Neural Network Layer Algebra: A Framework to Measure Capacity and Compression in Deep Learning

Alberto Badías and Ashis G. Banerjee, Senior Member, IEEE

Abstract—We present a new framework to measure the intrinsic properties of (deep) neural networks. While we focus on convolutional networks, our framework can be extrapolated to any network architecture. In particular, we evaluate two network properties, namely, capacity, which is related to expressivity, and compression, which is related to learnability. Both these properties depend only on the network structure and are independent of the network parameters. To this end, we propose two metrics: the first one, called layer complexity, captures the architectural complexity of any network layer; and, the second one, called layer intrinsic power, encodes how data are compressed along the network. The metrics are based on the concept of layer algebra, which is also introduced in this article. This concept is based on the idea that the global properties depend on the network topology, and the leaf nodes of any neural network can be approximated using local transfer functions, thereby allowing a simple computation of the global metrics. We show that our global complexity metric can be calculated and represented more conveniently than the widely used Vapnik–Chervonenkis (VC) dimension. We also compare the properties of various state-of-the-art architectures using our metrics and use the properties to analyze their accuracy on benchmark image classification datasets.

Index Terms—Artificial neural networks, computational complexity, deep learning, image classification.

I. INTRODUCTION

Deep learning models have a good ability to deal with challenging problems that are too complex for us to explain by means of simple and deterministic laws in closed forms. Some examples include the extraction of relevant information from images [1], image inpainting and denoising [2], natural language processing [3], the creation of music [4], and learning how to play a 3-D role-playing game properly [5].

In particular, if we consider the image processing problem, deep learning methods deal with data in their natural forms (images) and try to make the system capable of automatically extracting semantic information. In other words, we are not only dealing with the problem of prediction (here, classification) but also discovering the representations needed to extract discriminative features from raw data. It is, therefore, not only a question of obtaining a target function capable of classifying data that is already in a defined space but also a question of defining that space simultaneously.

The latter question is difficult to answer since the true dimension of the problem space is usually unknown and depends on both the input and output data. For example, the number of classes (labels) is of importance to the classifier. Experience tells us that some of the network structures are focused on the extraction of relevant descriptors, such as convolutions for images and recurrent units for time series data, while other structures have a direct prediction-related objective, such as dense groups of fully connected layers. A deep neural network for any challenging task usually includes both types of structures, creating a complex architecture of information flow channels. Since we have no a priori idea about the dimensionality of the problem space, we usually tend to create a large set of interconnected layers to achieve, as best as possible, high accuracy in solving a given task. Although it is true that people try to reduce the network size once the desired accuracy is achieved, they typically do not aim to attain any control based on the problem dimensionality.

For the moment, we think there are not many ways to estimate how big, small, or oversized a network is. In fact, to the best of our knowledge, there is no tool to compare the different architectures systematically (for example, we think that the sum of the parameters is not a representative comparison measure). While approximation and generalization errors are widely used to evaluate supervised learning models, we base our evaluation on two related but slightly different metrics of expressivity and learnability [6], as they pertain directly to the ability of networks to encode complex functions and identify the underlying patterns in the input data.

Accordingly, we draw inspiration from the capacity and compression of (data) flow in networks to define two corresponding metrics, complexity and intrinsic power, which are related to expressivity and learnability, respectively. Specifically, our complexity metric aims to encode the expressive capacity of a neural network without assuming any knowledge of the input data distribution or the network parameters. This metric is analogous to the VC dimension and bias-variance
analysis, which are commonly used to obtain a desired tradeoff between approximation (good data fitting) and generalization (low test error). The intrinsic power metric, on the other hand, is designed to represent the ability of a neural network to learn effectively, i.e., get trained properly, regardless of the data distribution. We first define our metrics on the individual network layers and then combine them to obtain (scalar) global cumulative values based only on the network topology. We show that these values are practically computable for extremely large networks, and are useful in estimating the ease of training and explaining prediction accuracy on benchmark image classification problems.

The rest of this article is organized as follows. Section II summarizes the related work on the other methods that are used to measure certain properties of neural networks. Section III presents an overview of our method. In Section IV, the net properties and metrics are introduced. Section V explains the evaluation of these metrics for a set of known local layers. Section VI defines the concept of layer algebra, a necessary tool to take into account the topology of the architecture into the metrics. Section VII describes the experiments carried out to assess the usefulness of the metrics. Finally, Section VIII provides some concluding remarks and outlines future work directions.

II. RELATED WORK

Although there has been a lot of interest in neural networks in the last decade, largely due to the computational power of modern devices, the theoretical principles were developed earlier. For example, we know that a neural network with a hidden layer is a universal approximator, which can approximate any continuous function on compact subsets of \( \mathbb{R}^n \) arbitrarily well, under some assumptions on the activation functions [7]. However, some aspects of neural networks, especially pertaining to deep architectures, are unknown to the scientific world.

There are many works that define the basic properties of neural networks and propose methods to estimate these properties. To this end, some researchers demonstrated properties, such as learning ability, generalization, and robustness [6], [8]. Baldi and Vesprynin [9] defined the capacity of a network as the binary logarithm of the number of functions it can implement.

A lot of attention has been given to estimating network complexity in particular since it is one of the most interesting but difficult-to-measure properties. An example is the development of a complexity theory based on the computability of functions by neural networks of a given type and size [10]. Subsequently, Montufar et al. [11] estimated the complexity of functions that are computable by deep feedforward neural networks with linear activation functions by observing the number of generated linear regions. Others proved that an upper bound to the number of linear regions scales exponentially with network depth but polynomially with width [12], [13], [14], while some works notice that size and depth of the networks affect the ability to approximate real functions [15]. Song et al. [16] derived a theoretical lower bound to imply that efficient training algorithms require stronger assumptions on the target functions and input data distributions than Lipschitz continuity and smoothness.

Cohen et al. [17] analyzed the complexity of the functions that can be expressed by a network using tensor analysis, thereby comparing deep and shallow architectures using factors such as locality, sharing, and pooling to establish an equivalence between the networks and hierarchical tensor factorization. Raghu et al. [18] focused on overall network expressivity and measured it as the length of the trajectory that captures the change in the output as the input sweeps along a one-dimensional path. They proved that expressivity complexity grows exponentially with network depth [19].

Another way of estimating the complexity of a network is by means of its sensitivity to input perturbations. Accordingly, Novak et al. [13] distinguished different network models based on their sensitivity and came up with a robustness measure through their input–output Jacobian. Alternatively, several researchers have connected neural networks to dynamical models to estimate the complexity from a theoretical perspective by establishing a parallel between the network architectures and stochastic partial differential equations [20], [21], [22]. A different approach, like Zhang et al. [23], is focused on the complexity estimation of recurrent neural networks by using the concept of the cyclic graph to define recurrent depth or recurrent skip coefficient to capture how rapidly information propagates over time.

