SplineCam: Exact Visualization and Characterization of Deep Network Geometry and Decision Boundaries

Ahmed Imtiaz Humayun
Rice University
imtiaz@rice.edu

Randall Balestriero
Meta AI, FAIR
rbalestriero@fb.com

Guha Balakrishnan
Rice University
guha@rice.edu

Richard Baraniuk
Rice University
richb@rice.edu

Abstract

Current Deep Network (DN) visualization and interpretability methods rely heavily on data space visualizations such as scoring which dimensions of the data are responsible for their associated prediction or generating new data features or samples that best match a given DN unit or representation. In this paper, we go one step further by developing the first provably exact method for computing the geometry of a DN's mapping – including its decision boundary – over a specified region of the data space. By leveraging the theory of Continuous Piece-Wise Linear (CPWL) spline DNs, ***SplineCam*** *exactly computes a DN's geometry without resorting to approximations such as sampling or architecture simplification. SplineCam applies to any DN architecture based on CPWL activation nonlinearities, including (leaky) ReLU, absolute value, maxout, and max-pooling and can also be applied to regression DNs such as implicit neural representations. Beyond decision boundary visualization and characterization, SplineCam enables one to compare architectures, measure generalizability, and sample from the decision boundary on or off the data manifold. Project website: bit.ly/splinecam.*

1. Introduction

Deep learning and in particular Deep Networks (DNs) have redefined the landscape of machine learning and pattern recognition [22]. Although current DNs employ a variety of techniques that improve their performance, their core operation remains unchanged, primarily consisting of sequentially mapping an input vector $\boldsymbol{x}$ to a sequence of L *feature maps* $\boldsymbol{z}^\ell$, $\ell = 1, \dots, L$ by successively applying simple nonlinear transformations, as in

$$\boldsymbol{z}^\ell = \sigma\left(\boldsymbol{W}^\ell \boldsymbol{z}^{\ell-1} + \boldsymbol{b}^\ell\right), \quad \ell = 1, \dots, L \qquad (1)$$

starting with $\boldsymbol{z}^0 = \boldsymbol{x}$. Here $\boldsymbol{W}^\ell$ and $\boldsymbol{b}^\ell$ denotes the weight matrix and the bias vector for layer ℓ, and σ is an activation operator that applies an element-wise nonlinear activation function. One popular choice for σ is the Rectified Linear Unit (ReLU) [12] that takes the elementwise maximum between its entry and 0. The parametrization of $\boldsymbol{W}^\ell, \boldsymbol{b}^\ell$ controls the type of layer, e.g., circulant matrix for convolutional layer.

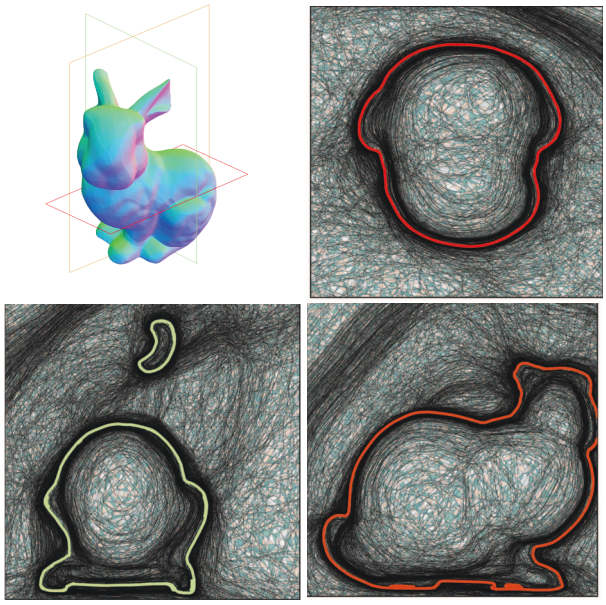

Figure 1. Exact visualization of the decision boundary and partition geometry of a 3D neural signed distance field (SDF). **(Top left)** Surface normals obtained from the learned signed distance field with annotations indicating slices used for visualization. For each of the slices, we can see the spline partition geometry of the learned SDF- each contiguous line represents a neuron, on either side of which it gets activated/deactivated. Neurons from different depths of the network create a partitioning of the input space into 'linear regions'. Here the colored lines represent the decision boundary learned by the SDF. Note that while the final neuron obtains the decision boundary, many neurons place their boundaries close to the ground truth surface to obtain the final SDF representation.

Interpreting the geometry of a DN is a nontrivial task since many different sets of parameters can lead to the same

979-8-3503-0129-8/23/$31.00 ©2023 IEEE
DOI 10.1109/CVPR52729.2023.00369

input-output mapping. One example is obtained by permuting the rows of $\boldsymbol{W}^\ell, \boldsymbol{b}^\ell$ and the columns of $\boldsymbol{W}^{\ell+1}$ for any two consecutive layers in a DN. Another example is to rescale $\boldsymbol{W}^\ell, \boldsymbol{b}^\ell$ by some constant κ and to rescale $\boldsymbol{W}^{\ell+1}$ by $1/\kappa$ for a ReLU-DN [33]; the list of such parameter manipulations preserving the underlying DN's function is an active area of research [32]. Since one cannot trivially use the DN's parameters to describe its mapping, practitioners have relied on different solutions to interpret what has been learned by a model by looking at the activations instead of the weights of the network [19, 41]. Activation based interpretability methods however can be susceptible to feature adversarial attacks, i.e., adversarial attacks that don't cross the decision boundary but changes the activation [11]. Some alternative empirical methods for model interpretation, therefore rely on sampling the decision boundary or finding the point on a model's decision boundary closest to a sample $\boldsymbol{x}$ [36]. Beyond interpretability, such methods find practical use in active learning [23] and adversarial robustness [15]. In this setting, gradient updates are performed from an initial guess for $\boldsymbol{x}$ based on an objective function that reaches its minimum whenever its argument lies on the model's decision boundary. While alternative and more efficient solutions have been developed, most of the progress in this direction has focused on providing more optimized losses and sampling strategies [15, 36]. In short, *there doesn't exist an exact (up to machine precision) method to compute the decision boundary of a DN.*

