Sharing Deep Neural Network Models with Interpretation

Huijun Wu
UNSW and Data61, CSIRO
Sydney, Australia
huijunw@cse.unsw.edu.au

Chen Wang
Data61,CSIRO
Sydney, Australia
chen.wang@data61.csiro.au

Jie Yin
The University of Sydney
Sydney, Australia
jie.yin@sydney.edu.au

Kai Lu
NUDT
Changsha, China
kailu@nudt.edu.cn

Liming Zhu
UNSW and Data61, CSIRO
Sydney, Australia
limingz@cse.unsw.edu.au

ABSTRACT

Despite outperforming humans in many tasks, deep neural network models are also criticized for the lack of transparency and interpretability in decision making. The opaqueness results in uncertainty and low confidence when deploying such a model in model sharing scenarios, where the model is developed by a third party. For a supervised machine learning model, sharing training process including training data is a way to gain trust and to better understand model predictions. However, it is not always possible to share all training data due to privacy and policy constraints. In this paper, we propose a method to disclose a small set of training data that is just sufficient for users to get the insight into a complicated model. The method constructs a boundary tree using selected training data and the tree is able to approximate the complicated deep neural network models with high fidelity. We show that data point pairs in the tree give users significantly better understanding of the model decision boundaries and paves the way for trustworthy model sharing.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; • **Software and its engineering** → *Open source model*;

KEYWORDS

Deep Neural Networks, Model Sharing, Interpretability, Decision Boundary

ACM Reference Format:
Huijun Wu, Chen Wang, Jie Yin, Kai Lu, and Liming Zhu. 2018. Sharing Deep Neural Network Models with Interpretation. In *WWW 2018: The 2018 Web Conference, April 23–27, 2018, Lyon, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3178876.3185995

1 INTRODUCTION

Complicated machine learning models like deep neural networks (DNN) have achieved great success in image classification [11, 18], speech recognition [10, 12], and classic games such as Go [29] 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 efforts of tuning. Sharing a trained model is cost-effective. As shown in Figure 1, machine learning models can be hosted in the cloud and run as a service in a Pay-As-You-Go model. The model developers train models using the data they collect. They may use machine learning as service platforms (MLaaS) such as Google CloudML [23], AmazonML [2] or Microsoft AzureML [21] to develop models and manage training data in the cloud. The application developers may then integrate these models into their applications through prediction APIs. However, a complicated machine learning model, particularly a DNN model remains a black-box and the model quality is difficult to assess, especially when the distribution of application data differs from that of the training data. Commonly used accuracy and confidence values are not sufficient to reveal model behaviors on the data collected from different sources. A simple example is that the confidence value of classifying a data point belonging to an unseen class by the model can be high (see Fig. 13). MLaaS platforms do not have effective means to address this problem, thus making it challenging for users to trust a shared model.

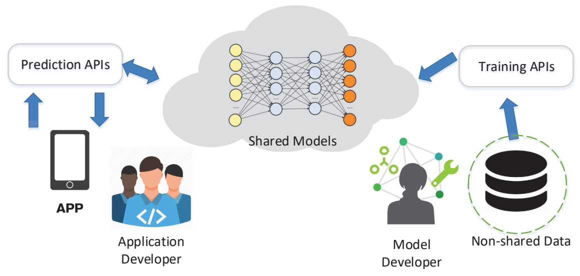


Figure 1: The Model Sharing Scenario: A model user calls prediction APIs to get predictions of her/his input data. The prediction is based on deep neural network models developed by the third party model developers on the data invisible to the model user.

Providing means to interpret a model is effective to improve model transparency and help model users to understand the potential weakness of the models on their data. There has been a growing interest in exploring the interpretability for DNNs.

A common approach is to find examples or prototypes to guide decision making, which is along the line of case-based reasoning [1].

This approach identifies training data points that are close to data points within their own classes and far away from those in different classes [5]. Prototype finding is solved as a set cover optimization problem in [5]. This approach is complemented by criticism finding as prototypes give little information to interpret the data points that do not fit a model well. MMD-critic [16] intends to find outliers in a class that differ the most to other data points belonging to the same class. These outliers are called criticisms. However, criticisms do not contain sufficient information to give users hints about why a model classifies a data point into one class, not another. Without information about similar data points in the neighboring classes, it is still difficult for a user to understand crucial model predictions around class boundaries. In other words, characterizing the differences between classes is able to explain these predictions.

Another approach is to mimic a complicated model using an interpretable simple model such as decision tree [6–8, 28]. The limitation of this approach is that it often requires structured training data. When the structure of data is complicated, a decision tree itself can be difficult to interpret.

Visualization is also used to examine features extracted by hidden neurons in DNNs for users to infer the relationship between a classification decision and these features [15, 35, 36]. The sample perturbation approach [9, 24] also attempts to understand features leading to certain predictions in image classification. The basic idea is to learn an image perturbation mask that minimizes a class score. The computational cost for perturbation based methods is non-trivial. Moreover, these methods rely on the change of confidence values to infer the influence of a changing individual feature. The change in confidence values may not accurately reflect the influence of features on predictions when there is a *concept drift* in data, which can be particularly common in model sharing or open set classification scenarios [27]. Moreover, in model sharing scenarios, model users may not have the access to the internal structure of a model [34].

Along a similar line, [17] proposes to use influence functions to identify the most important training data leading to certain predictions. This method requires the access to the whole training dataset, which is unrealistic in the model sharing settings.

In this paper, we propose a method to enhance users' understanding of a shared DNN model by providing a small set of training data that characterizes the model decision boundaries. These data points are informative for users to infer how a prediction is made in relation to them. Our method is relatively simple yet effective. Specifically, we employ a max-margin based approach to select the most representative training data that largely contributes to the forming of the decision boundaries of a DNN model. These training data points are organized via an Explicable Boundary Tree (EB-tree) based on the distances in the DNN transformed space. The EB-tree data structure embodies DNN decision boundaries. Thus, traversing data points in the EB-tree is able to approximate the predictions of the DNN model on these points with high fidelity.