Other representative efforts include the work by Bartlett and Mendelson [24] to compute the complexity of a classification function using the Rademacher and Gaussian complexities. Bianchini and Scarselli [25] used the topology of the layers (specifically, Betti number) to estimate the complexity of the network. On a related note, Rieck et al. [26] developed a topological complexity measure, called neural persistence, based on the use of persistent homology. Barron and Klusowski [27] determined the theoretical accuracy by measuring the statistical risk as defined by the mean squared prediction error. They also proposed methods to estimate metric entropy, in addition to the complexity and statistical risk, in [28].

From a more practical standpoint, it is common to quantify network complexity using the number of training parameters, the cost of memory storage, or the total number of operations required (FLOPS) to evaluate the data during inference [29]. Another simple way to compare the complexity of neural networks is by empirically measuring the time taken by a computer to perform the complete computation. However, as might be expected, this depends on the computer being used, the dataset, and the implementation itself. Other approaches directly measure the complexity based on the depth of the network as given by the number of layers. We believe that this is not an appropriate metric since a layer can be very complex, involving many operations, or it can be very simple, with only an activation function. In fact, Zhang et al. [23] demonstrated interesting performance characteristics of recurrent neural networks by defining depth as the maximum number of nonlinear transformations from the inputs to the outputs.

Beyond property estimation, researchers have compared deep architectures with shallow architectures, noting that deeper networks have better learning capabilities for really
Challenging problems [30]. They have proved that deep networks can approximate the class of compositional functions with the same accuracy as shallow networks but with an exponentially lower number of training parameters [31], as well as Vapnik-Chervonenkis (VC)-dimension [32]. They have also tried to answer how width affects the expressiveness of neural networks, thereby obtaining a universal approximation theorem for width-bounded rectified linear activation unit (ReLU) networks [33].

### III. OVERVIEW OF THE PROPOSED METHOD

As we have seen, several methods have been devised to estimate the complexity of neural networks. Here, we propose an alternate method that measures the properties of a network architecture precisely using specific values, rather than limits or bounds. On many occasions, we talk about deep networks, and different architectures are compared based on the number of layers. However, we think this is not the best approach since, in addition to the number of layers, it is necessary to analyze how each (local) layer affects the global complexity. Similarly, when the layers have large widths, it is possible for a network to be shallow but have many parallel channels that enable much greater complexity than a deeper network. Therefore, we believe that a rigorous mathematical framework is necessary to standardize neural network architectures and allow fair comparisons among the different architectures.

Our goal is to establish a methodology to compare the different architectures encompassing the concepts of depth, width, or resolution [34]. To do so, we present a set of metrics to estimate the local values of our properties in each layer (see Fig. 1). Subsequently, we apply the notion of layer algebra to obtain the global values. We propose two properties, capacity, and compression, for which we use two metrics: the first measures the layer complexity of a neural network, and the second measures the intrinsic power of the data. Note that our metrics are related to the expressivity and learnability properties established in [6], which affect the approximation and generalization errors on supervised learning problems.

According to our hypotheses, two networks with the same architecture but different parameters would obtain the same values of our proposed metrics. This happens because our measurements are only based on the operations or transformations that the data undergoes throughout the network. While we think it is better to use analytical functions to estimate our metrics, we rely on a data-driven approach to compute the values for certain network layers that use activation functions. We believe that our method could be used to predict if a network would be accurate enough, learnable, or computationally intensive before actually training or testing the network on large-scale datasets.

### IV. NET PROPERTIES

In this work, we focus on the steady-state properties of a neural network. We are not going to model the learning process from the transitory viewpoint of optimizing a set of parameters, such as weights and biases. To do this, we look at the changes that the data undergoes as it flows through the network based on its morphology.

Let us suppose we have a dataset \(X \in \mathbb{R}^{N \times 1}\) with \(N\) as the number of samples and \(I\) as the dimension of the input data. Let \(Y \in \mathbb{R}^{N \times C}\) be the output dataset, with \(C\) being the size of the output data. \(C\) is a continuous set if we are facing a regression problem or categorical if it is a classification problem. The changes applied by a neural network on the input data to obtain the output (assuming supervised learning) can be defined as a mapping function \(f(X) : X \rightarrow Y\). Depending on the problem complexity, the function \(f(X)\) may need to be highly nonlinear, where this global function is divided into a set of \(L\) compositional local functions \(f^l(X)\) that are linked in a multiplicative way to recover the function \(f(X)\). Note that \(L\) is often referred to as the number of layers. For example, in the image classification context, \(f^l\) comprises both linear transformations and componentwise nonlinear functions, also called activation functions. \(f^l\) can then be expressed in a matrix form as \(f^l(x) = \sigma(W^l x + b^l)\), where \(W^l\) is the set of linear coefficients or weights of layer \(l\), \(b^l\) is the set of independent terms or bias, and \(\sigma\) is the activation function.

Here, we adopt a somewhat different approach and model how the data evolves through the network to establish parallelism with dynamic systems. Accordingly, we define the following series of differential equations for any network property \(Z\) evaluated locally at layer \(l\) as:

\[
\frac{dz^l}{dl} = g^l(Z, \phi^l), \quad l = 1, \ldots, L.
\]

Note that \(Z\) is independent of the network parameters, but depends on the local transformation \(\phi^l\) to which the data is subjected. The form of the function \(g^l\), which depends on these transformations \(\phi^l\), must be set for each type of layer and is defined in Section V.

Subsequently, we define the cumulative property \(Z^{L_N}\) in a layer \(l = L_N\) as the sum of the transformations applied from the first to the current layer after going through all the previous layers as

\[
Z^{L_N} = \int_1^{L_N} g^l(Z, \phi^l) dl.
\]

Again, this is a scalar value that defines the cumulative evolution of the data at any point of the network. Finally,
we obtain the global cumulative property of the network, $Z_{GC}$, when it is evaluated in the output layer ($l = L$). A different global metric is also proposed, called the global sum property $Z_{GS}$. It is estimated as the direct sum of all of the local property values, but without taking into account the topology of the architecture. Mathematically

$$Z_{GS} = \sum_{l=1}^{L} \phi^l.$$ \hfill (3)

The proposed global metrics are easily estimated in a classical feedforward architecture, such as the multilayer perceptron, by adding the local values consecutively. However, parallel paths and shortcut (or skip) connections often appear in many modern architectures. Hence, the network topology must be taken into account. That is why we define the concept of layer algebra to estimate the global metric values by considering all the information paths (see Section VI). In the end, we obtain unique values of the capacity and compression of a network to model how the data flows through the network regardless of the learned (optimized) values of the network parameters.