In this paper, we focus on DNs employing Continuous Piece-Wise Linear (CPWL) activation functions σ, such as the (leaky-)ReLU, absolute value, and max-pooling. In this setting, the entire DN itself becomes a CPWL operator, i.e., its mapping is affine within regions of a partition of its domain. There has been previous studies dedicated to estimating the partition of such CPWL DNs and bridging empirical findings with interpretability. For example, Raghu et al. [34] shows that the partition density provides measures of DN expressivity, Hanin et al. [13] connects the DN partition density with the complexity of the learned function, Jordan et al. [20] approximates the DN partition to provide robustness certificates, Zhang et al. [43] interprets the impact of dropout with respect to DN partitions, Balestriero et al. [3] proposes to improve batch-normalization to further adapt DN partitions to the data geometry, Humayun et al. [17, 18] proposes methods to control pre-trained generative network output distributions by approximating the DN partition, Chen et al. [25] proposes a neural architecture search method based on partition statistics. Despite being successful, *all these methods rely on approximation of the DN partition.*

We propose *SplineCam*, a sampling-free method to compute the exact partition of a DN. Our method computes the partition on two-dimensional domains of the input space, easily scales with width and depth of DNs, can handle convolutional layers and skip connections, and can be scaled to discover numerous regions as opposed to previously existing methods. Our method also allows local characterization of the input space based on partition statistics, and enables one to tractably and efficiently sample arbitrarily many samples that provably lie on a DN's decision boundary - opening new avenues for visualization and interpretability. We summarize our contributions as follows:

- Development of a scalable enumeration method that, given a bounded 2D domain of a DN's input space, computes the DN's input space partition (aka, linear regions) and decision boundary.
- Development of **SplineCam** that leverages our new enumeration method to directly visualize a DN's input space partition, compute partition statistics and sample from the decision boundary.
- Quantitative analysis that demonstrates the ability of SplineCam to characterize the DN and compare between architectural choices and training regimes.

The **SplineCam** library, and codes required to reproduce our results are provided in our Github[1]. In Appendix E, we also demonstrate the usage of SplineCAM with example code blocks.

2. The Exact Geometry and Decision Boundary of Continuous Piece-Wise Linear Deep Networks

The goal of this section is to first introduce basic notations and concepts associated with CPWL DNs (Sec. 2.1), and then develop our method that consists of building the exact DN input space partition, and the DN's decision boundary that lives on it (Sec. 2.2); empirical studies based on our method will be provided in Sec. 4.1.

2.1. Deep Networks as Continuous Piece-Wise Linear Operators

One of the most fundamental functional form for a nonlinear function emerges from polynomials, and in particular, spline operators. In all generality a spline is a mapping which has locally degree D polynomials on each region ω of its input space partition Ω, with the additional constraints that the first $D-1$ derivatives of those polynomials are continuous throughout the domain, i.e., imposing a smoothness constraint when moving from one region to any of its neighbor. More formally and for the context of DNs we will particularly focus on affine splines, i.e., spline operators with

[1] https://bit.ly/splinecam-git

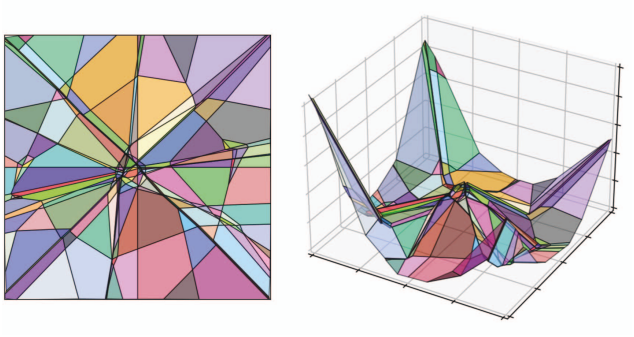

Figure 2. Visual depiction of Eq. 2 with a toy affine spline mapping $S : \mathbb{R}^2 \to \mathbb{R}^3$. **Left.** Input space partition Ω made of multiple convex regions shown with different colors and with boundaries shown in black. **Right.** Affine spline image $Im(S)$ which is a continuous piecewise affine surface composed of the input space regions affinely transformed by the per-region affine mappings. Colors maintain correspondence from the left to the right.

$D = 1$ and only constrained to enforce continuity throughout the domain.

Let $\boldsymbol{S}$ be a Deep Network (DN) with L layers and parameters $\{\boldsymbol{W}^\ell, \boldsymbol{b}^\ell\}_{\ell=1}^L$. Whenever $\boldsymbol{S}$ employs continuous piecewise affine (CPA) activation σ at each layer, i.e., layer ℓ outputs are given by Eq. 1, with $\boldsymbol{z}^0$ being an input $\boldsymbol{x} \in \mathbb{R}^S$.

Lemma 1. *The layer* 1 *to* ℓ *composition of a DN* $\boldsymbol{S}$*, denoted as* $\boldsymbol{S}^\ell$ *with output space* $\mathbb{R}^\ell$*, can be expressed as*

$$\boldsymbol{S}^\ell(\boldsymbol{x}) = \sum_{\omega \in \Omega} \left(\boldsymbol{A}_\omega^\ell \boldsymbol{x} + \boldsymbol{b}_\omega^\ell\right) \mathbb{1}_{\{\boldsymbol{x} \in \omega\}}, \tag{2}$$

with indicator function $\mathbb{1}_{\{.\}}$ *and per-region affine parameters given by,*

$$\boldsymbol{A}_\omega^\ell = \prod_{i=1}^{\ell} \text{diag}\left(\boldsymbol{q}_\omega^i\right) \boldsymbol{W}^i, \tag{3}$$

$$\boldsymbol{b}_\omega^\ell = \text{diag}\left(\boldsymbol{q}_\omega^\ell\right) \boldsymbol{b}^\ell + \sum_{i=1}^{\ell-1} \left(\prod_{j=i+1}^{\ell} \text{diag}\left(\boldsymbol{q}_\omega^j\right) \boldsymbol{W}^j \right) \text{diag}\left(\boldsymbol{q}_\omega^i\right) \boldsymbol{b}^i. \tag{4}$$

Here, $\boldsymbol{q}_\omega^\ell$ is the point-wise derivative of σ at pre-activation $\boldsymbol{W}^\ell \boldsymbol{z}^{\ell-1} + \boldsymbol{b}^\ell$, and diag(.) operator given a vector argument creates a matrix with the vector values along the diagonal. As a consequence of Thm. 1 from Balestriero et al. [2], $\boldsymbol{q}_\omega^\ell$ is unique for any region $\omega \in \Omega$.