EB-tree has the following advantages: First, it extracts training data points that characterize decision boundaries so that it is able to explain why a test data point is classified into one class, not the other through the traversal of the test point along training data pairs with different labels in the tree. These pairs reveal the difference between a test data point to representative training data belonging to different classes to understand how the classification decision is made by the DNN model. Second, interpreting classification results by traversing the EB-tree does not require the access of the internal structure of the DNN model, thus suitable for model sharing scenarios. Third, EB-tree is computationally efficient since it only needs to pass the training data once, which makes it scalable for dealing with large models.

Our experimental results show that EB-tree is able to achieve a high fidelity to the DNN model by only disclosing a small set of training data. In addition, we show through a human pilot study that the traversal process of a test point in the tree clearly improves model users' understanding about how predictions are made by DNN models. Compared to methods like MMD-critic, the boundary traversal in the tree can help users to better understand the classification results. We also demonstrate that in addition to giving model users insight into DNN model decision boundaries, EB-tree can also be used for identifying mislabeled training data and improving the efficiency of emerging new class detection, both of which are useful for understanding shared models.

The rest of the paper is organized as follows: Section 2 describes the decision boundary of a DNN model and the process of constructing an explicable boundary tree (EB-tree) associated with a shared deep learning model; Section 3 gives algorithms to approximate a model using EB-tree; Section 4 presents evaluation results and Section 5 concludes the paper.

2 EXPLICABLE BOUNDARY TREE

2.1 Interpretable Decision Boundaries

To interpret a DNN model, one effective way is to link the numerical values in a prediction made by the model to the training data related to this prediction. This process characterizes model decision boundaries using these training data points and ensures these boundaries are consistent with model predictions. We call the boundaries *interpretable decision boundaries.*

In the DNN transformed space, the interpretable decision boundaries formed by the training data points with different labels can be computed using a Voronoi diagram [4]. Specifically, let $P = \{P_k | k \in K\}$ denote a set of distinct points in space X, 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 how an interpretable boundary between two classes in a DNN transformed space is characterized. As shown in Figure 2, training data points in grey and white region have different labels. We build a Voronoi diagram for all these points (K equals to the total number of training data points) through Delaunay triangulation [19], shown by orange lines in Figure 2. 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 boundary between the two classes is a set of connected edges shared by the neighboring regions of different classes. To gain a different level of insight into the DNN model, the interpretable decision boundaries can be constructed in the latent or the output layer of the DNN transformed space.

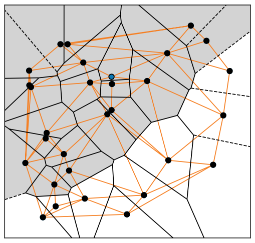


Figure 2: An interpretable decision boundary constructed using a Voronoi diagram.

However, computing Delaunay triangulation on n points in $\mathbb{R}^d$ has the time complexity of $O(n^{\lceil d/2 \rceil})$ [33]. It scales poorly with the increase of dimensions (d) and takes a long time if not impossible to compute for a moderate dataset. To address this problem, we give a method called Explicable Boundary Tree (EB-tree) to approximate interpretable decision boundaries (referred as decision boundaries for simplicity in the rest of the paper).

2.2 Boundary Tree

We use edges that cross the boundaries and their associated nodes in the Delaunay triangulation to approximate the decision boundaries. The boundary tree is a data structure to achieve this. The boundary tree (forest) algorithm [20] is 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 closest node x to y and uses the label of x to predict the class of y. If x has the same label with y, y is discarded. Otherwise, it is added to the tree. The process repeats for each training data point. A test data point is classified by the label of its closest 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. Similar to the edges that cross the decision boundary in Delaunay triangulation, the difference between two end nodes of an edge on the boundary tree can, therefore, serve as a local explainer to a prediction close to the boundary.

2.3 Explicable Boundary Tree

Even though its edges provide certain hints about the decision boundary between two classes, a basic boundary tree has three main limitations to support the interpretability of a shared DNN model: firstly, the tree may have relatively low fidelity to the DNN model in decision making; secondly, the number of training data points in the tree is not optimized and there is plenty of room to reduce the number to avoid unnecessary training data disclosure; thirdly, the training data point selection does not characterize a boundary clearly as two data points far away from the boundary may be connected by an edge. Some recent work [37] intends to further learn a distance metric of boundary tree edges using DNN. Specifically, for a given boundary tree structure, [37] provides a DNN to transform the data into a boundary tree-friendly representation. Note that the DNN used in [37] is not optimized for classification directly. By contrast, for a given DNN optimized for classification, EB-tree aims to provide a boundary tree that approximates the decision boundaries of the DNN classifier accurately with a small number of training data points. EB-tree intends to address these limitations of the basic boundary tree.

Figure 3 shows the architecture of EB-tree consisting of a DNN classifier f and an optimized boundary tree. The model translation module is responsible for constructing a boundary tree T to mimic the decision making of the DNN classifier. Each node of T is a training data point that is close to the decision boundary in the training dataset D. The DNN classifier transforms training data $t \in D$ to representation $f(t)$, which is a transformed feature vector. The features can be derived from different hidden layers or the output layer. We use the output of *softmax* layer in the DNN as the transformed representation of a training data point. The Euclidean distance is used to measure the distance between $f(t1)$ and $f(t2)$.

EB-tree selects a small portion of training data points that are close to the decision boundaries to approximate a DNN model. These data points are helpful for model users to gain insight into the key differences between different classes. For a test data point, traversing the tree to reach its closest training point pairs provides an interpretation of the decision choice of the model. To construct an EB-tree with good interpretability, it is important to ensure that the distance between two nodes connected by an edge is short. As an edge crosses the boundary, a short edge approximates the boundary with a narrow margin and gives a model user better visualization of the boundary. The vanilla boundary tree is trained in an online manner and it is impossible to change the order of the nodes fed into the tree. EB-tree, however, can leverage a carefully-designed training order to improve its ability to mimic the decision boundary of a DNN model. Specifically, we give a training data ordering algorithm for better boundary characterization in the following section. The algorithm produces high accuracy and fidelity to the DNN model.