V. LOCAL PROPERTIES ESTIMATION

Now that we have introduced the properties, in this section, we first present their general definitions before specifying them for some of the most common layers in a neural network. These properties are local and their contributions should be properly combined to obtain the global values (see Section VI).

A. Generic Definitions

Although this section defines the presented metrics in a practical way for the most used operations (convolution, pooling, fully connected layers, and so on), it is not possible to define them for all the existing operations. For this reason, we define the general theoretical framework so that these metrics can be extrapolated to other types of operations.

Let $A \in \mathbb{R}^{D \times P}$ be a tensor object at the input of a local operation (input data) (see Fig. 2) and $B \in \mathbb{R}^{Q \times O}$ be the tensor object obtained after applying the operation (output data). As defined earlier, $\phi^l$ is the mapping (transformation) between the input and output such that $B = \phi^l(A)$ and $\phi^l : \mathbb{R}^{D \times P} \mapsto \mathbb{R}^{Q \times O}$. Any local metric, as applied to $\phi^l$, is then defined in two different ways, depending on whether the operation involves weights (kernel-type operation) or is an activation function.

If the tensor operation involves weighting an input signal (convolution, transpose convolution, pooling, or fully connected, among others), the intrinsic power, $p^l$, is defined as the ratio of the output space dimension $\Phi_{out}$, with respect to the input space dimension $\Phi_{in}$. Mathematically

$$p^l = \frac{\Phi_{out}}{\Phi_{in}}.$$ \hfill (4)

Here, the size of the output (input) space is analogous to the volume of data flow, which can be correlated with the flow energy, and, therefore, power. The denominator term acts as a normalizing factor so that the ratio captures the local data compression due to $\phi^l$.

If the operation is an activation function, the value of the intrinsic power is defined as the ratio between the variance of the output data with respect to the input data, where

$$p^l = \frac{\sigma_{out}^{2}}{\sigma_{in}^{2}}.$$ \hfill (5)

This definition resembles the physical relationship between the transport power and the amplitude of a wave, where power is proportional to the square of the amplitude. In our case, network data flow is analogous to wave transport and amplitude is captured through the data (sample) variance.

Some other techniques could be applied, like the well-known K-L divergence, to measure the differences between the input and output distributions. However, the activation functions apply abrupt changes to the input data, causing problems in estimating the differences between the two distributions as they are defined over very different ranges. Therefore, we decided to use the ratio of the outputs and inputs variances as a measure of the intrinsic power of an activation function. Furthermore, since we are not evaluating the transitory learning process, we assume that the addition of the bias term would produce a readjustment of the classifier to obtain the same accuracy. Hence, it is not included in $p^l$ computation.

The second property to estimate is the capacity of the convolution operation using the complexity metric. As in the case of intrinsic power, we make a distinction between kernel-based operations and activation functions and define complexity differently in the two cases.

If the operation involves weighting the input signal, as in the case of convolution, transpose convolution, pooling, and fully connected layer, we define complexity, $c^l$, as the logarithm of the size of the local (weighting) tensor operator. Therefore

$$c^l = \log_2(\Phi_{local})$$ \hfill (6)

where $\Phi_{local}$ is the size (dimension) of $\phi^l$. This definition ties the layered complexity to a scaled (logarithmic) form of the local capacity (expressivity) of the corresponding operator in a simple manner. Note that the scaling, which preserves monotonicity, is needed to ensure that the complexity values are $O(1)$, just like for all the other ratio-based local property estimates.

If the operation is an activation function, we simply interpret complexity as the inverse of its intrinsic power. Therefore

$$c^l = \frac{1}{p^l} = \frac{\sigma_{in}^{2}}{\sigma_{out}^{2}}.$$ \hfill (7)

This interpretation is based on the notion that the capacity of a nonlinear activation function can be estimated by the relative reduction in output variability due to the selection (high weighting) of specific input neurons.

As in the case of intrinsic power, we only consider the changes that imply differences in the distribution of the data, which ultimately translates into modifying the classification capacity of the network. For example, some linear transformations, such as bias addition, do not have any effect on the decision capacity of a classifier and are, therefore, not taken into account.
Fig. 2. Two-Dimensional convolution operation involving several filters illustrating local intrinsic power and complexity metrics.

Fig. 3. Intrinsic power explanation in 1-D convolutions using different kernel sizes.

**B. Specific Definition: Convolution**

We start with a simple 1-D convolution example to understand our proposed properties and then apply it to 2-D convolutions. The process can be generalized to higher dimensions.

Let us assume that a kernel of size \( K = 1 \) and a stride of \( S = 1 \) is used in a 1-D convolution. After applying the convolution to the data vector, the result is the same vector multiplied by the kernel element \( k \). In other words, it is a linear combination of the initial values. This does not affect the classification process, since the separation margins of the classifier would be adapted by \( k \) resulting in the same accuracy. Hence, we assume that the intrinsic power of our data does not change, and \( p_l = 1 \) (see Fig. 3, left).

However, if a different kernel size of \( K = 3 \) with a stride of \( S = 1 \) is used, we can interpret the output as a softening of the input values by means of a smoothing function with three coefficients (Fig. 3, right). This operation reduces the variability in the data. The output of (Fig. 3, right) is computed as the product of the kernel weights \( (k_1, k_2, k_3) \) and the input vector over the cell \( a_i \) and its \( K - 1 \) (two) neighbors. This can be considered as the intrinsic power of that cell divided by \( K \), as a result of applying the smoothness function. If we extrapolate this operation to the rest of the input data, we can estimate a variation of the power of the data, defined as the particularization of (4), giving the following equation:

\[
p'_l = \frac{\Phi_{out,k}}{\Phi_{in,k}} = \frac{C_o \cdot S_o}{K \cdot S_i}.
\]

Correspondingly, complexity is given by

\[
c'_l = \log_2(\Phi_{local,k}) = \log_2 K.
\]

Here, \( C_o \) is the number of output connections in the filter (typically equal to 1), \( S_o \) is the size of the output data \( (Q \times O) \) in Fig. 2), \( K \) is the size of the kernel \( (K_H \times K_W) \), and \( S_i \) is the size of the input data \( (D \times P) \). Note that \( S_o \) is related to the stride and depends on the size of the input data and padding.

