Sparsity-Probe: Analysis tool for Deep Learning Models [*]

Ido Ben-Shaul[†] and Shai Dekel[†]

Abstract. We propose a probe for the analysis of deep learning architectures that is based on machine learning and approximation theoretical principles. Given a deep learning architecture and a training set, during or after training, the Sparsity Probe allows to analyze the performance of intermediate layers by quantifying the geometrical features of representations of the training set. We show how the Sparsity Probe allows measure the contribution of adding depth to a given architecture, to detect under-performing layers, etc., all this without any auxiliary test data set.

Key words. Deep Learning, Approximation Theory, Representation learning, Wavelets, Sparsity, Explainability.

AMS subject classifications. 68T07, 68T30, 65D15, 65Y20, 65D40, 65D10

1. Introduction.

Deep Neural Networks(DNN) have triumphantly improved benchmarks in a variety of different tasks. Remarkable works of architecture design [33, **?**, 51], optimization methods [45, 30, 14], and data mutation [27, 49, 9, 5] have been introduced and shown to empirically advance the fields of computer vision and natural language processing. Still, practitioners fail to justify the success of these models and lack the tools to test their real-world performance. Furthermore, while the basic notions of network architecture are understood [33, 19], it is difficult to assess the contribution of a certain layer of a trained model. Given these limitations, it is often unknown how to analyze a given architecture and how it can be improved. In many cases networks are treated as black boxes that lack explainability, and architectural experiments are conducted in a trial and error manner. An auxiliary test set is regularly presented as an approximation of the true dataset distribution [44, 32, 31] and used to quantify the model performance.

Given the supervised classification setting, as presented in [6], any machine learning algorithm seeks to find a geometrical transformation that separates the samples of different categories and gathers the samples from matching categories. This notion has been prevalent in the field of Self-Supervised Learning(SSL) [21, 4, 55]. In this paradigm, lacking the categorical information, a distorted image is generally compared to itself, to enforce the geometrical notion.

In the Deep Learning setting, the category labels are often represented as a one-hot-encoding [23], a vector in $\mathbb{R}^L$, where L is the number of categories. Intermediate Features have been shown to learn incrementally higher-level features throughout the model layers [18, 41]. The output of the model's k^{th} layer, as k grows, is expected to have a simpler structure, as the features contain more information that is class-specific. This concept corresponds to simpler mappings from incremental layers to the output labels.

In the approximation theory approach, the sparsity of a function given some representation can be a robust method for evaluating its simplicity [15, 16]. Functions in Deep Learning are generally not of a Sobolev nature, but rather in a general Besov Space [43]. The study of adaptive, nonlinear approximation[12], allows the computation of this complexity score on such inherently non-smooth

[*]Received by the editors DATE.

[†]Department of Applied Mathematics, Tel-Aviv University(idobenshaul@mail.tau.ac.il, shai@tauex.tau.ac.il).

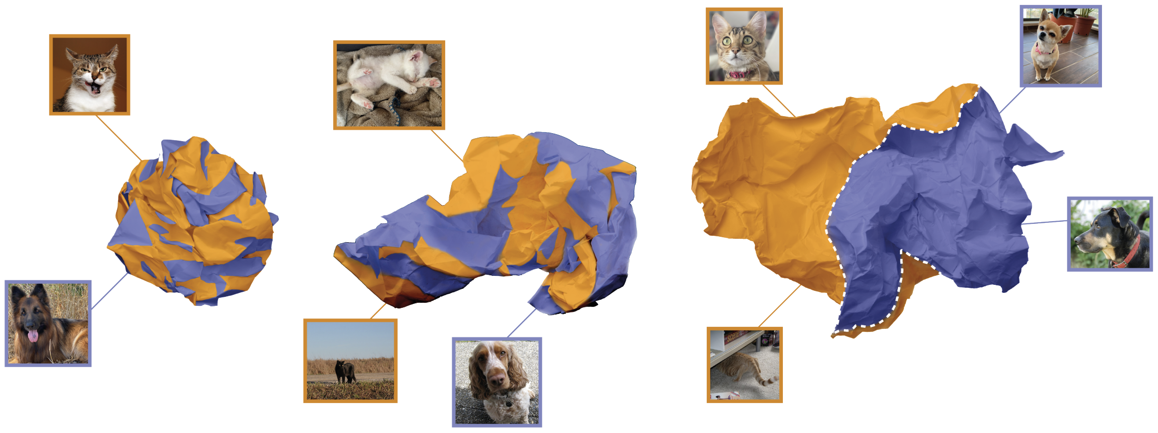
Figure 1: Demonstration of uncrumpling of the data representation through DL layers, as proposed in [6].

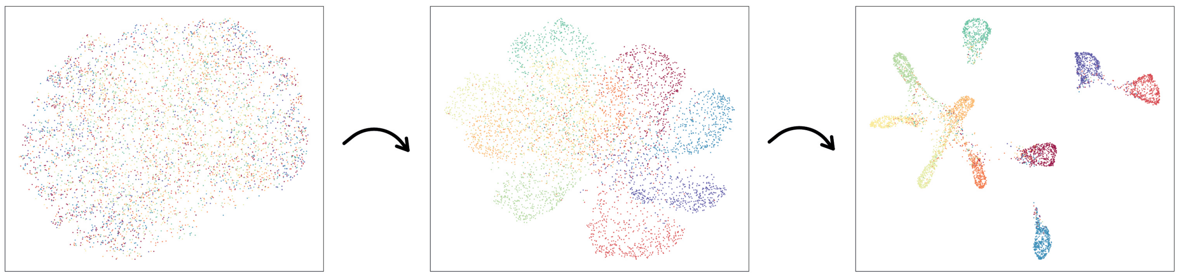
Figure 2: UMAP dimensionality reduction for the feature space of the input layer, the 2nd layer, and the 6th layer of a VGG-13 architecture well-trained on the CIFAR10 dataset. The improved clustering of the data representations is visually significant.

functions.

The main contributions of this paper are as follows:

(i) **Sparsity-Probe**, a mathematically grounded tool for investigating the performance of intermediate model layers is introduced, using solely the train data, and model architecture(without an auxiliary test set).
(ii) Extensive analysis is conducted showing the advantage of the Sparsity-Probe over classical clustering indices.
(iii) To support the theoretical basis, Sparsity-Probe is demonstrated on several known classification datasets.
(iv) We present examples where the Sparsity-Probe is able to detect faulty or buggy architectures and by pinpointing the problematic layer allows to fix them.

2. Related Research and Concepts.

2.1. Statistical Approach. Different statistical and mathematical theories that aim to explain the success of DL have been proposed. The authors of [47] provide an Information-Bottleneck theory in which the network is viewed as a Markov-Chain. The Mutual-Information is documented between the inputs and the labels throughout the layers. Other statistical approaches [34] propose equivalence between increasingly-wide networks and Gaussian Processes.

Classic Machine Learning algorithms rely heavily on the geometry of the input feature for their success. Methods like Support Vector Machines [8], KNN [37], and Random Forest [3] focus on leveraging the geometry of the data for training. DL models are required as automatic feature engineering tools, when there is no clear clustering of the classes in the original feature space (e.g. pixel representation in computer vision). This leads us to believe that without a clear understanding of the geometry in the hidden layers, one cannot hope to understand the prediction quality of the model.

2.2. Approximation-Theoretical Approach. Approximation Theory has given great importance in the field of Signal and Image Processing. Many methods have been offered for uncovering concealed relevant information from signals [43, 10, 15, 38]. There has been a large amount of interest in grounding the theoretical basis of Neural Networks from an approximation theoretical perspective. Many such works study the expressive power of deep feed-forward neural networks(FNN) for a certain target function $f \in \mathcal{B}$ networks, where $\mathcal{B}$ is a given Banach Space. In [46], the number of neurons is used to characterize the approximation rate for Hölder continuous functions using ReLU FNNs. Using both the width and depth of the network, [36] achieve optimal approximation characterization of deep ReLU networks for smooth functions. Upper and lower bounds for the capacity needed to approximate Sobolev Functions are demonstrated in [53]. The authors were able to show that deep ReLU networks are able to more efficiently approximate smooth functions than shallow networks. General continuous functions are considered in [54], where optimality is shown for constant-width fully connected in terms of approximation rates.

However, typically the authors assume that the input dataset can be represented as samples of a continuous or smooth function, such as functions in certain Sobolev spaces with sufficiently high smoothness index. Yet, evidence suggests that in most computer vision problems the input space is more correctly modelled by a discontinuous function. UMAP [39] is a nonlinear dimensionality reduction method that is commonly used to visualize data. In 2 we visualize the feature space of the input layer, the 2nd layer, and the 6th layer of a VGG-13 architecture trained on the CIFAR10 dataset. We sample 6000 random instances from the Train Set, and fit the UMAP reduction on their matching latent representation. It is obvious that the input space represents a discontinuous function. It is eyeopening to see that throughout the layers, the geometric clustering is apparent. Thus, in this work, we assume that the input dataset can be modeled as a function in some geometric Besov space, with relatively low smoothness index.

2.3. Sparsity-Based Approach. Sparsity has been shown paramount for representing complex signals and giving insights into their nature e.g. Wavelet and Fourier transforms [38, 10, 15]. It is intuitive to believe that DNNs employ sparsity methods to achieve successful representation learning. The Multi-Layer Convolutional Sparse Coding(ML-CSC) [42] provides a sparsity-based apprehension of Convolutional Neural Networks. Given a Dictionary D, it is shown that a ReLU Network forward pass is in fact equivalent to a layer-wise Nonnegative Sparse Coding pursuit, using Soft Nonnegative layered thresholding as a sparsity pursuit approximation. In [50], a holistic pursuit is proposed along with a method for such Dictionary Learning. The discovery that concatenated layers are sparse with respect to a proposed dictionary is an important one and helps bridge the gap between the sparsity theory and empirically found neural network architectures. Our study differs from this approach in two critical aspects. The ML-CSC model proves that under certain conditions, a sparse Dictionary and representation vectors can be found, and propose methods for finding them. In our work, a general Post-hoc technique is shown to reliably enhance the explainability of any given trained model, along with its train dataset. This, in turn, does not involve learning a specific dictionary and representation, but rather assessing the quality of a given state. More importantly, we focus on supervised learning, where the sample categories(whether provided or not) are integral to approximate the model quality. Indeed, our premise relies on the fact that the sparsity should be centered around the mapping between the latent features and the labels. To strengthen this claim, consider a certain representation space. For a specific label assignment, this representation can be extremely well clustered, yet completely intertwined for a different label assignment.

