SMINT: Toward Interpretable and Robust Model Sharing for Deep Neural Networks

HUIJUN WU, National University of Defense Technology, China and UNSW, Australia
CHEN WANG and RICHARD NOCK, Data61, CSIRO, Australia
WEI WANG, University of New South Wales, Australia
JIE YIN, University of Sydney, Australia
KAI LU, National University of Defense Technology, China
LIMING ZHU, Data61, CSIRO and University of New South Wales, Australia

Sharing a pre-trained machine learning model, particularly a deep neural network via prediction APIs, is becoming a common practice on machine learning as a service (MLaaS) platforms nowadays. Although deep neural networks (DNN) have shown remarkable successes in many tasks, they are also criticized for the lack of interpretability and transparency. Interpreting a shared DNN model faces two additional challenges compared with interpreting a general model. (1) Limited training data can be disclosed to users. (2) The internal structure of the models may not be available. These two challenges impede the application of most existing interpretability approaches, such as saliency maps or influence functions, for DNN models. Case-based reasoning methods have been used for interpreting decisions; however, how to select and organize the data points under the constraints of shared DNN models is not discussed. Moreover, simply providing cases as explanations may not be sufficient for supporting instance level interpretability. Meanwhile, existing interpretation methods for DNN models generally lack the means to evaluate the reliability of the interpretation. In this article, we propose a framework named Shared Model INTerpreter (SMINT)[1] to address the above limitations. We propose a new data structure called a boundary graph to organize training points to mimic the predictions of DNN models. We integrate local features, such as saliency maps and interpretable input masks, into the data structure to help users to infer the model decision boundaries. We show that the boundary graph is able to address the reliability issues in many local interpretation methods. We further design an algorithm named hidden-layer aware p-test to measure the reliability of the interpretations. Our experiments show that SMINT is able to achieve above 99% fidelity to corresponding DNN models on both MNIST and ImageNet by sharing only a tiny fraction of training data to make these models interpretable. The human pilot

[1]A preliminary version of this article was published as a research track paper in the 27th World Wide Web Conference (WWW'18).

This work is supported by National High-level Personnel for Defense Technology Program (2017-JCJQ-ZQ-013), NSF 61902405, the HUNAN Province Science Foundation 2017RS3045. Wei Wang is supported by ARC DPs 170103710 and 180103411, and D2DCRC DC25002 and DC25003.

Authors' addresses: H. Wu, National University of Defense Technology, 109 Deya Road., Changsha, China, UNSW, Sydney, Australia; email: huijunw@cse.unsw.edu.au; C. Wang and R. Nock, Data61, CSIRO, 13 Garden Street, Eveleigh, NSW 2015, Sydney, Australia; emails: {chen.wang, richard.nock}@data61.csiro.au; W. Wang, University of New SouthWales, UNSW, Kensington, NSW 2052, Sydney, Australia; email: weiw@cse.unsw.edu.au; J. Yin, University of Sydney, The University of Sydney, NSW 2006, Sydney, Australia; email: jie.yin@sydney.edu.au; K. Lu, National University of Defense Technology, 109 Deya Road, Kaifu, 410072, Changsha, China; email: kailu@nudt.edu.cn; L. Zhu, Data61, CSIRO and University of New South Wales, 13 Garden Street, Eveleigh, NSW 2015, Sydney, Australia; email: liming.zhu@data61.csiro.au.

1559-1131/2020/05-ART11 $15.00
https://doi.org/10.1145/3381833

study demonstrates that SMINT provides better interpretability compared with existing methods. Moreover, we demonstrate that SMINT is able to assist model tuning for better performance on different user data.

CCS Concepts: • **Information systems** → **Web applications**; *Web services*; • **Computing methodologies** → **Neural networks**; • **Software and its engineering** → *Collaboration in software development*;

Additional Key Words and Phrases: Deep neural networks, model sharing, interpretability, decision boundary

ACM Reference format:
Huijun Wu, Chen Wang, Richard Nock, Wei Wang, Jie Yin, Kai Lu, and Liming Zhu. 2020. SMINT: Toward Interpretable and Robust Model Sharing for Deep Neural Networks. *ACM Trans. Web* 14, 3, Article 11 (May 2020), 28 pages.
https://doi.org/10.1145/3381833

1 INTRODUCTION

Complicated machine learning models, particularly deep neural networks (DNN), have achieved great success in image classification [23, 34], speech recognition [22, 24], and classic games [56] in recent years. The success inspires the use of DNN in a rapidly increasing number of applications. Training a DNN model, however, often requires large amounts of labeled data and non-trivial tuning efforts. Thus, a cost-efficient practice is to share a pre-trained model built for common applications. As shown in Figure 1, machine learning models can be hosted in the cloud and run as a service in Pay-As-You-Go mode. The model developers train models using the data they collect. They may use machine learning as service platforms (MLaaS) such as Google CloudML [50], AmazonML [4], or Microsoft AzureML [40] to develop models and manage training data in the cloud. The application developers then integrate these models into their applications through prediction APIs. However, a complicated machine learning model, particularly a DNN model, is often seen as a black-box in decision making. The model quality is difficult to assess. The MLaaS platforms do not have effective means to address this problem. There are potential risks when integrating a shared model in user applications without a meaningful service level objective (SLO) for quality assurance.

MLaaS normally does not share the whole training data due to privacy concerns or business interests. The internal architecture of a shared model is also protected to prevent parameter leak. Recent study [63] shows that it is much easier for one to steal the parameters of a shared model if the training data are all available. Stealing the parameters allows malicious users to build their own proxy of the model without paying model providers. Although recent work proposes different ways to visualize or interpret a DNN model [18, 26, 28, 31, 52, 55, 57, 59, 67–69], they often assume the whole training data and model architecture are accessible. In model sharing, users have limited knowledge about how the model is trained and what training data have been used. Providing interpretability is challenging but highly necessary.

Case-based reasoning is one approach to interpreting the DNN models [27, 37, 47, 64]. Similar ideas have also been explored to build case-based reasoning systems for a long time [14, 51]. It is a promising direction for interpreting shared models as only a subset of training data (i.e., prototypes or examples) needs to be provided to explain predictions. Meanwhile, most of these methods do not require access to the internal structure of DNN models. Nevertheless, limitations exist as described below.

First, most of the existing work simply present prototypes to users and lacks useful data structures to organize them, e.g., Reference [37] proposes a network architecture that learns prototypes by an auto-encoder. It is not sufficient as prototypes alone do not embody the relationships among training data and, therefore, a lack of information about how a model generalizes features from

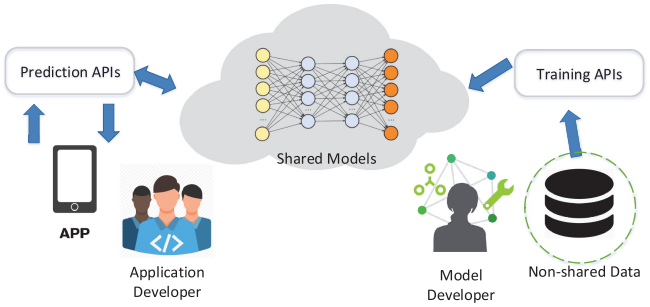


Fig. 1. The model sharing scenario: A model user calls prediction APIs to get predictions of her input data. The prediction is based on deep neural network models developed by the third-party model developers on the data invisible to the model user.

a set of data. For a shared model, what prototypes should be presented and how they should be presented is important. Humans often learns from multiple cases rather than a single case [32], after all.

Second, the existing case-based DNN interpretation work generally does not highlight features in prototypes that highly influence the decision, which may leave a wide space for arbitrary interpretation based on a prototype.

Third, case-based methods and most existing interpretation methods suffer from the reliability problem [2, 3, 20, 30]. A recent study [3] showed that both perturbation-based interpretations [52] and saliency maps [55, 57, 59] are not sensitive to the random perturbations of the input data or model parameters. Similarly, the influential training data selected by the influence function method [31] for a given test point may change due to small random perturbations of the test point. Adversarial rather than random perturbations can easily generate inconsistent interpretations [20]. Unreliable interpretations confuse users and make it difficult to debug a model, particularly when test cases are generated during the automatic model testing [49]. One key reason for the unreliability is that the test points may be drawn from data with different distribution with the training data, which results in unexpected interpretations. Existing interpretation methods lack of an effective means to detect out-of-distribution test data.

In this article, we propose a framework named SMINT to enhance the interpretability for shared models. The article makes the following contributions:

- SMINT uses a data structure named boundary graph to organize a subset of training data points. The boundary graph aims to characterize the decision boundaries of a DNN model with a minimal subset of training data. The boundaries are computed at the latent space, particularly the *softmax* layer outputs of a DNN model. Boundary characterizing is motivated by cognitive research [33, 41] claiming that humans learn from not only examples but also counterexamples. For a DNN classifier, neighboring training data points with different labels are likely to demonstrate key feature differences that lead to different predictions. A decision boundary exists between these neighboring data points. To characterize these boundaries, one example for each class may not be enough. This is mainly because the softmax vectors are dense so that multiple feature differences are difficult to distinguish in a particular region. To address this, we propose to use *supernodes* as the interpretation units. A supernode is a set of training examples that are close in the latent space. Supernodes have stronger modeling power to characterize decision boundaries.

- To provide intuitive hints to users, we associate the local features, such as saliency maps and feature masks [52], with the training points in the boundary graph. We further show that this integration has unique advantages compared with using these methods directly on testing points.
- Interpretations can be unreliable particularly when test data are out-of-distribution. We propose an algorithm called *hidden-layer-aware p-test* to measure whether the neighboring relationships of a test point in the boundary graph are statistically supported by the training data. This way, unreliable predictions, and interpretations can be detected.

Our experimental results show that SMINT achieves 99% fidelity to the DNN model on ImageNet by disclosing a small fraction of training data to users. SMINT also has satisfactory interpretation performance in a human pilot study and effective out-of-distribution detection capability.

The article is organized as follows: Section 2 introduces the preliminaries and related work; Section 3 gives an overview of the architecture; Section 4 introduces algorithms for boundary graph construction; Section 5 presents the local feature integration; Section 6 is the measurement of the reliability of interpretations. Section 7 gives evaluation results and Section 8 concludes the article.