In the same convolution layer, it is common to apply several filters at the same time, obtaining a third-order tensor. For example, we could apply \( n_k \) filters in the current layer so that the output after this layer would be a block of size \( n_k \) in the third dimension. Since we assume that the filters are completely independent with values other than zero, the intrinsic power of the data is multiplied by the number of filters. In other words, we are applying a smoothing function with each filter so that the power is reduced, but do it in parallel on all the filters so that the powers are added together. Therefore, the intrinsic power is written as

\[
p'_l = n_k \cdot p'_l.
\]

The local complexity must also be multiplied by the number of filters in the layer since the individual filter capacities have to be added up as well. Therefore

\[
c'_l = n_k \cdot c'_l.
\]

**C. Specific Definition: Transpose Convolution**

The intrinsic power of the transpose convolution is estimated using (8) and (10), with the difference that \( C_o \) is equal to \( K \) and the number of input connections, \( C_i \) is typically equal to 1. The expressions for complexity are the same as that for standard convolution since the local capacity is still equal to

\[
c'_l = n_k \cdot c'_l.
\]
the size of the convolution filter
\[ p' = \Phi_{out,k} = \frac{K \cdot S}{C_i \cdot S_i} \]  
\[ \phi = n_k \cdot p'_k \]  
\[ c'_k = \log_2 K; \quad c' = n_k \cdot c'_k. \]

D. Specific Definition: Pooling

Pooling is an operation that can be viewed as a specific form of convolution. For this reason, we apply the same equations to estimate the intrinsic power (8) and capacity (9) of the local layers. Although there are different types of pooling operations (i.e., average pooling or max pooling, among others), in this work, we apply the same equations based on the changes in the dimensions of the data to estimate the properties of these type of layers.

However, there is a type of pooling operation that is worth analyzing separately. This operation is known as global pooling, where the data are compressed along the width and height of the input matrix (in the case of 2-D operations). Based on our analysis, we interpret this operation as the application of a single global filter of the same size as the input data \((D \times P)\).

Hence, we continue using (8) and (9), but modify the filter term suitably.

E. Specific Definition: Fully Connected Layer

For this type of layer, we make use of (4) to define the local intrinsic power. Here, the output size is simply the number of neurons in the output layer, \(D_{out}\), and the input size is the number of neurons in the input layer, \(D_{in}\). Therefore
\[ p' = \frac{D_{out}}{D_{in}}. \]

Correspondingly, the local complexity is given by a particularization of (6) as
\[ c' = \log_2 (D_{out} \cdot D_{in}) \]

since the local operator is defined as a flattened weight matrix in \(\mathbb{R}^{D_{out} \times D_{in}}\).

F. Specific Definition: ReLU Activation

As we are not interested in the weight values, we assume that a ReLU activation function [35] removes half of the activations. This is an approximation, since the weights assume more positive values as the learning progresses, leading to nonzero-centered distributions. However, since we cannot estimate these variations for each network and dataset, we consider the intrinsic power value to be around 0.5.

The data-driven analysis leads us to the same conclusion (see the Supplementary Material), where we estimate the intrinsic power value for both normal and uniform data distributions. For this purpose, we apply the relation between the output–input variances to compare the data distribution before and after the activation layer. We directly consider the output–input relation as the local intrinsic power value, \(p' = 0.584\), and its inverse, \(c' = 1.713\), as the complexity of the ReLU activation function.

We can find other activation functions that are variants of ReLU (e.g., exponential linear unit (ELU), LeakyReLU [36], swish [37], and so on), some of which contain parameters that can be optimized. This fact means that it is challenging to estimate the properties of these types of activation functions since they depend on the data used and the learning step itself. Consequently, we approximate the intrinsic power and complexity with the same values used for the ReLU function.

G. Specific Definition: TanH Activation

Since this type of activation function is highly dependent on the input values, it is more difficult to estimate a theoretical value for our metrics. Therefore, we rely more on the data-driven analysis, setting a value of \(p' = 0.628\) for the local intrinsic power (see the Supplementary Material). The value of the local complexity is again estimated as the inverse of the local intrinsic power to be \(c' = 1.592\).

It is worth pointing out here that the gain of the activation function, related to the initialization values of the activations [36], [38], should not be confused with the idea proposed in our work. This gain is related to the stability of the activation distributions in deep networks, seeking convergence of the output values after crossing a large number of layers.

H. Specific Definition: Sigmoid Activation

Similar to the previous case, it is also difficult to obtain theoretical values of intrinsic power and complexity for this type of activation function. We, therefore, again make use of data-driven analysis to set the values of \(p' = 0.208\) and \(c' = 4.802\).

I. Specific Definition: Softmax Function

This type of function is often used in classification problems to obtain an output vector that estimates the probability of belonging to a specific class. The input to a softmax function is not bounded, while its output lies within the interval \((0,1)\). Hence, we can therefore, understand that this function normalizes the data to a posterior probability distribution [39].

The softmax activation function changes the data distribution substantially by scaling and centering the data using exponential functions. Hence, theoretical values of the intrinsic power and complexity are especially hard to estimate. That is why we again rely on the data-driven approach to obtain a local intrinsic power value of \(p' = 1.342e-05\) and a local complexity value of \(c' = 7.454e04\).

J. Specific Definition: Batch Normalization

Batch normalization is a procedure to improve the learning process and stabilize the activations along the network [40]. It can be interpreted as a statistical adjustment that centers and scales the range of the activation values. This implies that the data is not changed as far as the prediction accuracy of a classifier is concerned. Batch normalization implies a movement in the limits of the classifier, sometimes making the values more suitable for the activation functions, but without
modifying the relative internal variability of the data [41]. Therefore, even though it is proven to improve the training process [40], it does not modify the capacity of the network. It improves the training process from the point of view of data adequacy and optimization, which, as already mentioned, is outside the scope of this work. Therefore, we conclude that batch normalization does not modify the intrinsic power of the data nor adds complexity to the network.

K. Specific Definition: DropOut

Dropout is a technique to increase the effectiveness of the training process in deep networks by reducing overfitting through co-adaptation [42]. It consists of dropping out some random units during the training step. However, the network recovers the complete set of neurons during the inference step. This implies that the overall capacity of the network remains unchanged, meaning that both the intrinsic power and network complexity are unaffected.

VI. LAYER ALGEBRA

In Section III, we defined the local intrinsic power and complexity values for some of the most common layers of a neural network. However, to estimate the global properties we have to take into account how the information flows throughout the network. The goal of this section is to develop this process with the definition of a simple concept that we call layer algebra, understood as a set of simple operations to estimate the global properties of the network from the local values.