Once constructed, the EB-tree is able to answer queries of test data points from a model user as follows: A query sample y is firstly transformed by the DNN model to $f(y)$; then a traversal process locates the closest node to $f(y)$ in the tree, denoted by x; finally, the label of x is used to predict the class of y. The traversal path is used as an explainer of the prediction.

3 MODEL TRANSLATION

The model translation module in Figure 3 is responsible for identifying the most representative training data that characterizes the decision boundaries of a DNN model and constructing an EB-tree to approximate the decision boundaries.

The basic boundary tree algorithm is not able to achieve sufficient fidelity as shown in our experiment (Table 1), neither does it optimize the size of the tree, which may lead to unnecessary training data disclosure. We address this problem by training point reordering and boundary stitching described as below.

3.1 Training Data Point Reordering

Figure 4 gives an example to illustrate the training point selection problem. Node B may add either node A or A' as its child. The choice leads to different classification results for query Q. Specifically,

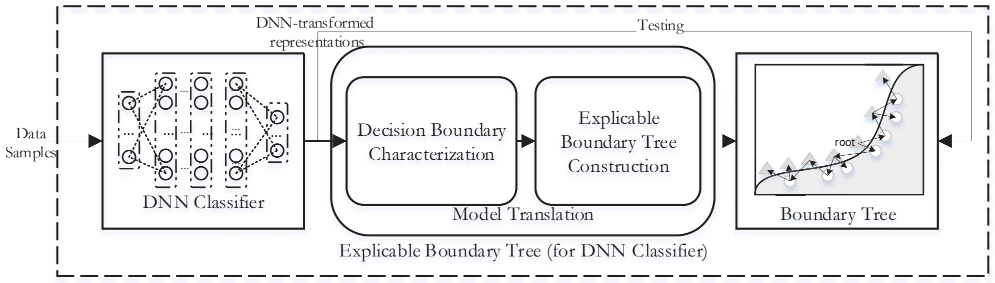

DNN-transformed representations
Testing
Data Samples
DNN Classifier
Decision Boundary Characterization
Explicable Boundary Tree Construction
Model Translation
root
Boundary Tree
Explicable Boundary Tree (for DNN Classifier)

Figure 3: Architecture of Explicable Boundary Tree: A model user calls the prediction APIs to calculate the DNN transformed representations of input data. The representations traverse the boundary tree for predictions and interpretations.

when $A^{'}$ is selected, query of test data point Q is classified as class A because it is closer to node A (on the left of dash line $l^{'}$); otherwise, it is misclassified because Q is closer to node B than A.

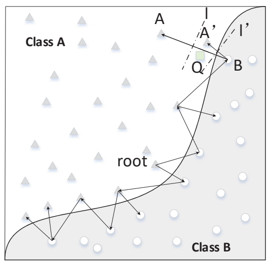


Figure 4: A boundary tree for a two-class classification problem. A and $A^{'}$ are two possible children for node B. Q is a test point.

Selecting training data pairs close to the boundary from a different side and close to each other reduces the probability a test data point is misclassified, thereby enabling the constructed tree to achieve high fidelity to the DNN model. In order to do this, we need to derive a metric to characterize the distance between a data point and a boundary.

EB-tree uses a support vector machine to measure the distance. Since the DNN-transformed space can often be considered linearly separable, we first use a support vector machine to get the decision boundaries. Let w denote a vector orthogonal to a decision boundary, b denote a scalar "offset" and $\{x_i, y_i\}$ denote the DNN transformed representations of training points and the corresponding predicted label given by the DNN model. Given a set of training data points x, the decision boundary of two classes can be represented as

$$w^T x + b = 0. \tag{1}$$

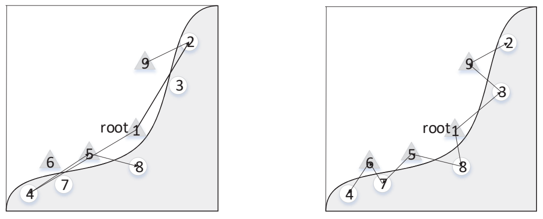
The margin from the boundary to the nearest data points on each side of the boundary satisfies

$$(w^T x_i + b) \cdot y_i \geq 1. \tag{2}$$

The margin between two classes is, therefore, $d = \frac{2}{||w||}$. We identify a boundary that maximizes the margin between two classes, which is equivalent to minimizing $||w||$. We use the one-vs-all scheme [25] to obtain a boundary for each class based on max-margin. For each training data point (x, y) in the DNN transformed space, we compute its minimal distance to the decision boundary of its corresponding class. Training points are sorted in ascending order according to their distances to boundaries.

The EB-tree construction process fetches sorted data points to insert into the tree. The order has a significant impact on both node number and interpretability of the boundary tree constructed. Consider that training points are randomly included in the tree, a training data point that is far away from a decision boundary is likely to be added to the tree first. This may result in the discard of the subsequent adjacent data points that are closer to the boundary because they share the same label of the node already in the tree. Our experiments show that random order tends to include many nodes in the tree but achieving sub-optimal model mimicking performance.

On the interpretability aspect, randomly ordered data points are likely to include many long edges in the boundary tree, which make the feature or visual difference between a parent and a child node difficult to infer. In contrast, an ascending order can keep the data points near a boundary in the constructed tree, meanwhile avoiding the inclusion of long edges. It is also likely to discard data points far away from boundaries. The classification of these data points is often consistent with human intuition and including them in the tree does not contribute to boundary characterization much. This approach helps to minimize the training data disclosure.



(a) decision distance increasing order. **(b) boundary stitching order.**

Figure 5: The comparison of distance increasing order and boundary stitching order on constructed boundary trees: the numbers on nodes follow the increasing order of the distances to decision boundaries.

