What does it mean to learn in deep networks? And, how does one detect adversarial attacks?

Ciprian A. Corneanu
Univ. Barcelona

Meysam Madadi
CVC, UAB

Sergio Escalera
Univ. Barcelona

Aleix M. Martinez
OSU

Abstract

The flexibility and high-accuracy of Deep Neural Networks (DNNs) has transformed computer vision. But, the fact that we do not know when a specific DNN will work and when it will fail has resulted in a lack of trust. A clear example is self-driving cars; people are uncomfortable sitting in a car driven by algorithms that may fail under some unknown, unpredictable conditions. Interpretability and explainability approaches attempt to address this by uncovering what a DNN models, *i.e*., what each node (cell) in the network represents and what images are most likely to activate it. This can be used to generate, for example, adversarial attacks. But these approaches do not generally allow us to determine where a DNN will succeed or fail and *why*. *i.e*., does this learned representation *generalize* to unseen samples? Here, we derive a novel approach to define what it means to learn in deep networks, and how to use this knowledge to detect adversarial attacks. We show how this defines the ability of a network to generalize to unseen testing samples and, most importantly, *why* this is the case.

1. Introduction

Deep Neural Networks (DNNs) have enough capacity to learn from very large datasets, without the need to define hand-crafted features, models or hypotheses for every problem. This has revolutionized computer vision and other areas of Artificial Intelligence (AI) [15].

Using high-capacity DNNs to learn from huge datasets, however, does not tell us how the network learns what it does and why. This so called "black-box" problem has resulted in a lack of trust [5, 35]. For example, why did a given DNN identified a specific deep representation as more appropriate than other possible representations?

Interpretability and explainability methods [34] allow us to see what a DNN has learned, but not *how* and *why* it learned it. For example, if the goal is to know what type of image activates a specific node (cell) of the network, one can optimize the DeepDream or other related criteria [1]. When our goal is to generate plausible class output images, optimizing softmax and related objectives may be performed [27]. But neither method tells us how and why these deep features (representations) were chosen by the network and *where* they may fail.

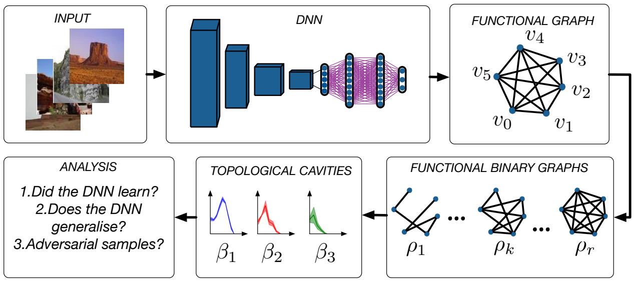


Figure 1. Given a DNN its functional graph is defined by the correlation of activation of distant nodes. This allows us to compute binary graphs defining local and global topological properties of the network (Algorithm 1). Global topological properties define generalization, while local properties identify overfitting (Algorithm 2). The same topological properties are used to detect adversarial attacks.

The main problems of lack of interpretability and explainability are: 1. If we do not know how DNNs learn, we can only improve them by trail-and-error. This is slow and does not solve the trustability problem. And, 2. If we do not know why DNNs learn a specific deep representation or model, we will not know where and why the network fails, *e.g*., detect an adversarial attack.

With the help of algebraic topology, we derive a set of algorithms that help us address the problems stated above. Simple local properties of the network (*e.g*., the degree of a node) as well as global properties (*e.g*., the path length between nodes) have been found to be insufficient to explain how and why DNNs learn [3].

This paper shows how topological properties (*e.g*., n-cliques and nD holes) of the functionality of the network describe how a DNN learns, Figure 1. We use these properties to demonstrate we can predict when a DNN successfully learns, and when and why it is likely to misclassify an unseen test sample (Algorithms 1 and 2). We also show how to use the derived approach to successfully detect adversarial attacks.

2. Related Work

In recent years, high capacity Deep Neural Networks (DNNs) have outperformed many other machine learning algorithms in a number of applications like object recognition [13, 32] and speech recognition [11, 22], especially when large labeled datasets [28, 17] and/or means of obtaining information from large datasets were available [8].

Why or how DNNs achieve this feat and where and why DNNs fail remains a mystery. This makes DNNs untrustworthy "black-box" models to many [35], with some researchers claiming DNNs are too unpredictable to make them a general solution to most AI problems [26, 6, 20].

To solve this problem, researchers are trying to understand what DNNs do. Two approaches are *interpretability* and *explainability*, with an emphasis on defining what the nodes (cells) of the network represent and how they respond to different inputs [12, 14, 34, 25, 38, 19, 23].

For example, DNNs used in object recognition generally learn to extract low-level, Gabor-like features in the first layers, combine these low-level features to represent parts of an object in mid-level layers, and finally combine those to generate highly complex representation of objects that are invariant to image changes, *e.g*., pose, illumination and affine transformations [30, 24, 39].

Another group of methods focuses on the analysis of these features. One way is by exploring the semantic meaning of filters [33] or computing feature distributions of different attributes [2]. Saliency maps, heatmaps, sensitivity maps, and attention maps display class relevant information that is used by the model to make its predictions [7, 18, 7, 40].

The goal of this paper is to go beyond these approaches of interpretability and explainability and ask, instead, what does it mean to learn in DNNs? Specifically, how does the graph defining the network evolve during the learning process? And, how can we use this knowledge to know where the network has successfully learned and where it will fail?

Making "black-box" models like DNNs interpretable is of great importance for several reasons, as for example to: $i)$ increase trust; $ii)$ facilitate transferability to other problems, e.g., use a network pre-trained on a specific problem in a different problem [37]; $iii)$ predict where a network is likely to work and where it will most probably fail; $iv)$ help researchers and practitioners design better networks, because we now know what works and what does not; $v)$ derive unsupervised learning algorithms [36, 39], because we will know what the network needs to do to generalize to unseen samples; and $vi)$ better prepare our algorithms for fair and ethical decision makings (i.e., unbiased performance) [4, 9].