2 PRELIMINARIES AND RELATED WORK

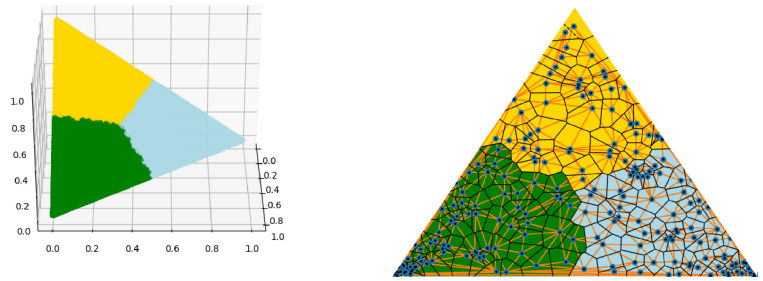
2.1 Characterizing Decision Boundary by Instances

The decision boundaries of a DNN model are abstract and hard to understand. Studies of human reasoning show that humans often learns from examples, or prototypes [15, 42]. To interpret a DNN model, one effective way is to link the numerical values of a prediction made by the model to examples. Specifically, as a technique of case-based interpretation tool, the interpretability in this article provides the users with the ability to infer the potential reasons for classification decisions by comparing the examples near the decision boundaries. We model the problem of characterizing the decision boundary as a data point selection problem that selects data points close enough to the boundary to make the decision boundary "visible." We call these boundaries *interpretable decision boundaries*. We measure the distances in the latent space, particularly the softmax layer of a DNN model, as this layer retains the most knowledge learned by the DNN model. The nearest neighbor classifier built from the selected boundary points is able to mimic the predictions made by a DNN model, thus serving the purpose of interpretability. Unless specified explicitly, the *decision boundaries* in the rest of this article mean the decision boundaries of a DNN model itself. The *interpretable decision boundaries*, or *interpretable boundaries*, however, indicate the ones formed by the output of the nearest neighbor classifier that mimics the DNN model.

The interpretable decision boundaries can be computed using a Voronoi diagram [7]. Specifically, let $P = \{P_k | k \in K\}$ denote a set of distinct points in space X, where a Voronoi diagram of P is the subdivision of the space into K regions. Data points in a region, denoted by R_k, have the following property: Their distance to P_k is not greater than their distance to any $P_j, j \neq k$. Formally, given a distance function $d(x, P_k)$, a region in a Voronoi diagram is defined as below:

$$R_k = \{x \in X | \forall j \neq k, d(x, P_k \leq d(x, P_j))\}.$$

We give an example to show an interpretable boundary between three classes in the latent space. As shown in Figure 2, regions with different colors represent different classes. Geometrically speaking, the softmax function maps the vector space $\mathbb{R}^K$ to the interior of the standard $K-1$-simplex, reducing the dimension by one (the range is a $(K-1)$-dimensional simplex in K-dimensional space), due to the linear constraint that all dimensions in an output vector sum to 1 meaning the vector space lies on a hyperplane. The decision boundaries, therefore, are like what



(a) Decision boundaries at the softmax layer.

(b) Interpretable decision boundaries built by a Voronoi diagram.

Fig. 2. The decision boundaries and interpretable decision boundaries for a 3-class classification task.

Figure 2(a) shows. We build a Voronoi diagram for all these points (K equals the total number of training data points) through Delaunay triangulation [36], shown by lines in Figure 2(b). Delaunay triangulation is the dual graph of a Voronoi diagram and is commonly used for Voronoi diagram construction. With a Voronoi diagram for all the points constructed, the interpretable decision boundaries between the classes are sets of connected edges shared by the neighboring regions of classes. However, computing Delaunay triangulation on n points in $\mathbb{R}^d$-space has the time complexity of $O(n^{\lceil d/2 \rceil})$ [62]. It is prohibitively expensive, if not totally impossible, to compute with high-dimension data.

2.2 Boundary Tree

To make the boundary characterization computationally feasible, previous work has attempted to use an optimized boundary tree named EB-tree to approximate interpretable decision boundaries [64]. The boundary tree (forest) algorithm [39] was initially proposed for fast online learning. Each node of the boundary tree represents a training point. For a query of training point y, the algorithm looks up the nearest node x to y. The lookup is done through tree traversal. Specifically, starting from the root, we find the nearest child to the query node y and go to that child to do a further search until there are no children closer to the query node than the current parent node x. Then, the label of x is used to predict the class of y. If x has the same label with y, then y is discarded. Otherwise, it is added to the tree and becomes the child of x. The process repeats for each training data point. A test data point is classified by the label of its nearest node in the boundary tree. As each edge in the tree crosses a decision boundary, all the nodes on the boundary tree, in essence, sketches the contours of the decision boundaries.

2.3 Locality Sensitive Hashing

Locality Sensitive Hashing (LSH) [6, 17, 25, 35] refers to a family of functions (known as LSH families) to hash data points into buckets. By doing so, data points near each other are located in the same buckets with high probability, while data points far from each other are likely to appear in different buckets.

LSH can be used to find the approximate nearest neighbors efficiently, as seen in constructing the interpretable decision boundaries [64] and automatic model debugging [44]. In this work, we use LSH to search for the nearest neighbors for training points while constructing the boundary graph.

LSH searches incur sublinear time complexity for similarity search even in high-dimensional space. Specifically, LSH uses a set of hash functions to map data points that are close to each other

to the same bucket with a high probability. With Euclidean distance, we choose hash functions as follows:

$$h_{a,b}(v) = \left\lfloor \frac{a^T v + b}{u} \right\rfloor,$$

where $a \in \mathbb{R}^d$ with each $a_i (i = 1, \ldots, d)$ independently drawn from a normal distribution $g(a_i \propto \exp(-a_i^2/2))$, u is a scaling number, and $b \in [0, u)$ is the bias. The probability that two points v_1 and v_2 collide under $h_{a,b}$ is then

$$P(h_{a,b}(v_1) = h_{a,b}(v_2)) = \int_0^w \frac{1}{c} g\left(\frac{t}{c}\left(1 - \frac{t}{u}\right)\right) dt,$$

where $c = ||v_1 - v_2||$. The constructed family of hash functions is $(r_1, r_2, \alpha, \beta)$-sensitive for $\alpha = p(1), \beta = p(c), \frac{r_2}{r_1} = c$. In practice, multiple hashing functions and probes are used to get more accurate results. We use the implementation in Reference [5] for our experiments.

2.4 Maximum Mean Discrepancy (MMD)

In this article, we use MMD as a metric to remove redundant points used to characterize boundaries.

Let $\mathcal{F}$ be a class of functions: $f \to \mathbb{R}$. Given two distributions $P(X)$ and $Q(Y)$, the maximum mean discrepancy (MMD) is defined as follows:

$$\text{MMD}(\mathcal{F}, P, Q) = sup_{f \in \mathcal{F}}(E_{X \sim P}[f(X)] - E_{Y \sim Q}[f(Y)]).$$

If $\mathcal{F}$ is the reproducing kernel Hilbert space (RKHS) with the kernel function $\mathcal{X} \times \mathcal{X} \to \mathcal{R}$, then the suprenum can be achieved at:

$$f(x) = E_{X' \sim P}[k(x, X')] - E_{X' \sim Q}[k(x, X')].$$

The function $f(x)$ measures the maximum discrepancy between the two expectations in $\mathcal{F}$. A biased empirical estimate of the MMD can be obtained by replacing the population expectations with empirical expectations computed on samples X and Y, namely

$$\text{MMD}(\mathcal{F}, X, Y) = sup_{f \in \mathcal{F}}\left(\frac{1}{m}\Sigma_{i=1}^m f(x_i) - \frac{1}{n}\Sigma_{i=1}^n f(y_i)\right).$$

2.5 Related Work

2.5.1 Interpretability for DNN Models. Visualizing a DNN model by examining features extracted by hidden neurons in DNNs may help users to infer the relationship between a prediction and these features [26, 67, 68]. However, it does not work for a shared model as visualization requires the disclosure of the internal architecture of the DNN model. The leak of model parameters makes the model vulnerable to malicious users [63].

The perturbation approaches [18, 52] attempt to learn an image perturbation mask that minimizes the prediction score for a given class so that a user can infer the significance of certain features on the prediction. However, this approach incurs a non-trivial computational cost. Recent study [31] proposes to use influence functions to identify the most useful or harmful training points (removing harmful training points leads to the decrease of test loss) for predictions. This method requires access to the whole training dataset, which is unrealistic in the model sharing settings.

Saliency maps [55, 57, 59] interprets DNN models by tracing a prediction back to the input data and weigh the importance of different features through the gradients of these features leading to the prediction.

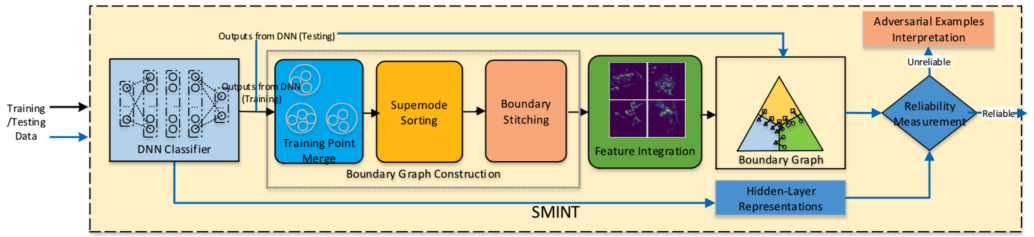


Fig. 3. The architecture of SMINT.

Some other works [19, 65, 69] try to build deep learning architectures that are self-interpretable. For example, in Reference [69], each filter in a high convolutional layer is forced to represent a specific object part. Reference [65] proposes to use tree regularization to improve the interpretability of DNN models. These methods, however, are not model-agnostic, and they often result in a lower accuracy due to their generalization ability.

Another approach of achieving interpretability is to mimic the predictions of a DNN model with an interpretable model. A typical method is to mimic a complicated model with a decision tree [10, 13, 16, 54]. However, this approach often only handles structured data. When the structure of data is complicated, a decision tree itself can be difficult to interpret.

There is work seeking prediction interpretability through finding examples or prototypes in training data, which is along the line of case-based reasoning [1]. These work identify training data points that are close to other points within their own classes and far away from those in different classes [8]. These data points are called prototypes. Prototype finding is solved as a set cover optimization problem in Reference [8]. Prototype finding is complemented by criticism finding. Criticisms are outliers in a class. They do not fit a model well. MMD-critic [27] intends to use these criticisms together with prototypes to characterize a model. However, MMD-critic does not provide any data structure to organize the training data it selects and contains little information for users to understand why the model classifies a data point into one class, not another.

2.5.2 Reliability of Model Interpretations. Most existing interpretation methods suffer from the reliability problem [2, 3, 20, 30]. Recent study [3] shows that both perturbation-based interpretations [52] and saliency maps [55, 57, 59] are not sensitive to the random perturbations of the input data or model parameters. Similarly, the influential training data selected by the influence function method [31] for a given test point may change due to small random perturbations of the test point. Adversarial rather than random perturbations can easily generate inconsistent interpretations [20]. The generated examples look like normal examples and are correctly predicted by the DNN model used, but the corresponding saliency maps mark different regions in the original input for their significance on the prediction. Unreliable interpretations confuse users and make it difficult to debug a model, particularly when test cases are generated during the automatic model testing [49]. One key reason for the unreliability is that the test points may be drawn from data with different distribution with the training data, which results in unexpected interpretations. Existing approaches lack of efficient means to detect out-of-distribution inputs.