2.4. Linear and Kernel Probes.

Definition 2.1 (Linear Separability). *Two sets $\Omega_1, \Omega_2 \subset \mathbb{R}^n$ are linearly separable, if their convex hulls do not intersect.*

Definition 2.2 (Non-Linear Separability). *We say the sets $\Omega_1, \dots \Omega_k \subset \mathbb{R}^n$ are non-linearly separable if for every Ω_i there exists a domain $M_i \subset \mathbb{R}^n$ with a smooth boundary, such that $\Omega_i \subset M_i$ and $M_i \cap \bigcup_{j \neq i} M_j = \emptyset$.*

Classic machine learning algorithms like SVMs and CART[35] seek to find the best separation in the feature space between clusters of different classes. In the field of Representation Learning [2], contrastive losses [**?**, 29, 24] aim to separate samples of different underlying category, whilst clustering samples of the same category. It is then of interest to quantify the wellness of separability between classes in the latent space.

Recent works propose to compute the linear separability of the intermediate layers [1, 7]. Linear Separability is simple to define and compute, yet fails to grasp any separation which is not linear.

We present three synthetic toy datasets, with feature space of dimension 2, and two outcome classes: Spiral, Circles, and Gaussian Quantiles(GQ) - see figure 3. It is clear that the Linear Classifier methods cannot differentiate between the classes, as they are not linearly separable. A more sophisticated measure of separability is considered in [40], by using radial basis kernel PCA to map the latent space to a different representation (selecting d leading singular values), and measure linear separability in the projected dimension. This is problematic as d is difficult to choose. In fact, if d is large enough, linear separability becomes trivial, and so the authors consider small d. In general, one should prefer to measure the true separability in the given dimension, instead of measuring the linear separability in a projected (potentially lower-dimensional) space.

The separability of feature space in an intermediate layer is equivalent to the smoothness of the mapping to the sample labels, in the one-hot-encoding scheme. This work relies on **sparsity** to measure the smoothness of such functions. In [6], a DNN is compared to an uncrumpling of a high dimensional paper ball, such that every layer decouples between the classes incrementally. This concept is visualized in 1. In this visualization, at the final layer, we show an example of data that are well separated, yet clearly not linearly separable. We propose the Sparsity-Probe as a tool to quantify this type of separation.

3. Formulation.

3.1. Preliminaries. Evaluating the sparsity of a high dimensional function can be a daunting task. We base our notion on a classic approach that arrives from Image Compression [13], where

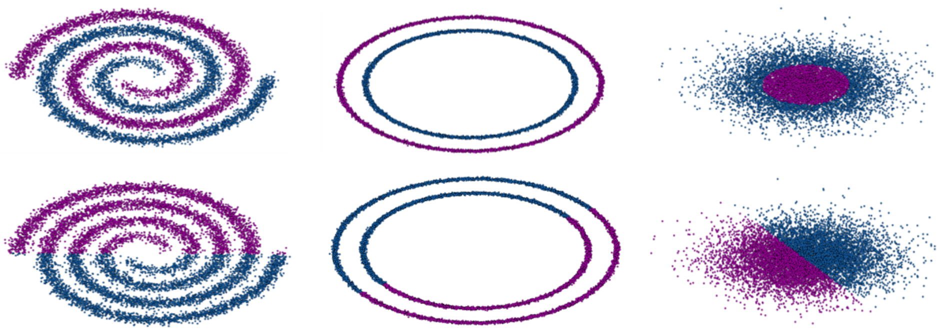
Figure 3: Three synthetic datasets which are all non-linearly separable: Top: Spiral, Circles, and Gaussian Quantiles datasets are shown. Bottom: Output of Linear Probes on these datasets. It is clear to see that a Linear Probe cannot capture the sparsity of such a function.

Wavelets[10] are used to approximate a signal. The Geometric Wavelet [11], was shown to extend this notion to the terms of adaptive non-linear approximation. The importance of function sparsity in terms of of signal processing and representations has been thoroughly emphasized in [15] . An in-depth introduction to Geometric Wavelets is shown in [16].

In order to use the appropriate functional tools, we first need to state the problem in a functional setting. Assume we have a square image input sample for the model of side length N. We normalize its values and unravel the image into a long vector of size N^2. At each output layer, we again normalize the representation and unravel them into a vector, these will be the inputs for our functions. We now need to address the labels. In the multiclass classification setting, a common representation for a label is the one-hot-encoding. We use these encodings to represent every label as a vector in $\mathbb{R}^L$, for L - the number of classes. At each layer, the unraveled vector representations along with their vector label value, are considered as samples of a vector-valued function associated with the layer.

The complete NN can be modeled as a function $f : \mathbb{R}^{N^2} \to \mathbb{R}^L$. At the same time, for each layer k, assume there exists a function $f_k : \mathbb{R}^{n_k} \to \mathbb{R}^L$, that maps the unraveled feature vector of the k-th layer, to the label vector representation in $\mathbb{R}^L$. Certainly, any sample of the training set produces simultaneously a sample for each of these functions f_k. A NN f is obviously well-trained, if for a given input x with label y, $f(x)$ is close to the one-hot-encoding of y. Furthermore, for $x_1, ..., x_n$ of the same underlying class, a well trained network will aim to cluster their representations at the intermediate levels. The most common loss functions are built to do just this. Although this is the penalty that is minimized, we claim a well trained model also aims to cluster the intermediate layers, based on the GT labels, and so yields a more clustered representation to be passed to the following layer. To state this more thoroughly, We argue that in a well trained network, each input space for such f_k is more clustered in terms of class label, and so the function mapping it to the class label is smoother. Let us refine the concept of such clustering. Although we do wish that input samples of the same label be close in the k^{th} output space, we need a measure that captures the possibility of several different clusters from a same class. This is a slightly different criterion than that of clustering. We are looking for a notion of good behavior, demanding that similar inputs be mapped to similar outputs.

Using normalization (e.g.) of pixel values, we may assume that our samples x_i are sampled from a convex domain Ω_0, such as $[0,1]^{N^2}$. Our dataset is then of the form

$$\{x_i, f(x_i)\}_{i \in I} \in \left(\Omega_0, \mathbb{R}^L\right).$$

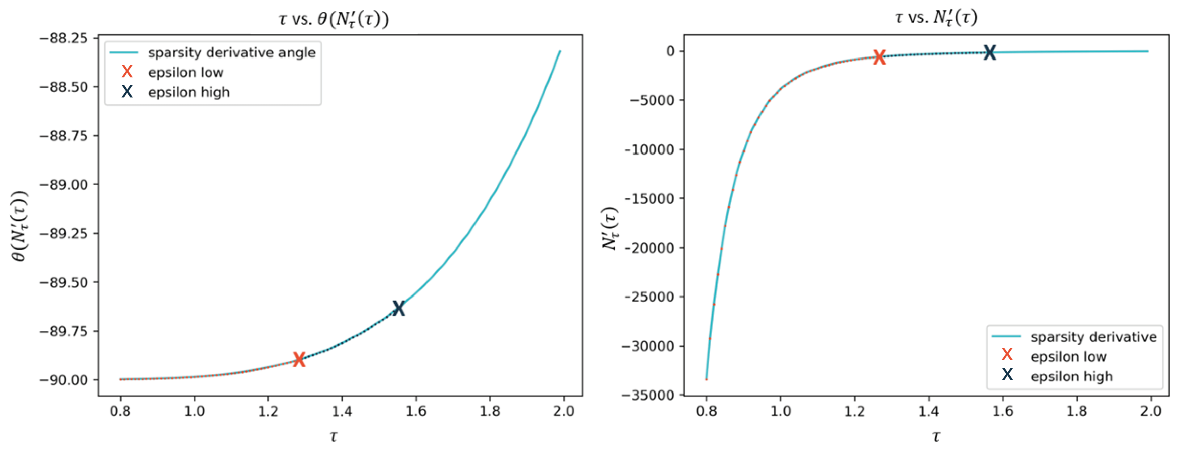

Figure 4: Numerical algorithm for estimating the transition index τ^*. The two meta-parameters, ε_{low} in red and, $\varepsilon_{\text{high}}$ in blue.

We approximate and quantify sparsity of high-dimensional non-smooth functions, using the Wavelet Decomposition of a Random Forest.

3.1.1. Wavelet Decomposition of Random Forest.

We begin with an overview of single trees. Decision Trees aim to find a sparse and efficient representation for the true, underlying function. At each stage, the algorithm seeks the optimal dividing hyperplane, w.r.t a Cost Function. The process is continued until a certain stopping criterion is fulfilled. The resulting domains are labeled as **leaves**. The general setting is as follows:

We begin with the convex domain Ω_0 as the root of the decision tree. At a given stage, for node $\Omega \subset \mathbb{R}^n$, a cost minimizing partition by a hyperplane is found to split Ω into Ω', Ω'', $\Omega' \cup \Omega'' = \Omega$. In the Variance Minimization setting, the cost that is minimized is defined as

$$\sum_{x_i \in \Omega'} \left\| f(x_i) - \vec{E}_{\Omega'} \right\|^2_{l_2(\mathbb{R}^L)} + \sum_{x_i \in \Omega''} \left\| f(x_i) - \vec{E}_{\Omega''} \right\|^2_{l_2(\mathbb{R}^L)} \tag{3.1}$$

Where:

$$\vec{E}_{\hat{\Omega}} := \frac{1}{\#\left\{x_i \in \hat{\Omega}\right\}} \sum_{x_i \in \hat{\Omega}} f\left(x_i\right) \qquad \vec{E}_{\hat{\Omega}} \in \mathbb{R}^L$$

The Random Forest [3] algorithm is a significant generalization of the single Decision Tree which is a locally 'greedy' algorithm. Several decision trees are constructed over random subsets of the training data, and inference is applied through a voting mechanism over the trees. For any point $x \in \Omega_0$, the approximation associated with the j^{th} tree, $\tilde{f}_j\left(x\right)$, is computed by finding the leaf $\Omega \in \mathcal{T}_j$ in which x is contained and assigning $\tilde{f}_j(x) := \vec{E}_\Omega$, where $\vec{E}_\Omega$ is the corresponding mean value of leaf Ω computed during training. The approximate value of a point $x \in \Omega_0$ is then given by

$$\tilde{f}\left(x\right) = \frac{1}{J}\sum_{j=1}^{J} \tilde{f}_j\left(x\right).$$