Such formulations of DNs have previously been employed to make theoretical studies amenable to actual DNs without any simplification, while leveraging the rich literature on spline theory, e.g., in approximation theory [6], optimal control [7], statistics [8] and related fields.

Algorithm 1 Find Partitions

Input: 2-polytope P and hyperplanes $\mathcal{H} \in \mathbb{R}^2$, s.t., $\mathcal{H} \cap P \neq \emptyset$.

Output: G, C. $G = (E, N)$, where E are edges and N are nodes of the graph G. C are cycles/cells/faces of G.

Step 1: Solve for $N = \{h_i \cap (h_j \cup P_e)\ \forall h_i, h_j \in \mathcal{H} : j \neq i\}$, where P_e are edges of P.

Step 2: For each h_i, sort $\{h_i \cap (h_j \cup P_e)\ \forall h_j \in \mathcal{H} : j \neq i\}$ and connect in sorted sequence to obtain edges E.

Step 3: Obtain set of cycles C from graph G via Alg. 2.

2.2. Exact Computation of Their Partition and Decision Boundary

Suppose, $\boldsymbol{w}_i^\ell$,$\boldsymbol{b}_i^\ell$ are the i-th rows of $\boldsymbol{W}^\ell, \boldsymbol{b}^\ell$. The following lemma provides us a framework to back-project to $\mathbb{R}^S$ a hyperplane $h_i^\ell \in \mathbb{R}^{\ell-1}$ from layer ℓ with parameters $\boldsymbol{w}_i^\ell, \boldsymbol{b}_i^\ell$, expressed as,

$$h_i^\ell \triangleq \{\boldsymbol{z} \in \mathbb{R}^{\ell-1} : \langle \boldsymbol{w}_i^\ell, \boldsymbol{z} \rangle + \boldsymbol{b}_i^\ell = 0\ \}. \tag{5}$$

Lemma 2. *Given a hyperplane* $h_i^\ell \in \mathbb{R}^{\ell-1}$*, it can be projected onto the tangent space of region* $\omega \in \Omega$ *in* $\mathbb{R}^S$ *as,*

$$proj_\omega(h_i^\ell) = \{\boldsymbol{x} \in \mathbb{R}^S : \langle \boldsymbol{w}_i^\ell, \boldsymbol{A}_\omega^{\ell-1} \boldsymbol{x} + \boldsymbol{b}_\omega^{\ell-1} \rangle + \boldsymbol{b}_i^\ell = 0\}. \tag{6}$$

The proof of Lem. 2 is direct since $\boldsymbol{z}_i^\ell = \langle \boldsymbol{w}_i^\ell, \boldsymbol{A}_\omega^{\ell-1} \boldsymbol{x} + \boldsymbol{b}_\omega^{\ell-1} \rangle + \boldsymbol{b}_i^\ell$, with $\boldsymbol{z}_i^\ell$ the i-th element of layer ℓ activation.

Theorem 1. *Let,* $\boldsymbol{S}$ *is a binary classifier DN therefore with a single output neuron. In* $\mathbb{R}^{L-1}$*, the decision boundary is the hyperplane* h_1^L*. The decision boundary in* $\mathbb{R}^S$ *is therefore* $\cup_{\omega \in \Omega} \{proj_\omega(h_1^L) \cap \omega\}$.

Thm. 1 can be proven by repeatedly applying Lem. 2 to back-project the hyperplane h_1^L for all $\omega \in \Omega$. While the above is general, we want to compute Ω on a 2-polytope $P \in \mathbb{R}^S$ for tractability [1] and ease of visualization.

SplineCam. Let's denote the partition in the input space formed by the composition of layers 1 to ℓ as Ω^ℓ. Using Alg. 1, SplineCam starts by partitioning P into Ω^1 via hyperplanes h_i^1 from layer $\ell = 1$, the first layer. Then for each $\omega \in \Omega^1$, we use Lem. 1 and Lem. 2 to obtain $proj_\omega(h_i^2)$ for layer two. Therefore, we use Alg. 1 on each region to obtain a finer partition. For a given polytopal region ω, the target of Alg. 1 is to first compute the graph G formed via intersections between the edges of ω and a set of hyperplanes $\mathcal{H}$. Next, via Alg. 2, it finds the unique cycles in G (also referred to as circuits or faces for a planar graph). By repeating Alg. 1 for each region $\omega \in \Omega^1$, we can obtain Ω^2.

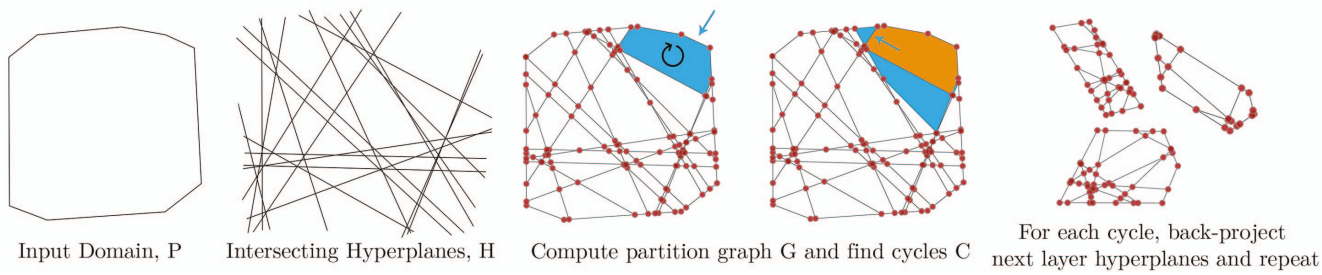

Figure 3. Given an input domain P and a set of hyperplanes $\mathcal{H}$, SplineCam first produces a graph G using all the edge and hyperplane intersections (as in Alg. 1). To find all the cycles in the graph, we select a boundary edge e_s (blue arrow), do a breadth-first search (BFS) to find the shortest path through the graph between vertices of e_s and obtain the corresponding cycle (blue). The edges obtained via BFS are enqueued and the search is repeated for each. Each non-boundary edge is allowed to be traversed twice, once from either direction (see Alg. 2). Once new regions are found, we back-project deeper layer hyperplanes, compute partition graphs, and repeat.