3 OVERVIEW OF SMINT

Figure 3 shows the architecture of SMINT. The black and blue lines illustrate the workflows for training and test data, respectively. For training data, SMINT first feeds the data into the DNN model. The softmax outputs of the DNN model is then used to construct a boundary graph, which is the core data structure to characterize the interpretable decision boundaries. The construction

of a boundary graph includes three steps, namely training point merge, training point sorting, and boundary stitching. After these steps, local features, such as saliency maps and perturbation masks are integrated to the nodes in the boundary graph. For test data, the softmax outputs of the DNN model is used to find their nearest neighbors in the boundary graph as the interpretation of predictions. SMINT will evaluate the reliability of the interpretation by utilizing the hidden-layer neighborhood information. In the following, we describe the details of the four design elements of SMINT, namely boundary graph, supernode, local feature integration, and interpretation reliability measurement.

3.1 Boundary Graph

A boundary graph approximates the interpretable decision boundaries. It is essentially the nearest neighbor graph (NN-graph) in the latent space, particularly the softmax outputs of a DNN model. Using softmax outputs is motivated by the following two observations. First, the softmax layer is generally linear separable, and a simple distance metric works well to characterize the similarity between data points. Second, the DNN model is often optimized with a softmax layer and the cross entropy loss so that representations at the softmax layer contains the most knowledge learned by the model about separating training data points. In a boundary graph, two training points are connected if they are the nearest neighbors to each other (this graph was called the symmetric nearest neighbor graph in Reference [43]). The boundary graph can be regarded as an NN-graph constructed from the data points close to the decision boundaries. All the edges connecting the training points within the same class are pruned and the remaining edges are used to characterize the decision boundaries. The key advantage of a boundary graph is that it is much faster and feasible to compute compared with the Delaunay triangulation based methods as described in Section 2.

3.2 Interpretation with Supernodes

SMINT is a case-based reasoning method. It provides interpretability through identifying relevant training examples that form the decision boundaries and assisting users to understand feature differences along a decision boundary. Example selection is critical for achieving good interpretability. Simply selecting two neighboring examples across a boundary at the softmax layer has limitations as the representations at the softmax layer for DNN models are dense, which results in examples with different features are mapped to the same neighborhood. SMINT uses *supernodes* to address this problem. A supernode is defined as a group of examples that have similar representations at the softmax layer. This enables the boundary graph to capture rich feature differences between different classes, thus improving the interpretability. We also provide a means to measure the diversity of selected examples to maximize their feature representativeness and eliminate redundancy.

To enable nearest neighbor search in the boundary graph, each supernode is represented by their centroid. Compared to 1-NN classifier, the supernode approach greatly reduces the variance in boundary example selection and therefore improves the reliability of interpretable decision boundaries. Note that the Voronoi diagram formed by supernodes has a similar computing complexity with a higher-order Voronoi diagram [9].

3.3 Local Feature Integration

Boundary examples alone may not be sufficient for users to understand features differentiating two classes. SMINT addresses this problem by presenting the critical features together with these examples. Local explanation methods are useful for identifying these features. For instance, the saliency maps [55, 57, 59] and LIME [52] highlight the critical regions in the inputs that lead to

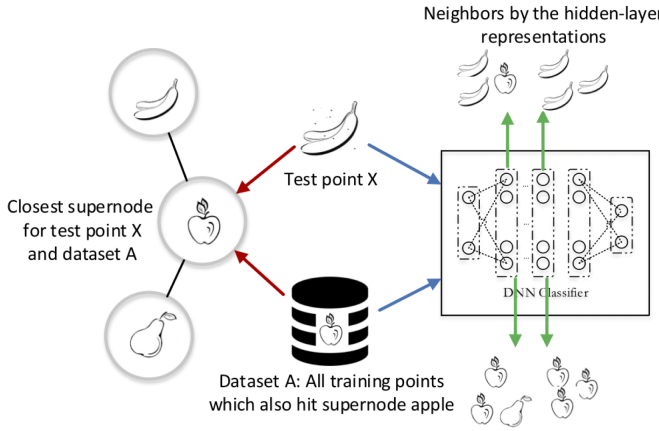


Fig. 4. An example explaining the rationale of HLA p-test.

predictions. SMINT associates the critical regions leading to a prediction for each training point in the boundary graph. Recent study [2, 3, 20, 30] shows out-of-distribution test examples may trigger unreasonable saliency maps or image masks. Combining local explanation with supernodes achieves better reliability in interpretation. This is because the DNN models are optimized upon the training data and the integrity of training data has guarantee.

3.4 Interpretation Reliability Measurement

The key reason for the unreliability of the existing interpretation methods is that these approaches do not check whether inputs are within a model's capacity. If the inputs are out-of-distribution examples that do not lie on the same manifold with the training points, then the interpretations become unreliable.

For SMINT, the interpretations to a test point provided by the boundary graph are the nearest supernodes. To measure the reliability, we compare the test point with the subset of training points, denoted by S, that also hit the same supernode. The set S can be recorded by feeding all the training points to the boundary graph (one-off process). These supernodes are regarded as unreliable interpretations if the features in the query point are statistically different from the features of points in S, which indicates that the features in the test data are not seen in training.

SMINT does not only rely on the softmax layer for reliability measurement. For out-of-distribution data, if a sample belonging to class A is "confidently" misclassified as class B, then it is difficult to distinguish this sample from training data points that are "confidently" classified as class B based on their softmax outputs. Consequently, the nearest supernodes of this sample are also from class B. However, each layer of the DNN model extracts certain features from the training data and it is likely some differences between this sample and the training data of class B are captured in one of the hidden layers. SMINT utilizes the information of neighboring training points of a sample at these hidden layers to measure the interpretation reliability.

As shown in Figure 4, a network used for classifying apple, pear, and banana misclassifies an image of a banana (X) as an apple. In the boundary graph, apple images dominate X's neighbors as their softmax outputs are close to X's. When we look into the hidden-layer representations of X and the training points that also hit the supernode in the boundary graph, we find that the dominant neighbors for X in the hidden layers are bananas. In other words, the neighboring relationships of X in the boundary graph are inconsistent with those in hidden layers and thus fail to provide correct

ALGORITHM 1: Candidate selection for supernodes.

Input: D: Data points in the latent space.
L: Labels of different classes.
Output: Union_find sets S indicating the grouped supernodes.

```
Procedure Group_Supernodes_by_NN_Graph()
    foreach l ∈ L do
        S[l] = union_find_set(D_l)
        // Construct a LSH query object q for D_l
        foreach idx ∈ range(D[l].size()) do
            nearest_neighbor_idx = q.find_nearest(D[l][idx])
            S[l].union(idx, nearest_neighbor_idx)
        end
    end
return S
```

interpretation to the prediction. By making use of the inconsistency, we propose an algorithm named Hidden-Layer-Aware (HLA) p-test to measure the reliability of interpretations.

4 BOUNDARY GRAPH CONSTRUCTION

In this section, we describe the detailed algorithms for boundary graph construction.

4.1 Training Point Grouping

Supernodes contain training data points showing a variety of feature differences near the decision boundaries. Clustering methods like k-means or hierarchical clustering are not feasible for forming supernodes due to expensive similarity comparison when the number of clusters k and the data point dimension d is large, which is common for the outputs of DNN models. We propose a fast two-phase algorithm to group training points into supernodes. The first phase groups candidates of the training points with similar softmax outputs (or a given representation layer output when the number of classes is small). The selected candidates may contain some redundant training examples (i.e., visually similar). The second phase eliminates redundant examples.

In the first phase, the algorithm (see Algorithm 1) uses *union-find sets* to group all the points in the same neighborhood as candidates of a supernode. Specifically, if point A and B are neighbors according to LSH, then they are put into the same union-find set. We pass all the training points and union each point and its nearest neighbor. Each union-find-set, therefore, contains all the candidates for a supernode. For the simplicity of complexity estimation, we assume each class has a similar amount of data points (k). The whole set of data points, denoted by D has, therefore, $|D| = k \cdot |C|$ data points, in which C is a set of classes. The time complexity of the algorithm is $O(|C|(k \cdot log(k) + d \cdot k^{\rho})(\rho < 1))$. For each class c, the union-find-set $S[c]$ contains all the connected supernodes of the class-wise nearest neighbors.

The first phase may form big supernodes containing many similar data points. Similar points showing the same feature differences should be removed to keep the supernodes concise. To achieve this, we utilize the higher-dimensional feature embedding of training points before the softmax layer. As the feature embeddings normally have more dimensions and contain richer information about fine-grained features, we can identify characteristically different points even though the softmax outputs of these data points are similar to each other. Consider a supernode $\mathcal{N}$ containing m examples and n is a pre-defined threshold for the maximum size of a supernode, the target is to remove $n - m$ examples from $\mathcal{N}$. We remove the examples one by one as follows:

ALGORITHM 2: Greedy Algorithm for Boundary Graph Construction

Input: *S*: Union-find sets indicating the supernodes for different classes.
Label: Label for different supernodes.
Output: A boundary graph mimicking the decision boundary of the DNN model being interpreted.

```
Procedure Construct_Boundary_Graph()
    //Sort all supernodes by the distance from their centroids to the decision boundaries.
    G = Graph(S)
    sorted_supernodes = sort_by_dist_2_boundary(S.supernode())
    foreach supernode ∈ sorted_supernodes do
        closest = find_nearest(supernode)
        if Label[closest] != Label[supernode] then
            G.node(supernode)
            G.edge(closest, supernode)
        end
    end
    //Connect the supernodes that are closer than the average length of connected ones in the graph.
    avg_edge_len = (Σ_{e∈G.edges} e.len) / |G.edges|
    foreach s1, s2 ∈ G.nodes do
        if dist(s1, s2) ≤ avg_edge_len then
            G.edge(s1, s2)
        end
    end
    return G
```

For each example x_i, we evaluate the maximum mean discrepancy between $\mathcal{N}$ and $\mathcal{N} - x_i$; the one with the minimal value, i.e.,

$$\underset{i}{\operatorname{argmin}}(\mathrm{MMD}(\mathcal{N}, \mathcal{N} - x_i)),$$

is eliminated from the supernode. The reason is that the removal of x_i changes the distribution of the examples in the supernode the least. The process repeats for the remaining data points in the supernode until its size falls below the threshold n.

4.2 Supernode Re-ordering

The boundary graph is constructed using supernodes obtained as above. Each supernode is represented by its centroid. We give a greedy algorithm (Algorithm 2) to construct the boundary graph. The supernodes that are close to the decision boundaries are retained in the graph while those far from the decision boundaries are discarded.