We are now ready to define the **Geometric Wavelet Decomposition of a Random Forest**. Let $\mathcal{T}$ be a decision tree for function f. First we denote the 'father' wavelet as the constant function $\psi_{\Omega_0} := \vec{E}_{\Omega_0}$. In going further, let Ω' to be a child of Ω in tree $\mathcal{T}$, i.e. $\Omega' \subset \Omega$ and Ω' was created by a partition of Ω. The wavelet $\psi_{\Omega'} : \mathbb{R}^n \to \mathbb{R}^L$ is defined as

$$\psi_{\Omega'}(x) := \mathbf{1}_{\Omega'}(x) \left(\vec{E}_{\Omega'} - \vec{E}_{\Omega} \right),$$

where $\mathbf{1}_{\Omega'}$ is the indicator function of Ω'. The wavelet norm is given by

$$\|\psi_{\Omega'}\|_{L_2} = \left\| \vec{E}_{\Omega'} - \vec{E}_{\Omega} \right\|_{l_2(\mathbb{R}^L)} |\Omega'|^{1/2} \tag{3.2}$$

Under certain mild conditions on a decision tree $\mathcal{T}$, the following holds [16]:

$$f = \sum_{\Omega \in \mathcal{T}} \psi_{\Omega} \tag{3.3}$$

We can then define the wavelet decomposition of a RF as:

$$\tilde{f}(x) = \frac{1}{J} \sum_{j=1}^{J} \sum_{\Omega \in \mathcal{T}_j} \psi_{\Omega}(x) \tag{3.4}$$

With the Geometric Wavelet Decomposition of a Random Forest at hand, we can define the notion of sparsity.

3.1.2. τ-Sparsity.

We define the τ-Sparsity of tree $\mathcal{T}$ and parameter $0 < \tau < 2$,

$$N_\tau(f, \mathcal{T}) = \left(\sum_{\Omega \neq \Omega_0, \Omega \in \mathcal{T}} \|\psi_\Omega\|_2^\tau \right)^{1/\tau} := \|\{\|\psi_\Omega\|_2\}_{\Omega \in \mathcal{T}}\|_{l_\tau} \tag{3.5}$$

It is easy to see that:

$$\lim_{\tau \to 0} N_\tau(f, \mathcal{T})^\tau = \{\#\Omega \in \mathcal{T} : \|\psi_\Omega\|_2 \neq 0\} := \|\{\|\psi_\Omega\|_2\}_{\Omega \in \mathcal{T}}\|_0 \tag{3.6}$$

This coincides with sparsity described in [15]. Let us further denote the τ-sparsity of a forest $\mathcal{F}$, by

$$N_\tau(f, \mathcal{F}) := \frac{1}{J} \left(\sum_{j=1}^{J} N_\tau(f, \mathcal{T}_j)^\tau \right)^{1/\tau} = \frac{1}{J} \left(\sum_{j=1}^{J} \sum_{\Omega \neq \Omega_0, \Omega \in \mathcal{T}_j} \|\psi_\Omega\|_p^\tau \right)^{1/\tau}. \tag{3.7}$$

The $\|.\|_\tau$ norm is monotonically non-increasing in τ.

3.2. τ-Sparsity Motivation.

3.2.1. Smooth-curve Separator in $\mathbb{R}^2$. We begin with a lemma that gives a bound on the τ-sparsity of a smooth curve separator in the binary classification setting, with features in $\mathbb{R}^2$, and a dyadic non-adaptive tree.

Lemma 3.1. *Let $f(x) = \mathbf{1}_{\tilde{\Omega}}(x)$, where $\tilde{\Omega} \subset [0,1]^2$ is a compact domain with a smooth boundary. Then, for $1 < \tau < 2$, $N_\tau(f, \mathcal{T}_I) < \infty$, where $\mathcal{T}_I$ the tree with isotropic dyadic partitions, creating dyadic cubes of area 2^{-2k} at level $2k$.*

Lemma 3.2. *Let $f(x) = \sum_{k=1}^{K} c_k \mathbf{1}_{B_k}(x)$, where $B_k \subset \Omega_0$ are disjoint cubes with sides parallel to main axes, $c_k \in \mathbb{R}$. Then, there exists an adaptive tree $\mathcal{T}$, such that for every $0 < \tau < 2$, $N_\tau(f, \mathcal{T}) < \infty$.*

Based on lemma 3.2 we can define the τ^*, as the transition index

Definition 3.3 (transition τ-index). *We define:*

$$\tau^* := \inf_{0<\tau<2} \left\{ \tau | N_\tau(f, \mathcal{F}) < \infty \right\}, \tag{3.8}$$

where $N_\tau(f, \mathcal{F})$ is given in (3.7).We notice that due to the monotonicity of the $N_\tau(f, \mathcal{F})$, the transition index is the smallest τ such that the τ-sparsity is finite.

It was shown in [16], [17], that under certain mild conditions, the Forest Besov-Smoothness of the function is equivalent to it's τ-sparsity.

3.3. Numerical estimation of τ^*.

Since τ^*, defined in (3.8), is a complicated transition index, the task of estimating it is highly non trivial. Here we propose a more robust method then the methods proposed in [16], [17]. First, we use (3.7), to create a series of samples $N_{\tau_k}(f, \mathcal{F})$, for a set of discrete samples $\{\tau_k\}$, $0 < \tau_k < 2$. We then approximate at these samples numerical derivatives

$$N'_\tau(\tau_k) := \frac{\partial N_\tau(f, \mathcal{F})}{\partial \tau}(\tau_k).$$

We use the angles of the derivatives

$$\theta(\tau_k) := \arctan(N'_\tau(\tau_k)),$$

to estimate the transition index τ^*. Now, observe that the transition index is associated with an 'infinite' derivative, or equivalently an angle of $-\pi/2$. To this end, we use two meta parameters: $\varepsilon_{\text{low}}, \varepsilon_{\text{high}}$, and define

$$S := \{\tau_k : -\frac{\pi}{2} + \varepsilon_{\text{low}} \leq \theta(\tau_k) \leq -\frac{\pi}{2} + \varepsilon_{\text{high}}\}.$$

We now define the transition index by $\tau^* := \frac{1}{|S|} \sum_{\tau_k \in S} \tau_k$. A demonstration is shown in Figure 4.

4. Experiments.

The experiments throughout the paper use $\varepsilon_{\text{low}} = 0.1$, $\varepsilon_{\text{high}} = 0.4$. We use a three trees, with maximal depth 15. The sparsity-probe is deployed on the input layer, and all intermediate model layers. We do not test the sparsity at the final model layer. For each dataset, each model is trained with three different initialization seeds, and approximated throughout the layers.

Table 1: τ-Sparsity of synthetic datasets. Although the datasets are completely separable by a smooth curve, due to its non-linearity - the Linear Probes cannot quantify this separability.

Dataset	τ^*
Spiral	**0.98**
Circles	**0.93**
GQ	**0.99**

4.1. Sparsity Probe on the synthetic datasets. We saw that the Linear Probes cannot quantify the separability of the synthetic datasets presented in Figure 3. We show the τ^* estimate on these datasets. Lemma 3.1 that provides a bound for sparsity using a non adaptive decision tree, suggests that when using adaptive tree partitions, we should expect a sparsity $\tau^* \leq 1$. The numerical estimates for the τ^* values are reported in Table 1. As the distance between the classes in the Circles dataset is largest, the τ^* is indeed the lowest. The Spiral dataset has more distance between the classes than in the Gaussian Quantiles, and so its τ^* is slightly lower.

4.1.1. Comparison of the Sparsity Probe transition index to Clustering indices. Although clustering is closely related to sparsity, there are caveats that yield it inaccurate when trying to evaluate the separation of a latent space:

(i) Most clustering methods do not deal well with non linearly separated data. KMeans, for example cannot handle well the synthetic datasets of Figure 3, and its outcome will be similar to those of the Linear Probes. More advanced Hierarchical Clustering methods can be used to improve this issue in some scenarios.

(ii) When we look at the latent spaces of deep intermediate layers, we expect to observe well-separated clusters. However, in the shallow layers this is simply not true and there are scenarios where the geometry of shallow layers is too vague for most clustering indices.

4.2. Sparsity-Probe on Neural Networks. In order to fully approximate the true functional setting of Deep Neural networks, which are known to be unstable[56], we train each network with 3 different seeds, and approximate τ^* for each of the intermediate layers. We then set the mean of the 3 sparsity index estimates as the estimated index. In some of the figures we also render the certainty intervals of the graphs of τ^*.

Definition 4.1 (α-Score). *A closely related definition to τ-sparsity is the α-score. For the critical $0 < \tau^* < 2$, the α^*-index is defined as*

$$\alpha^* := \frac{1}{\tau^*} - \frac{1}{2} > 0 \tag{4.1}$$

In the following sections we report the critical α-score found by the algorithm. Since they have an inverse relation, we are looking for the highest α-score.