3.2 Boundary Stitching Algorithm

Simply inserting training data points according to the ascending order of their distances to boundaries is not sufficiently effective for constructing a boundary tree with good interpretability. It is mainly due to that two training points with similar distances to the boundary may be located at different ends of the boundary and far away from each other. The features of such two points are unlikely to share sufficient commonality for a model user to understand the decision boundary. Figure 5a illustrates the case with an example, in which data points like node 3, 6 and 7 are not included in the tree because their corresponding closest nodes with the same label but closer to the boundary are already in the tree at the time they are being processed. In the following, we give a boundary stitching algorithm to address this problem and the algorithm aims o construct a tree well approximating the boundary as illustrated in Figure 5b.

The boundary stitching algorithm (Algorithm 1) considers the distances between the current node in the tree and the candidate points to insert. It prioritizes candidate points with a different label based on their distances to the current node in the tree. The algorithm first computes the DNN transformed representations of training data points and their distances to the boundaries via max-margin. It then sorts these data points according to the ascending order of their distances to the boundaries. The construction of the boundary tree starts from the node with the shortest distance to a boundary. The node is inserted into the tree as the root. A search for k-nearest neighbors (kNNs) of the newly added node in the tree is then performed. If there is a neighboring data point belonging to a different class compared with the current node, the closest one to the current node is selected to insert into the tree. The insertion process traverses the selected data point to its closest node. If the data point has a different label with its closest node in the tree, it is inserted as a child of this node; otherwise, the data point is discarded as a similar node belonging to the same class has already been added to the tree. The process continues until all data points are processed.

Finding k nearest neighbors for massive high-dimensional data incurs high computing cost. Hence, we use *locality sensitive hashing* (LSH) [3] to reduce the cost. For an n-point dataset in d-dimensional space, doing so achieves query computing complexity of $O(dn^{\rho+O(1)})$ where $\rho = \frac{1}{2c^2-1}$ for the Euclidean distance and approximation $c > 1$.

Algorithm 1: Boundary Stitching Algorithm

```
Input: R: A list of the DNN transformed representations of training
       points and corresponding DNN predictions.
Output: The EB-tree T for the DNN model.
1  Procedure EB-tree Construction()
2      //sort points according to distances to boundaries
3      Q = sortedByDistanceToBoundaries(R)
4      current = Q.removeFirst()
5      T.insert(null, current) // insert root
6      while Q is not empty do
7          child = getCandidate(current,Q,T, k)
8          // traverse T to find the closest node to child
9          parent = findParent(T, child)
10         if parent.label ≠ child.label then
11             T.insert(parent, child)
12             current = child
13         end
14     end
15 Function getCandidate(current, Q, T, k)
16     currentIndex = Q.index(current)
17     //find k nearest candidates for the head of Q by LSH
18     kNearests = LSH[currentIndex/($\frac{N}{n}$)].query(current, k)
19     if kNearests is not empty then
20         foreach $n \in kNearests$ do
21             if n.label ≠ current.label then
22                 return n
23             end
24         end
25     end
26     return Q.removeFirst()
```

4 EVALUATION

We evaluate the effectiveness of EB-tree on three image classification tasks. The shared models are a convolutional net (CNN) [32] for MNIST hand-written digit dataset, all convolutional net (All-CNN) [30] for CIFAR-10 dataset and Inception-v3 network [31] for ImageNet dataset [26]. We set k in Algorithm 1 and the maximum number of children for each node in EB-trees based on the number of classes in the datasets. We evaluate our algorithm on the following two aspects: model mimicking performance and the interpretability of the constructed EB-tree. The model mimicking performance is measured by the decision consistency between an EB-tree and its corresponding DNN classifier, or *fidelity*. The metric we use is the *F-measure* of the EB-tree predictions against the DNN model predictions. We measure the interpretability of EB-trees by conducting a human pilot study.

4.1 Model Accuracy and Fidelity

4.1.1 Boundary Tree and EB-tree. As shown by the error rate in Table 1, EB-tree achieves a comparable accuracy of the DNN classifiers. EB-tree also approximates the DNN classifiers with high fidelity (99.90% with MNIST-CNN and 99.12% with CIFAR-ALL-CNN). Expectedly, EB-tree significantly outperforms the original boundary tree algorithm in terms of accuracy and fidelity with a smaller number of nodes in the tree. It is worth noting that, for the MNIST dataset, the resulting EB-tree only needs to disclose 21 out of 60,000 training data points (0.035%) to show model users interpretable decision boundaries. For CIFAR-10 dataset, EB-tree only needs to disclose 145 out of the 50,000 training data points (0.29%) to model users. For the ImageNet dataset with 1.28 million images and 1,000 classes, EB-tree achieves a low error rate (22.51%) and a high fidelity (97.05%) with 11,927 training data points to disclose. Note that for an EB-tree with a large number of children at each node (e.g., EB-tree for ImageNet-Inception-v3), traversing the tree may lead to a local closest node, so that the final node might share little similarity with the test data point. The boundary tree algorithm solves this by building a forest. This approach is infeasible for EB-tree as it aims to reduce the number of training data points to disclose. To address this, we allow a global nearest

Table 1: Comparison of boundary tree, EB-tree and the original DNN classifiers. ***SD*** **is standard deviation.**

Dataset	Model	Error Rate	Fidelity	Error Fidelity	Avg. Edge Length	Parameters	# of Nodes
MNIST	CNN	0.53%	100%	-		1.66M	-
	Boundary Tree	0.57%(SD=0.04%)	99.81(SD=0.07%)	92.45% (SD=3.7%)	0.998 (SD=0.023)	-	46(SD=7)
	EB-tree	0.53%	99.90%	94.33%	0.716	-	21
CIFAR-10	ALL-CNN	7.90%	100%	-		1.37M	-
	Boundary Tree	7.87%(SD=0.33%)	98.88%(SD=0.27%)	86.96% (SD = 4.05%)	0.259 (SD=0.055)	-	277(SD=34)
	EB-tree	7.73%	99.12%	93.79%	0.165	-	145
ImageNet	Inception-v3	22.05%	100%			24.7M	-
	Boundary Tree	24.97%(SD=0.51%)	87.82%(SD=4.16%)	66.29% (SD = 7.03%)	0.501 (SD = 0.046)	-	43986(SD=3572)
	EB-tree	22.51%	97.05%	91.87%	0.371	-	11927

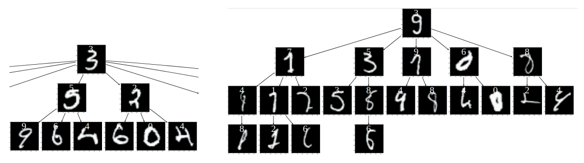
neighbor search when the final node of the traversal path has a different label to the prediction of the test data point.