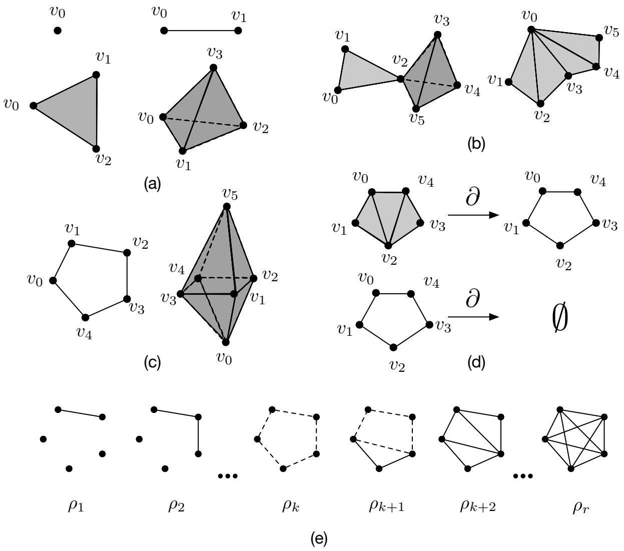

v_0
v_0 v_1
v_0 v_1 v_2
v_0 v_1 v_2 v_3
(a)
v_0 v_1 v_2 v_3 v_4 v_5
v_0 v_1 v_2 v_3 v_4 v_5
(b)
v_0 v_1 v_2 v_3 v_4
v_0 v_1 v_2 v_3 v_4 v_5
(c)
v_0 v_1 v_2 v_3 v_4
∂
v_0 v_1 v_2 v_3 v_4
v_0 v_1 v_2 v_3 v_4
∂
$\emptyset$
(d)
ρ_1
ρ_2
ρ_k
ρ_{k+1}
ρ_{k+2}
ρ_r
(e)

Figure 2. (a) Shown here are examples of a 0-clique (a point or node of a graph), 1-clique (two nodes connected by a 1D undirected edge), 2-clique (three nodes connected by the 1D edges and 2D faces filling in the space between the edges), and 3-clique (four nodes connected by their 1D edges, 2D faces, and a 3D solid filling in the space between the faces). These simplicies correspond to topological realization of a 0-clique, 1-clique, 2-clique, and 3-clique, respectively. Note that a n-clique defines a nD space. (b) Examples of clique complexes: cliques glue on common faces to form topological objects. (c) Example cavities in the topological space. (d) The boundary operator of a chain surrounding a clique complex (top) and a cavity (bottom). (e) Persistent homology detects birth and destruction of a cavity (shown with dashed line) during the mapping of a weighted graph onto binary graphs (Algorithm 1).

3. Approach

The structure of a DNN is generally defined by a weighted graph. Similarly, the "behavior" of a DNN (*i.e*., the activation of its nodes) are given by a functional graph, *i.e*., the correlations between the activation of each pair of nodes.

We study the topological changes of this functional graphs during training and testing. *We show that networks that generalize to unseen testing samples converge to a common topology, and that this topology is differential from that of networks that fail to generalize*. We then demonstrate how we can exploit this new knowledge to determine whether an unseen testing sample will be correctly analyzed by the network and when it will not. Specifically, we focus on adversarial attacks.

3.1. Preliminaries

Let $G = (V, E)$ be an undirected graph that consists of a pair of finite sets (V, E), where the elements v_i of V are called the vertices, and the elements e_{ij} of E are called the edges.

A *n-clique*, $\sigma \subset G$ is a subset of $n+1$ vertices of G that are connected to each other, and n is the degree of the clique. Any subgraph $\phi \subset \sigma$ of a n-clique is itself a clique of lower degree and is called a *face* of that clique. A clique that is not a face of a clique of larger degree, is called a *maximal clique*.

The *clique complex* of a graph G is the collection of all its cliques, $S(G) = \{S_0(G), S_1(G), \ldots S_n(G)\}$, where $S_k(G)$ is the set of all $(k+1)$-cliques in G [31]. A clique complex defines a topological space.

The *geometric realization* of this topological space is an object. A few examples of such objects are shown in Figure 2(a). As seen in this figure, a 0-clique is realized by a single node, a 1-clique by a line segment (*i.e.*, edge) connecting two nodes, a 2-clique is the filled triangle that connects three nodes (*i.e.*, the three edges of the three nodes define a filled triangle, that is, a 2D face), etc.

The geometric realizations of cliques in a clique complex intersect on common faces forming a well-defined object as shown in Fig. 2(b).

We define the correlation activation of nodes in a DNN as topological objects, as those illustrated in Fig. 2(a-c).

We study the topological properties of these objects. For this, we turn to several concepts necessary to compute homology, a method in algebraic topology capable of counting *cavities* in topological objects.

3.2. Homology and cavities

We define a *chain complex* $C(S)$ of a clique complex to be the sequence $\{C_n(S, \mathbb{F}_2)\}_{n \geq 0}$ (abbreviated C_n), with C_n the $\mathbb{F}_2$-vector space whose bases are the (n+1)-cliques $\sigma \in S_n$, $\forall n \geq 0$, and $\mathbb{F}_2 = \{0, 1\}$. In other words the elements of C_n are interconnections of (n+1)-cliques in S. For example, elements of C_1 are linear combinations of edges (2-cliques) and elements of C_2 are linear combinations of *filled* triangles (3-cliques, *i.e.*, 2D faces).

For each $n \geq 1$, there exist a linear transformation called a *boundary operator* that maps C_n onto a lower dimensional chain complex:

$$\partial_n : C_n \rightarrow C_{n-1}, \tag{1}$$

with

$$\partial_n(\sigma_{0,1,\ldots,n}) = \sum_{k=0}^{n} \sigma_{0,1,\ldots,k-1,k+1,\ldots,n}. \tag{2}$$

Geometrically, the boundary of a n-clique is the set of $(n-1)$-cliques bordering it. Fig. 2(d) shows an example. Hence, the boundary operator takes a collection of n-cliques and maps them to their boundary, *i.e.*, a collection of $(n-1)$-cliques.