To better understand the idea, let us consider a control theoretic point of view, where a neural network is defined as $\mathcal{A}(X, \theta)$ and the contribution of each network layer $l$ is interpreted as a modification of the original signal $u$ introduced to the network. Through simple operations, such as additions and multiplications, the amount of information evolves as it flows through the network. In this way, each layer can be understood as a local transfer function $G(L_i)$ that modifies the input signal $u$ until finally obtaining the output value $y_{L_N} = u \cdot \prod_{i=1}^{L_N} G(L_i)$ in the layer $l = L_N$. Of course, the geometry of the neural network defines the operations to be carried out. For example, when the data passes through a layer, the process can be interpreted as the product of the input signal and the local function of the layer (see Fig. 4). In architectures with parallel channels or shortcut connections, as, for example, in the residual blocks of ResNets [43], the contribution of both the parallel signals of the block must be taken into account (see Fig. 5).

A. Intrinsic Power

To estimate the cumulative value of the intrinsic power of the data, we assume that the network entry power has a value $u = 1$. Note here the parallelism with control theory. The fact that the input value is unitary instead of encoding the dimensions of the training data (tensor for images) is because they are considered in the first transfer function ($G(L_1)$ in Fig. 4), corresponding to the first layer of the network. This implies that we are moving all the data transformations to the transfer functions estimated in any local layer, allowing us to measure the intermediate cumulative values of the intrinsic power of the data at any point of the network ($y_n$ in Fig. 4).

In simple feedforward architectures, like a multilayer perceptron (without parallel connections), the procedure is shown in Fig. 4. The value of the transfer function in any layer is equal to the local intrinsic power value, i.e., $G(L_i) = p^i$. The result of the cumulative intrinsic power after passing through the first layer is computed as the product of the transfer function $G(L_1)$ and the input intrinsic power, in this case, $u = 1$. Using the general nomenclature of layer algebra,
we define it as

\[ y_1 = u \cdot G(L_1). \quad (17) \]

Now, specifying for intrinsic power using the nomenclature defined in Section IV, the cumulative intrinsic power is expressed as

\[ P^{L_i} = P^{L_{i-1}} \cdot p^i. \quad (18) \]

The cumulative intrinsic power value is computed successively across all the layers until the global value is obtained.

In the case of an architecture with parallel paths, a slightly different analysis is necessary. For example, in the architecture shown in Fig. 5, a second branch appears in the output of layer \( l = L_1 \). According to our approach, this bifurcation implies that the power of the data is maintained in the input of the following layers, where \( y_1 = u_2 = u_3 \). Continuing with the example of Fig. 5, the output information of layers \( L_2 \) and \( L_4 \) are received at the input of layer \( l = L_5 \). Since we are measuring the intrinsic power as a form of entropy of the data, we consider the cumulative value of intrinsic power in the input of layer \( L_5 \) as the largest value of the previous outputs \( u_5 = \max(y_2, y_4) \), since the value is stronger.

Although it is true that the activation values for the outputs of multiple converging layers are added (for example, in a residual ResNet block), their powers can be very different. Some of the paths can make deep transformations on the data, whereas, others may copy the data without modifications. In other words, there is a mixing of information at different scales even though the data has the same size. For this reason, we believe that the highest value of intrinsic power should dominate the others. Therefore, at a path convergence point, we have

\[ P^{L_{i-1}} = \max(P^{L_{in,i}}) \quad (19) \]

where \( P^{L_{i-1}} \) is the value of intrinsic power before layer \( L_i \) and \( P^{L_{in,i}} \) is the set of intrinsic power values of the paths entering in layer \( L_i \). Through (19), we enforce consistency in the value of our metrics, since during the previous branching (bifurcation), we assumed duplication of the power by assuming a constant value of the intrinsic power, regardless of the number of paths that were created.

B. Complexity

In the case of complexity, again we first estimate the local values previously defined in Section V, depending on the type of layer. These values are represented by the transfer functions \( G(L_i) \) in Figs. 4 and 5, for the complexity.

The computation of the cumulative values of the complexity is carried out in the same way, through the product of a unit value of the input complexity with the local values of the layers that the data goes through. However, in the case of parallel paths (Fig. 5), the cumulative complexity is not computed as the maximum value of the inputs. Here, the cumulative complexity is estimated as the sum of all the values of the input paths for the layer \( l = i \)

\[ C^{L_{i-1}} = \sum_{j=1}^{N_{in}} (C^{L_{in,j}}) \quad (20) \]

where \( N_{in} \) is the number of inputs received by the layer \( i \).

In this case, we do not penalize the duplication of the complexity when branches appear, since we are measuring the capacity of the architecture, unlike intrinsic power that captures data compression.

VII. Experiments

In this section, we apply our metrics to four different experiments to demonstrate their usefulness. The first experiment is a comparison of a well-known metric (VC-dimension) with our complexity metric. The second experiment analyzes our metrics on a symmetric autoencoder. The third experiment compares the family of ResNets and their PlainNet counterparts using our metrics. Finally, section four gathers a set of widely used state-of-the-art architectures and tries to establish a correspondence between the complexity measure and the test accuracy of the architectures.

A. Comparisons With VC Dimensions

In this experiment, we consider a set of simple architectures and estimate their complexities using both our metric and the VC dimension. The VC dimension is a well-established way of estimating the complexity, and, therefore, the generalization error of certain classifiers. However, it can be quite challenging to estimate the VC dimension for deep networks comprising different types of layers. Recently, some theoretical estimates have been developed for fully connected networks [44] and convolutional networks with ReLU activation functions [45].

Fig. 6 shows the results of our complexity metric and VC dimensions for these relatively simple architectures.
Fig. 6 shows that our complexity estimates follow the same monotonically increasing trends as the VC dimensions for all the architectures. However, the magnitudes of our estimates increase at substantially slower rates than VC dimensions for increasing network complexity. This implies that VC dimension can only be practically computed for simple architectures, whereas, our metric allows the evaluation of very large or deep networks. Besides, our method allows flexibility in estimating the complexity of any convolutional or residual network comprising different types of layers. On the other hand, to the best of our knowledge, the theoretical approximations of the VC dimension have not yet been developed for many layer types. Therefore, we cannot compare our complexity metric and VC dimension to the state-of-the-art architectures used in large-scale image classification problems.