We further examine classification errors made by EB-tree and the original boundary tree. We measure whether a test data point is likely to be misclassified to the same class as the model. We call this the *error fidelity*, defined as N_c/N_m where N_m is the total number of misclassified test data points and N_c is the number of consistent predictions of the misclassified test data points between the interpretation model and the DNN model. As shown in Table 1, the original boundary tree has a lower error fidelity, indicating it does not mimic the model faithfully on misclassified data points. With a better boundary characterization mechanism, EB-tree outperforms the original boundary tree significantly.

We measure the average edge length of an EB-tree and a vanilla boundary tree. As shown in Table 1, an EB-tree has a much shorter average edge length than a vanilla boundary tree. Figure 6 shows a part of the vanilla boundary tree and a part of the EB-tree built for MNIST-CNN. It is clear that the nodes sharing the same edges in the EB-tree have better visual similarity than those in the boundary tree. An EB-tree characterizes boundaries in a more interpretable manner and is capable of explaining how a data point is classified into one class, but not another by the model.

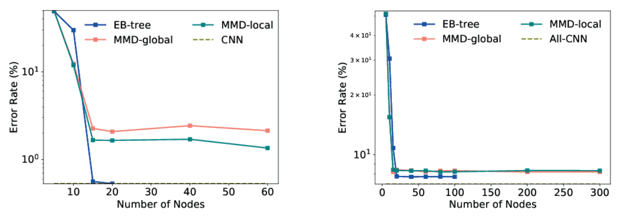
We also measure the EB-tree construction and traversal time. The support vector machines are used to measure the distance to decision boundaries for training data points. For a complex model (Inception-v3) built on a large dataset (ImageNet), the training time for a support vector machine is within 6 hours with 60 Intel Xeon E5 cores and 64GB memory on an HPC cluster. The training time is short compared to that of training the DNN model. Moreover, it takes 47.61s, 35.65s, and 3,812.71s to construct an EB-tree for CNN (MNIST), ALL-CNN (CIFAR-10) and Inception-v3 (ImageNet), respectively. For Inception-v3 (ImageNet), the EB-tree construction time is mainly dominated by execution time of LSH queries. The EB-tree traversal time depends on the depth and number of children nodes. For the above three EB-trees, the average traversal time for a test data point is 0.062s, 0.065s, and 0.194s, respectively.

4.1.2 MMD-critic and EB-tree. We also compare the training data selection performance between EB-tree and MMD-critic. MMD-critic is a representative method for interpreting deep learning models with examples. We first compare the error rates of 1-NN (the nearest neighbor) classifiers built from the MMD-selected prototypes and EB-tree nodes. For MMD-critic, image embeddings before the fully connected layer are used for the prototype selection. As shown in Figure 7, the classifier built on EB-tree selected



(a) Boundary Tree (b) EB-tree

Figure 6: Comparison of the boundary tree built by random training data order and the EB-tree for MNIST-CNN.



(a) MNIST. (b) CIFAR-10.

Figure 7: Error rate vs. number of prototypes (nodes).

data achieves a lower error rate with a significantly smaller number of training data points. This indicates that the EB-tree has a stronger capability of model mimicking and reflects model decision boundaries better.

(a) Prototypes (b) Criticisms

Figure 8: Sample prototypes and criticisms (MNIST-CNN).

Apart from the prototypes, MMD-critic also selects criticisms to assist in interpreting the outlier points as shown in Figure 8. However, MMD-critic does not provide any data structure to help users to explain a prediction. For a large dataset, it is difficult and time-consuming for a user to go through these selected data points one by one to get insight into a prediction. For an EB-tree, similar nodes belonging to different classes are organized as parent-children pairs in the tree. Traversing a test data point along these pairs is effective

Table 2: The comparison of number of test data points whose supporting training data points can be identified via MMD-critic and EB-tree.

Dataset/Tool	Correctly classified data	Incorrectly classified data	Time/image
MNIST/MMD-critic	21	14	17s
MNIST/EB-tree	17	20	6s
CIFAR-10/MMD-critic	17	13	88s
CIFAR-10/EB-tree	15	18	25s
ImageNet/MMD-critic	19	12	72s
ImageNet/EB-tree	18	17	12s

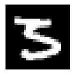
for a user to understand why certain predictions are made by a DNN model. We show our experimental results with human judgments in the following use cases.

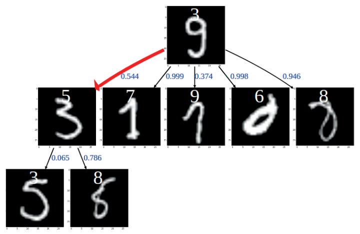
4.2 Use Cases of EB-tree

4.2.1 Case 1: Explaining Predictions of Individual Data Points. To evaluate the effectiveness of an EB-tree in helping a user understanding the decision making of a model, we conduct a human pilot study. We define their effectiveness as follows: a method is effective for assisting a user in interpreting a DNN model when the user is able to identify the data points that support the most likely classes in a prediction. For example, if a test data point is confidently classified into class A, a training data point from class A that is visually similar to the test data point should be identified as the explainer. If the prediction of a test data point is ambiguous between class A and class B, an effective data point selection method should be able to assist the user in identifying training data points belonging to each class to demonstrate the ambiguity.