Similarly, a n-cycle is a path in our graph that forms closed structures, *e.g.*, $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_1$. Since the boundary of any clique is a cycle, the boundary operator provides a mechanism to compute cycles. Formally, a n-cycle is given by any element $l \in C_n$ that satisfies $\partial_n(l) = 0$, Fig. 2(d).

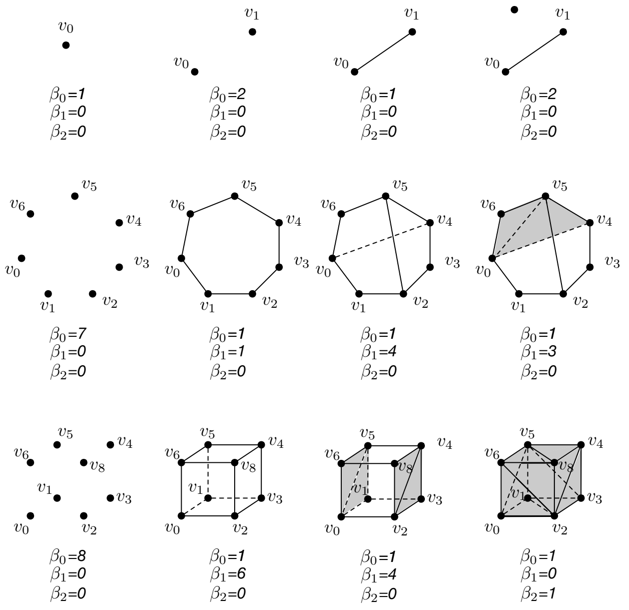


Figure 3. The Betti numbers are sequences of natural numbers counting the cavities of a topological object in a corresponding dimension. Low dimensional Betti numbers have intuitive interpretation: β_0 counts connected components, β_1 2D cavities, β_2 3D cavities. We show several examples of clique simplices and corresponding Betti numbers. By adding edges, lower dimensional cavities are destroyed while higher dimensional are formed. For example, note how a first cavity $l_0 = (v_0, v_1, ..., v_6)$ is formed (middle row, second column), then by adding two more edges (v_2, v_4) and (v_0, v_4) four new cavities appear $l_1 = (v_0, v_4, v_5, v_6)$, $l_2 = (v_0, v_1, v_2, v_3, v_4)$, $l_3 = (v_2, v_3, v_4, v_5)$ and $l_4 = (v_0, v_1, v_2, v_5, v_6)$ (middle row, third column). By definition, a geometrical realization of a clique always includes all the enclosed space between the nodes (simplices). Adding another edge (v_0, v_5) fills l_1, thus β_1 drops to 3. Some edges are dashed for facilitating visualization.

Let us call the subset of n-cycles, $k(\partial_n)$. And, let the n-cycle that defines the boundary of C_n be $b(\partial_n)$.

Note that two n-cycles, $l_1, l_2 \in C_n$, are equivalent if their sum in $\mathbb{F}_2$ defines the boundary of a path of $n+1$ vertices, *i.e.*, a $(n+1)$-chain.

Formally, we define this homology of dimension n as,

$$H_n(S, \mathbb{F}_2) = k(\partial_n)/b(\partial_n + 1), \tag{3}$$

for $n \geq 1$, and

$$H_0(S) = C_0/b(\partial_1). \tag{4}$$

Therefore, H_n defines the vector space spanned by the class of n-cycles. Its dimensionality is the number of nontrivial n-cycles; *i.e.*, the number of $(n+1)$-dimensional cavities in the objects (graphs) in Fig. 2(c). And a *cavity* is a non-filled face of dimensionality n, Fig 2(d).

Since cycles are chain complexes that map to $\emptyset$ by the boundary operator, *cavities* are the nullspace of this boundary operator, $null(\partial_n) = k(\partial_n) \subset C_n$.

3.3. Cliques in DNNs

The above is defined in binary graphs, *i.e.*, in $\mathbb{F}_2$. For weighted graphs we will compute all the possible binary graphs. Let us define an approach to achieve this.

Let $G = (V, E)$ be the graph defining the DNN we wish to study, and $X = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{m}$ the labeled training samples, with m the number of samples, $\mathbf{x}_i \in \mathbb{R}^p$, and $\mathbf{y}_i \in \mathbb{Z}$ in classification and $\mathbf{y}_i \in \mathbb{R}^q$ in regression ($q \geq 1$).

Passing the sample vectors $\mathbf{x}_i$ through the network ($i = 1, \dots, m$), allows us to compute the correlation c_{ij} between the activation (a_i, a_j) of each pair of nodes (v_i, v_j) of G. That is given by,

$$c_{ij} = \frac{m(m-1)}{2} \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \frac{(a_i - \mu_{a_i})(a_j - \mu_{a_j})}{\varsigma_{a_i} \varsigma_{a_j}}, \quad (5)$$

where μ and ς indicate the mean and standard deviation over X.

Let us now order the absolute values of these correlations from largest to smallest: $|c_{i_1 j_1}| \geq |c_{i_2 j_2}| \geq \cdots \geq |c_{i_r j_r}| > T$, r the number of correlations larger than T in G given X, written $corr(G|X, T)$. In this notation, the subscripts of each correlation indicate the node pairs in that correlation: $(v_{i_1}, v_{j_1}), \dots, (v_{i_r}, v_{j_r})$.

Binary graphs are obtained by iteratively adding a node pair at a time. We start by adding the nodes with largest correlation, (v_{i_1}, v_{j_1}), and continue adding node pairs up to the last one, (v_{i_r}, v_{j_r}).

We will refer to the number of edges included in a binary graph as its density, $\rho_k = k/$number of non-zero correlations; the number of non-zero correlations is computed as the number of $|c_{ij}| > 0$, $\forall i$ and $j > i$. Thus, the density of our binary graph will increase at each iteration (as we add more and more nodes).