Algorithm 2 Find cycles

Input: $G = (E, N)$ an undirected graph, $P_e \subset E$ boundary edges and starting edge $e_s \subset P_e$.
Output: C cycles.

```
Initialize C = ∅, E_q = ∅
G' =bidirectional(G)            ▷ connect edges both ways
APPEND E_q ← e_s                ▷ append to end of queue
REMOVE e_s from G'
while E_q ≠ ∅ do
    POP e ← E_q                 ▷ get from top of queue
    REMOVE e' from G'
                                ▷ e' is e with its direction inverted
    E_c = bfs(G', v_e, v_s)
                                ▷ v_s, v_e are start and end vertices of e
    ▷ E_c are edges forming shortest path from v_e to v_s
    APPEND E_q ← e'_c, ∀e_c ∈ E_c if e_c ∉ P_e
    REMOVE e_c from G', ∀e_c ∈ E_c
    REMOVE e'_c from G', ∀e_c ∈ E_c s.t. e_c ⊂ P_e
    APPEND E_c ← e'             ▷ append edge to form cycle
    APPEND C ← E_c
end while
```

We repeat this process for all layers up to $\ell = L$ to obtain the final partition. We have provided pseudocode for the search algorithm in python script in Suppl. E List. 3.

Scalability and Complexity. SplineCam is scalable, all the SplineCam operations can be vectorized except for the search algorithm, which finds cycles in a given graph G. For the vectorized operations, we can trade-off time complexity with space complexity by allocating more memory, preferably on GPU. For Alg. 2, scaling requires distributing sets of $(\omega, \mathcal{H})$ across CPU threads. Given a set of n intersecting hyperplanes and a 2-polytope P, the operation of Alg. 1 to find the partition reduces to an arrangement of lines problem. Therefore, the number of intersections, edges and cycles $\leq \mathcal{O}(n^2)$ [1]. As the number of hyperplanes $n \to \infty$, the expected number of edges per cycle $\approx \mathcal{O}(1)$. We can also see this in Table. 1, where we see that regardless of the architecture or training dataset, the average number of edges per region converges to 4. Therefore, the average case complexity of Alg. 2 as $n \to \infty$ is also of the order $\mathcal{O}(n^2)$. In Suppl. Fig. 10, we present the wall time required for SplineCam for a randomly initialized single layer MLP with variable width and input dimensionality. For an MLP with width 1000, input dimensionality 8002, and 8M params, it takes SplineCam $134s$ to find $132K$ regions. We present in Suppl. Fig. 18 partition visualization for such a setting. For deeper networks, e.g., 138M parameter VGG16 pre-trained on tinyImagenet, SplineCam takes $\sim$ 7 minutes to find 1K regions. We also provide statistics and visualizations for deeper networks, in Fig. 8 and Suppl. Materials.

The methods closest to SplineCam in the literature are by Yuan et al. [39] that uses an exponential complexity linear programming based algorithm to compute the DN partitions and Gamba et al. [10] a method that computes the intersection of partition boundaries with one-dimensional lines connecting pairs of training samples. SplineCam computes the partition of 2D input domains and can compute the per region affine functions as well. *SplineCam is the first exact method that is fast, scalable and computes the partition on 2D slices for a wide array of architectures.*

3. Visualizing and Understanding Implicit Neural Representations

We start our journey into the geometry of DNs by looking at implicit neural representations (INRs). INRs are generally multi-layer perceptrons (MLPs) that are trained to produce a continuous mapping from 1D/2D/3D signal coordinates to the intensity of the signal at those coordinates. They are pervasive in applications like 3D view synthesis [27] and inverse problems [37]. The low input dimensionality, and interpretability of ground truth functions make INRs

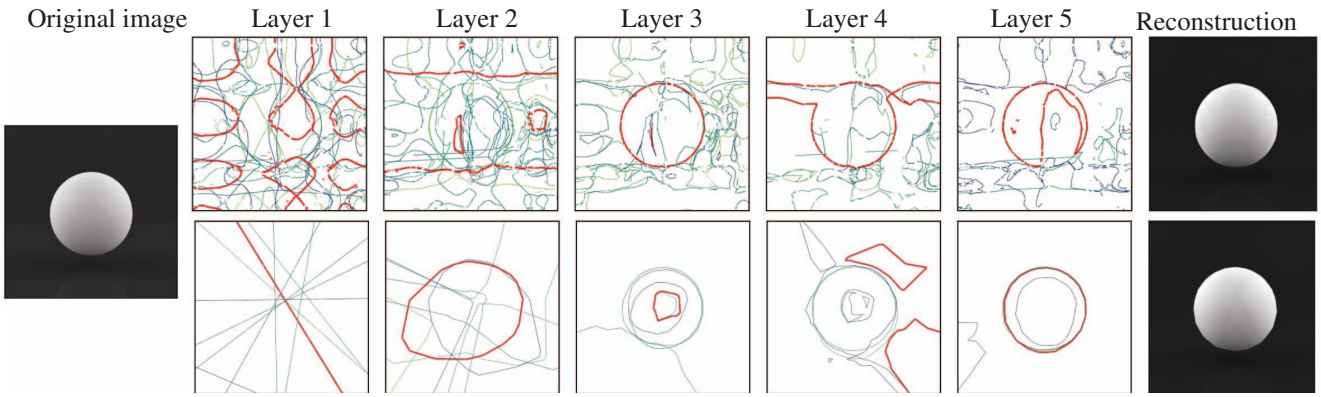

Figure 4. SplineCam visualizations of neurons from different layers of an MLP trained with **(top)** and without **(bottom)** periodic position encoding on a 2D image fitting task. All the neurons are visualized in the input space, color coded by the same color, and one neuron from each layer is highlighted in red. The trained MLP has a width of 10 and depth of 5 and has ReLU activations for every layer. For the positionally encoded (PE) network, boundaries of some neurons seem to be periodically repeating in the input domain, significantly increasing the number of unique ω where the ReLU is active. *The increased weight sharing, i.e., same weights/neurons being used to represent/fit non-contiguous parts of the learned function, could be a possible reason for improved convergence of PE MLP [29].*