We sort the supernodes by the distances from their centroids to the decision boundaries in an ascending order. The key is how to measure the distance. The decision boundaries at the softmax layer are already known as hyperplanes in the $(C-1)$-dimensional space when there are C classes. Let $\mathbf{x} = (x_1, x_2, \ldots, x_C)$ denote the softmax output for an example, the decision boundary B_{ij} between class i and class j can be described as a portion of a hyper-plane H, $H = \{\mathbf{x} \in \mathbb{R}^C | \Sigma_{i=0}^{C} x_i = 1\}$, which satisfies the following conditions:

$$x_i = x_j$$

and

$$x_c < x_i, (\forall c \in [C], c \neq i, c \neq j).$$

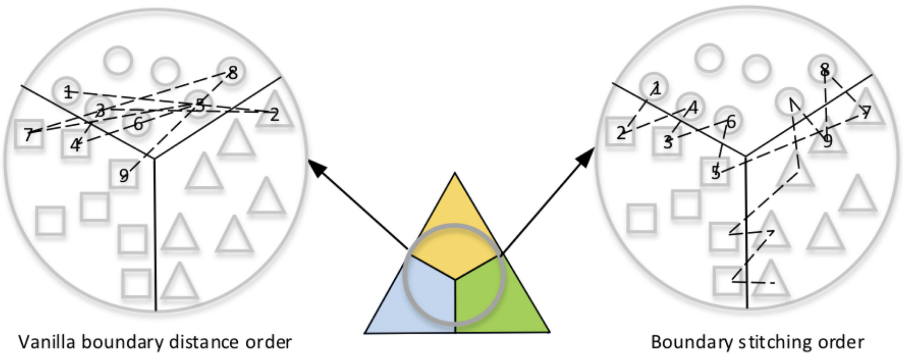


Fig. 5. The comparisons between the vanilla boundary distance order and boundary stitching order.

Calculating the distance from a supernode $\hat{\mathbf{x}}$ to a decision boundary B_{ij} between class i and j is essentially to find a data point $\mathbf{x}'$ on B_{ij} to minimize $d_{\hat{\mathbf{x}}',\hat{\mathbf{x}}}$. The computing complexity of distances is $\frac{C\cdot(C-1)\cdot m}{2}$, in which m is the number of supernodes and C is the number of classes. This can be intractable when m and C are large. Instead of using one-versus-one to compute the distance to each boundary, we use one-versus-all to reduce the complexity. Assuming $\hat{\mathbf{x}}$ belongs to class c, the one-versus-all decision boundary B_c between class c and all the rest classes is the boundary between c and the second most likely class for $\hat{\mathbf{x}}$, i.e.,

$$b = \underset{\mathbf{x}}{\text{argmax}}(\hat{\mathbf{x}} - \hat{x}_c).$$

This enables the distance computing to be performed in a two-dimensional space. The distance between $\hat{\mathbf{x}}$ and B_c is essentially the distance between point $(\hat{x}_c, \hat{x}_b)$ to the line $x_c = x_b$. The number of distance computations is therefore reduced to $C \cdot m$.

For each data point, we calculate its shortest distance to all decision boundaries and use the distance as a metric sort all points in ascending order. We then insert these supernodes to the boundary graph according to the order. During insertion, the supernode is discarded if it has the same label with its nearest neighbor in the graph. Otherwise, the supernode is added to the graph by connecting to its nearest neighbor in the graph.

One may also use the support vector machine (SVM) for finding the representative examples for decision boundaries. However, training SVMs for a large number of classes may not be feasible for many complex tasks. Also, for the softmax layer of a deep neural network model, each dimension of the embedding has the actual meaning (i.e., the probability that the example belongs to a certain class). Directly training SVMs may not be optimal as it would abandon this information.

One problem with this graph construction method is that it may include long edges with weak interpretability in the graph. This is due to that two supernodes that are close to the decision boundaries are not necessarily close to each other. The long distance between the two supernodes is likely to results in few common features between them, making it difficult for users to infer how the decision boundary between them looks like. To address this problem, we propose an algorithm named boundary stitching to explicitly select the next point to insert into the graph.

4.3 Boundary Stitching

The main purpose of boundary stitching is to avoid long edges to link supernodes that share little common features. As shown in Figure 5, simply following the boundary distance of supernodes may yield a boundary graph on the left. The edges tend to be long and the supernodes sharing an edge does not characterize the local decision boundary well.

ALGORITHM 3: Boundary Stitching Algorithm

```
Input: LSH: An LSH query object constructed for supernodes.
G: The current boundary graph.
current: the supernode being inserted into the boundary graph.
Function get_next_to_insert(current, LSH, G, k)
    //find sorted k nearest candidates for the current supernode using LSH
    kNearests = LSH.query(current, k)
    if kNearests is not empty then
        foreach n ∈ kNearests do
            if n ∉ G && n.label ≠ current.label then
                return n
            end
        end
    end
    return NULL
```

The boundary stitching algorithm (Algorithm 3) makes use of the distances between the current supernode in the graph and the candidates to be added to optimize the boundary characterization. It selects a candidate with a different label but close to the current supernode to add. To do this, it searches for the k-nearest neighbors (kNNs) of the current supernode. The nearest candidate among the k neighbors with a different label is added to the graph regardless of its boundary distance order. This procedure, just as the name suggests, connects the supernodes by "stitching" so that long edges are only introduced when there is no supernode with a different label in the k-nearest neighbors of the current supernode.

Greedy boundary construction (Algorithm 2) and boundary stitching characterize the interpretable decision boundaries with the following properties: (1) The non-boundary supernodes far away from the decision boundaries are not in the graph. Supernodes are added to the graph according to their distances to boundaries; therefore there always exists a supernode in the graph closer to the boundaries than a non-boundary supernode. (2) For a supernode $\hat{\mathbf{x}}$ in the graph, if there is an edge to one of its k nearest neighbors, denoted by $\hat{\mathbf{x}}^{'}$, crossing the boundary, $\hat{\mathbf{x}}^{'}$ is in the graph as well. The first property ensures that the graph gives an overview of boundary nodes. The second property ensures that boundaries around local regions are continuous.

5 FEATURE INTEGRATION IN BOUNDARY GRAPH

To help users to easily identify the feature differences among boundary nodes, SMINT includes local interpretation methods such as saliency maps and interpretable perturbations for boundary nodes. As the local interpretations are computed on the training points in the boundary graph, they have better reliability than on arbitrary test data.

We show one simple method that fools the state-of-the-art saliency interpretation method [59]. Given a testing point x and the saliency map method S, the malicious user can generate a test case x' to minimize $\mathcal{L} = -\Sigma_{i=1}^{N} y_i \log_{P_i} - \Sigma_j \Sigma_m \Sigma_n ||S(x')_{jmn} - S(x)_{jmn}||$ under the constraint $||x' - x||_\infty \leq \epsilon$. The idea here is to mess up the saliency map but keep the prediction without changing the testing point much. Compared with general adversarial examples [11, 46], the attack here is mainly to fool the interpretation method rather than the DNN model itself. For vanilla gradient-based saliency map, Figure 6 shows that the above attack method can generate the test example (see Figure 6(b)), which is also correctly classified as "paper towel." However, the corresponding

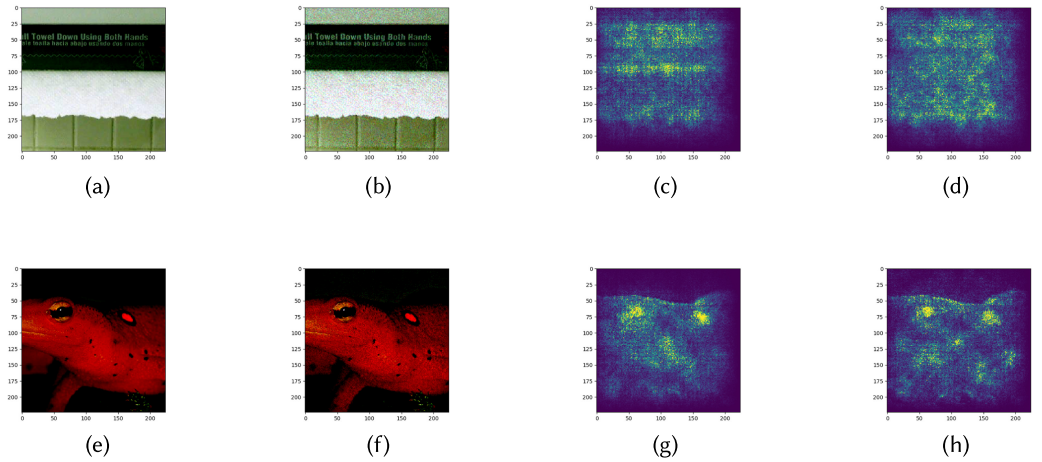


Fig. 6. Two examples of generating adversarial examples to fool the saliency map methods. For each row, the order of the images are as follows: raw input, perturbed input, raw saliency map, saliency map after the attack, respectively.

saliency map (Figure 6(d)) is in disarray. Even for integrated gradients [59], which are supposed to be more robust, we found that the saliency maps could also be affected (see Figure 6(h)).

Local interpretations of boundary data points do not suffer from this problem. First, it is easier to guarantee the integrity of the training data. Second, as the model is optimized upon training points, the corresponding local interpretations such as saliency maps are also more reliable.

Moreover, the differences in local interpretations of boundaries nodes give users a better understanding of model behaviors compared to presenting local interpretations on a single node alone. As indicated in some recent study [41], an explanation should be contrastive. In other words, people request a contrastive explanation where "Why P rather than not-P?" is preferred over simply "Why P?" Correspondingly, in SMINT, the neighboring structure provides the guide to compare between classes while local feature visualizations further differentiate those classes. However, since we only need to generate the features for the data points included in the graph, the computational overhead for local interpretations is also greatly reduced.

6 HIDDEN-LAYER-AWARE P-TEST

For a test data point, we give a hidden-layer-aware (HLA) p-test method (see Algorithm 4) to measure the reliability of its interpretations. The idea of HLA p-test is to compare the neighboring relationships of a test data and those of nodes in the supernode hit by the test data.

Each layer of a DNN model extracts certain features from an input data point. For a pre-trained network f with n layers, the outputs for a training point x at different hidden layers and the softmax layer can be represented as $f_l(x)(l \in \{1, 2, \ldots, n\})$. We use the aggregated neighboring information from each layer of a data point to measure the reliability of interpretations.

The algorithm is shown in Algorithm 4. First, a one-off procedure performs preprocessing on training data once a model is trained. The process finds kNN for each training data point x at each layer using the hidden layer output $f_l(x)$. For each training point, it obtains a vector containing the numbers of neighbors belonging to each class, i.e., $v_x^l = \{|knn^l(x)|_{c_1}, |knn^l(x)|_{c_2}, \ldots, |knn^l(x)|_{c_N}\}$, where $|knn^l(x)|_{c_i}$ is the number of neighbors of x with label c_i at layer l. The vectors of nodes that are closest to a boundary supernode, denoted by $h(x)$ are aggregated. For each boundary supernode, a non-conformity value is computed. The value represents the total number of neighbors

ALGORITHM 4: Hidden-Layer-Aware p-test

Input: f: pre-trained network;
n: # of layers in f;
$D = (X, Y)$: training data;
$\hat{x}$: test point;
k: # of nearest neighbors for each point;
G: boundary graph built for f;
C: # of classes
Output: reliability of interpretation of $\hat{x}$'s prediction.