This process allows us to investigate how the homology of the graph defining the DNN of interest evolves as a function of the density, Fig. 2(e).

This approach is summarized in Algorithm 1. This algorithm maps a DNN defined by a weighted graph onto the set of binary graphs $\{G_1, \dots, G_r\}$. And, these binary graphs are topological objects as shown in Fig. 2.

Algorithm 1 allows us to compute any topological property of a DNN given X as a function of learning.

Algorithm 1 Weighted to binary graph mapping.

```
Let $G = (V, E)$, $X = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{m}$, and define $T > 0$.
Let $corr(G|X, T) = \{|c_{i_1 j_1}|, |c_{i_2 j_2}|, \dots, |c_{i_r j_r}|\}$, s.t. $|c_{i_1 j_1}| \geq |c_{i_2 j_2}| \geq \cdots \geq |c_{i_r j_r}| > T$.
Let $G_0 = \emptyset$ and k=0.
repeat
    $k = k + 1$.
    $G_k = G_{k-1} \bigcup \{v_{i_k}, \hat{e}_{ij} v_{j_k}\}$, with $\hat{e}_{ij}$ the undirected edge defining $v_{i_k}$— $v_{j_k}$.
    $\rho_k = k/r$.
until $k = r$.
```

We start with the analysis of the number of n-cliques. To illustrate this, we will use the LeNet derived in [16]. LeNet is a historical network and will be used here as a proof-of-concept. This network was originally derived to recognize written digits, *i.e.*, 0-10. We use the MNIST dataset, consisting of $60,000$ training images and $10,000$ test images, as X [16].

The cross-validation (CV) classification accuracy of this network is shown in Fig. 4(a), where the x-axis defines the epoch and the y-axis the 5-fold CV accuracy, and the thickens of the blue curve defines the variance over this CV procedure.

Given the graph of LeNet G_{LeNet} and the training set X, we use Algorithm 1 to get G_k s.t. $\rho_k = .25$. The number of n-cliques ($n \in [0, 20]$) in G_k are shown in Fig. 4(b), where each plot represents the results at the indicated epoch, the x-axis specifies the value of n, and the y-axis the number of n-cliques.

Note that the number of n-cliques in an untrained DNN must be zero or very close to zero. That is because the number of nodes working together (*i.e.*, correlated, as given by c_{ij}) before any training is performed must be zero or tiny. This is shown in the first plot of Fig. 4(b). But as the network *learns*, the nodes in the graph start to work together to solve the problem of mapping $\mathbf{x}_i$ onto $\mathbf{y}$.

Note that by the time the learning process has made its major gains (by about epoch 10), the number of n-cliques is maximum. As the learning process continuous tightening the knobs of the network (*i.e.*, adjusting its parameters), the number of cliques starts to decrease. This is the result of overfitting to X.

Are cavities an even better indicator of how well the network learns a dataset X? We explore this next.

3.4. Cavities in DNNs

The rank of the n-homology group H_n in a topological space is called the n^{th} Betti number. In other words, Betti numbers compute the maximum amount of cuts that must be made to separate a surface into two k-cycles, $k = 1, 2, \dots$.

Hence, the first Betti number, β_0, computes the number of connected elements in $S(G)$; the second Betti number, β_1, calculates 1D cavities (or holes); β_2 gives the number of

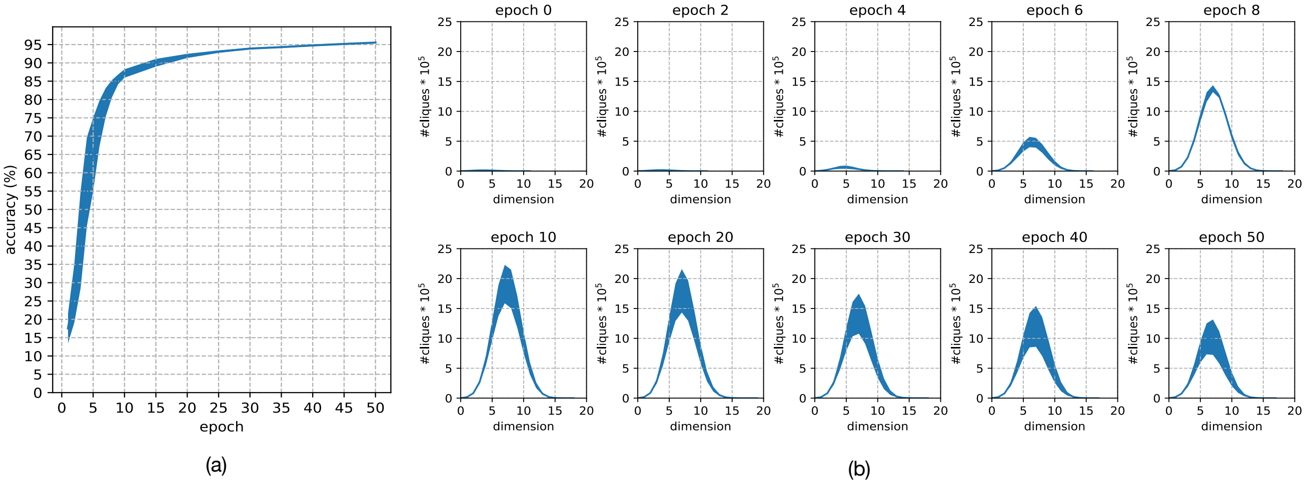


Figure 4. Testing accuracy for LeNet training on MNIST (a) vs the number of cliques (b). Mean and standard deviation given by 5-fold cross-validation.

2D cavities; and, more generally, β_n computes the number of nD cavities, Figure 3.

Key to understanding Figure 3 is to note that all kD faces of the simplicies of a n-clique are filled, *e.g.*, see the last object in the second row and the last two in the last row in Figure 3; note how these kD faces eliminate cavities of lower dimensionality and add cavities of higher dimensionality.