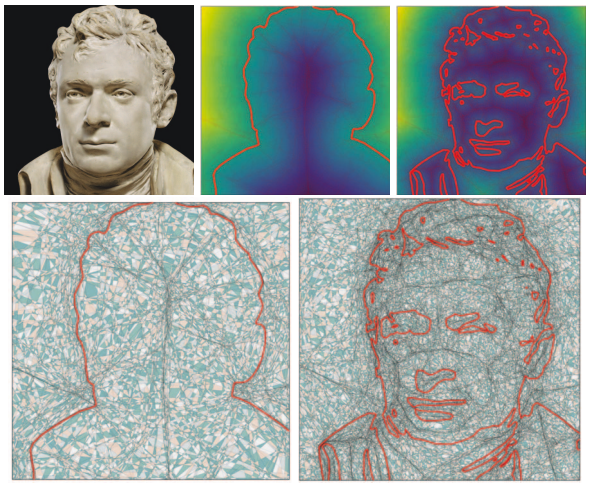

Figure 5. Visualization of the decision boundary and partition geometry of a 2D neural SDF with width 256 and depth 6. A single training image is thresholded at 0.01 and 0.5 to create two signed distance fields (**top-middle and top-right**), on which an MLP is trained. We use SplineCam to obtain the analytical zero level set (decision boundary, in red) and also visualize the partition geometry **(bottom)**. Note that even with identical architecture, the partition density differs significantly based on the task complexity.

a good setting to qualitatively validate SplineCam. There also exists a lack of theoretical understanding for current INR practices [42]. For example, while ReLU MLPs were primarily used in NeRF [26]- one of the most popular INR applications- the current practice has moved towards using periodic activations to encode of the input coordinates and following up with a ReLU MLP. In this section, we look at the effect of periodic encodings and visualize the geometry of the regions learned by INRs.

3.1. Decision Boundary of Signed Distance Functions

A signed distance function (SDF) is an implicit continuous representation of a surface or boundary; the output of an SDF is the signed distance of an input from the boundary represented by the function. The zero level set of an SDF therefore denotes the surface or boundary of the function. Training an INR as a signed distance function is essentially a regression task, where a ground truth distance field is fit by the model to implicitly learn a continuous boundary. We train a 2D and 3D SDF and visualize the analytical zero level set, à la decision boundary, using our method, and provide the spline partitioning learned by the functions in Fig. 1 and Fig. 5.

To train an INR as a 2D SDF, we take the image as in Fig. 5 from the MetFaces [21] dataset and threshold it at .001 and .05 to create two binary images. Each binary image is used to create separate ground truth SDFs, one with a simpler boundary separating the background (ESDF) and another with a more convoluted boundary (HSDF). We train an identical ReLU-MLP architecture with width 256 and depth 6 on both ESDF and HSDF. In Fig. 5 we present the analytical decision boundary of the SDF overlaid on the ground truth signed distance field for both ESDF and HSDF. While the network capacity remains the same for both, we can see that the spline partition of the two figures vary dramatically, with a finer partition and higher region density for the HSDF task compared to ESDF. For the harder HSDF task, creating significantly more regions allows the network to learn the curvature of the decision boundary. *This indicates that harder tasks may utilize more of the network parameters compared to easier tasks.*

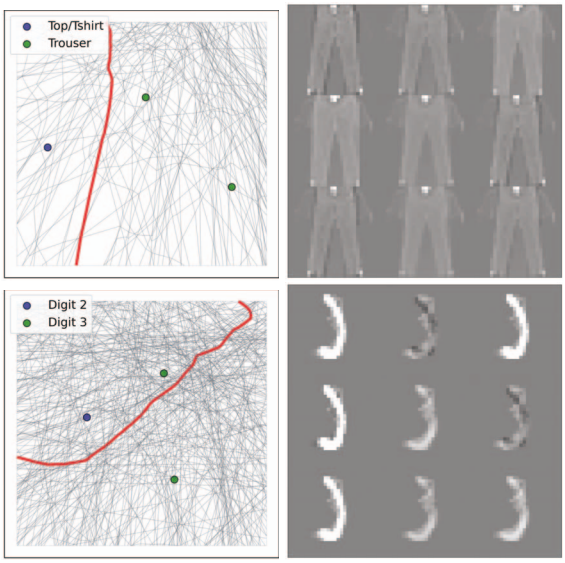

Figure 6. **(Top Left)** Decision boundary visualization for an MLP of width 50 and depth 3 trained on fashion-MNIST. Red lines represent the learned decision boundary while black lines represent the spline partition of the network. **(Top Right)** Samples from the decision boundary between classes Top and Trouser. The samples are ambiguous with distinguishable attributes present from both classes, indicating a good local decision function. **(Bottom Left)** SplineCam visualization for a CNN trained on MNIST, with two convolutional layers and one hidden fully connected layer of width 50. One of the digit 3 samples is misclassified by the network as digit 2. **(Bottom Right)** Samples from the decision boundary between digits 2 and 3 of MNIST. Some of the samples clearly look like the digit three, indicating that the decision boundary here could be sub-optimal.

For the 3D SDF task, we train a leaky-ReLU-MLP with width 256 depth 6 on a Stanford Bunny SDF, and present in Fig. 1 the normal map of the learned SDF, as well as the spline partition and decision boundary on three 2D slices, $\{x = 0, y = 0, z = 0\}$. What is noticeable here is that, apart from the final layer hyperplane (final output neuron weights), many neurons from the deeper layers of the network also place their boundaries near the zero level set of the SDF. Meaning, while the decision boundary is denoted by the output neuron, there are multiple neurons that learn the surface boundary. *This is indicative of a hierarchical nature of signal fitting by INRs.*

3.2. The Effect of Positional Encoding on INRs

Empirical evidence shows that INRs trained with periodically encoded (PE) coordinates instead of input space coordinates, are able to fit the input signal better with faster convergence rates. While initially inspired by its use in transformer networks there has been very less theoretical investigation of how positional encoding affects learning. We train a 2D INR to regress the grayscale intensity of an image, for given pixel coordinates (Fig.4). We use a ReLU MLP as the INR backbone, with width 10 and depth 5, and visualize the partition induced with/without a periodic position encoding front-end. Since SplineCam works with affine nonlinearities only, we use a piecewise approximation of sine/cosine while training, with 5 linear regions for each period of the sine/cosine. We see that using this encoding has negligible effect on performance compared to using continuous cosine and sine functions for encoding. In Fig. 4 we present a layerwise visualization where we separately show for each layer the neurons in the input space. We also highlight in red, the boundary of a single neuron from each layer.