Function `Preprocessing`(X, Y, f, k)
  Find k nearest *neighbors* for each point in X at layer l ($l \in \{1, 2, \ldots, n\}$).
  for $x \in X$ **do**
    for $l \in 1 : n$ **do**
      Construct neighboring label vector v_x^l.
    end
  end
  //Group the neighboring label vectors for training points by their centers.
  for $x \in X$ **do**
    $V_{h(x)}.add(\{v_x^1, v_x^2, \ldots, v_x^l\})$
  end
  //Compute the non-conformity values for training data in each node group
  for *node* $s \in G$ **do**
    for $v \in V_{h(s)}$ **do**
      $\alpha_v = \sum_{l=1}^{n} (\sum_{\substack{i=0 \\ i \neq Y_s}}^{C} v^l[i])$
      $\alpha_s.add(\alpha_v)$
    end
  end
  $\alpha_G = \{\alpha_s | s \in G\}$
  return α_G

Procedure `p-test`$(\hat{x}, \alpha_G)$
  Construct neighboring label vector $v_{\hat{x}}^l$ for $\hat{x}$ at layer l ($l \in \{1, 2, \ldots, n\}$)
  Retrieve $\alpha_{h(\hat{x})}$ from α_G
  Compute non-conformity value $\alpha_{\hat{x}}$ for $\hat{x}$
  $p_{\hat{x}} = \frac{|\forall \alpha_r \in \alpha_{h(x)} : \alpha_r > \alpha_x|}{|\alpha_{h(\hat{x})}|}$

that do not belong to the same class of the supernode. The non-conformity values are stored for p-test. Note that the non-conformity value of a supernode is calibrated with a set of reference data to establish what the normal non-conformity threshold is.

Second, when measuring the reliability of interpretation to a test data point $\hat{x}$, a vector containing the numbers of k nearest neighbors belonging to each class is constructed. A non-conformity value is computed for this vector. Assume $h(\hat{x})$ is the boundary supernode that is the closest to $\hat{x}$. The p-test procedure compares the non-conformity value of the vector of $\hat{x}$ and that of $h(\hat{x})$ and uses the result as the reliability metric. A large value produced by p-test indicates a likely out-of-distribution test data point.

Our implementation speeds up the kNN search with FALCONN, a fast LSH (locality sensitive hash) algorithm [5]. The complexity of kNN search can be reduced to $O(s + d' \log d')$, in which s denotes the number of non-zero items in an input vector and d' is the reduced dimension of the input vector. d' is much smaller than the dimension of the input vector.

The method we propose is generic and model-agnostic. Service providers can provide this functionality to enhance the interpretability for the models built by any model developers. The hidden-layer outputs do not need to be provided to the application developers using the pre-trained models. The interpretation score or reliability metric is computed at the server-side and given to the application developers as extra information. After out-of-distribution samples are detected, various following-up operations can be added to the service. For instance, service providers can mark users who provide multiple adversarial examples as malicious.

An interpretation with a low reliability score can be caused by either a benign or an adversarial test data point. In both cases, they have different distribution with the training data. SMINT provide a means to differentiate the two cases as they result in different problems to a service provider.

A recent study [38] found that the adversarial examples lie off, yet close to the data sub-manifold. The benign out-of-distribution examples, however, are very far from the data sub-manifold. Adversarial examples are perturbed slightly and purposely to get misclassified. There is a high chance to further perturbing the data so that it can escape from the adversarial subspace. The predictions of benign out-of-distribution examples, in contrast, are resistant to slight perturbations. Therefore, when SMINT finds an example with a low interpretation score, it attempts to transform the example and use the interpretation of the transformed data to better fix and interpret the prediction. A service provider may check the misclassified label of the adversarial examples and the recovered interpretation for the non-adversarial variant to identify malicious users. SMINT uses the idea of an ensemble of squeezers [66] to reduce the unnecessary large dimensions exploited by the adversaries. For image inputs, fast squeezers like color depth, spatial smoothing, and image compression methods are available. SMINT squeezes the identified examples with low interpretation scores and obtains the interpretation for the squeezed inputs. The results of the ensemble methods are used to vote for the final interpretation provided for users.

7 EVALUATION

We evaluate SMINT on two image classification tasks. We also examine the applicability of SMINT on a text classification task. For the image classification tasks, the models are the convolutional net (CNN) [61] for MNIST hand-written digit dataset and InceptionResNet2 network [60] for ImageNet dataset (ILSVRC2012) [53]. The maximum supernode size is set to 20 in all three cases. The value is chosen to avoid overwhelming human on checking boundary graphs. We evaluate our algorithm on the following two aspects: model mimicking performance and the interpretability. The model mimicking performance is measured by the decision consistency between a boundary graph and its corresponding DNN classifier. The metric we use is the *F-measure* of the boundary graph predictions against the DNN model predictions. We measure the interpretability of SMINT by conducting a human pilot study. SMINT is compared against the vanilla boundary tree [39], EB-tree [64], and MMD-critic [27].

7.1 Model Accuracy and Fidelity

As shown in Table 1, the boundary graph achieves a comparable accuracy of the original DNN models. For both MNIST and ImageNet datasets, the boundary graph outperforms the vanilla boundary tree and the EB-tree. For F1-score, the boundary graph outperforms EB-tree significantly. On the ImageNet dataset, it achieves the improvement of over 1%. The average length of the edges in the tree/graph is an important metric for the interpretability. Shorter edges ensure that the nodes

Table 1. Comparisons of Boundary Tree, EB-tree, and Boundary Graph

Model	Accu.	F1-score	$\overline{Edge}$	nodeCnt
MNIST(Network)	0.9918	—	—	—
MNIST(Boundary Tree)	0.9917 ± 0.0024	0.9985 ± 0.0042	0.5879 ± 0.0178	47 ± 7
MNIST(EB-tree)	0.9919	0.9987	0.5607	24
MNIST(Boundary graph)	**0.9922**	**0.9995**	**0.4636**	82
ImageNet(Network)	0.7643	—	—	—
ImageNet(Boundary Tree)	0.7526 ± 0.0038	0.9664 ± 0.0067	0.6739 ± 0.0193	21.630K ± 1032
ImageNet(EB-tree)	0.7539	0.9789	0.4911	15.667K
ImageNet(Boundary graph)	**0.7640**	**0.9911**	**0.3732**	**9.975K**

connected share more features and visually close to each other, thus providing better interpretability. The last column shows the total number of training points included in the tree/graph and disclosed to users. The MNIST dataset is relatively simple. Its decision boundaries between different classes are clear. Boundary tree and EB-tree use fewer boundary nodes to characterize the boundaries. Boundary graph uses slightly more nodes to provide diversity, but the total number of training images disclosed to users is still small. For the large ImageNet dataset, boundary graph shows its advantage in characterizing boundaries. It reduces the number of training points included in the graph by around 36.9% while providing better boundary characterizing performance. Boundary graph is highly scalable in terms of the percentage of nodes to be disclosed among training data. Specifically, for MNIST dataset with 60,000 training points and 10 classes, the average number of training points required for each class is 8.2 while the number is 9.9975 for ImageNet, which has 1.28 million training points and 1,000 classes.

7.2 Global-View Interpretability

For a DNN model, SMINT is able to present its boundary graph as a global view of the model. The global view gives interpretability on the model weakness. Users are able to see what kind of training errors and ambiguities exist in the model.

As shown in Figure 7, the boundary graph is able to locate the ambiguous part of the model. For example, comparing the supernode of "9" at the bottom of the graph and its neighboring supernode of "4," we can see that there exist training data points indistinguishable even for human eyes. We also notice many training errors from the boundary graph marked by blue circles in Figure 7. For some supernodes, such as "8" and "2," all the points are training errors. In other words, the boundary graph identifies the weakness of the model by showing that the model has not learned the knowledge given by the labels of those training points. This is represented as a tangled decision boundary. Similarly, the training points in the supernode "0" all have the true label "6" and those in supernode "2" have the true label "7," indicating a high chance of test errors between classes 0 and 6 and classes 2 and 7. As expected, the corresponding test errors for these classes are higher (in average 6) than others (in average 1).

By looking at the boundary graph at Figure 7, the overall observation is that the network is not sufficiently trained as it exhibits under-fitting. Indeed, the network was deliberately trained with only 10 epochs. After we retrain the same network with 50 epochs, we get the boundary graph shown in Figure 8. It is obvious that the boundaries are clearer and more precise than the previous one. There are significantly less training errors in the boundary graph.

Figure 8(b) shows the saliency maps for the training points in each supernode, by which one can infer the features the DNN model uses for decision making. For example, the key feature for

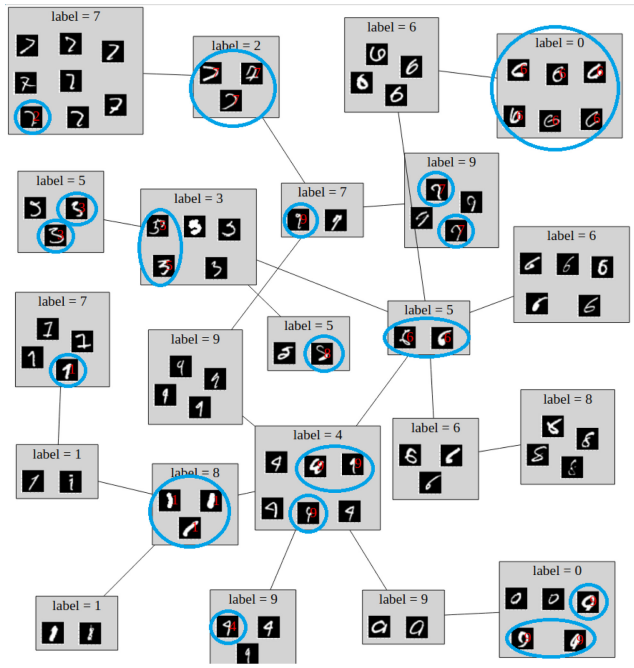


Fig. 7. The boundary graph built for the DNN model trained on MNIST dataset. Nodes with blue circles are training errors (i.e., the DNN-predicted label is different from the true label); epoch number = 10.

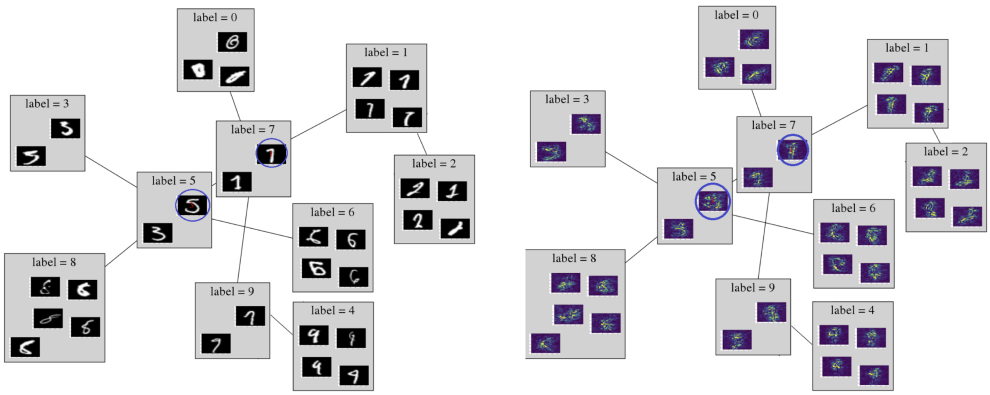


Fig. 8. The boundary graph built for the DNN model trained on MNIST dataset; epoch number = 50. DNN model accuracy = 0.9950 and F1-score = 0.9996.