To further clarify this important point, consider the second example in the second row in Figure 3. The functional representation of this DNN indicates that node v_0 and v_1 work together to solve the classification or regression problem the network is tasked to address, *i.e.*, their activation patterns are highly correlated (Algorithm 1). Similarly, v_i and v_{i+1} ($i = 1, \dots, 6$) and v_6 and v_0 also work together to solve the problem the network is faced with. This creates a 1D cavity, $\beta_1 = 1$, because ∂ maps this chain to $\emptyset$. But if v_0 and v_4 and v_0 and v_5 are also highly correlated, this forms cliques with filled simplicies, yielding two additional 1D cavities, $\beta_1 = 3$; as shown in the last objects in the second row in Fig. 3.

The Betti numbers of a clique complex S of a graph G are formally given by,

$$\beta_n(S) = null(\partial_n) - rank(\partial_{n+1}). \tag{6}$$

Note that in order to compute β_n, we need the null space and the rank of the matrix formed by cliques of dimension n and dimension $n-1$. This means that we only need to compute the cliques in the first n dimensions to calculate $\beta_1, \dots, \beta_n$. Since, in most DNNs, $\beta_n = 0$ for n larger than 4 or 5, this means that the computations can be performed efficiently.

3.5. Learning and generalization

The evolution of the Betti numbers of the LeNet network as a function of T and epochs is shown in Fig. 5. Recall, T defines the density our approach explores, *i.e.*, T is inversely proportional to ρ (see Algorithm 1). The y-axis in the plots indicates the number of cavities properly normalized by the number of nodes (*i.e.*, number of cavities/node).

As we can see in this *key* figure, the number of 1D cavities moves from higher to lower densities as the DNN learns. That is, as the DNN learns, the correlation pattern of activation of the nodes in the network that are further apart as well as those that generate larger simplicial complexes start to appear.[1] This demonstrates that the network is constructing *global* structures to learn the function that maps the input samples to their corresponding outputs, $\mathbf{y}_i = g(\mathbf{x}_i)$, *i.e.*, the network is learning to generalize by using large portions of the graph.

But when the network can no longer generalize, it starts to memorize the samples that do not abide by the learned functional mapping $g(\cdot)$. This results in a decrease of global structure and an increase in *local* chain complexes. This is clearly visible in Figure 5: note the maximum number of cavities starts to move toward higher densities. As seen in Figure 5, this happens after about epoch $= 40$, when the network has done all it can to learn to generalize, Figure 4(a). An example of this topological effect was illustrated

[1] This effect is due to the fact that nodes that work together start to form cliques whose simplices fill in the holes of the functional graph. That is, rather than adding additional nodes when increasing the density from ρ_k to ρ_{k+1}, we add edges between the nodes already available at density ρ_k. We refer to this as a *global* property, because rather than having a chain of semi-related nodes (*e.g.*, $v_1 \to v_2 \to \cdots \to v_r$), we have n-cliques (*i.e.*, all nodes connected to all other nodes). Note that the cliques appearing at higher densities delete the cavities we observe a lower densities, Figure 3.

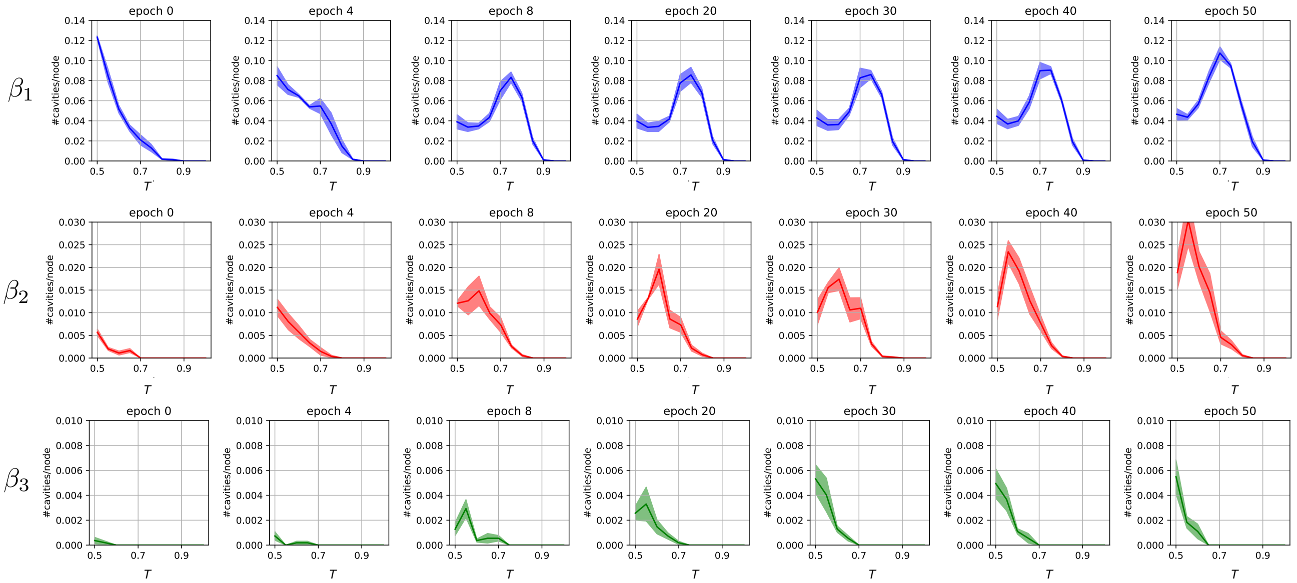


Figure 5. Betti number dynamics during LeNet [16] training on MNIST. Mean and standard deviation given by 5-fold cross-validation. Refer to Fig. 4(a) for testing accuracy.