4.2.1. Analyzing the contribution of adding depth to a network. Suppose we wish to create a model for the CIFAR10 dataset, and decide to use a VGG[48] architecture. It is natural to ask

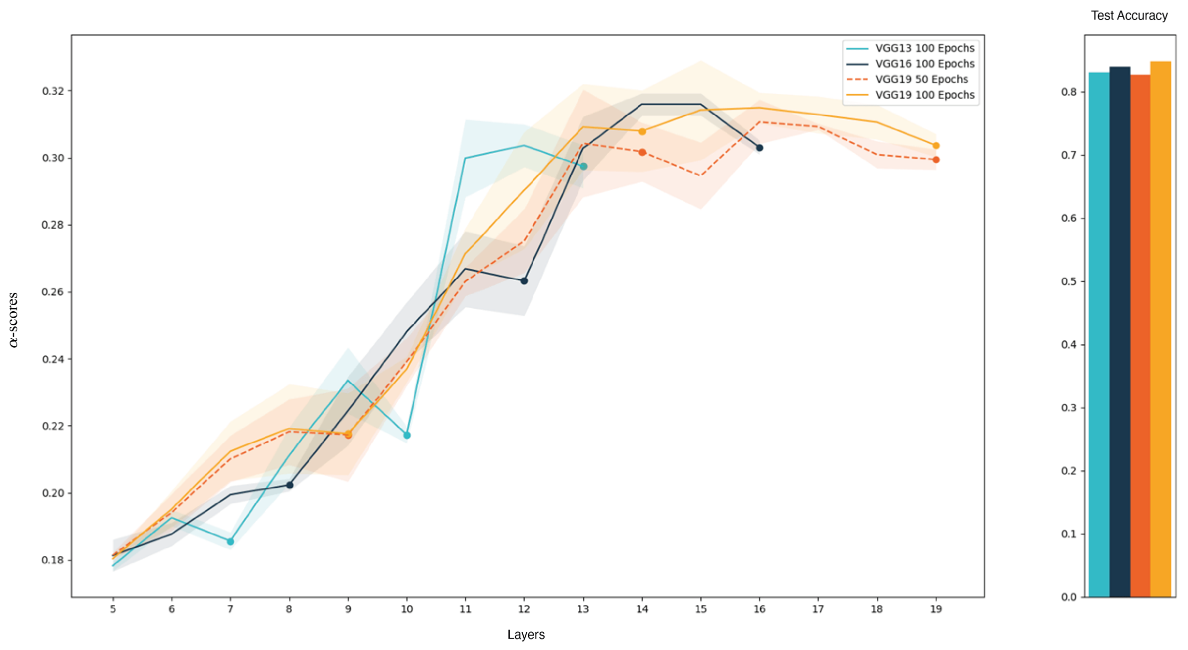

Figure 5: Comparing VGG-{13, 16, 19} on the CIFAR-10 datasets for 100 Epochs. VGG19 is also shown after 50 Epochs. Layers are either MaxPooling(Markers) or Convolutional. We omit the first 5 layers of each network.

- how deep should our model be? Too few layers and the accuracy could be low, too many and the model capacity will be too high, and lead to overfitting. Furthermore, can we actually quantify the contribution of each added layer to the outcome? We train the VGG{ 13, 16, 19} architecture variants on CIFAR10 for 100 Epochs and estimate the sparsity at the output of every MaxPooling layer and of every Convolutional layer beginning from the 5th layer. For VGG19, we also report α^* after 50 Epochs. Results are shown in 5. Observe that the Sparsity Probe reveals certain interesting properties of the different architectures:

(i) As expected, in general, deeper architectures have the capacity to increase the sparsity and this correlates with the accuracy testing results.

(ii) However, we see a certain sparsity saturation phenomena with the VGG16 architecture, where the added layers of VGG19 do not drastically improve sparsity. This correlates with the testing results, where both architectures produce similar accuracy.

(iii) When comparing the VGG19 trained on 50 epochs with the same architecture trained for 100, it is apparent that the separability improves, most noticeably towards the end of the network.

(iv) We can also see that the MaxPooling layers usually cause a dip in α^*. This can be attributed to the fact that max-pooling is a non-learned, coarse discretization layer.

4.2.2. Using the Sparsity Probe to compare different architectures.

We wish to asses the probe's ability to analyze problematic architectures that do not perform well on the testing data, e.g, that are not able to generalize [28] (recall that our probe only uses the training data). We report our results on the Fashion-MNIST[52] dataset. As mentioned before, modern architectures in Machine Learning consist of two parts - the feature extractor and the classifier. The main role of the feature

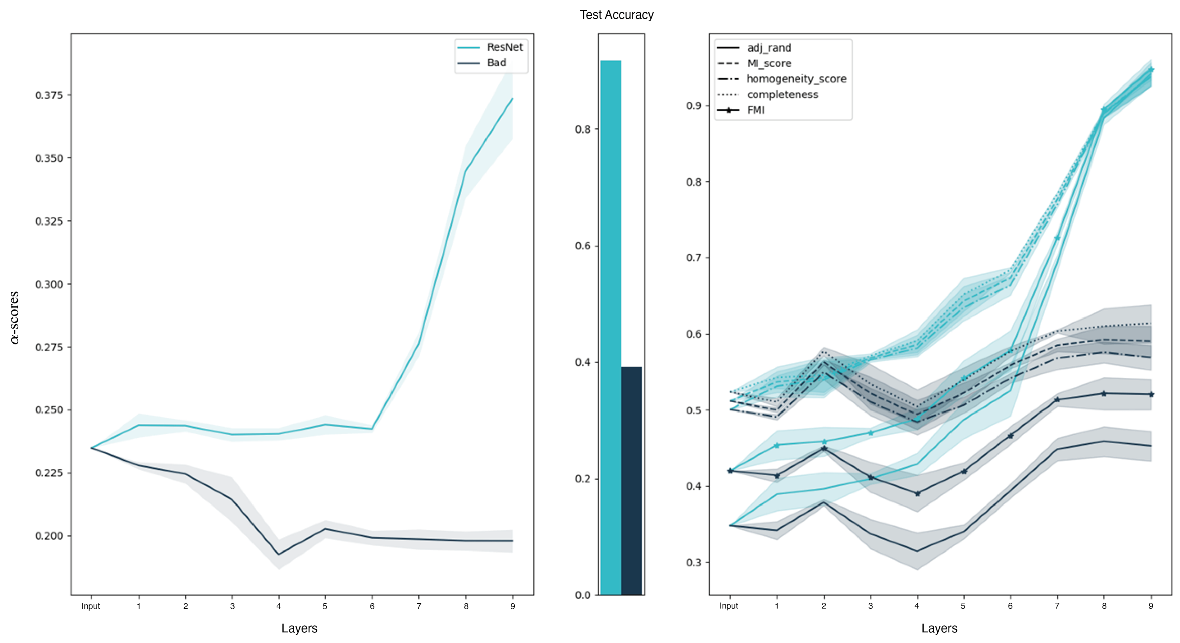

Figure 6: Comparing good and bad architectures α-scores on the Fashion-MNIST dataset. The Sparsity-Probe is able to differentiate the quality of the models, and see the inter layer improvements.

extractor is to transform the latent features into more easily separable latent embeddings. Convolutional Layers(ConvLayers) are used to learn increasingly complicated features across layers, by using spatial relations. Suppose one were to create an entirely different architecture that alternates between ConvLayers and linear layers. Every output of a linear layer is transformed into a square as an input into the ConvLayer. Looking at the α^* in Fig. 6(Left), we see that for the dark-blue line, the scores decrease throughout the layers. The light-blue line reports a model trained on a Resnet18[25] variant, with smaller channel sizes, and inputs of gray-scale images. α-scores are measured at the end of every residual connection, and in edge layers. A general theme which is shown is that the true rise of the α^* happens towards the end of the network. This can be explained by the strength of the gradients that arrive at the earlier layers.

4.2.3. Fashion-MNIST - Clustering Correlation. It is natural to ask how α^* behaves compared to clustering statistics. We would expect high correlation between the metrics when the model latent features are well clustered, and low correlation when the features are not well clustered, or clustered into many different clusters per-label.

To test this, we run the KMeans clustering algorithm with $k = 10$ on each of the layer features, using the models from the previous section. Figure 6(Right) shows the clustering statistics compared in each intermediate layer. In the good models, the correlation between the sparsity and the clustering statistics is positive, demonstrating the data gathers into better label-groups as the layers progress. However, in under performing models, the clustering indices do not manage to asses how bad is the geometry of the represnetations in the intermediate layers. Moreover, in this example, the clustering indices fail to capture the fact that the representations get ‘worse’ throughout the layers.

Table 2: Pearson Correlation: α-scores vs. Clustering on the Fashion-MNIST dataset

Metric	Bad	Good
Rand Index - adjusted for chance	-0.41	0.96
Adjusted Mutual Information	-0.41	0.93
Homogeneity	-0.39	0.94
Completeness	-0.43	0.93
Fowlkes-Mallows Index	-0.43	0.96

We can demonstrate this by using the Pearson Correlation coefficient between each of the clustering statistics and α^* on all model layers. These results are reported in table 2.

4.2.4. Case Study - MNIST-1D. Recent work [20] propose a 1D parallel to the well-known MNIST. The intention of this dataset is to scale down the dimension of the MNIST, and essentially turn the classes into different signals. The authors also show how, as opposed to MNIST, the signals are intertwined in the input space, and are then much less separable. One could ask - does the model improve throughout the epochs? We use this dataset, with a simple 7-layer Convolutional Neural Network, and monitor the α^* at every 10 epochs. The results are shown in figure 7. It is clear that the model not only improves in the final layer, but is able to create a smoother increase in the intermediate layers, as we progress through the epochs.

4.2.5. Case study- Image classification from the magnitude of Fourier coefficients. The problem of Phase Retrieval(PR) is defined as recovering the an image solely using the magnitude of its Fourier transform coefficients. This is a problem that arises in many applications and is obviously an ill-posed inverse problem. Here, we experiment with DL architectures that aim to classify images from the MNIST dataset, again, using only the magnitude of the Fourier coefficients as input. At first, it seems natural to use a standard convolutional network for this classification task. However, as is clear from Figure 8, this approach completely fails. As the plot of the α^* score for this model shows, the network fails to 'unfold' the data and the test accuracy score is very low. The explanation for this phenomena is that architectures based on convolutions assume there are spatial correlations between neighbouring pixels in the input or features in the feature maps. However, this is not true in the Fourier domain. As we see in Figure 8, when one applies a fully-connected network architecture, it is able to learn features which are not of spatial nature. We see how the Sparsity Probe is able to capture the fit of the network to the problem.

5. Analyzing and debugging architectures.

In this section we leverage the sparsity-probe to analyze model architectures and detect problematic layers. We use a 7-layer Convolutional Network with Batch Norm [27], RELU activation function, and two linear classification layers. Each model is trained with 3 different initialization seeds, for 100 Epochs on the MNIST-1D dataset. We show direct correlations between the α^* and the test accuracy, even though the α^* are computed solely from the train dataset.