"4" is the top left curve. For "2," the critical part is whether a horizontal line exists at the bottom. For "0," the closed left part is an important feature. The key difference between "5" and "6" is whether the left part is closed. For training points in the supernode "8," the cross in the middle is the key feature. Compared to the model trained for 10 epochs, this model has better generalization capability and suffers less from mislabelled training data. For example, the mislabelled "3" in the supernode "5" is still regarded as a "5." We then train the DNN model with 100 epochs and get

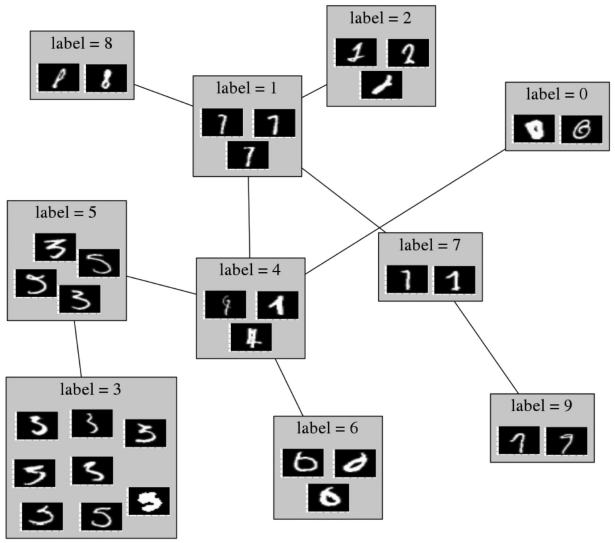


Fig. 9. The boundary graph built for the DNN model trained on MNIST dataset. Nodes with blue circles are training errors (i.e., the DNN-predicted label is different from the true label); epoch number = 100. DNN model accuray = 0.9942 and F1-score = 0.9998.

the boundary graph (see Figure 9). Although no training error exists in the boundary graph, the model has worse generalization ability and lower accuracy due to over-fitting.

Regardless of the model accuracy, the boundary graph always mimics the model with fidelity over 0.999 on MNIST dataset. The average edge length is an indicator of how a model fits the data. A shorter average edge length often means a clear boundary. When the number of epochs increases from 10 to 100, the average length of the edges increases from 0.4636 to 0.8102.

Users often expect to exploit the interpretability information to improve the model. In the following, we show a simple yet effective method to use the information provided by SMINT to refine the model. It is commonly known that training points may contain ambiguity. Those ambiguous training data often lead to lower predictive accuracy. Boundary graphs can assist users to identify them. Given a validation dataset, we can feed the validation points to the boundary graph and get the nearest node that the validation points hit in the boundary graph. The misclassifications within the validation dataset can be interpreted in a geometrical manner. Intuitively, in the boundary graph, a misclassification tends to happen if the supernodes do not characterize a faithful decision boundary between classes, which indicates that the decision boundary in this region is entangled by these training points. This can be caused by mislabeled data. One solution to address this problem is to remove the training points that are responsible for the ambiguous local decision boundary and retrain the model to obtain a decision boundary that separates different classes better in this region, which achieves higher predictive accuracy on the validation dataset.

However, it is not feasible to simply remove the training points in the supernode hit by the misclassified validation points, as the local boundary is characterized by training points at different sides of the boundary. SMINT addresses this problem by first identifying the nearest neighbor (N) of the supernode (H) hit by a validation point in the graph, which shares the label of the validation point and then recovering training points (R) close to these supernodes but discarded during graph construction. SMINT deletes data points in N, H, and R and retrains the model to retest on the

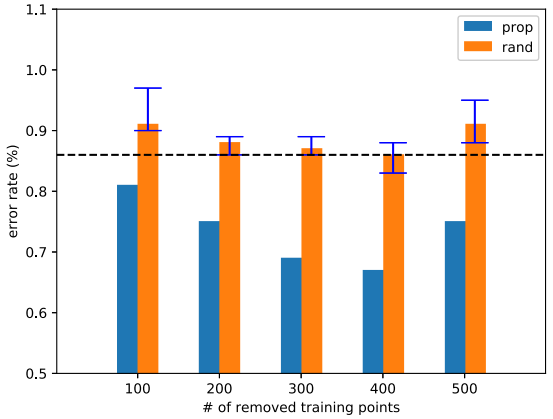


Fig. 10. Error rate on the MNIST test data. *prop* is the proposed method to remove ambiguous training points and *rand* is a random method. The x-axis indicates the number of training points removed. The dashed line shows the error rate when no training points are removed.

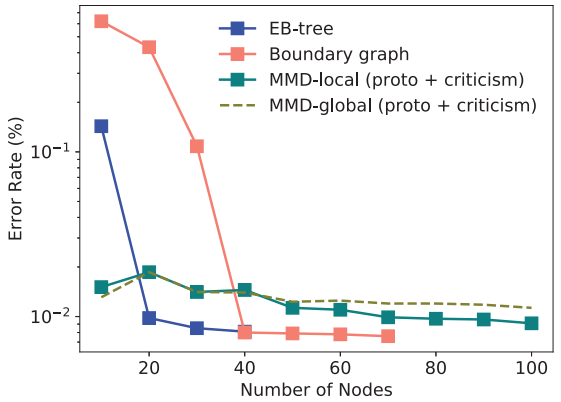


Fig. 11. Error rate vs. number of prototypes (nodes).

validation dataset. By doing so, SMINT is able to remove the ambiguous patterns that lead to the tangled local decision boundary in the training dataset.

We use LSH to identify the data points to remove. Figure 10 shows the evaluation of the proposed method on MNIST dataset. We vary the total number of data points to remove from 100 to 500 and compare our method with a method that randomly deletes the same number of training data points. Our method is able to reduce the error rate by up to 0.18% without changing any other hyper-parameters of the model. When the data that a user applies the model to have a relatively stable distribution (i.e., can be captured well by a validation dataset), the boundary graph enables a model customization service for the user, in which the user submits their validation dataset and gets the model retrained on a subset of the original training dataset with less ambiguity for their own applications.

Training data selection is the main function of forming a global view of a model. We compare the data selection performance among boundary graph, EB-tree, and MMD-critic ($\gamma = 0.05$). Running MMD-critic on the whole ImageNet requires excessive memory so that we do the experiments on the MNIST dataset. Figure 11 shows the error rate of different training data selection methods.

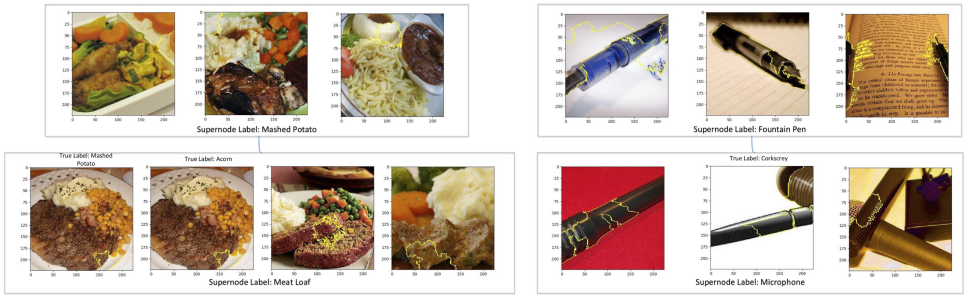


Fig. 12. The local decision boundary between (a) mashed potatoes and meat loaf and (b) fountain pen and microphone in the boundary graph.

Since a constructed boundary graph has a fixed number of nodes, we plot the error rate during the graph construction for comparison. For MMD-critic, besides the prototypes, we also generate 35 criticisms in attempt to improve the accuracy by providing diverse data distribution. However, the improvement is trivial as the criticism patterns do not appear often in the test data, and therefore the criticisms are not counted. The boundary graph node selection achieves high accuracy with a small number of nodes. As the boundary graph is optimized to characterize the decision boundary from the most ambiguous data points, it shows a high error rate initially during graph construction when the number of nodes is small. However, this node selection approach helps the boundary graph capture the model knowledge faithfully by putting ambiguous data first.

7.3 Local-View Interpretability

The boundary graph can be regarded as a compact human-readable summary of the knowledge learned by the DNN model. SMINT allows users to navigate from a global view into a local view to obtain interpretations within supernodes. This supports a few important use-cases.

SMINT also uses perturbation-based local interpretations [52] to identify key features for each training point in the supernodes. As shown in Figure 12, the data points of different classes are distinguished by these key features. Between mashed potatoes and meatloaf, potato and meat are marked, respectively. For fountain pens and microphones, we can easily infer that the nib of pens and head of the microphones are the key features to differentiate those two classes. One may notice that the first image in the supernode "meatloaf" appears twice. This is because that there are multiple objects in the image. The image is, therefore, duplicate with different labels. Generally, one may detect conflicting labels of training points with supernode. Conflicting labels for the duplicate training data are harmful to the model training. The capability of detecting conflicting labels with the boundary graph is helpful to re-assess the model performance, particularly when labels are interchangeable. For the ImageNet dataset trained on InceptionResNet2, SMINT detects 204 pairs of conflicting labels using the boundary graph. Most of are interchangeable. For example, identical image "n04152593_23560" and "n03782006_3422" are labeled as "screen" and "monitor," respectively. "n02979186_1542" and "n04392985_1095" are identical but labeled as a cassette player and tape player, respectively. The actual accuracy of the model should be higher as a result. This confirms the findings that the accuracy of the DNN model is underestimated [58].

Misclassified data points are often of the interest for interpretation. The boundary graph is able to identify over 98.2% test errors through the local decision boundaries. When two supernodes belonging to different classes share an edge, most test errors are presented as displacement of data points in these two supernodes.

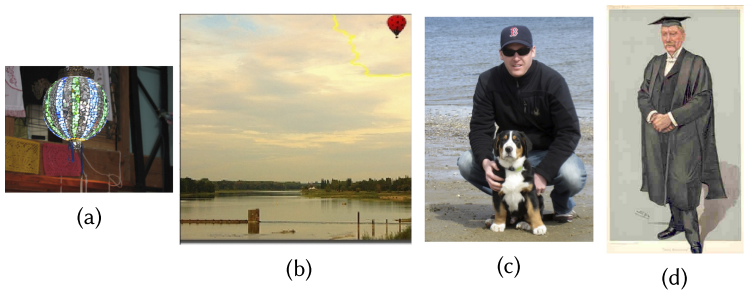


Fig. 13. (a) A misclassified "lampshade" and its neighbor sample (b) in the boundary graph. (c) A greater Swiss mountain dog misclassified as an Appenzeller.