Algorithm 2 Generalization.

```
Let $G = (V, E)$ define a DNN with $\theta$ its parameters.
Let $X$ be the training set, and set $T > 0$.
Set t=0, and $n$ to either 1, 2, or 3.
repeat
    Use $X$ and the selected loss to optimize $\theta$.
    t=t+1.
    Use Algorithm 1 to obtain $G_k$, $k = 1, \ldots, r$.
    Compute the clique complexes $S_k$ of $G_k$.
    $\widehat{k}_t = \arg\max_k \beta_n(S_k)$.
until $\widehat{k}_t > \widehat{k}_{t-1}$.
```

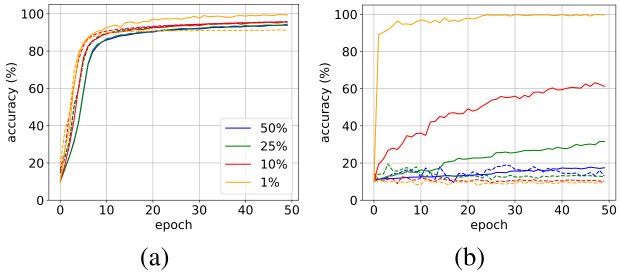
in the second and third rows of Fig. 3.

This means *learning in DNNs is equivalent to finding the smallest density $\rho_{\widehat{k}}$ of nD cavities in the functional binary graphs that define the network.* This peak density allows us to identify when a network has reach its limits of global learnability and generalization to unseen samples.

This approach is summarized in Algorithm 2.

If we wish to find global properties with Algorithm 2, we set $n = 1$ to detect 1D cavities. But if we wish to identify even larger topological connections, we set $n = 2$ or 3. Hence, larger n values mean we are interested in increasingly general properties of the underlying, unknown function $g(\cdot)$ our DNN needs to learn.

The smaller n is, the more we allow the DNN to adapt to our specific dataset. Thus, a smaller n will yield more accurate results on a testing set that is representative of the training, but less accurate results on test sets that diverge from the training set.



(a) (b)

Figure 6. Training/testing accuracy with decreasing proportion of training data and permuted/non-permuted labels.

Additional results are in the supplementary file.

3.6. Learning vs. failing to learn

We can also use the above defined topological properties of our DNN to determine where the networks fails to learn to generalize to unseen samples.

Let us illustrate this in an example using LeNet. In this example, we trained the network with either 50%, 25%, 10% or 1% random selection of the training samples (plus the data augmentation typically used in LeNet). The training and testing accuracies are shown in Figure 6(a). Solid lines indicate training accuracy; dashed lines testing accuracy. As can be appreciated in the figure, for LeNet the testing accuracy follows the same pattern as the training, regardless of the percentage of training data used.

But, when we redo the above experiment with permuted labels,[2] the results are very different, Figure 6(b).

[2] This means that the labels $\mathbf{y}_i$ have been permuted by multiplying the

Note how the network is able to learn from small nonsensical (permuted) datasets during training. But the testing accuracy curves shows this is just an illusion, *i.e.*, the network is able to memorize these nonsensical sample pairs $\{\mathbf{x}_i, \tilde{\mathbf{y}}_i\}$, where $\tilde{\mathbf{y}}_i$ are the permuted labels, but this serves no purpose when it comes to testing the performance of the system on independent samples. This means we cannot relay on the training accuracy as a measure of *generalization*; a well-known fact.

Fortunately, from our previous section, we know that nD cavities move toward lower densities of our functional binary graphs when the network learns to generalize. We will now use this insight to derive a simple algorithm that knows whether the network is learning or not.

Figure 7 shows the Bettis at different densities, percentages of training data (50 to 1%), and epochs (epoch $= 2, 10$ and 50). The blue curves correspond to the Bettis computed with Algorithm 2 and non-permuted data. Orange curves show the Bettis obtained when using the permuted labels $\tilde{\mathbf{y}}_i$. Figure 7(a) are the results at epoch $= 2$, Figure 7(b) at epoch $= 4$, Figure 7(c) at epoch $= 10$ and Figure 7(d) at epoch $= 50$.

As expected, the maximum number in β_1 moves to lower densities when the labels are *non-permuted*, but does *not* when the labels are *permuted*. Even more telling is the lack of 2D and 3D cavities when using *permuted* labels.

Our algorithm to detect lack of training is thus simple: *a.* Lack of 2D and 3D cavities, and *b.* $1D$ cavities move toward lower densities.

What does it mean to learn in DNN? Learning to generalize in DNN is defined by the creation of 2 and 3D cavities in the functional binary graphs representing the correlations of activation of distant nodes of the DNN, and the movement of 1D cavities from higher to lower graph density ρ. Overfitting is indicated by a regression of these cavities toward higher densities in these functional binary graphs.

3.7. Detecting adversarial attacks

We can also use the approach derived in the previous sections to know where our network is likely to fail during testing. We illustrate this by demonstrating how to detect an adversarial attack, which is a guaranteed, easy way to make a DNN fail [33].

To do this, we use the algorithm of [21] to generate images for an adversarial attack on a trained LeNet. As above, LeNet was trained on MNIST.

For testing, we used the the independent MNIST testing set and the set of images prepared for the adversarial attack.

As above, we expect 1D cavities to increase and move to lower densities as well as an increase in the number of 2 and 3D cavities at lower densities on unaltered testing data. But for the data in the adversarial set, we expect only 1, 2 and 3D cavities at the highest densities, indicating local processing but a lack of global engagement of the network.

The results are in Fig. 8, with the blue curves indicating the cavities on the functional binary graphs G_k when using the unaltered testing sample, and the orange curve those observed when processing the samples in the adversarial attack set. Thus, our predictions are confirmed, and we know that the images in the adversarial attack set cannot be correctly classified by LeNet.

This approach, for the first time allows us to identify *test* images that the network is bound to misclassify.

See additional results in the supplementary file.