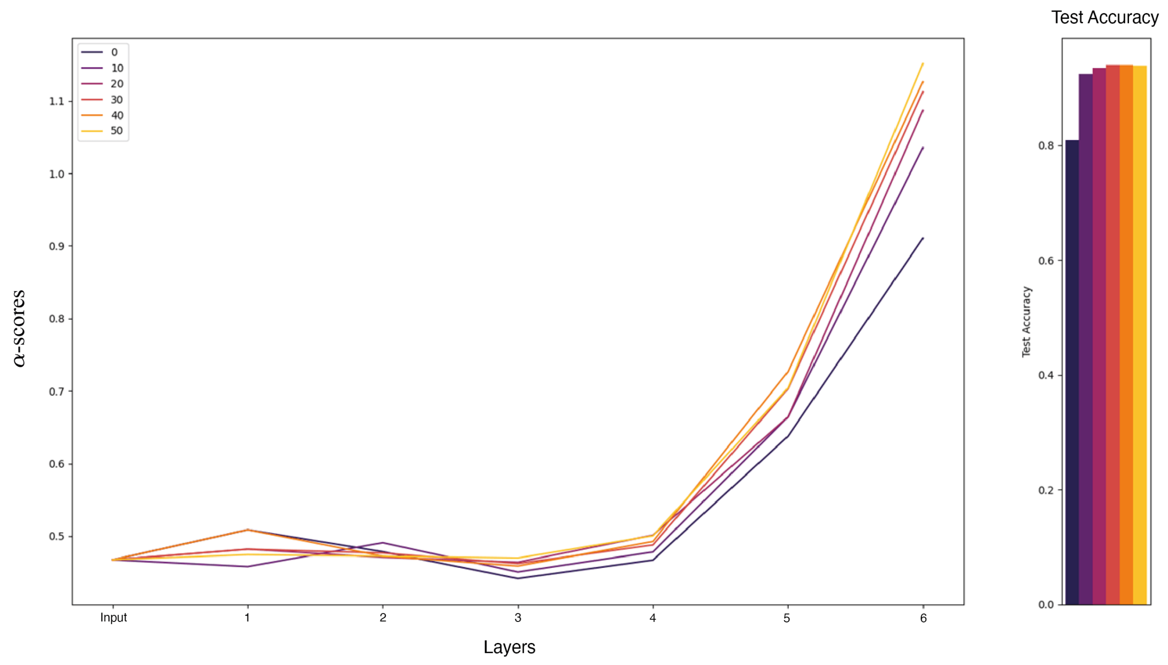

Figure 7: Comparing different epochs during train of a 7-Layer convolutional network on the MNIST-1D dataset. It is clear that the α-scores improve through the epochs, and show a smoother rise, with more contribution at every layer.

5.0.1. Picking the Batch Size. Given a network architecture, meta-parameters still need to be fine-tuned for network training. The batch size is of critical importance as it is constrained by the compute, but also by the affects in the optimization algorithm. If the batch-size is too large, many gradients of different directions can lead to slower convergence. However, if the batch size is too small, the batch can be non-representative of the dataset and lead to a wrong gradient step. In Figure 9 we compare different Batch Sizes. From the analysis, it is apparent that a batch-size of 512 is too large, and results in lower α^*, and accordingly test scores. Batch sizes of 64 and 128 behave similarly in terms of α^*, with a slight advantage to 64, matching the test-scores. Using the sparsity-probe to investigate, we can understand the trade-off between the computational cost and the added gain.

5.0.2. Picking the Activation Function. The activation function is one of the most dismissed architecture choices, yet known to be vital. In modern architectures, mostly ReLU activations are used. Suppose we are trying to create a new activation function, by using a simple step function:

$$f(x) = \begin{cases} 1, & \text{if } x > 0. \\ 0, & \text{otherwise.} \end{cases}$$

This is obviously a problematic activation value, as value magnitudes are not considered. We wish to compare it to other nonlinearities. During first works in Neural Networks, the Sigmoid[22] was proposed as a nonlinearity. It was later shown to promote several issues, such as vanishing gradients. Different alternatives to ReLU have been proposed such as Leaky-ReLU, and GELU [26]. We compare these activations on the specific task in figure 10. For this particular dataset, we see a dominance of

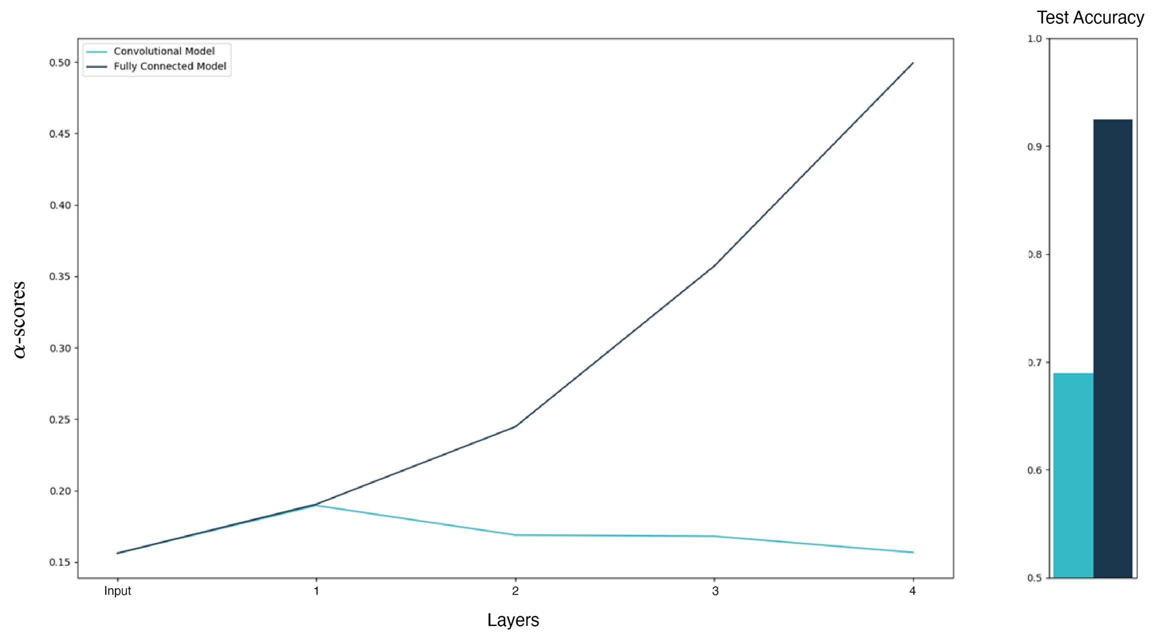

Figure 8: Comparing Fully-Connected and Convolutional architectures on MNIST classification using only Fourier-intensity of the images. It is clear that the α-scores of the architectures match the intuition in this task, as the Fourier Domain does not consist of spatial features, and will therefore fail when using convolutional layers.

the ReLU to other activations. The ReLU variants - Leaky ReLU and GELU are relatively close in performance. The Sigmoid is indeed far lower in terms of α^*, and we see a dip during train. Lastly, the proposed nonlinearity fails magnificently, and the α^* get worse throughout the layers. Remarkably, the Sparsity-Probe perfectly matches the ordering of the test scores!

5.0.3. Increasing the Stride. Suppose we are trying to improve our architecture, by increasing the stride at the 4^{th} layer, from 2 to 5, while the kernel size remains the same(3). This would of course result in a loss of information, as the receptive field does not cover the entire input. A comparison of the α^* is shown in Figure 11. It is clear that during the 4^{th} layer there is a big dip in the α^*. This affects the performance of earlier layers, yet we see an increase after the problematic layer. Using our tool, without looking at the test scores, we can pinpoint the exact location where the network fails.

5.0.4. Batch Normalization. Batch Normalization has proved extremely helpful for the stability of training, especially in very deep network architectures. Suppose we are trying to test the affects of the Batch Norm(BN), by only applying it at the first k layers. In figure 12 we report such comparison, with respect to the base architecture, that includes Batch Normalization at all ConvLayers. It is significant to see that the addition of each BN layer increases the α^*, and accordingly, the test scores.

6. Conclusion.

In this paper, we present the **Sparsity-Probe**, a new method for measuring supervised model quality using sparsity considerations. We give an in-depth explanation of the numerical algorithm and its theoretic background. We give motivation to why the mathematical

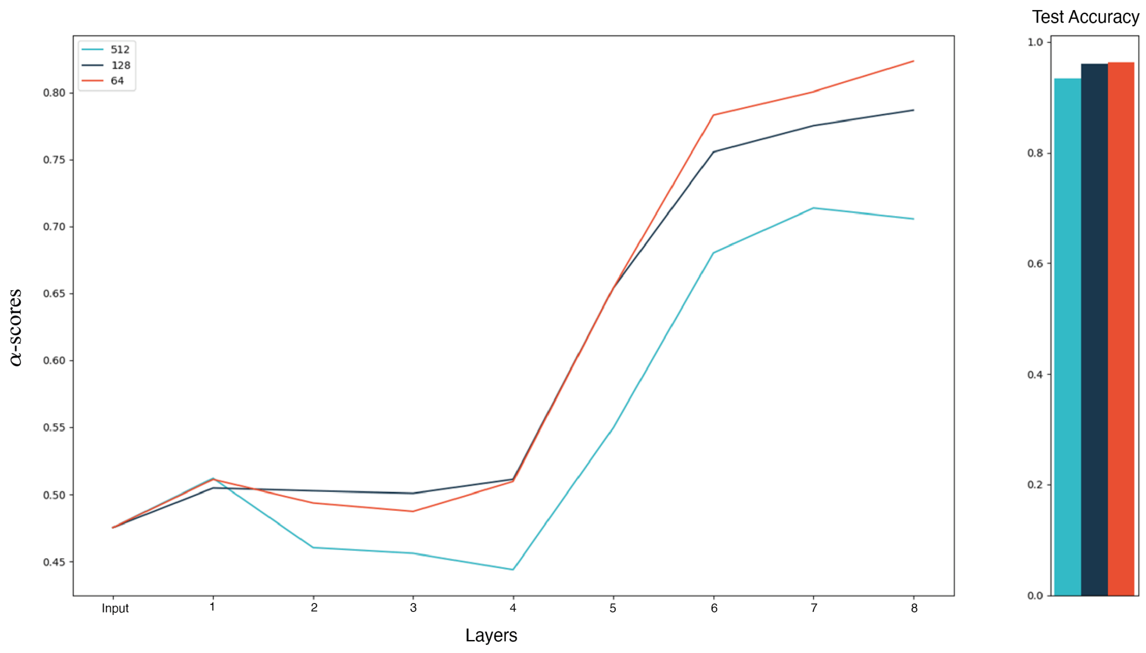

Figure 9: Comparing different Batch-Sizes on the MNIST-1D dataset using the Sparsity-Probe.

complexity in this method is necessary for promising results. We show how this method relates to clustering metrics, and show that our method approximates the theoretical bound on a simple 2D synthetic dataset. Our experiments are conducted onto different datasets and have various end goals. We show how the Sparsity-Probe can be used to assess a model quality without an auxiliary test set. This leads to many downstream capabilities such as finding flaws in the architecture and selecting a robust model.