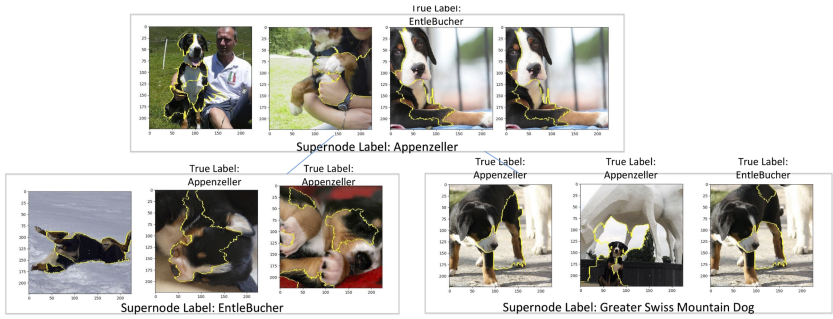


Fig. 14. Local neighboring supernodes for testing point in Figure 13(c).

Some misclassifications happen due to the existence of similar patterns. For example, a lampshade (see Figure 13(a)) is misclassified as a balloon with confidence of 0.9857. One example image of its neighbors in the graph is shown in Figure 13(b).

Some misclassifications are due to mislabelling and ambiguous concepts. For example, a greater Swiss mountain dog (see Figure 13(c)) is misclassified as Appenzeller. The neighboring supernodes of the point are shown in Figure 14. One may notice that the local decision boundaries between Appenzeller, Entlebucher, and greater Swiss mountain dog are tangled. For example, the third image in the supernode of Appenzeller appears twice and is labeled both as Entlebucher and Appenzeller. Dual labeling also happens for the first image in the supernode of greater Swiss mountain dog. The lengths of edges between those supernodes are much shorter than the average length of all edges. This also indicates unclear decision boundaries.

There were also misclassifications that happen due to the existence of multi-objects. As shown in Figure 15, the local decision boundary between the class mortarboard and academic gown is ambiguous as both objects appear in most of the images in these two classes. Also, double-labeling exists for the third (fourth) image in the supernode of the academic gown. The second image for the supernode mortarboard is due to overfitting. Even though the mortarboard exists in the original image, it has been cropped while feeding into the model. The local features highlighted on the image also demonstrates this, since none of the features makes sense to the prediction of the mortarboard. We also confirmed this by using the influence of this training point by influence function [31]. The test image (see Figure 13(d)) has the true label mortarboard. However, it is misclassified as an academic gown. We noticed that the cropped image also removed the mortarboard, thus making the mortarboard features lost in the image. When we randomized the crop configurations to keep the mortarboard, the score for class "mortarboard" increases from 0.072 to 0.583.

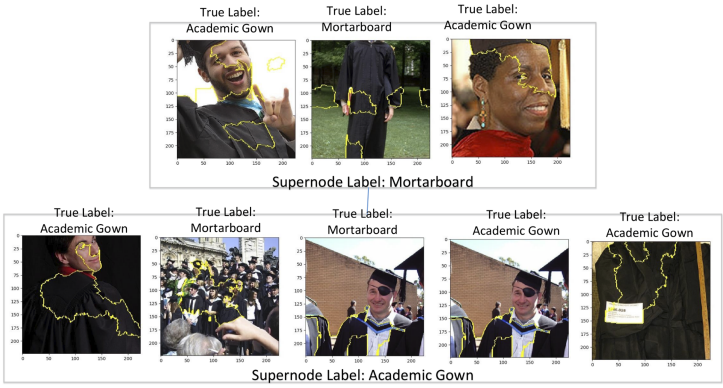


Fig. 15. Local neighboring supernodes for testing point in Figure 13(d).

Table 2. The Comparison of Intepretation Satisfactory Score among MMD-critic, EB-tree, and the Boundary Graph

	Method	Correctly Classified	Misclassified	Time
MNIST	MMD-critic	7.57 (SD = 0.43)	7.22 (SD = 0.59)	19.1s
	EB-tree	7.69 (SD = 0.50)	7.38 (SD = 0.65)	5.8s
	Boundary Graph	7.92 (SD = 0.37)	7.81 (SD = 0.47)	8.7s
ImageNet	MMD-critic	6.86 (SD = 0.59)	5.12 (SD = 0.77)	75.9s
	EB-tree	6.98 (SD = 0.63)	5.36 (SD = 0.85)	14.5s
	Boundary Graph	7.82 (SD = 0.41)	6.91 (SD = 0.53)	21.2s

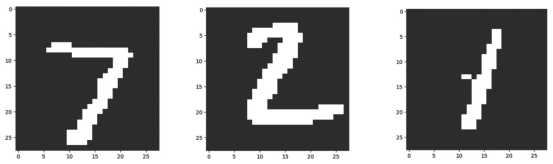
To further verify the level of interpretability of the boundary graph, we conducted a human study to compare it with EB-tree and MMD-critic. The study involves 20 users with only basic machine learning knowledge (i.e., no machine learning research or development experience). We randomly choose 50 images in both MNIST and ImageNet datasets. The participants are asked to give a score (1–10), which measures how well they feel the interpretation method helps them to find the key features for the prediction. This, in essence, is to evaluate which representation of explanations provides more information to the users. For prototypes and criticisms in MMD-critic, we get those from the same classes as in the neighborhood of the boundary graph for fair comparison. As shown in Table 2, users prefer the interpretation of the boundary graph. This demonstrates the effectiveness of the supernode-based graph structure and integrated features in the boundary graph. Compared to EB-tree, which does not provide any feature hints, users spend more time on the boundary graph exploring the integrated visualizations.

7.4 Interpretations of Adversarial Examples

We evaluate SMINT's reliability with adversarial examples. For adversarial examples, we set the parameters $\epsilon = 0.3$ for Fast Gradient Sign Method (FGSM) attack [21], which makes the model accuracy on MNIST drop to 0.2410. We set the change made to pixels, $\theta = 1$, and maximum distortion, $\gamma = 0.15$, for Jacobian-based Saliency Map Attack (JSMA) [48] and $c = 10^{-4}$, $i = 100$ for CW-L2 (Carlini-Wagner) attack [12] (a larger c increases the probability of successful attack, but also makes the difference of adversarial examples noticeable). The learning rate is 0.1. We use spatial smoothing squeezer with the window size of 2×2 and 1-bit for color depth squeezer as those parameters turn out to work well in most cases [66].

Table 3. Model Accuracy on Different Datasets and Squeezers

Dataset	No.	Tandom	Smoothing (2 × 2)	Color depth (1-bit)
Normal	0.9914	0.9914	0.9906	0.9911
Noisy	0.1623	0.1689	0.0593	0.1005
Rotation	0.2830	0.2844	0.2803	0.1015
FGSM	0.2410	0.2490	0.3412	0.9260
JSMA	0.0070	0.0070	0.7520	0.1260
CW-L2	0.0000	0.3080	0.8490	0.9850



(a) (b) (c)

Fig. 16. Adversarial examples generated CW-L2 attack.

Once an example is marked with a low interpretation score, SMINT starts the transforming procedure to recover the input. Table 3 shows the model accuracy on different datasets with different squeezers. With the input transforming, there is a high chance that the real label can be recovered if it is an adversarial example. For the benign out-of-distribution examples, the squeezers do not improve but sometimes negatively affect the accuracy. Therefore, after the input transforming, SMINT re-calculates the interpretation score and provides the model users with the interpretation with a higher score. If the interpretation score cannot be improved by transforming, then SMINT does not interpret the prediction further but report the input as an out-of-distribution example.

Figure 16 shows the adversarial examples generated by the CW-L2 attacks. The three examples are misclassified as 9, 1, and 4 with interpretation scores of 0.0198, 0.0435, and 0.0376, respectively. After transforming (one-bit color depth), we have all of them correctly classified and interpreted with interpretation scores of 0.2571, 0.3925, and 0.3780, respectively. This demonstrates the reliability of SMINT on interpreting adversarial examples.

7.5 Applying SMINT beyond Image Data

The idea of the proposed approach is generic and can be applied to different types of data. In the following, we show the use case of SMINT on a text classification task. We use a movie review dataset [45] with one sentence per review. The classification task is to detect positive and negative reviews. We use a convolutional neural network [29] as a base to construct a model. The network contains an embedding layer that transforms the input sentences to tensor with the embedding dimension equal to 256. On top of the embedding layer are three parallel convolutional layers with filter sizes 3, 4, and 5 and maximum pooling layers on each of the convolutional layer. The concatenated output is then fed into the representation layer with 10 fully connected units before the dropout and final softmax layer.

The boundary graph is constructed by extracting the output from the representation layer. Table 4 shows the overall performance of the boundary graph in comparison with the original model.

We show classification examples below.

Sentence “visually striking and viscerally repellent” and sentence “in visual fertility treasure planet rivals the top japanese animations of recent vintage” are classified as positive reviews. They

Table 4. Performance of Boundary Graph on Text

Model	Accu.	F1-score	$\overline{Edge}$	nodeCnt
Original Network	0.7586	—	—	—
Boundary Graph	0.7581	0.9733	0.2760	47

are both linked to the following sentence in the boundary graph: "what jackson has accomplished here is amazing on a technical level." However, the following sentence is wrongly classified as negative by the model: "a terrifically entertaining specimen of spielbergian sci-fi." Its representation is closer to the same sentence in the boundary graph. The three test sentences are all linked to the boundary node in the technical aspect of a movie.

The result shows that our boundary graph is able to mimic model decisions for text data. It captures certain semantic relationships between test data and relevant training data near the decision boundaries even for a binary classification case.

8 CONCLUSION AND DISCUSSION

Model sharing raises new challenges for interpretability. Constraints such as the number of training points to disclose to users and lack of full access to the internal structure of DNN models add difficulties for existing approaches. In this article, we presented SMINT, a framework that uses a new data structure called boundary graph to improve the interpretability for shared DNN models. The boundary graph faithfully characterized the decision boundaries of a DNN model with a small number of training data points. It enabled model users to gain a global view of DNN models and at the same time helped them to gain insight into the decision making of DNN models by presenting the local decision boundaries. The supernodes in the boundary graph embodied key features leading to the decisions. We demonstrated that the boundary graph enabled model users to fine-tune DNN models. To address the unreliability issues of many interpretation methods, we also showed that SMINT could automatically evaluate the reliability of interpretations and correctly interpret adversarial examples. Essentially, the boundary graph is a powerful data structure for mimicking DNN models and providing interpretability for shared models.

REFERENCES

[1] Agnar Aamodt and Enric Plaza. 1994. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Commun.* 7, 1 (1994), 39–59.

[2] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. 2018. Sanity checks for saliency maps. In *Advances in Neural Information Processing Systems*. 9505–9515.