The study involves 20 users without machine learning research and development experience. We randomly choose 50 test images in each of the three datasets (MNIST, CIFAR-10, and ImageNet). Among each dataset, 25 are chosen from misclassified images while the other 25 are chosen from images that are correctly classified. For each dataset, we provide two means for supporting the interpretation. One is the EB-tree constructed based on the models and the other is the prototypes/criticisms selected by MMD-critic. We design tasks to ask a user to identify "explainers" among MMD-selected training data points or from the EB-tree. To make a fair comparison, the node number of the tree and the data point number of prototypes/criticisms are set to equal. For ImageNet, the computation of prototypes/criticisms overwhelms MMD-critic and we only select the classes related to the classification of the 50 test images. To avoid the interference of the MMD-critic and EB-tree method, we randomly split participants into two groups with one group using MMD-critic method and the other using EB-tree method.

The results of our experiment are shown in Table 2. Overall, users take significantly shorter time to identify supporting training data points in EB-trees than in MMD-critic selected data points. The results on CIFAR-10 dataset are relatively worse than the other two datasets mainly because that the image resolution of the CIFAR-10 dataset is low and participants of the pilot study have difficulty in recognizing some images. For correctly classified test data, MMD-critic exhibits better performance on identifying their supporting training points. For incorrectly classified test data, EB-tree outperforms MMD-critic. As EB-tree mainly selects the boundary nodes,


(a)



(b)

Figure 9: A test image with the label 3 (Figure 9a) in MNIST dataset and its traversal path in the EB-tree (Figure 9b).

correctly classified data points are likely to be further away from these nodes than from prototypes selected by MMD-critic. However, we argue that model interpretability can be better achieved through the understanding of boundaries. Incorrectly classified data points reveal the weakness of a model and a user is able to infer the model decision boundaries through training data points that support the misclassification. The insight into the boundaries is important for model sharing scenarios. On the other hand, users find it straightforward to interpret the correctly classified results even without giving supporting training data.

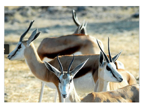
EB-tree also shows model predictions of the training data points in the tree. This reveals a few training data labeling errors and model training errors. To further illustrate the training data points identified in the EB-tree, we show examples of randomly chosen test data points and their corresponding traversal paths in Figure 9, Figure 10 and Figure 11. CNN classifies the "3" in Figure 9a as "5". Its traversal path marked in red is shown in Figure 9b. A traversal path moving from node x_i to node x_j indicates that node x_j is closer to the test point based on the DNN model. One may easily discover that the misclassification is because that the closest boundary node to the test data point is visually similar to "3", but mislabeled as "5".

The "Appenzeller" in Figure 10a is misclassified as 'Greater Swiss Mountain Dog' by Inception-v3. When looking at the traversal path, the dog in the final node is indeed similar to the test data point. The traversal paths of EB-trees indeed provide participants with hints to understand the prediction results.

Moreover, the distances between the test image to the training points on the traversal path can be used to compute the confidence vector for a prediction. For EB-tree, given a test point y, let x_i denote the features of nodes on the traversal path of y in the EB-tree. C_x indicates the label of node x. Hence, the probability that the test point belongs to class c ($p(C_y = c)$) is computed as follows:

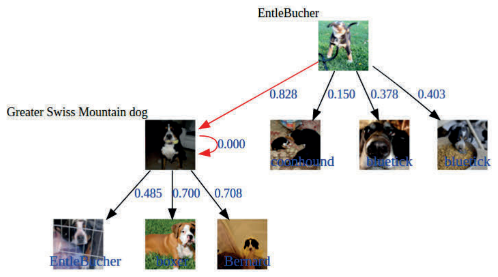
$$p(C_y = C_{x_i}) = \mathrm{SoftMax}(-d(x_i, y)), \tag{3}$$

(a) Test data A

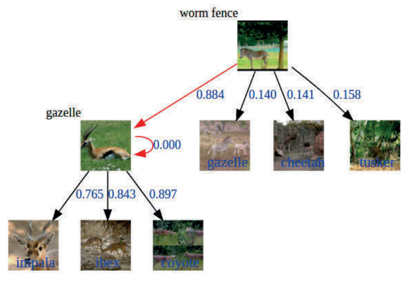

(b) Test data B

(c) Test data C

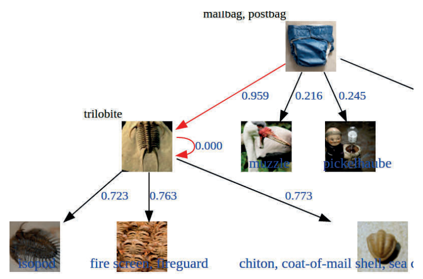
Figure 10: Misclassified test samples in ImageNet.



(a) Traversal path for the sample A.



(b) Traversal path for the sample B.



(c) Traversal path for the sample C.

Figure 11: Sample A (Figure 10a) with label "appenzeller" is misclassified as "greater swiss mountain dog" by Inception-v3. Sample B (Figure 10b) with label "impala, aepyceros melampus" is misclassified as 'gazelle'. Sample C (Figure 10c) with label "microphone" is misclassified as "trilobite".

$$p(C_y = c) = \sum\nolimits_{x_i = c} p(C_y = C_{x_i}), \qquad (4)$$

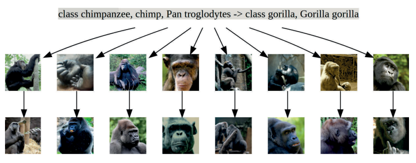
where $d(x, y)$ is the distance between x and y. For the "Appenzeller" example, the confidence vector given by the DNN model is [Greater Swiss Mountain Dog: 0.812, EntleBucher: 0.069, other classes: 0.119]. Following Equation 3 and 4, we obtain the confidence of EB-tree: [Greater Swiss Mountain Dog: 0.891, EntleBucher: 0.073, other classes: 0.0359]. This indicates that the EB-tree makes predictions in a similar way to the original DNN model. For the test data in Figure 10b, nearly all the participants write the interpretation like 'The direction and the profile of the gazelle look much more similar to the test image than the 'impala' (the 1st child of the final node). The test data C in Figure 10c is interesting since before looking at the traversal path, none of the participants have an idea about why the image (true label is microphone) is classified with the confidence vector like ['trilobite':0.603, 'isopod':0.036, ...]. The traversal path clearly gives visual hints about the prediction results; the similar texture of the trilobite and the isopod leads to the misclassification.