Appendix A. Proofs.

Proof of Lemma 3.1. Let $\mathcal{T}_I$ be the non-adaptive tree with partitions at dyadic values along the main axes. $\mathcal{T}_I$ partitions $[0,1]^2$ into 2^k rectangles of area 2^{-k} on level k. Since we are in the binary setting, the output range is the interval $[0,1]$ and for $\Omega \in \mathcal{T}_I$, E_Ω is a scalar.

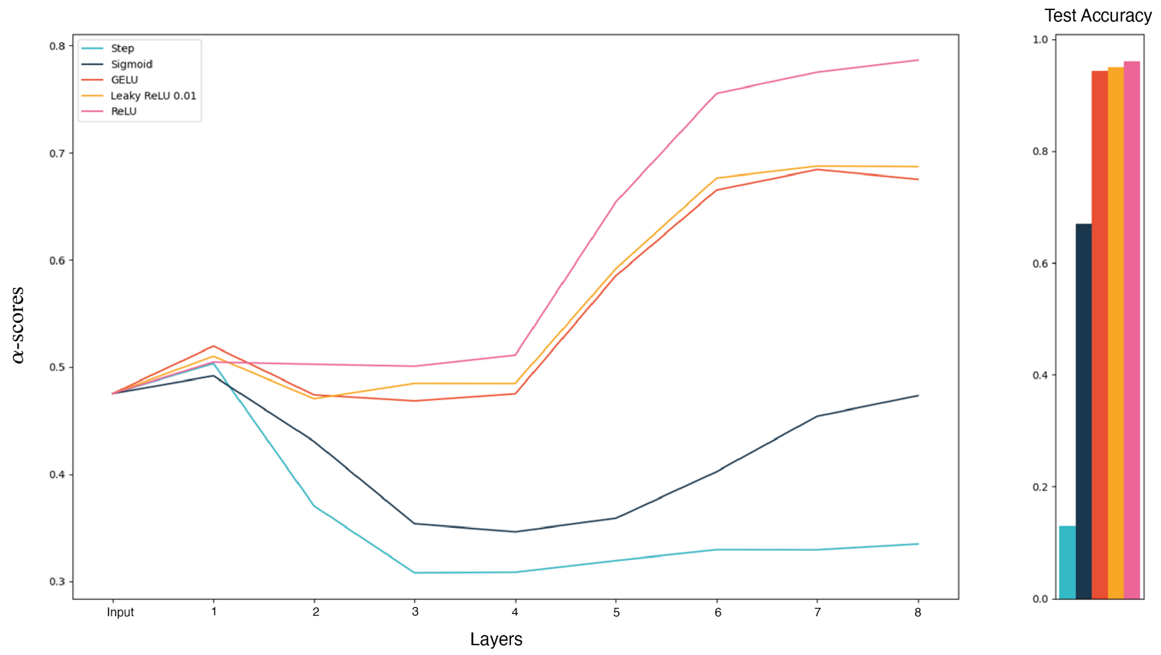

Figure 10: Comparing different Activation Functions on the MNIST-1D dataset using the Sparsity-Probe. We see correlation between the Test Scores and the $\alpha*$ scores in the last level. The proposed 'Step' activations α^* scores deteriorate throughout the layers.

Let $l(\Omega)$ be the level in which domain Ω was created in $\mathcal{T}_I$. The τ-Sparsity of $\mathcal{T}_I$ is given by:

$$\begin{aligned} N_\tau\left(f, \mathcal{T}_I\right) &:= \left(\sum_{\Omega' \in \mathcal{T}_I, \Omega' \neq [0,1]^2} \|\psi_{\Omega'}\|_2^\tau \right)^{1/\tau} \\ &= \left(\sum_{\Omega' \in \mathcal{T}_I, \Omega' \neq [0,1]^2} |E_{\Omega'} - E_\Omega|^\tau |\Omega'|^{\frac{\tau}{2}} \right)^{1/\tau} \\ &= \left(\sum_{\Omega' \in \mathcal{T}_I, l(\Omega') = k, k>0} |E_{\Omega'} - E_\Omega|^\tau |\Omega'|^{\frac{\tau}{2}} \right)^{1/\tau} \end{aligned}$$

Let $\Omega' \in \mathcal{T}_I$, and $\Omega \in \mathcal{T}_I$ be its parent in $\mathcal{T}_I$. If $\Omega' \cap \partial\tilde{\Omega} = \emptyset$, and $\Omega \cap \partial\tilde{\Omega} = \emptyset$ then $E_{\Omega'} - E_\Omega = 0$. Otherwise, if $l(\Omega') = k$,

$$|E_{\Omega'} - E_\Omega|^\tau |\Omega'|^{\frac{\tau}{2}} \leq 2^{\frac{-\tau k}{2}}$$

Therefore,

$$N_\tau\left(f, \mathcal{T}_I\right)^\tau \leq \sum_{\Omega' \in \mathcal{T}_I, k>0} 2^{\frac{-\tau k}{2}} \# A_k$$

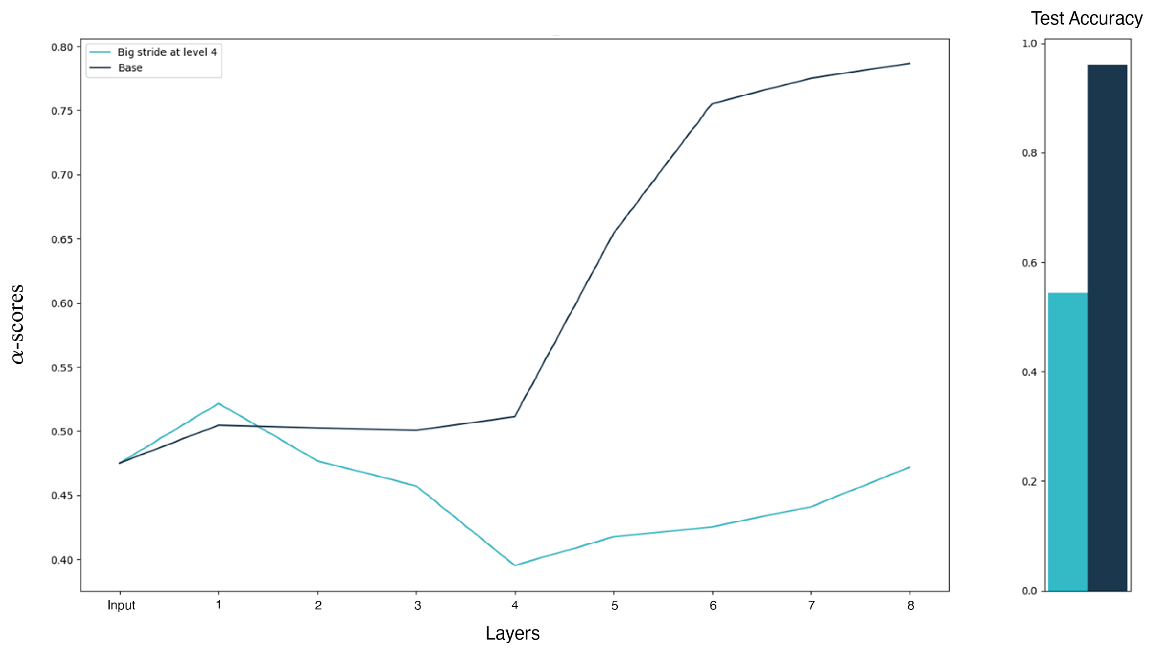

Figure 11: Adding a disproportionately big stride in the 4th layer. We see a sharp decrease in the 4th layer α^*-scores, and a slow increase after that. It is interesting to notice that there is a decrease in α^*-scores in the first three layers. This is because the gradients are blocked by the 4th layer bottleneck.

Where $\text{Pa}(\Omega')$ is the parent of Ω' in $\mathcal{T}_I$, and

$$A_k := \{\Omega' : \Omega' \in \mathcal{T}_I, l(\Omega') = k, \text{Pa}(\Omega') = \Omega, (\Omega' \cap \partial\tilde{\Omega} \neq \emptyset \vee \Omega \cap \partial\tilde{\Omega} \neq \emptyset)\}.$$

We claim that

$$\#A_k =\leq C(\tilde{\Omega})2^{\lfloor \frac{k+1}{2} \rfloor}. \tag{A.1}$$

Where $C(\tilde{\Omega})$ is a constant dependant of the domain $\tilde{\Omega}$. This implies that

$$N_\tau\left(f, \mathcal{T}_I\right)^\tau \leq C(\tilde{\Omega})\left(\sum_{k=1}^{\infty} 2^{\frac{-\tau k}{2} + \lfloor \frac{k+1}{2} \rfloor}\right)$$
$$= C(\tilde{\Omega})\left(\sum_{j=1}^{\infty} 2^{\frac{-\tau(2j)}{2} + \lfloor \frac{2j+1}{2} \rfloor} + \sum_{j=1}^{\infty} 2^{\frac{-\tau(2j+1)}{2} + \lfloor \frac{2j+2}{2} \rfloor}\right)$$
$$= C(\tilde{\Omega})\left((1 + 2^{1-\frac{\tau}{2}})\sum_{j=1}^{\infty} 2^{-j(\tau-1)}\right)$$

