose convolutional filters are dilated, *i.e.* incorporate gaps between their elements. Each layer is characterized by a different dilation, twice as large as that of its preceding layer. As before, we study functions realized by networks through the hierarchical decompositions that represent their grid tensors (see sec. 3). In the case of the baseline dilated convolutional network, the hierarchical decomposition adheres to a tree over tensor modes as illustrated in fig. 8(a). Modifying the structure of this tree yields a hierarchical decomposition that corresponds to a network with modified dilations throughout its layers – see illustration of a particular example in fig. 8(b).

For the analysis of networks with different connectivity schemes, we introduce the notion of *mixed tensor decompositions*. Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be two dilated convolutional networks whose cor-
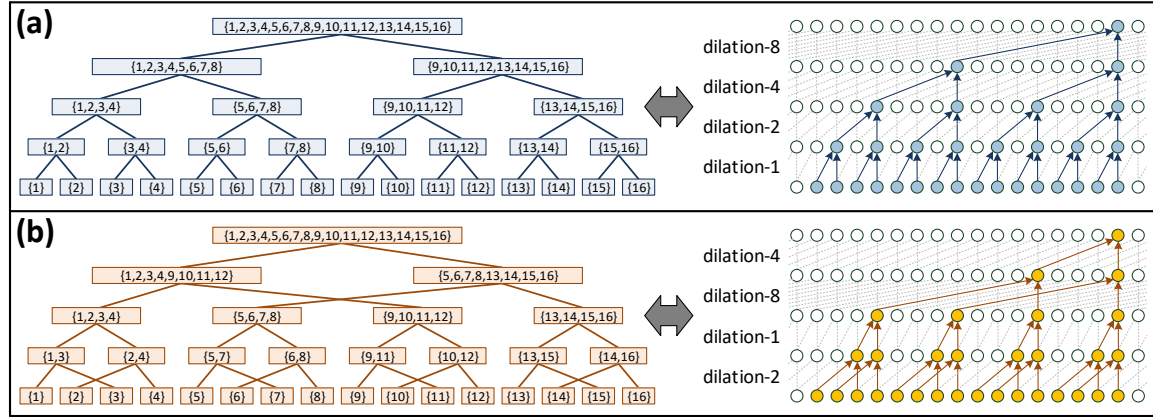
16

Figure 8: Illustrations of trees over tensor modes and the corresponding dilated convolutional networks. Different trees give rise to different hierarchical tensor decompositions, which represent the grid tensors of networks with different dilations throughout their layers. *(a)* Tree corresponding to baseline network (fig. 7). *(b)* Alternative tree corresponding to a network with alternative dilations.

responding hierarchical tensor decompositions are based on mode trees $\mathcal{T}_1$ and $\mathcal{T}_2$ (respectively). The mixed tensor decomposition of $\mathcal{T}_1$ and $\mathcal{T}_2$ runs their hierarchical decompositions in parallel, while exchanging tensors at different points along the way. It represents the grid tensors of a mixed network $\mathcal{M}$, obtained by interconnecting the intermediate layers of $\mathcal{N}_1$ and $\mathcal{N}_2$.

We would like to show that $\mathcal{M}$ is expressively efficient with respect to $\mathcal{N}_1$ and $\mathcal{N}_2$, thereby exemplifying the ability of interconnectivity to introduce efficiency. As discussed in sec. 2, this requires proving two propositions:

 (i) Any function realized by $\mathcal{N}_1$ or $\mathcal{N}_2$ can be replicated by $\mathcal{M}$ with no more than linear growth in size

 (ii) There exist functions realized by $\mathcal{M}$ that cannot be replicated by $\mathcal{N}_1$ or $\mathcal{N}_2$ unless their size is allowed to grow super-linearly

Proposition (i) follows from the fact that the mixed network $\mathcal{M}$ reduces to one of the networks it comprises ($\mathcal{N}_1$ or $\mathcal{N}_2$) if we set the opposite network's weights to zero. For proposition (ii), we compare matricization ranks under the hierarchical decompositions of $\mathcal{T}_1$ and $\mathcal{T}_2$, to those brought forth by their mixture. This results in the following theorem (see Cohen et al. (2017) for proof):

Theorem 12 *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be different trees over tensor modes, and consider the hierarchical decompositions they give rise to. These decompositions must grow (in terms of the number of intermediate tensors – sec. 3) at least quadratically to replicate tensors generated by their mixture.*

From a network perspective, theorem 12 translates to:

Corollary 13 *Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be different dilated convolutional networks, and let $\mathcal{M}$ be a network obtained by interconnecting their intermediate layers. $\mathcal{M}$ realizes functions that cannot be replicated by $\mathcal{N}_1$ or $\mathcal{N}_2$ unless these are at least quadratically larger.*

We conclude that ***with dilated convolutional networks, interconnectivity brings forth expressive efficiency***. Moreover, even a single connection between intermediate layers of different networks

already leads to a quadratic gap, which in large-scale settings typically makes the difference between a model that is practical and one that is not. Empirical evaluation of the analyzed models (carried out in Cohen et al. (2017)) demonstrates how adding connections between intermediate layers of different networks improves accuracy, with no additional cost in terms of computation or model capacity. This serves as yet another indication that in general, expressive efficiency and improved accuracies go hand in hand.