We briefly consider another comparison experiment, where we set the VC dimension to a reasonable interval (for example, $10^{11} - 2 \times 10^{12}$), and investigate different architectures that yield values with this interval. We observe that a large fully connected network, with 18 layers and an input of $8196 \times 8196$, has the same value as a very small convolutional network, comprising just one internal layer and an input of $55 \times 55 \times 3$. On the other hand, our complexity metric yields drastically different values; the convolutional and fully connected networks have values of $O(100)$ and $O(10^{12})$, respectively. These values correspond much more closely to the actual expressivity of the two networks, further validating the practical usefulness of our metric.

B. Estimation of Properties for Autoencoder Network

We now present a simple example to explain our metrics visually. It consists of a completely symmetric autoencoder with linear activation functions, where the arrangement of the layers is shown in the left section of Fig. 7. The intrinsic power plots are shown in the central section of Fig. 7, where the local values in each layer and the cumulative values are depicted in the lower and upper rows, respectively. Similarly, the layerwise local and cumulative complexity values are shown in the bottom right and top right sections of Fig. 7, respectively. The global value of the metrics is taken as the cumulative value in the final layer of the network [global cumulative intrinsic power (GCIP) and global cumulative complexity (GCC) for intrinsic power and complexity, respectively].

As seen in Fig. 7, the value of global intrinsic power is the same as the initial value, meaning that the power of the output data is the same as that of the input data. This is because we have used a completely symmetric architecture with linear activation functions in each layer. However, this does not imply that the value of the predicted labels would be the same as the input labels. It is also worth noting that a nonsymmetric autoencoder with nonlinear activation functions, or even a final classification function, is not likely to obtain the same output and input values of intrinsic power.

Although the power of the autoencoder output data reaches the same value as the input, the network does have a learning capacity, captured through a positive complexity value. Therefore, in the upper right part of Fig. 7, we see the cumulative complexity of the network with an increasing trend throughout most of the local layers. The last value of the GCC is related to the expressivity of the autoencoder.

C. Comparisons Between ResNets and PlainNets

After understanding how the metrics are computed and the relationship between the local and global values for a simple neural network, we now enumerate the contribution of our metrics. In this case, we measure the differences in the
metrics values between the ResNet and the PlainNet family of architectures.

The results of the global metrics for five models of the ResNet family and five models of the PlainNet family are shown in Fig. 8. Both network families use the same number of processing blocks. However, as shown along the horizontal axis of Fig. 8, the number of equivalent local layers, corresponding to indivisible operations, are different. This happens as ResNet requires more operations to add the data in the residual block outputs, and certain operations, such as activations, occur independently along parallel network paths.

In the right section of Fig. 8, we see very small differences between the GCC values of the ResNet and PlainNet models. This means that the models have an equivalent capacity in that they can express or learn, essentially, the same set of functions for classification. Furthermore, as might be expected, the complexity increases consistently with the number of layers, indicating that the deeper networks are capable of learning more complex patterns or structures from data. In other words, the deeper networks can achieve better tradeoffs between high degrees of approximation and low generation errors, provided they can be trained.

Although the theoretical capacity of both types of architectures is equivalent, their intrinsic powers are not. The left section of Fig. 8 shows that there are notable differences between the ResNet family and the PlainNet family. These values can be understood as the power to compress data, where a small value indicates a large compression. Therefore, we can say that the PlainNet152 model has much higher compression than the ResNet152 model. In fact, all the models of the PlainNet family lead to a larger data compression than any member of the ResNet family. Within each family, of course, intrinsic power decreases, or, equivalently, data compression increases with network depth. Now, since the training process becomes more challenging with increasing data compression due to the difficulty in information flow across the network, ResNets are easier to train than PlainNets as already pointed out in [46]. This observation is also indicated via both theoretical and experimental results in [6, Chs. 3.3.3 and 3.3.4], which further validates our choice of the intrinsic power metric and establishes its link to model learnability.

The fact that the learning process is improved in the ResNet family as the data propagates more fluently is related to the final accuracy result. The fact that a network has the capability to express a classification function is a necessary but not sufficient condition. Through the training process, we must also ensure that the network is capable of learning the suitable weights to express the function. In other words, an architecture can be very complex (and, therefore, capable), but it may not be able to express its entire power if it is not trained adequately. Note though that a multitude of other factors plays a role during the training process, such as the loss function, learning algorithm, batch normalization, data preparation, and data augmentation. Here, we only consider the intrinsic power
Fig. 10. GCIP and GCC values for some known network models grouped and colored by family. Low GCIP corresponds to a high transformation of the data, indicating that a moderately large value is desired to facilitate easy training. High GCC indicates that the network has a large capacity, implying that a high value is preferred, especially for challenging prediction problems, requiring both high approximation and generalization capabilities.

to evaluate the learning capability of the network, as it is a very influential, data-independent factor in the training process.

It is interesting to note that the intrinsic power curve in the ResNet family of models produces small waves (Fig. 8), due to the summation of the powers from the shortcut channels. We also observe that the slope of the curves is steeper for the smaller models (with 18 and 34 layers) as compared with the larger models with a greater number of layers. This observation indicates that higher compression rates start happening earlier in small architectures, as might be conjectured intuitively.

Fig. 9 shows further analysis of the differences between the ResNet and PlainNet architectures for the same number of layers. The upper section of Fig. 9 shows the differences in GCIP values, while the lower section shows the differences in test accuracy on the CIFAR-10 dataset [47]. These graphs indicate that, although the complexity of the ResNet and PlainNet models is nearly identical for the same number of layers (Fig. 8, right), test accuracy (generalization capability) is substantially affected by the training process. As mentioned earlier, we believe the GCIP is a good estimator of the ease of training a model based solely on its architecture, and this is further supported by the similar trends observed in the two comparison graphs. It should be noted that the PlainNet-101 and PlainNet-152 models are so hard to train that we could not achieve any meaningful test performance, due to which, the accuracy difference between the ResNet and PlainNet models for 101 and 152 layers is the absolute maximum.
D. Comparisons Among State-of-the-Art Architectures

Finally, we test our metrics on a large set of deep neural networks. The network models are selected based on widespread use and easy access via open-source implementations in platforms, such as TensorFlow [48] and PyTorch [49]. Fig. 10 visualizes the GCIP and GCC values for all the models in the upper and lower sections, respectively. This visualization allows us to quickly understand the variations of the metrics within the same family of networks as well as compare the different architectures families. Consequently, we can choose the networks with reasonably large GCIP values to facilitate training and high GCC values for challenging prediction problems requiring a high degree of approximation with a low generalization error.

While a quantitative comparison of the complexity of the different network models is a contribution in itself, as in the previous experiment, we go a step further to investigate the correlation between our metrics and the accuracy of the models. To this end, we use a challenging benchmark classification dataset in the form of ImageNet [50] and explain the variations in accuracy from a theoretical standpoint, for the first time to the best of our knowledge.