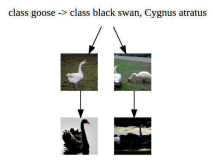
The results indicate that EB-tree is able to give model users explanation about why a data point is classified into a particular class, not other classes along the traveral path. A model user can also traverse their own data in the tree to gain insight into the model. For model providers, EB-tree is also helpful for debugging their models and identifying mislabeled training data.

4.2.2 Case 2: Understanding the Model Weakness via Visualizing Decision Boundaries. The decision boundaries learned through DNN classifiers are difficult to "see" by a human without examples. EB-tree provides a means for model users to see a small number of training data points that characterize the boundaries. It is also helpful for improving the model training. We implement an operation called *boundary projection* in the EB-tree to visualize boundaries. This operation traverses the EB-tree and finds all the edges with two end nodes from each pair of neighboring classes. A sequence of these node pairs along a boundary produces a visualization of the boundary between two classes.

Figure 12a shows the decision boundary between class "chimpanzee" and class "gorilla" for the EB-tree built for the Inception-v3 model on ImageNet. One may notice that some training points are at the wrong side of the boundary. For example, the third image on the "chimpanzee" side has a label of "gorilla". We identify that these training points are not correctly classified due to model training errors. We also find that if a training point is mislabeled, it is likely to also appear at the decision boundary. The reason is that when a training point of class B is mislabeled as class A, the data point still has significant similarity with many samples in class B in the DNN transformed space. Therefore, it is likely to be inserted into the EB-tree as a neighbor of some training points in class B. This enables us to identify incorrectly labeled training data through unexplainable observations on the decision boundaries. Similarly, as shown in Figure 6b, a few training points (e.g., the root the tree) are clearly mislabeled as "3". To further evaluate EB-tree's capability of discovering mislabeled training points, we randomly mislabel 100 images (10 for each class) in MNIST and CIFAR-10. Through the boundary projection, there are 99 and 97 mislabeled training points being identified respectively by the projected boundaries.

The separability of decision boundaries can reveal the weakness of a model. EB-tree links training data points that are close to each other but belong to different classes together. If a decision boundary contains many nodes but the differences among these nodes are unclear, the model is weak on differentiating these classes and misclassification is likely to happen among these classes. In contrast, if the difference of nodes is clear along a boundary, the model can generalize the difference between two classes well and is able to differentiate the corresponding data points belonging to these classes.


class chimpanzee, chimp, Pan troglodytes -> class gorilla, Gorilla gorilla
(a)



(b)

Figure 12: Projected boundaries for Inception-v3 (ImageNet) (a) the boundary between class "chimpanzee" and "gorilla"; (b) the boundary between "goose" and "black swan".

For test images in class "goose" and "black swan" in Figure 12b, the projected boundary is simple and the two classes can be separated by the color. As expected, we rarely observe misclassification between the two classes. For "chimpanzee" and "gorilla", the projected boundary contains many nodes and the majority of our participants claim that the decision boundary is difficult to recognize. Correspondingly, we find that one-third of the misclassified images in class "chimpanzee" are classified as "gorilla".

4.2.3 Case 3: Detecting Emerging New Classes. Using a pre-trained model on a new dataset faces challenges that the dataset may contain classes unseen in the training process, which may result in incorrect predictions or missing important discoveries. This is the emerging new class detection problem in model sharing.

Existing new class detection methods require massive distance computation [14, 22] to compare the incoming data with the existing training data points. EB-tree is able to reduce the computational complexity by only computing distances between a test data point and the training data points that share the same traversal path with the test data in the EB-tree. Formally, for an incoming data point z with a predicted class D that reaches node N in an EB-tree traversal, we are able to detect new classes by only comparing a subset D_N that reach node N in the EB-tree through the same traversal path. As in [14], we use conformal evaluation to compute the similarity between z and data points in D_N through p-values. The p-value $p_z^{D_N}$ of a test data point z is calculated as below:

$$p_z^{D_N} = \frac{|\{\forall i \in D_N : A(D_N \backslash z_i, z_i) \geq A(D_N, z)\}|}{|D_N|}, \tag{5}$$

where A is a distance function. The distance between two data points can be computed by the Euclidean distance of their confidence values just as in [14]. The p-value $p_z^{D_N}$ for z indicates how different the new data point is from existing data points that share the same traversal path. A low p-value indicates that the prediction of the test data lacks statistical support for fitting the prediction model.

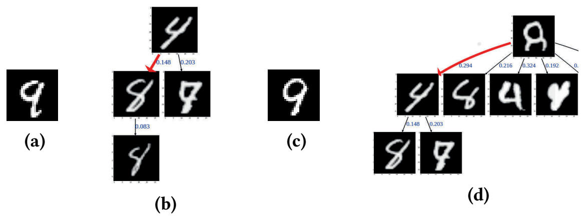


Figure 13: (a) and (c) are test data from the unseen class "9". (b) and (d) show the traversal paths of the two samples.

To demonstrate the effectiveness of EB-tree in detecting emerging new classes, we simulate a model sharing scenario with the MNIST dataset. Initially, the DNN model is only trained by data points from class "0" to "8". The EB-tree constructed for the DNN model contains 52 nodes. We then mix samples from class "9" in the test data and check whether they can be accurately identified. We observe that the test points in class "9" are misclassified in different ways. For instance, Figure 13a is classified as "8" with a confidence value of 93.76% while Figure 13c is misclassified as "4" with a confidence value of 96.43%.