9. Conclusion

Expressive efficiency and inductive bias are the primary notions for formally reasoning about expressiveness – the driving force behind convolutional networks. Perhaps more important than their role in formalizing common beliefs and explaining empirically observed phenomena, is the potential of expressive efficiency and inductive bias to provide new tools for network design. Expressive efficiency can be viewed as the enhancement of a network's expressiveness, whereas inductive bias corresponds to making better use of expressive resources given the needs of a task at hand. Mounting empirical evidence shows time and time again that both procedures directly lead to improved performance (accuracies in particular).

Through an equivalence to hierarchical tensor decompositions, we analyzed the expressive efficiency and inductive bias of various architectural features in convolutional networks. Specifically, we studied the effects of network depth, layer widths, geometry of pooling windows, overlapping convolutions, and interconnectivity schemes. The results derived are not only explanatory – they provide concrete steps for controlling expressive efficiency and inductive bias. For example, guidelines are given for setting layer widths and pooling geometries in accordance with input correlations one wishes to model. We hope the series of works reviewed in this paper will serve as a first step towards extensive use of hierarchical tensor decompositions for more principled convolutional network design.

Acknowledgments

References

Gregory Beylkin and Martin J Mohlenkamp. Numerical operator calculus in higher dimensions. *Proceedings of the National Academy of Sciences*, 99(16):10246–10251, 2002.

Nadav Cohen and Amnon Shashua. Convolutional rectifier networks as generalized tensor decompositions. *International Conference on Machine Learning (ICML)*, 2016.

Nadav Cohen and Amnon Shashua. Inductive bias of deep convolutional networks through pooling geometry. *International Conference on Learning Representations (ICLR)*, 2017.

Nadav Cohen, Or Sharir, and Amnon Shashua. Deep simnets. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016a.

Nadav Cohen, Or Sharir, and Amnon Shashua. On the expressive power of deep learning: A tensor analysis. *Conference On Learning Theory (COLT)*, 2016b.

Nadav Cohen, Ronen Tamari, and Amnon Shashua. Boosting dilated convolutional networks with mixed tensor decompositions. *arXiv preprint arXiv:1703.06846*, 2017.

Shawn X Cui, Michael H Freedman, Or Sattath, Richard Stong, and Greg Minton. Quantum max-flow/min-cut. *Journal of Mathematical Physics*, 57(6):062206, 2016.

Olivier Delalleau and Yoshua Bengio. Shallow vs. deep sum-product networks. In *Advances in Neural Information Processing Systems*, pages 666–674, 2011.

Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. *arXiv preprint arXiv:1512.03965*, 2015.

W Hackbusch and S Kühn. A New Scheme for the Tensor Representation. *Journal of Fourier Analysis and Applications*, 15(5):706–722, 2009.

Wolfgang Hackbusch. *Tensor Spaces and Numerical Tensor Calculus*, volume 42 of *Springer Series in Computational Mathematics*. Springer Science & Business Media, Berlin, Heidelberg, February 2012.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*, 2016.

Tamara G Kolda and Brett W Bader. Tensor Decompositions and Applications. *SIAM Review ()*, 51(3): 455–500, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, pages 1106–1114, 2012.

Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10), 1995.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

Yoav Levine, David Yakira, Nadav Cohen, and Amnon Shashua. Deep learning and quantum entanglement: Fundamental connections with implications to network design. *arXiv preprint arXiv:1704.01552*, 2017.

Xingyi Li, Fuxin Li, Xiaoli Fern, and Raviv Raich. Filter shaping for convolutional neural networks. *International Conference on Learning Representations (ICLR)*, 2017.

Hrushikesh Mhaskar, Qianli Liao, and Tomaso Poggio. Learning real and boolean functions: When is deep better than shallow. *arXiv preprint arXiv:1603.00988*, 2016.

Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2924–2932, 2014.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

Román Orús. A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Annals of Physics*, 349:117–158, 2014.

Razvan Pascanu, Guido Montufar, and Yoshua Bengio. On the number of inference regions of deep feed forward networks with piece-wise linear activations. *arXiv preprint arXiv*, 1312, 2013.

MARTIN B Plenio and SHASHANK Virmani. An introduction to entanglement measures. *Quantum Information and Computation*, 7(1):001–051, 2007.

Tomaso Poggio, Fabio Anselmi, and Lorenzo Rosasco. I-theory on depth vs width: hierarchical function composition. Technical report, Center for Brains, Minds and Machines (CBMM), 2015.

Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 689–690. IEEE, 2011.

Frank Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, DTIC Document, 1961.

Or Sharir and Amnon Shashua. On the expressive power of overlapping operations of deep networks. *arXiv preprint arXiv:1703.02065*, 2017.

Or Sharir, Ronen Tamari, Nadav Cohen, and Amnon Shashua. Tensorial mixture models. *arXiv preprint arXiv:1610.04167*, 2017.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. *CVPR*, 2015.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

Matus Telgarsky. Representation benefits of deep feedforward networks. *arXiv preprint arXiv:1509.08101*, 2015.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR abs/1609.03499*, 2016.

Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.