and so,

$$N_\tau\left(f, \mathcal{T}_I\right) < \infty \Leftrightarrow \tau > 1$$

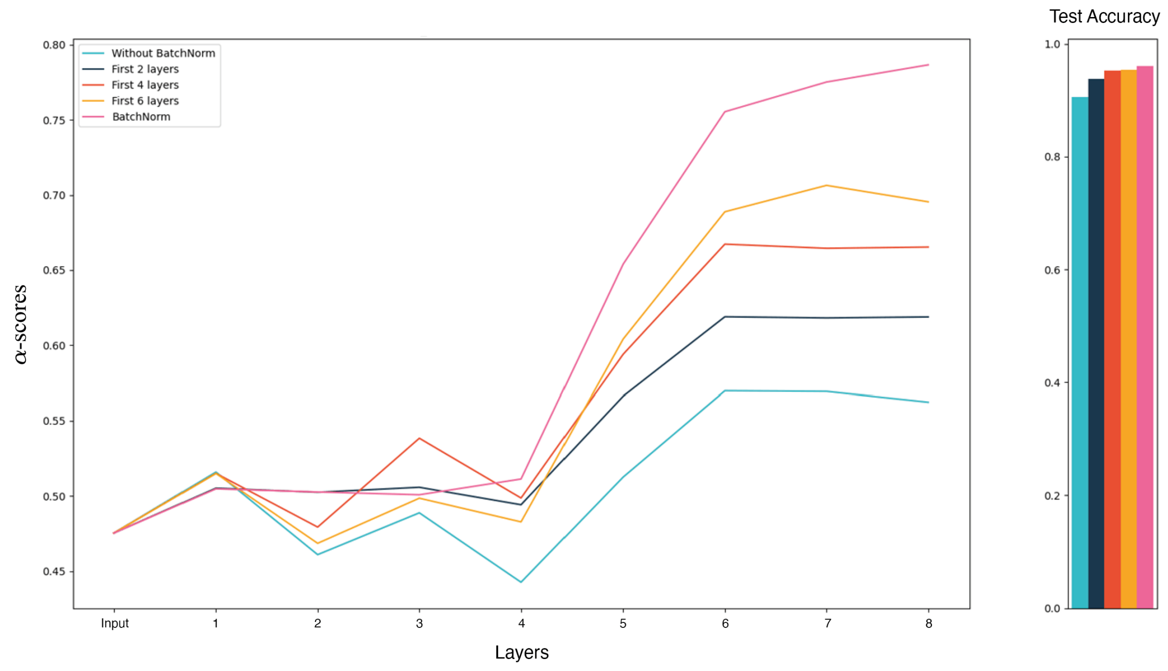

Figure 12: Comparing BatchNorm on a different array of layers using the Sparsity-Probe. We see the added benefit of adding every BN layer.

Let us return to the estimate A.1. We define:

$$B_k := \{\Omega' : \Omega' \in \mathcal{T}_I, l(\Omega') = k, \Omega' \cap \partial\tilde{\Omega} \neq \emptyset\}.$$

We notice that if:

$$\#B_k \leq C(\tilde{\Omega})2^{\lfloor \frac{k+1}{2} \rfloor}, \tag{A.2}$$

then the following relation holds:

$$\begin{aligned} \#A_k &\leq \#B_k + \#B_{k+1} \\ &\leq C_1(\tilde{\Omega})2^{\lfloor \frac{k+1}{2} \rfloor} + C_2(\tilde{\Omega})2^{\lfloor \frac{k+2}{2} \rfloor} \\ &\leq C(\tilde{\Omega})2^{\lfloor \frac{k+1}{2} \rfloor}. \end{aligned}$$

Let us now show A.2. First, we notice that it is enough to show that for every even layer $2k$, $\#B_{2k} < C(\tilde{\Omega})2^k = C(\tilde{\Omega})2^{\lfloor \frac{2k+1}{2} \rfloor}$. Once this is shown, for every odd layer, $2k+1$, the amount of domain intersections with the $\partial\tilde{\Omega}$ is at most the number of intersections in the next layer:

$$\#B_{2k+1} \leq \#B_{2k+2} \leq C(\tilde{\Omega})2^{k+1} = C(\tilde{\Omega})2^{\lfloor \frac{2k+1}{2} \rfloor}$$

Let us look at the boundary $\tilde{\Omega}$ at an even layer $2k$. There are a finite number of points where the gradient of the boundary is aligned with one of the main axes. Between these points the boundary segments

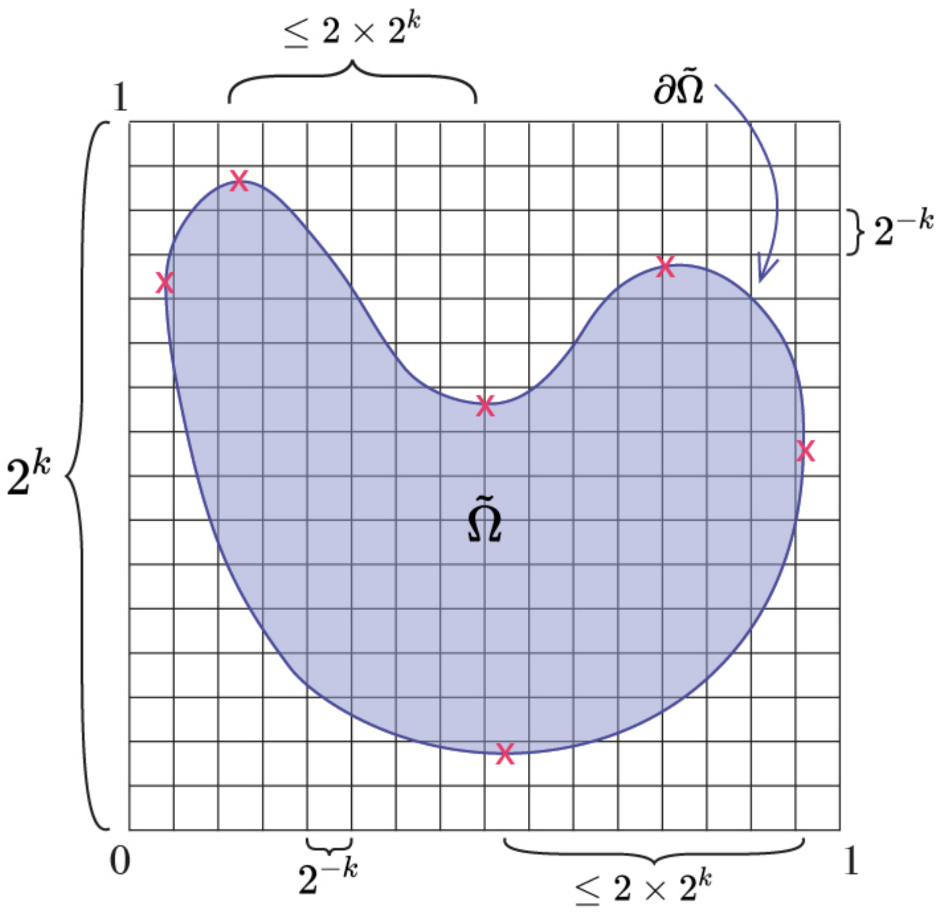

Figure 13: Visualization of bound on the number of cubes that intersect with the boundary $\partial\tilde{\Omega}$, $(\#B_{2k})$ at level $2k$.

are monotone in x_1 and x_2 and therefore the amount of cubes it intersects is at most 2×2^k. This is because on axis x_1, the boundary is monotone, and so it can intersect at most 2^k dyadic cubes. The same goes for axis x_2. We can then bound the total number of intersections at level $2k$ by $C(\tilde{\Omega})2^k$, where $C(\tilde{\Omega})$ is determined by the number of points where the boundary gradient is aligned with one of the main axes. This bound is visualized in figure 13. ■

Proof of Lemma 3.2. Let $f(x) = \sum_{k=1}^{K} c_k \mathbf{1}_{B_k}(x)$, where $B_k \subset \Omega_0$ are disjoint cubes with sides parallel to main axes, $c_k \in \mathbb{R}$. Since the cubes $\{B_k\}$ are disjoint, it is possible to find a tree $\mathcal{T}_A$ such that at level N_0, each cube B_k is in a separate domain Ω_{B_k}. The sides of B_k are parallel to the main axes, and so it is possible to partition Ω_{B_k} 4 times, such that the resulting partition Ω'_{B_k} is exactly B_k with value C_k. We notice, that any partition after this level, $l = N_1$, the value of f does not change, and therefore the for wavelets $\psi_{\Omega'}$

$$\vec{E}_{\Omega'} - \vec{E}_{\Omega} = \vec{0}$$

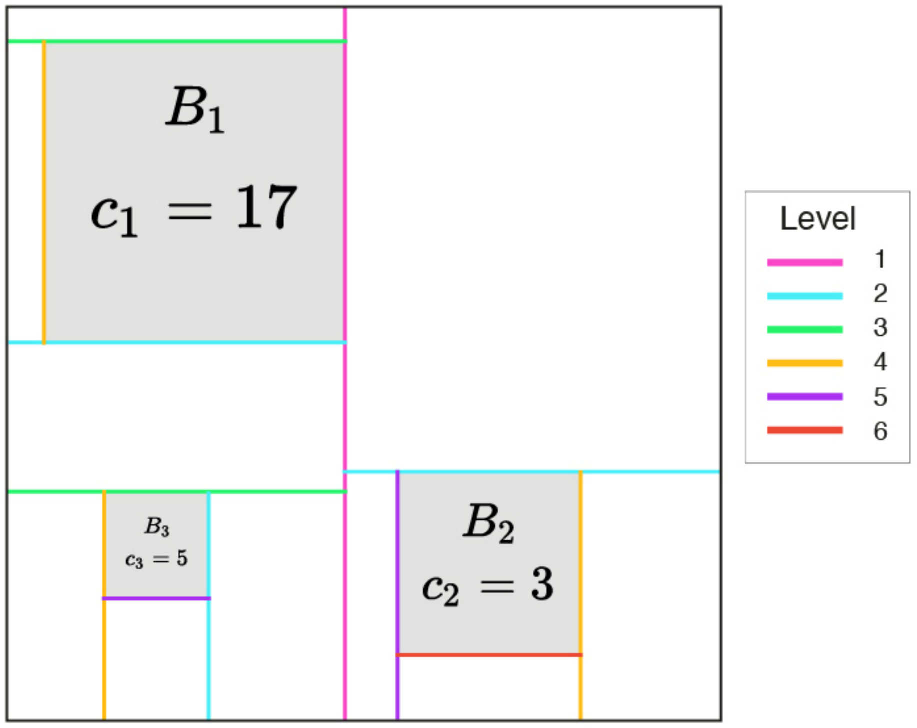

Figure 14: Example adaptive tree $\mathcal{T}_A$ on $f(x) = \sum_{k=1}^{K} c_k \mathbf{1}_{B_k}(x)$.

and,

$$\|\psi_{\Omega'}\|_{L_2} = \left\|\vec{E}_{\Omega'} - \vec{E}_{\Omega}\right\|_{l_2(\mathbb{R}^L)} \left|\Omega'\right|^{1/2} = 0.$$

Therefore,

$$\begin{aligned} N_\tau(f, \mathcal{T}_A)^\tau &:= \sum_{\Omega' \in \mathcal{T}_A, \Omega' \neq [0,1]^n} \|\psi_{\Omega'}\|_2^\tau \\ &= \sum_{\Omega' \in \mathcal{T}_A, \Omega' \neq [0,1]^n, l(\Omega') \leq N_1} \|E_{\Omega'} - E_\Omega\|_{l_2}^\tau |\Omega'|^{\frac{\tau}{2}} \\ &< \infty \end{aligned}$$

where the last transition is true since we are adding a finite amount of finite values. A visualization of a possible $\mathcal{T}_A$ is shown in figure 14. ■

REFERENCES

[1] G. Alain and Y. Bengio, *Understanding intermediate layers using linear classifier probes*, (2017).