For the PE network, boundaries of some neurons seem to be periodically repeating across the input domain. This is due to the periodic wrapping of space induced by PE, i.e., the input domain is wrapped around in the embedded space between $[-1, 1]$ for each dimension, which gets cut by subsequent ReLU hyperplanes. The effect of periodicity is most evident for the first layer hyperplanes, as can be seen from the highlighted neuron in Fig. 4. Such repetition of neurons across different parts of the input space, *significantly increases the number of regions and weight sharing across input space*, which could be a possible reason for improved convergence [29]. We also see a layerwise learning of the boundary of the sphere, indicating that multiple neurons across different depths coordinate to complete the final regression task. The absence of some neurons from the input space domain also shows that not all neurons actively partake in the regression task. For example, while for the first layer of the non-PE network, all 10 hyperplanes intersect the input domain, for the last layer only 4 of the 10 neurons intersect the input domain. *This shows how different neurons create redundancy by remaining active/inactive for all possible inputs observed during training.*

4. How Training Decisions Impact Your Spline

Recall from Sec. 2.1 that any DN with CPWL nonlinearities is a CPWL mapping or affine spline. Affine splines have been widely studied and many of their properties, e.g. number of regions in their partition, are known to convey information on function complexity [13]. In this section, we explore the effect of different training choices, e.g., architecture, data-augmentation, on DN partition.

4.1. Impact of Architecture on Partitions Properties

In this section we explore the impact of the DN's architecture on the partition. To start off, we look at the partitions induced by MLPs and CNNs when trained on fashion-MNIST and MNIST datasets. We quantify the characteristics of the partitions via the following measures- Average Region Volume (ARV), Average Number of Vertices (ANV), Number of Regions (NR), and Average Region Ec-

Table 1. Statistics of the spline partitions formed by fully connected (MLP) and convolutional (CONV) neural networks. For each dataset, the same 2D slice and input domain is used to find the partition regions. Convolutional neural networks have a significantly higher number of regions with lower volume compared to MLPs even with less parameters.

Architecture	Dataset	Parameters	Avg. Reg. Vol	Avg. Number of Vertices	Ecc.	Number of Regions
MLP	MNIST	44,860	3.144e-4	4	102e7	318
	Fashion-MNIST	44,860	4.991e-4	4	36e7	1364
CONV	MNIST	39,780	1.134e-5	4	17e7	8814
	Fashion-MNIST	39,780	3.54e-5	4	14e7	28222

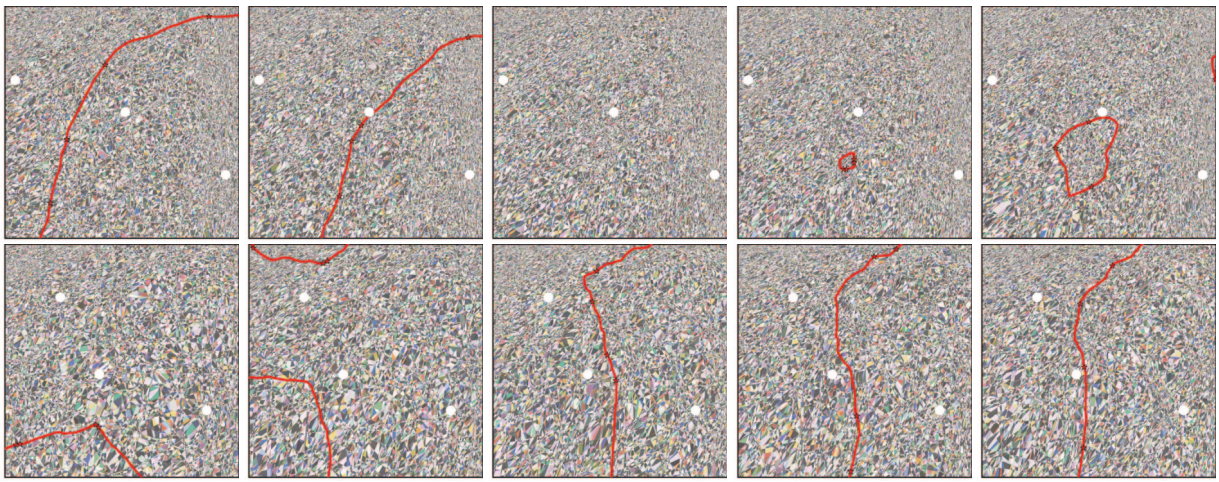

Figure 7. Decision boundary of a 5 layer convnet during training epochs $\{50,100,150,200,300\}$ (**columns**) trained on a binary classification task to classify between Egyptian cat and Tabby Cat classes of tinyimagenet. White points are images from the dataset that are used to determine the $2D$ input domains. We can observe that between epochs 50 and 100, not much change occurs to the decision boundary (**red line**). However, between epoch 100 and 150, a sudden change occurs for both cases. Especially for top row, the decision boundary moves out of the input domain. Following that, between epochs 200 and 300 the boundary stays roughly identical until the end of training. In Suppl. Fig. 12 we present the change of partition statistics at different train, test and off-manifold locations, while training a CNN on CIFAR 10. In Suppl. Sec. D.1 we present further discussions.

centricity (ARE) which is defined as the ratio of the max and min pairwise distance between vertices. For a given dataset, we fix the input domain and switch between convolutional and fully connected architectures to draw emphasis on the effect of the convolutional layers. We see that in convolutional architectures, there is a significantly higher number of partition regions formed. This is intuitive since convolutional architectures repeat the same set of parameters across sets of dimensions via a circulant weight matrix. This increases the number of cuts by the same neurons in the input space, demonstrating higher complexity for the same number of parameters [28]. In Tab. 1 we present partition statistics comparisons between architectural choices and datasets. We see that the eccentricity and volume of the polytopes are significantly smaller for convolutional architectures compared to fully connected architectures. These can also be visualized in Fig. 6. In Suppl. Fig.12 we present the variation of partition statistics with training, for training points, test points and points off the data manifold. We see that partition density increases for sample on the data manifold regardless of being on training or testing.