3.8. Experimental details

The experiments reported above with LeNet were trained using stochastic gradient descent with momentum of .9 and weight decay 5×10^{-4}. Learning rate was initialized at 10^{-4} and reduced by half when accuracy stagnated. For training, data was augmented using random horizontal flips, random rotations of ± 5 and random crops of equal image size with padding $= 4$. The images of the adversarial attack were generated using the approach of [21]. This approach computes minimal perturbations, that are mostly indistinguishable to a human observer but have devastating effects on DNNs. In our case, the initial testing accuracy of above 90% (Fig. 4(a)) dropped to random decision (10%) after the adversarial perturbations were used.

4. Discussion and conclusions

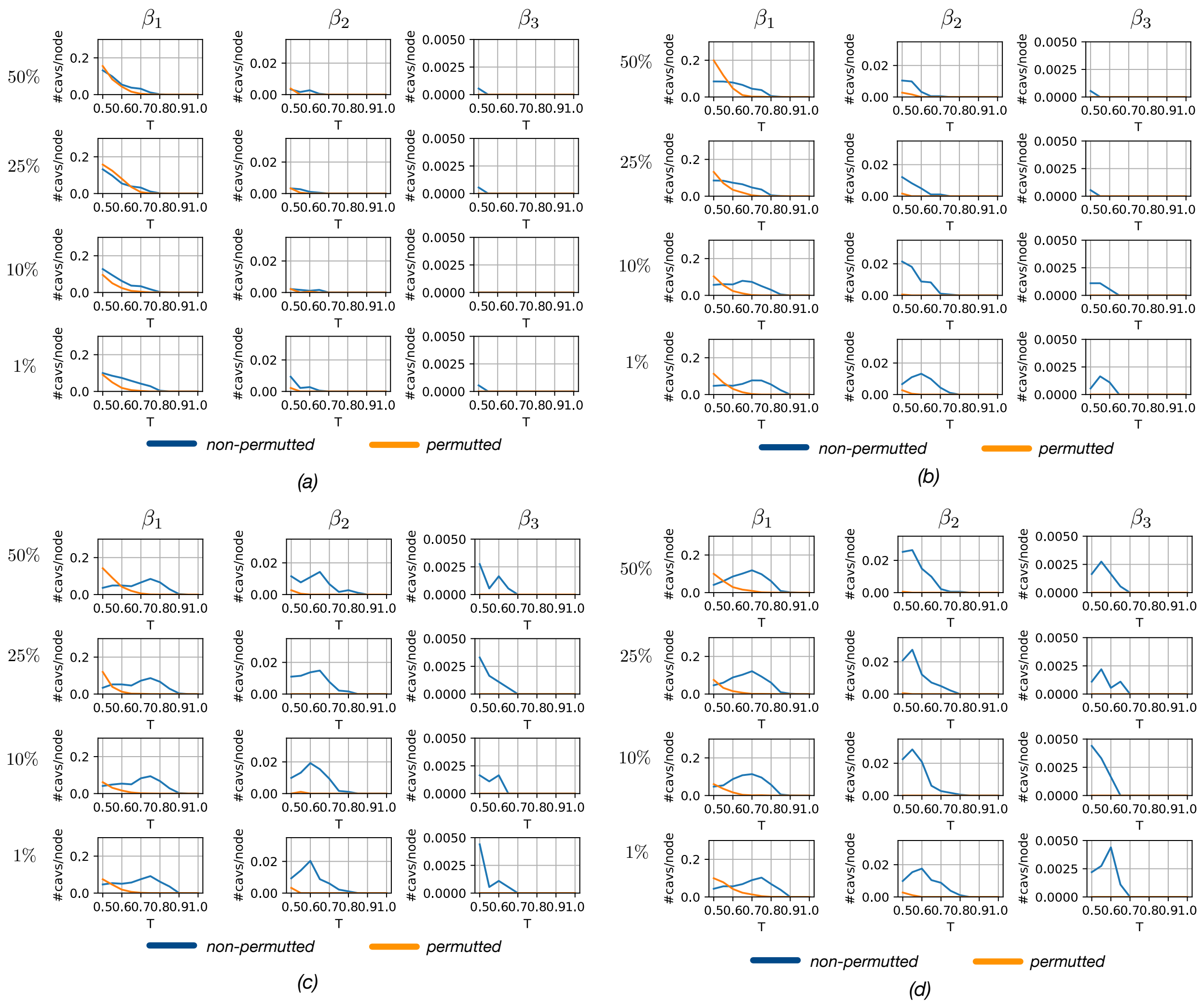
DNNs are used in a large number of real life products, including face recognition, facial expression analysis, object recognition, speech recognition, and language translation, to name but a few. Many companies depend on DNNs to design their products.

The main problem with DNNs is that they behave in a "black-box" manner, *i.e.*, we do not know how they learn, what they learn, or where these learned deep representations will fail. This has led to a loss of trust by many [5, 35, 26, 6, 20]. Clear examples of these are self-driving cars and medical diagnoses, with variants of the following argument: If we do not know what the network learned and why and where it will fail, how can I trust it to drive my car or give me a medical diagnosis.

In this paper, we have introduced a set of tools and algorithms to address these problems.

We accomplished this by defining what learning in deep networks means. Specifically, we demonstrated that learning to generalize to unseen (test) samples is equivalent to creating cliques among large number of nodes, whereas memorizing specific training samples (*i.e.*, a type of learning that does not translate to good generalizations) is equiv-

vector $(\mathbf{y}_1, \ldots, \mathbf{y}_m)^T$ with a randomly generated $m \times m$ permutation matrix $\mathbf{P}$. A permutation matrix is obtained by permuting the rows of an identity matrix.



(a)

(b)

(c)

(d)

Figure 7. Betti numbers obtained when using 50%, 25%, 10% or 1% of the training data (top to bottom row, respectively). Blue curves indicate non-permuted labels; orange curves indicate permuted labels. 1D cavities (β_1) are shown in the first column; 2D cavities (β_2) in the second column; and 3D cavities (β_3) in the third column. Results plotted at the following epochs (a) 2, (b) 4, (c) 10 and (d) 50.