[2] Y. Bengio, A. Courville, and P. Vincent, *Representation learning: A review and new perspectives*, IEEE transactions on pattern analysis and machine intelligence, 35 (2013), pp. 1798–1828.
[3] L. Breiman, *Random forests*, Mach. Learn., 45 (2001), pp. 5–32, https://doi.org/10.1023/A:1010933404324, https://doi.org/10.1023/A:1010933404324.
[4] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, *A simple framework for contrastive learning of visual representations*, in International conference on machine learning, PMLR, 2020, pp. 1597–1607.
[5] T.-H. Cheung and D.-Y. Yeung, {*MODALS*}*: Modality-agnostic automated data augmentation in the latent space*, in International Conference on Learning Representations, 2021, https://openreview.net/forum?id=XjYgR6gbCEc.
[6] F. Chollet, *Deep Learning with Python*, Manning, Nov. 2017.
[7] U. Cohen, S. Chung, D. D. Lee, and H. Sompolinsky, *Separability and geometry of object manifolds in deep neural networks*, Nature Communications, 11 (2020), 746, p. 746, https://doi.org/10.1038/s41467-020-14578-5.
[8] C. Cortes and V. Vapnik, *Support-vector networks*, Machine learning, 20 (1995), pp. 273–297.
[9] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, *Autoaugment: Learning augmentation policies from data*, 2019, https://arxiv.org/pdf/1805.09501.pdf.
[10] I. Daubechies, *Ten Lectures on Wavelets*, SIAM, 1992, https://doi.org/10.1137/1.9781611970104, https://doi.org/10.1137/1.9781611970104.
[11] S. Dekel and D. Leviatan, *Adaptive multivariate approximation using binary space partitions and geometric wavelets*, SIAM J. Numer. Anal., 43 (2005), p. 707–732, https://doi.org/10.1137/040604649, https://doi.org/10.1137/040604649.
[12] R. A. DeVore, *Nonlinear approximation*, Acta Numerica, 7 (1998), p. 51–150, https://doi.org/10.1017/S0962492900002816.
[13] R. A. DeVore, B. Jawerth, and B. J. Lucier, *Image compression through wavelet transform coding*, IEEE Transactions on Information Theory, 38 (1992), pp. 719–746, https://doi.org/10.1109/18.119733.
[14] J. C. Duchi, E. Hazan, and Y. Singer, *Adaptive subgradient methods for online learning and stochastic optimization*, in J. Mach. Learn. Res., 2011.
[15] M. Elad, *Sparse and Redundant Representations - From Theory to Applications in Signal and Image Processing*, Springer, 2010, https://doi.org/10.1007/978-1-4419-7011-4, https://doi.org/10.1007/978-1-4419-7011-4.
[16] O. Elisha and S. Dekel, *Wavelet decompositions of random forests - smoothness analysis, sparse approximation and applications*, Journal of Machine Learning Research, 17 (2016), pp. 1–38, http://jmlr.org/papers/v17/15-203.html.
[17] O. Elisha and S. Dekel, *Using function space theory for understanding intermediate layers*, 2018.
[18] D. Erhan, Y. Bengio, A. Courville, and P. Vincent, *Visualizing higher-layer features of a deep network*, (2009).
[19] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, The MIT Press, 2016.
[20] S. Greydanus, *Scaling down Deep Learning*, arXiv e-prints, (2020), arXiv:2011.14439, p. arXiv:2011.14439, https://arxiv.org/abs/2011.14439.
[21] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. D. Guo, M. Gheshlaghi Azar, B. Piot, K. Kavukcuoglu, R. Munos, and M. Valko, *Bootstrap your own latent: A new approach to self-supervised Learning*, arXiv e-prints, (2020), arXiv:2006.07733, p. arXiv:2006.07733, https://arxiv.org/abs/2006.07733.
[22] J. Han and C. Moraga, *The influence of the sigmoid function parameters on the speed of backpropagation learning*, in From Natural to Artificial Neural Computation, J. Mira and F. Sandoval, eds., Berlin, Heidelberg, 1995, Springer Berlin Heidelberg, pp. 195–201.
[23] D. Harris and S. Harris, *Digital Design and Computer Architecture*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.
[24] K. He, H. Fan, Y. Wu, S. Xie, and R. B. Girshick, *Momentum contrast for unsupervised visual representation learning*, 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), (2020), pp. 9726–9735.
[25] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.
[26] D. Hendrycks and K. Gimpel, *Gaussian error linear units (gelus)*, arXiv: Learning, (2016).
[27] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, in International conference on machine learning, PMLR, 2015, pp. 448–456.
[28] Y. Jiang, B. Neyshabur, H. Mobahi, D. Krishnan, and S. Bengio, *Fantastic generalization measures and where to find them*, ArXiv, abs/1912.02178 (2020).

[29] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, *Supervised contrastive learning*, ArXiv, abs/2004.11362 (2020).
[30] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, CoRR, abs/1412.6980 (2015).
[31] R. Kohavi, *A study of cross-validation and bootstrap for accuracy estimation and model selection*, in Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2, IJCAI'95, San Francisco, CA, USA, 1995, Morgan Kaufmann Publishers Inc., p. 1137–1143.
[32] D. T. Larose, *Discovering Knowledge in Data: An Introduction to Data Mining*, Wiley-Interscience, USA, 2004.
[33] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning*, Nature, 521 (2015), pp. 436–444, https://doi.org/10.1038/nature14539, https://doi.org/10.1038/nature14539.
[34] J. Lee, Y. Bahri, R. Novak, S. Schoenholz, J. Pennington, and J. Sohl-Dickstein, *Deep neural networks as gaussian processes*, ArXiv, abs/1711.00165 (2018).
[35] W. Loh, *Classification and regression trees*, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 1 (2011).
[36] J. Lu, Z. Shen, H. Yang, and S. Zhang, *Deep network approximation for smooth functions*, arXiv 2001.030402v2, (2020).
[37] J. B. MacQueen, *Some methods for classification and analysis of multivariate observations*, 1967.
[38] S. Mallat, *A Wavelet Tour of Signal Processing, Third Edition: The Sparse Way*, Academic Press, Inc., USA, 3rd ed., 2008.
[39] L. McInnes and J. Healy, *Umap: Uniform manifold approximation and projection for dimension reduction*, ArXiv, abs/1802.03426 (2018).
[40] G. Montavon, M. L. Braun, and K.-R. Müller, *Kernel analysis of deep networks*, Journal of Machine Learning Research, 12 (2011), pp. 2563–2581, http://jmlr.org/papers/v12/montavon11a.html.
[41] C. Olah, A. Satyanarayan, I. Johnson, S. Carter, L. Schubert, K. Ye, and A. Mordvintsev, *The building blocks of interpretability*, Distill, (2018), https://doi.org/10.23915/distill.00010. https://distill.pub/2018/building-blocks.
[42] V. Papyan, Y. Romano, and M. Elad, *Convolutional neural networks analyzed via convolutional sparse coding*, J. Mach. Learn. Res., 18 (2017), pp. 83:1–83:52.
[43] D. R. and L. G., *Constructive approximation*, 1993.
[44] B. D. Ripley and N. L. Hjort, *Pattern Recognition and Neural Networks*, Cambridge University Press, USA, 1st ed., 1995.
[45] H. Robbins and S. Monro, *A Stochastic Approximation Method*, The Annals of Mathematical Statistics, 22 (1951), pp. 400 – 407, https://doi.org/10.1214/aoms/1177729586, https://doi.org/10.1214/aoms/1177729586.
[46] Z. Shen, H. Yang, and S. Zhang, *Deep network approximation characterized by number of neurons*, Communications in Computational Physics, 28 (2020), pp. 1768–1811.
[47] R. Shwartz-Ziv and N. Tishby, *Opening the black box of deep neural networks via information*, ArXiv, abs/1703.00810 (2017).
[48] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, CoRR, abs/1409.1556 (2015).
[49] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, *Dropout: A simple way to prevent neural networks from overfitting*, Journal of Machine Learning Research, 15 (2014), pp. 1929–1958, http://jmlr.org/papers/v15/srivastava14a.html.
[50] J. Sulam, V. Papyan, Y. Romano, and M. Elad, *Multilayer convolutional sparse modeling: Pursuit and dictionary learning*, IEEE Transactions on Signal Processing, 66 (2018), pp. 4090–4104.
[51] A. Vaswani, N. M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention is all you need*, ArXiv, abs/1706.03762 (2017).
[52] H. Xiao, K. Rasul, and R. Vollgraf, *Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms*, ArXiv, abs/1708.07747 (2017).
[53] D. Yarotsky, *Error bounds for approximations with deep relu networks*, Neural networks : the official journal of the International Neural Network Society, 94 (2017), pp. 103–114.
[54] D. Yarotsky, *Optimal approximation of continuous functions by very deep relu networks*, in COLT, 2018.
[55] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny, *Barlow twins: Self-supervised learning via redundancy reduction*, ArXiv, abs/2103.03230 (2021).
[56] S. Zheng, Y. Song, T. Leung, and I. Goodfellow, *Improving the robustness of deep neural networks via stability training*, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), (2016), pp. 4480–4488.