Specifically, we evaluate our global weighted and global sum complexity (GSC) metrics with respect to the Top-1 and Top-5 accuracy in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) validation set. The results are shown in the upper section of Fig. 11, where we observe that both the complexity metrics are correlated with the accuracy achieved by the networks. However, the correlation shows a fair amount of dispersion, and we believe that a more useful relationship between complexity and accuracy can be obtained by reducing this dispersion. For this reason, we define a new variable called the global weighted complexity (GWC), $C_W$, where the contributions of the GCC and the GSC are combined as

$$C_W = C_L \ast C_{GS}$$

with $l = L$ as the output layer of the neural network.

GWC yields a tighter relationship between network complexity and ImageNet validation accuracy, as shown in the bottom section of Fig. 11. We apply curve fitting to obtain a numerical approximation of the relationship between the variables, where a power function gives correlation values of $R^2 = 0.751$ and $R^2 = 0.755$ for the Top-1 and Top-5 accuracy, respectively (orange curves in Fig. 11). This means
that we are able to obtain a curve that suitably approximates the relation of the complexity of the networks with their classification accuracy (see the Supplementary Materials for more details). Therefore, our framework can not only describe the network properties and explain their predictive capabilities but it can also be potentially used to design new architectures for challenging learning tasks, by imposing a desired accuracy or generalization error value on the fit GWC curves, without requiring training.

VIII. Conclusion

It is still an open problem to estimate the inherent capacity of a (deep) neural network model to learn the target functions from the input to the output space. While the sum of parameters, the number of layers, or the memory size are often used in practice, we believe that they do not adequately capture the capacity of a model, and are, therefore, not suitable for comparing various models. Instead, we define a new layerwise metric, called complexity, and combine the local values to come up with GCC to encode the data-independent capacity of the network models. It is, however, important to note that the entire (theoretical) network capacity may not be available for any given dataset. In fact, capacity availability depends on several factors, such as the ease and quality of the training process and the distribution of the input data.

In addition to network complexity, we define a set of similar local and global intrinsic power metrics to estimate the compression applied to the data as they flow through the network. Deep compression implies that the data undergo a big transformation during the inference process (direct propagation), and the gradients undergo large transformations during the learning process. In our opinion, this effect is closely related to the ease of learning. We understand that abrupt data transformations imply slower convergence, which translates into a more expensive learning process. Furthermore, since we assign a specific value for each of these metrics, we are able to compare the different network families and the different members within a family. This capability allows us to identify potent architectures with high complexities that are challenging to train.

Furthermore, the slope of the cumulative curves shows the rate at which compression and complexity increase as the data passes through a given network, thereby offering hints on the desired architecture and depth to solve a given learning problem. Our proposed metrics also provide the possibility of estimating the image classification accuracy for any given neural network a priori, since network complexity and prediction accuracy are found to be correlated. Therefore, we intend to use these metrics in the future to search for new architectures with the right balance of high complexity, relatively large intrinsic power (low compression), and reasonably low memory size (number of network parameters).

We acknowledge that our framework does not consider all the contributing factors and performance measures of neural networks. In fact, we omit the effects of the initial values of the network weights, the learning algorithm, and the corresponding hyperparameters (e.g., learning rate, batch size, use of momentum, and so on), and the optimization strategy in determining the prediction performance. Correspondingly, we do not include useful performance criteria, such as robustness and optimization error. Future work would systematically investigate the tradeoff between practical computability and a comprehensive a priori analysis of networks. Such an investigation, along with further development of the generic property definitions, could also help in analyzing other types of networks, such as recurrent neural networks and transformers.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in Proc. Adv. Neural Inf. Process. Syst., 2012, pp. 1097–1105.
[2] J. Xie, L. Xu, and E. Chen, “Image denoising and inpainting with deep neural networks,” in Proc. Adv. Neural Inf. Process. Syst., 2012, pp. 341–349.
[3] R. Collobert and J. Weston, “A unified architecture for natural language processing: Deep neural networks with multitask learning,” in Proc. Int. Conf. Mach. Learn., 2008, pp. 160–167.
[4] C. Carr and Z. Zukowski, “Generating albums with SampleRNN to imitate metal, rock, and punk bands,” 2018, arXiv:1811.00653.
[5] O. Five. Playing Dota 2 World Champions. Accessed: May 1, 2019. [Online]. Available: https://openai.com/ivef
[6] D. Rolnick, “Towards an integrated understanding of neural networks,” Ph.D. dissertation, Massachusetts Inst. Technol., MIT, Cambridge, MA, USA, 2018.
[7] B. Hanin, “Universal function approximation by deep neural nets with bounded width and ReLU activations,” Mathematics, vol. 7, p. 992, Oct. 2019.
[8] R. Vidal, I. Bruna, R. Giryes, and S. Soatto, “Mathematics of deep learning,” 2017, arXiv:1712.04741.
[9] P. Baldi and R. Vershynin, “On neural complexity,” in Proc. Adv. Neural Inf. Process. Syst., 2018, pp. 7729–7738.
[10] P. Orponen, “Computational complexity of neural networks: A survey,” Nordic J. Comput., vol. 1, no. 1, pp. 1–18, 1994.
[11] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio, “On the number of linear regions of deep neural networks,” in Proc. Adv. Neural Inf. Process. Syst., 2014, pp. 2924–2932.
[12] R. Pascanu, G. Montufar, and Y. Bengio, “On the number of response regions of deep feed forward networks with piece-wise linear activations,” 2013, arXiv:1312.6098.
[13] R. Novak, Y. Bahri, D. A. Abolafia, J. Pennington, and J. Sohl-Dickstein, “Sensitivity and generalization in neural networks: An empirical study,” 2018, arXiv:1802.08760.
[14] Q. Hu, H. Zhang, F. Gao, C. Xing, and I. An, “Analysis on the number of linear regions of piecewise linear neural networks,” IEEE Trans. Neural Netw. Learn. Syst., vol. 33, no. 2, pp. 644–653, Feb. 2020.
[15] V. Vardi, D. Reichman, T. Pitassi, and O. Shamir, “Size and depth separation in approximating benign functions with neural networks,” 2021, arXiv:2102.00314.
[16] L. Song, S. Vempala, J. Wilmes, and B. Xie, “On the complexity of learning neural networks,” in Proc. Adv. Neural Inf. Process. Syst., 2017, pp. 5514–5522.
[17] N. Cohen, O. Sharir, and A. Shashua, “On the expressive power of deep learning: A tensor analysis,” in Proc. Conf. Learn. Theory, 2016, pp. 698–728.
[18] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. S. Dickstein, “On the expressive power of deep neural networks,” in Proc. Int. Conf. Mach. Learn., 2017, pp. 2847–2854.
[19] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein, “Survey of expressivity in deep neural networks,” 2016, arXiv:1611.08083.
[20] S. Goldt and U. Seifert, “Stochastic thermodynamics of learning,” Phys. Rev. Lett., vol. 118, no. 1, Jan. 2017, Art. no. 010601.
[21] W. E, J. Han, and Q. Li, “A mean-field optimal control formulation of deep learning,” Res. Math. Sci., vol. 6, no. 1, pp. 1–44, Mar. 2019.
[22] Q. Sun, Y. Tao, and Q. Du, “Stochastic training of residual networks: A differential equation viewpoint,” 2018, arXiv:1812.00174.
[23] A. R. Barron and J. M. Klusowski, “Approximation and estimation for high-dimensional deep learning networks,” in Proc. Adv. Neural Inf. Process. Syst., 2016, pp. 1822–1830.