4.2. Data-Augmentation

Data-Augmentation is a ubiquitous technique that has led to great improvements into DN performance [9]. It is still unclear what is the impact of DA onto the DN's mapping. In fact, while explicit regularizers of DA has been found [5,16] and while many empirical studies have emerged [40], none truly pinpoint what is different between DNs trained with DA and DNs trained without.

In order to provide a first quantitative understanding of what actually changes within a DN when DA is applied, we propose to rely on SplineCam. In Fig. 8 we provide the ARV, NR, and ARE for spline partitions computed from randomly oriented 2D domains centered on 90 tinyImageNet test sam-

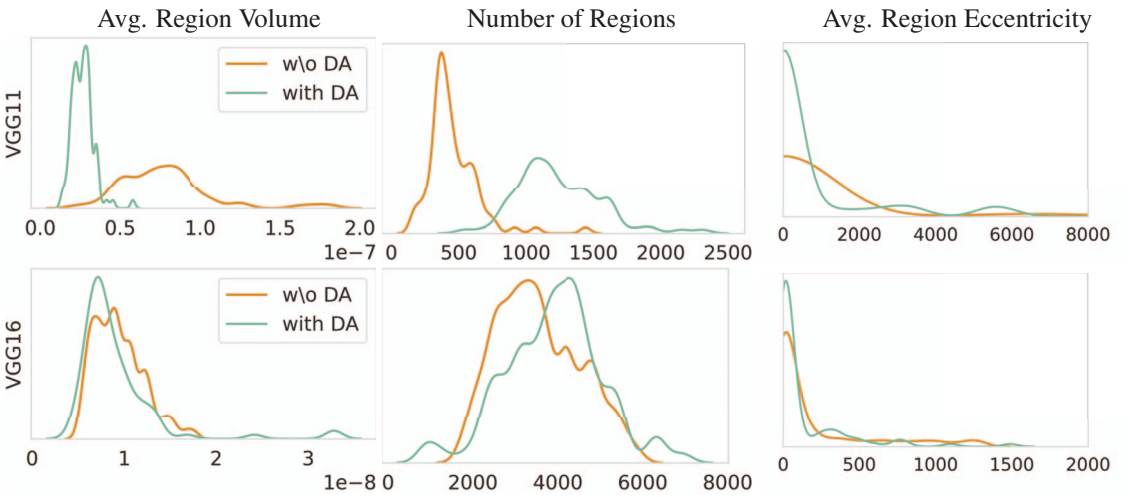

Figure 8. Average partition statistics for 90 tinyimagenet test samples with and without DA training for VGG11 and VGG16. The average volume and number of regions are indicative of partition density whereas eccentricity is indicative of the shape of the regions. For VGG11 a distinct difference in the statistics can be visualized between DA and non-DA training. DA training significantly increases the partition density at test points, which is indicative of better generalizability. On the other hand, the difference reduces for VGG16 while the overall region density increases. This is expected behavior since the VGG16 has significantly more parameters. For both case, the DA models acquired a similar accuracy of 51% on the tinyimagenet-200 classification task. In Suppl. Fig.15,14,16,17 we present partition statistics at random training and test set samples, while varying architecture and hyperparameters.

Figure 9. Images from the decision boundary of a CNN trained to perform binary classification between Tabby Cat and Egyptian Cat classes of tinyimagenet. Larger sets of samples from the decision boundary along with partition visualizations are provided in the Suppl. Fig. 19. Samples from the decision boundary are necessarily, linear combinations of the anchor points, the weights of which are determined by the geometry of the decision boundary.

ples. For each sample, computation of partition statistics take $\sim$ 7mins for VGG11. We see that partition statistics vary significantly between DA and non-DA for VGG11 but not as significantly for VGG16. We provide partition visualizations for VGG11 in Suppl. Fig. 13. Region volumes can vary within a given 2D input domain as well as between input domain orientations. In Suppl. Sec. D.2 we present SplineCam partition statistics for random orientations at different training points and discuss the variability of statistics between orientations.

5. Conclusions

In this paper, we have developed SplineCam, the first provably exact method to compute the geometry of the input-space partition of a DN based on CPWL nonlinearities. We have demonstrated SplineCam's ability to visualize a large-scale DN's decision boundary, obtain samples that are provably on the boundary, and characterize DNs based on their partition statistics. SplineCam and its underlying theory open up several new avenues of exploration for practical neural networks, including quantifying the quality of initialization at training data points during transfer learning, improving INR initialization, and visualizing partition dynamics for different training strategies, to name a few.

Acknowledgements

Humayun and Baraniuk were supported by NSF grants CCF-1911094, IIS-1838177, and IIS-1730574; ONR grants N00014-18-12571, N00014-20-1-2534, and MURI N00014-20-1-2787; AFOSR grant FA9550-22-1-0060; and a Vannevar Bush Faculty Fellowship, ONR grant N00014-18-1-2047.