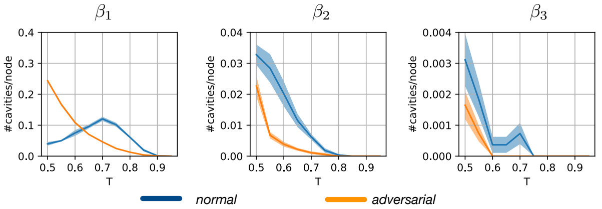


Figure 8. Betti numbers obtained when using unaltered and adversarial testing samples. LeNet trained on MNIST.

alent to topological changes between low-correlated sets of nodes (*i.e.*, nodes that are not highly cooperative among themselves). We then showed how we can use these same principles to determine when the network works correctly during testing. Specifically, we showed we can reliably identify adversarial attacks.

Beyond what we have demonstrated in this paper, it is worth mentioning that our approach is general and by no means limited to feedforward neural networks or the use of backpropagation. Our approach works equally well on networks that have all types of directed and undirected edges, *e.g.*, [10], alternative to backpropagation [29], or different types of activation functions.

Acknowledgements. NIH grants R01-DC-014498 and R01-EY-020834, Human Frontier Science Program RGP0036/2016, TIN2016-74946-P (MINECO/FEDER, UE), CERCA (Generalitat de Catalunya) and ICREA (ICREA Academia).

Contributions. AMM defined main ideas, AMM and CC designed research and experiments, CC, with MM and SE, ran experiments, CC and AMM wrote the paper.

References

[1] https://deepdreamgenerator.com. Accessed: 2018-11-16. 1

[2] M. Aubry and B. C. Russell. Understanding deep features with computer-generated imagery. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2875–2883. IEEE, 2015. 2

[3] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017. 1

[4] J. Buolamwini and T. Gebru. Gender shades: Intersectional accuracy disparities in commercial gender classification. In *Conference on Fairness, Accountability and Transparency*, pages 77–91, 2018. 2

[5] D. Castelvecchi. Can we open the black box of ai? *Nature News*, 538(7623):20, 2016. 1, 7

[6] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song. Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1625–1634, 2018. 2, 7

[7] R. C. Fong and A. Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. *arXiv preprint arXiv:1704.03296*, 2017. 2

[8] S. Ganju, O. Russakovsky, and A. Gupta. Whats in a question: Using visual questions as a form of supervision. *arXiv preprint arXiv:1704.03895*, 2017. 2

[9] N. Garg, L. Schiebinger, D. Jurafsky, and J. Zou. Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences*, 115(16):E3635–E3644, 2018. 2

[10] D. George, W. Lehrach, K. Kansky, M. Lázaro-Gredilla, C. Laan, B. Marthi, X. Lou, Z. Meng, Y. Liu, H. Wang, et al. A generative vision model that trains with high data efficiency and breaks text-based captchas. *Science*, 358(6368):eaag2612, 2017. 8

[11] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012. 2

[12] U. Johansson, C. Sönströd, U. Norinder, and H. Boström. Trade-off between accuracy and interpretability for predictive in silico modeling. *Future medicinal chemistry*, 3(6):647–663, 2011. 2

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 2

[14] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015. 2

[15] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015. 1

[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 4, 6

[17] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 2

[18] Z. C. Lipton. The mythos of model interpretability. *arXiv preprint arXiv:1606.03490*, 2016. 2

[19] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5188–5196, 2015. 2

[20] M. Mitchell. Artificial intelligence hits the barrier of meaning. *New York Times, Op-Ed*, 2018. 2, 7

[21] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2574–2582, 2016. 7

[22] T. N. Sainath, A.-r. Mohamed, B. Kingsbury, and B. Ramabhadran. Deep convolutional neural networks for lvcsr. *Acoustics, Speech, and Signal Processing, 1988. ICASSP-88., 1988 International Conference on*, pages 8614–8618, 05 2013. 2

[23] C. Olah, A. Mordvintsev, and L. Schubert. Feature visualization. *Distill*, 2(11):e7, 2017. 2

[24] A. J. OToole, C. D. Castillo, C. J. Parde, M. Q. Hill, and R. Chellappa. Face space representations in deep convolutional neural networks. *Trends in cognitive sciences*, 2018. 2

[25] M. T. Ribeiro, S. Singh, and C. Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144. ACM, 2016. 2

[26] A. Rosenfeld, R. Zemel, and J. K. Tsotsos. The elephant in the room. *arXiv preprint arXiv:1808.03305*, 2018. 2, 7

[27] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986. 1

[28] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 2

[29] J. Sacramento, R. P. Costa, Y. Bengio, and W. Senn. Dendritic cortical microcircuits approximate the backpropagation algorithm. *NIPS*, 2018. 8

[30] R. Salakhutdinov, J. B. Tenenbaum, and A. Torralba. Learning with hierarchical-deep models. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1958–1971, 2013. 2

[31] E. H. Spanier. *Algebraic topology*. Springer, 1994. 3

[32] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 2

[33] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013. 2, 7

[34] R. Turner. A model explanation system. In *Machine Learning for Signal Processing (MLSP), 2016 IEEE 26th International Workshop on*, pages 1–6. IEEE, 2016. 1, 2

[35] P. Voosen. How ai detectives are cracking open the black box of deep learning. *Science, July*, 2017. 1, 2, 7

[36] X. Wang and A. Gupta. Unsupervised learning of visual representations using videos. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2794–2802, 2015. 2

[37] A. R. Zamir, A. Sax, W. Shen, L. Guibas, J. Malik, and S. Savarese. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3712–3722, 2018. 2

[38] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014. 2

[39] B. Zhou, D. Bau, A. Oliva, and A. Torralba. Interpreting deep visual representations via network dissection. *IEEE transactions on pattern analysis and machine intelligence*, 2018. 2

[40] L. M. Zintgraf, T. S. Cohen, T. Adel, and M. Welling. Visualizing deep neural network decisions: Prediction difference analysis. *arXiv preprint arXiv:1702.04595*, 2017. 2