[24] P. L. Bartlett and S. Mendelson, “Rademacher and Gaussian complexities: Risk bounds and structural results,” J. Mach. Learn. Res., vol. 3, pp. 463–482, Mar. 2002.

[25] M. Bianchini and F. Scarselli, “On the complexity of neural network classifiers: A comparison between shallow and deep architectures,” IEEE Trans. Neural Netw. Learn. Syst., vol. 25, no. 8, pp. 1553–1565, Aug. 2014.

[26] B. A. Rieck et al., “Neural persistence: A complexity measure for deep neural networks,” in Proc. Int. Conf. Learn. Representations, 2019.

[27] A. R. Barron and J. M. Klusowski, “Approximation and estimation for high-dimensional deep learning networks,” 2018, arXiv:1809.03909.

[28] A. R. Barron and J. M. Klusowski, “Complexity, statistical risk, and metric entropy of deep nets using total path variation,” 2019, arXiv:1902.08000.

[29] A. Canziani, A. Paszke, and E. Culurciello, “An analysis of deep neural network models for practical applications,” 2016, arXiv:1605.07678.

[30] D. Rolnick and M. Tegmark, “The power of deeper networks for expressing natural functions,” in Proc. Int. Conf. Learn. Represent., 2018, pp. 1–14.

[31] H. Mhaskar, Q. Liao, and T. Poggio, “Learning functions: When is deep better than shallow,” in Proc. 31st AAAI Conf. Artif. Intell., 2017, pp. 2343–2349.

[32] P. L. Bartlett and W. Maass, “Vapnik-Chervonenkis dimension of neural nets,” in The Handbook of Brain Theory and Neural Networks. Cambridge, MA, USA: MIT Press, 2003, pp. 1188–1192.

[33] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, “The expressive power of neural networks: A view from the width,” in Proc. Adv. Neural Inf. Process. Syst., 2017, pp. 6231–6239.

[34] M. Tan and Q. V. Le, “EfficientNet: Rethinking model scaling for convolutional neural networks,” in Proc. Int. Conf. Mach. Learn., 2019, pp. 6105–6114.

[35] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in Proc. 14th Int. Conf. Artif. Intell. Statist., Jun. 2011, pp. 315–323.

[36] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” in Proc. IEEE Int. Conf. Comput. Vis., Jun. 2015, pp. 1026–1034.

[37] P. Ramachandran, B. Zoph, and Q. V. Le, “Searching for activation functions,” 2017, arXiv:1710.05941.

[38] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in Proc. Int. Conf. Artif. Intell. Statist., 2010, pp. 249–256.

[39] B. Gao and L. Pavel, “On the properties of the softmax function with application in game theory and reinforcement learning,” 2017, arXiv:1704.08005.

[40] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in Proc. Int. Conf. Mach. Learn., 2015, pp. 448–456.

[41] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, “How does batch normalization help optimization?” in Proc. Adv. Neural Inf. Process. Syst., 2018, pp. 2483–2493.

[42] N. S. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” J. Mach. Learn. Res., vol. 15, no. 1, pp. 1929–1958, 2014.

[43] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Jun. 2016, pp. 770–778.

[44] P. L. Bartlett, N. Harvey, C. Liaw, and A. Mehrabian, “Nearly-tight VC-dimension and pseudodimension bounds for piecewise linear neural networks,” J. Mach. Learn. Res., vol. 20, no. 1, pp. 2285–2301, 2019.

[45] Y. Jiang, B. Neyshabur, H. Mobahi, D. Krishnan, and S. Bengio, “Fantastic generalization measures and where to find them,” 2019, arXiv:1912.02178.

[46] K. He, X. Zhang, S. Ren, and J. Sun, “Identity mappings in deep residual networks,” in Proc. Eur. Conf. Comput. Vis., 2016, pp. 630–645.

[47] A. Krizhevsky, “Learning multiple layers of features from tiny images,” M.S. thesis, Univ. Toronto, Toronto, ON, Canada, 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/

[48] M. Abadi et al., “Tensorflow: A system for large-scale machine learning,” in Proc. 12th USENIX Symp. Operating Syst. Design Implement., 2016, pp. 265–283.

[49] A. Paszke et al., “PyTorch: An imperative style, high-performance deep learning library,” in Proc. Adv. Neural Inf. Process. Syst., 2019, pp. 8026–8037.

[50] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Jun. 2009, pp. 248–255.

Alberto Badías received the B.S. degree in mechanical engineering, the M.S. degree in industrial engineering (industrial automation and robotics), the M.S. degree in biomedical engineering, and the Ph.D. degree in mechanical engineering from the University of Zaragoza, Zaragoza, Spain, in 2011, 2014, 2016, and 2020, respectively. He is currently an Assistant Professor with the Polytechnic University of Madrid, Madrid, Spain. His research interests include computer vision and robotics developing 3-D reconstructions and new image processing tools, model order reduction methods, and artificial intelligence in applied mechanics.

Ashis B. Banerjee (Senior Member, IEEE) received the B.Tech. degree in manufacturing science and engineering from IIT Kharagpur, Kharagpur, India, in 2004, and the M.S. and Ph.D. degrees in mechanical engineering from the University of Maryland, College Park, MD, USA, in 2006 and 2009, respectively. He was a Research Scientist and a Post-Doctoral Associate with the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA. He was a Research Scientist with GE Global Research, Niskayuna, NY, USA. He is currently an Associate Professor of industrial and systems engineering and mechanical engineering with the University of Washington, Seattle, WA, USA. His research interests include autonomous robotics, predictive analytics, and statistical learning.