References

[1] Pankaj K Agarwal. Partitioning arrangements of lines II: Applications. *Disc. & Comp. Geom.*, 1990. 3, 4

[2] Randall Balestriero and Richard Baraniuk. A spline theory of deep networks. In *Proc. ICML*, pages 374–383, 2018. 3, 2

[3] Randall Balestriero and Richard G Baraniuk. Batch normalization explained. *arXiv preprint arXiv:2209.14778*, 2022. 2

[4] Randall Balestriero, Romain Cosentino, Behnaam Aazhang, and Richard Baraniuk. The geometry of deep networks: Power diagram subdivision. In *NeurIPS*, pages 15806–15815, 2019. 2

[5] Randall Balestriero, Ishan Misra, and Yann LeCun. A data-augmentation is worth a thousand samples: Analytical moments and sampling-free training. In *NeurIPS*, 2022. 7

[6] Elliott Ward Cheney and William Allan Light. *A Course In Approximation Theory*, volume 101. American Mathematical Soc., 2009. 3

[7] Magnus Egerstedt and Clyde Martin. *Control theoretic splines: optimal control, statistics, and path planning*. Princeton University Press, 2009. 3

[8] Cesare Fantuzzi, Silvio Simani, Sergio Beghelli, and Riccardo Rovatti. Identification of piecewise affine models in noisy environment. *International Journal of Control*, 75(18):1472–1485, 2002. 3

[9] Alhussein Fawzi, Horst Samulowitz, Deepak Turaga, and Pascal Frossard. Adaptive data augmentation for image classification. In *ICIP*, pages 3688–3692, 2016. 7

[10] Matteo Gamba, Adrian Chmielewski-Anders, Josephine Sullivan, Hossein Azizpour, and Marten Bjorkman. Are all linear regions created equal? In *AISTATS*, pages 6573–6590, 2022. 4

[11] Amirata Ghorbani, Abubakar Abid, and James Zou. Interpretation of neural networks is fragile. In *AAAI*, volume 33, pages 3681–3688, 2019. 2

[12] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *AISTATS*, pages 315–323, 2011. 1

[13] Boris Hanin and David Rolnick. Complexity of linear regions in deep networks. *arXiv preprint arXiv:1901.09021*, 2019. 2, 6

[14] L. A. Hannah and D. B. Dunson. Multivariate convex regression with adaptive partitioning. *J. Mach. Learn. Res.*, 14(1):3261–3294, 2013. 1

[15] Warren He, Bo Li, and Dawn Song. Decision boundary analysis of adversarial examples. In *ICLR*, 2018. 2

[16] Alex Hernández-García and Peter König. Data augmentation instead of explicit regularization. *arXiv preprint arXiv:1806.03852*, 2018. 7

[17] Ahmed Imtiaz Humayun, Randall Balestriero, and Richard Baraniuk. MaGNET: Uniform sampling from deep generative network manifolds without retraining. In *ICLR*, 2022. 2

[18] Ahmed Imtiaz Humayun, Randall Balestriero, and Richard Baraniuk. Polarity sampling: Quality and diversity control of pre-trained generative networks via singular values. In *CVPR*, pages 10641–10650, 2022. 2

[19] Mohammad AAK Jalwana, Naveed Akhtar, Mohammed Bennamoun, and Ajmal Mian. Cameras: Enhanced resolution and sanity preserving class activation mapping for image saliency. In *CVPR*, pages 16327–16336, 2021. 2

[20] Matt Jordan, Justin Lewis, and Alexandros G Dimakis. Provable certificates for adversarial examples: Fitting a ball in the union of polytopes. In *NeurIPS*, 2019. 2

[21] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proc. CVPR*, pages 8110–8119, 2020. 5

[22] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. 1

[23] Andrea Locatelli, Alexandra Carpentier, and Samory Kpotufe. An adaptive strategy for active learning with smooth decision boundary. In *Algorithmic Learning Theory*, pages 547–571. PMLR, 2018. 2

[24] A. Magnani and S. P. Boyd. Convex piecewise-linear fitting. *Optim. Eng.*, 10(1):1–17, 2009. 1

[25] Joe Mellor, Jack Turner, Amos Storkey, and Elliot J Crowley. Neural architecture search without training. In *ICML*, pages 7588–7598, 2021. 2

[26] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, pages 405–421. Springer, 2020. 5

[27] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021. 4

[28] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *NeurIPS*, pages 2924–2932, 2014. 7, 3

[29] Steven J Nowlan and Geoffrey E Hinton. Simplifying neural networks by soft weight-sharing. *Neural computation*, 4(4):473–493, 1992. 5, 6

[30] Adam Paszke et al. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, pages 8024–8035. 2019. 2, 10

[31] Tiago P. Peixoto. The graph-tool python library. *Figshare*, 2014. 2

[32] Henning Petzka, Martin Trimmel, and Cristian Sminchisescu. Notes on the symmetries of 2-layer relu-networks. In *Proceedings of the Northern Lights Deep Learning Workshop*, volume 1, pages 6–6, 2020. 2

[33] Mary Phuong and Christoph H. Lampert. Functional vs. parametric equivalence of ReLU networks. In *ICLR*, 2020. 2

[34] Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl Dickstein. On the expressive power of deep neural networks. In *ICML*, pages 2847–2854, 2017. 2

[35] Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. On the expressive power of deep neural networks. *arXiv preprint arXiv:1606.05336*, 2016. 4

[36] Gowthami Somepalli, Liam Fowl, Arpit Bansal, Ping Yeh-Chiang, Yehuda Dar, Richard Baraniuk, Micah Goldblum, and Tom Goldstein. Can neural nets learn the same model twice? Investigating reproducibility and double descent from the decision boundary perspective. In *CVPR*, pages 13699–13708, 2022. 2

[37] Yu Sun, Jiaming Liu, Mingyang Xie, Brendt Wohlberg, and Ulugbek S Kamilov. Coil: Coordinate-based internal learning for imaging inverse problems. *arXiv preprint arXiv:2102.05181*, 2021. 4

[38] Shuning Wang and Xusheng Sun. Generalization of hinging hyperplanes. *IEEE Transactions on Information Theory*, 51(12):4425–4431, 2005. 1

[39] Yuan Wang. Estimation and comparison of linear regions for relu networks. In *IJCAI*, 2022. 4

[40] Yulin Wang, Xuran Pan, Shiji Song, Hong Zhang, Gao Huang, and Cheng Wu. Implicit semantic data augmentation for deep networks. *NeurIPS*, 32, 2019. 7

[41] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *ICML Deep Learning Workshop*, page 12, 2015. 2

[42] Gizem Yüce, Guillermo Ortiz-Jiménez, Beril Besbinar, and Pascal Frossard. A structured dictionary perspective on implicit neural representations. In *CVPR*, pages 19228–19238, 2022. 5

[43] Xiao Zhang and Dongrui Wu. Empirical studies on the properties of linear regions in deep neural networks. *arXiv preprint arXiv:2001.01072*, 2020. 2