[3] David Alvarez-Melis and Tommi S. Jaakkola. 2018. On the robustness of interpretability methods. https://arxiv.org/pdf/1806.08049.pdf.

[4] Amazon. 2017. Machine Learning on AWS. Retrieved July 7, 2017 from https://aws.amazon.com/machine-learning/.

[5] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. 2015. Practical and optimal LSH for angular distance. In *Advances in Neural Information Processing Systems*. 1225–1233.

[6] Alexandr Andoni and Ilya Razenshteyn. 2015. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing*. ACM, 793–801.

[7] Franz Aurenhammer. 1991. Voronoi diagrams: A survey of a fundamental geometric data structure. *ACM Comput. Surv.* 23, 3 (1991), 345–405.

[8] Jacob Bien and Robert Tibshirani. 2011. Prototype selection for interpretable classification. *Ann. Appl. Stat.* 5, 4 (2011), 2403–2424.

[9] Jean-Daniel Boissonnat, Olivier Devillers, and Monique Teillaud. 1993. A semidynamic construction of higher-order voronoi diagrams and its randomized analysis. *Algorithmica* 9, 4 (1993), 329–356.

[10] Olcay Boz. 2002. Extracting decision trees from trained neural networks. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 456–461.

[11] Nicholas Carlini and David Wagner. 2017. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. ACM, 3–14.
[12] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP'17)*. IEEE, 39–57.
[13] Zhengping Che, Sanjay Purushotham, Robinder Khemani, and Yan Liu. 2016. Interpretable deep models for icu outcome prediction. In *Proceedings of the AMIA Annual Symposium Proceedings*, Vol. 2016. American Medical Informatics Association, 371.
[14] Daqing Chen and Phillip Burrell. 2001. Case-based reasoning system and artificial neural networks: A review. *Neur. Comput. Appl.* 10, 3 (2001), 264–276.
[15] Marvin S. Cohen, Jared T. Freeman, and Steve Wolf. 1996. Metarecognition in time-stressed decision making: Recognizing, critiquing, and correcting. *Hum. Factors* 38, 2 (1996), 206–219.
[16] Mark Craven and Jude W. Shavlik. 1996. Extracting tree-structured representations of trained networks. In *Advances in Neural Information Processing Systems*. 24–30.
[17] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the 20th Annual Symposium on Computational Geometry*. ACM, 253–262.
[18] Ruth Fong and Andrea Vedaldi. 2017. Interpretable explanations of black boxes by meaningful perturbation. In *Proceedings of the IEEE International Conference on Computer Vision*. 3429–3437.
[19] Nicholas Frosst and Geoffrey Hinton. 2017. Distilling a neural network into a soft decision tree. https://arxiv.org/pdf/1711.09784.pdf.
[20] Amirata Ghorbani, Abubakar Abid, and James Zou. 2019. Interpretation of neural networks is fragile. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 3681–3688.
[21] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. https://arxiv.org/abs/1412.6572.pdf.
[22] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. In *Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'13)*. IEEE, 6645–6649.
[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.
[24] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, et al. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Sign. Process. Mag.* 29, 6 (2012), 82–97.
[25] Qiang Huang, Jianlin Feng, Qiong Fang, Wilfred Ng, and Wei Wang. 2017. Query-aware locality-sensitive hashing scheme for lp norm. *VLDB J.* 26, 5 (2017), 683–708.
[26] Mayank Kabra, Alice Robie, and Kristin Branson. 2015. Understanding classifier errors by examining influential neighbors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3917–3925.
[27] Been Kim, Rajiv Khanna, and Oluwasanmi O. Koyejo. 2016. Examples are not enough, learn to criticize! criticism for interpretability. In *Advances in Neural Information Processing Systems*. 2280–2288.
[28] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, et al. 2018. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *Proceedings of the International Conference on Machine Learning*. 2673–2682.
[29] Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP'14)*. Association for Computational Linguistics, 1746–1751. DOI: 10.3115/v1/D14-1181
[30] Pieter-Jan Kindermans, Sara Hooker, Julius Adebayo, Maximilian Alber, Kristof T. Schütt, Sven Dähne, Dumitru Erhan, and Been Kim. 2017. The (Un) reliability of saliency methods. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Springer, 267–280.
[31] Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning*, Vol. 70. JMLR. org, 1885–1894.
[32] J. Kolodner. 2014. Selecting the best case for a case-based reasoner. In *Proceedings of the 11th Annual Conference Cognitive Science Society Pod.* 155–162.
[33] Janet L. Kolodner. 1992. An introduction to case-based reasoning. *Artif. Intell. Rev.* 6, 1 (1992), 3–34.
[34] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet classification with deep convolutional neural networks. *Commun. ACM* 60, 6 (May 2017), 84–90. DOI: https://doi.org/10.1145/3065386
[35] Brian Kulis and Kristen Grauman. 2009. Kernelized locality-sensitive hashing for scalable image search. In *Proceedings of the 2009 IEEE 12th International Conference on Computer Vision*. IEEE, 2130–2137.
[36] Der-Tsai Lee and Bruce J. Schachter. 1980. Two algorithms for constructing a Delaunay triangulation. *International J. Comput. Inf. Sci.* 9, 3 (1980), 219–242.

[37] Oscar Li, Hao Liu, Chaofan Chen, and Cynthia Rudin. 2018. Deep learning for case-based reasoning through prototypes: A neural network that explains its predictions. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[38] Xingjun Ma, Bo Li, Yisen Wang, Sarah M. Erfani, Sudanthi Wijewickrema, Michael E. Houle, Grant Schoenebeck, Dawn Song, and James Bailey. 2018. Characterizing adversarial subspaces using local intrinsic dimensionality. *Proceedings of the International Conference on Learning Representations (ICLR'18)*.

[39] Charles Mathy, Nate Derbinsky, José Bento, Jonathan Rosenthal, and Jonathan S. Yedidia. 2015. The boundary forest algorithm for online supervised and unsupervised learning. In *Proceedings of the AAAI Conference on Artifical Intelligence (AAAI'15)*. 2864–2870.

[40] Microsoft. 2017. Microsoft Azure Machine Learning Studio. Retrieved July 7, 2017 from https://studio.azureml.net/.

[41] Tim Miller. 2018. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence* 267 (2019), 1–38.

[42] Allen Newell, Herbert Alexander Simon, et al. 1972. *Human Problem Solving*. Vol. 104. Prentice-Hall, Englewood Cliffs, NJ.

[43] Richard Nock, Marc Sebban, and Didier Bernard. 2003. A simple locally adaptive nearest neighbor rule with Aapplication to pollution forecasting. *Int. J. Pattern Recogn. Artif. Intell.* 7, 8 (2003), 1369–1382.

[44] Augustus Odena, Catherine Olsson, David Andersen, and Ian Goodfellow. 2019. TensorFuzz: Debugging neural networks with coverage-guided fuzzing. In *International Conference on Machine Learning*. 4901–4911.

[45] Bo Pang and Lillian Lee. 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 115–124.

[46] Nicolas Papernot, Fartash Faghri, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Alexey Kurakin, Cihang Xie, Yash Sharma, Tom Brown, Aurko Roy, Alexander Matyasko, Vahid Behzadan, Karen Hambardzumyan, Zhishuai Zhang, Yi-Lin Juang, Zhi Li, Ryan Sheatsley, Abhibhav Garg, Jonathan Uesato, Willi Gierke, Yinpeng Dong, David Berthelot, Paul Hendricks, Jonas Rauber, and Rujun Long. 2018. Technical report on the CleverHans v2.1.0 adversarial examples library. https://arxiv.org/pdf/1610.00768.pdf.

[47] Nicolas Papernot and Patrick McDaniel. 2018. Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. https://arxiv.org/abs/1803.04765.pdf.

[48] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *Proceedings of the 2016 IEEE European Symposium on Security and Privacy (EuroS&P'16)*. IEEE, 372–387.

[49] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Deepxplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 1–18.

[50] Google Cloud Platform. 2018. Cloud Machine Learning Engine. Retrieved February 1, 2018 from https://cloud.google.com/ml-engine/.

[51] Jim Prentzas and Ioannis Hatzilygeroudis. 2009. Combinations of case-based reasoning with other intelligent methods. *Int. J. Hybrid Intell. Syst.* 6, 4 (2009), 189–209.

[52] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. 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*. ACM, 1135–1144.

[53] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet large scale visual recognition challenge. *Int. J. Comput. Vis.* 115, 3 (2015), 211–252. DOI: https://doi.org/10.1007/s11263-015-0816-y

[54] Gregor P. J. Schmitz, Chris Aldrich, and Francois S. Gouws. 1999. ANN-DT: An algorithm for extraction of decision trees from artificial neural networks. *IEEE Trans. Neur. Netw.* 10, 6 (1999), 1392–1401.

[55] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. 2017. Grad-CAM: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the International Conference on Computer Vision (ICCV'17)*. 618–626.

[56] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.

[57] J. Springenberg, Alexey Dosovitskiy, Thomas Brox, and M. Riedmiller. 2015. Striving for simplicity: The all convolutional net. In *Proceedings of the International Conference on Learning Representations (ICLR'15)*.

[58] Pierre Stock and Moustapha Cisse. 2017. Convnets and imagenet beyond accuracy: Explanations, bias detection, adversarial examples and model criticism. https://arxiv.org/abs/1711.11443.pdf.

[59] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, Vol. 70. 3319–3328.

[60] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi. 2017. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'17)*, Vol. 4. 12.

[61] Tensorflow. 2017. Deep MNIST for Experts. Retrieved July 7, 2017 from https://www.tensorflow.org/get_started/mnist/pros.

[62] Csaba D. Toth, Joseph O'Rourke, and Jacob E. Goodman. 2004. *Handbook of Discrete and Computational Geometry*. CRC Press.

[63] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. 2016. Stealing machine learning models via prediction apis. In *Proceedings of the USENIX Security Conference*.

[64] Huijun Wu, Chen Wang, Jie Yin, Kai Lu, and Liming Zhu. 2018. Sharing deep neural network models with interpretation. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 177–186.

[65] Mike Wu, Michael C. Hughes, Sonali Parbhoo, Maurizio Zazzi, Volker Roth, and Finale Doshi-Velez. 2018. Beyond sparsity: Tree regularization of deep models for interpretability. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[66] Weilin Xu, David Evans, and Yanjun Qi. 2017. Feature squeezing: Detecting adversarial examples in deep neural networks. *Proceedings of the Network and Distributed System Security Symposium (NDSS'18)*.

[67] Pierre Stock and Moustapha Cisse. 2018. Convnets and imagenet beyond accuracy: Understanding mistakes and uncovering biases. In *Proceedings of the European Conference on Computer Vision (ECCV'18)*. 498–512.

[68] Matthew D. Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*. Springer, 818–833.

[69] Quanshi Zhang, Ying Nian Wu, and Song-Chun Zhu. 2018. Interpretable convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 8827–8836.

Received April 2019; revised October 2019; accepted February 2020