For Figure 13a, we compare it with all nodes that also reach node "8" in the EB-tree shown in Figure 13b. Although all these nodes are predicted as "8", we get $p_S = 0.0073$. This value indicates that sample A is statistically different from previous data points being classified as "8" through the same node. Therefore it is likely to belong to a new class. Similarly, we get $p_S = 0.0036$ for Figure 13c by comparing with all nodes that reach node "4" shown in Figure 13d. Using this method, we correctly identify 991 out of 1,000 data points belonging to class "9" in the test data (99.1% accuracy). Comparing to the existing method [14] that achieves an accuracy of 99.2%, EB-tree reduces the computation cost by up to 97.1% through only computing the distances between an incoming sample and existing data points ending their EB-tree traversals on the same node. EB-tree provides a unique advantage for detecting emerging new classes and significantly reduces the detection cost.

In practice, each node in the EB-tree of a trained model is associated with a list of DNN-transformed features (e.g., confidence values) to track how the training data points hit the node for model users to detect new classes.

5 CONCLUSION AND DISCUSSION

We presented an Explicable Boundary Tree (EB-tree) to improve the trustworthiness of DNN model sharing. EB-tree supports model interpretability by disclosing a small set of representative training data. A model user gained insight into the decision making of a DNN model via EB-tree traversal. We showed that an EB-tree approximated the corresponding DNN model with high fidelity and improved a model user's understanding of a complicated model. We also showed an EB-tree was effective in detecting mislabeled training data and model training errors. EB-tree also significantly reduced the computation cost of detecting emerging new classes.

Discussions. An EB-tree can be built on the information at a different hidden layer of a DNN model. A tree built on the output of the last layer (softmax) often approximates the model the best with the least number of nodes, while a tree built on the output of one of the hidden layers often shows details of boundaries with additional nodes. This helps users to better understand the similarity or dissimilarity of features embodied in a boundary and in the test data. The EB-tree built on the output of the last fully connected layer of the MNIST-CNN model is available[1] for comparison. There exists a tradeoff between achieving interpretability and optimal model approximation.

In addition, the construction of a tree relies on the distance measure between data points. Euclidean distance shows good performance in the output of the last layer even though some work [13] shows the advantage of cosine similarity based distance metric in softmax output. There is still lots of room to exploit on what distances are suitable for the output of different hidden layers to provide an optimal model approximation.

6 ACKNOWLEDGMENT

The authors would like to thank Richard Nock and Xifeng Guo for insightful discussions. This work is partially supported by the The National Key Research and Development Program of China (2016YFB0200401), by the program for New Century Excellent Talents in University, by National Science Foundation (NSF) China 61402492, 61402486, 61379146, by the laboratory pre-research fund (9140C810106150C-81001), by the HUNAN Province Science Foundation 2017RS3045.

[1]https://github.com/sktzwhj/explicable_boundary_tree_sample/blob/master/MNIST_EB_tree_hidden_layer.png

REFERENCES

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

[2] Amazon. 2017. Machine Learning on AWS. https://aws.amazon.com/machine-learning/. (2017). [Online; accessed 07-July-2017].

[3] Alexandr Andoni and Ilya Razenshteyn. 2015. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*. ACM, 793–801.

[4] Franz Aurenhammer. 1991. Voronoi diagramsâĂŤa survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)* 23, 3 (1991), 345–405.

[5] Jacob Bien and Robert Tibshirani. 2011. Prototype selection for interpretable classification. *The Annals of Applied Statistics* (2011), 2403–2424.

[6] Olcay Boz. 2002. Extracting decision trees from trained neural networks. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 456–461.

[7] Zhengping Che, Sanjay Purushotham, Robinder Khemani, and Yan Liu. 2016. Interpretable deep models for icu outcome prediction. In *AMIA Annual Symposium Proceedings*, Vol. 2016. American Medical Informatics Association, 371.

[8] Mark Craven and Jude W Shavlik. 1996. Extracting tree-structured representations of trained networks. In *Advances in neural information processing systems*. 24–30.

[9] Ruth Fong and Andrea Vedaldi. 2017. Interpretable Explanations of Black Boxes by Meaningful Perturbation. *arXiv preprint arXiv:1704.03296* (2017).

[10] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE, 6645–6649.

[11] 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.

[12] 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 Signal Processing Magazine* 29, 6 (2012), 82–97.

[13] Shota Horiguchi, Daiki Ikami, and Kiyoharu Aizawa. 2016. Significance of softmax-based features over metric learning-based features. (2016).

[14] Roberto Jordaney, Kumar Sharad, Santanu K. Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. 2017. Transcend: Detecting Concept Drift in Malware Classification Models. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 625–642. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/jordaney

[15] 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.

[16] 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.

[17] Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. *arXiv preprint arXiv:1703.04730* (2017).

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet Classification with Deep Convolutional Neural Networks. *Commun. ACM* 60, 6 (May 2017), 84–90. https://doi.org/10.1145/3065386

[19] Der-Tsai Lee and Bruce J Schachter. 1980. Two algorithms for constructing a Delaunay triangulation. *International Journal of Computer & Information Sciences* 9, 3 (1980), 219–242.

[20] Charles Mathy, Nate Derbinsky, José Bento, Jonathan Rosenthal, and Jonathan S Yedidia. 2015. The Boundary Forest Algorithm for Online Supervised and Unsupervised Learning.. In *AAAI*. 2864–2870.

[21] Microsoft. 2017. Microsoft Azure Machine Learning Studio. https://studio.azureml.net/. (2017). [Online; accessed 07-July-2017].

[22] Xin Mu, Feida Zhu, Juan Du, Ee-Peng Lim, and Zhi-Hua Zhou. 2017. Streaming Classification with Emerging New Class by Class Matrix Sketching.. In *AAAI*. 2373–2379.

[23] Google Cloud Platform. 2018. Cloud Machine Learning Engine. https://cloud.google.com/ml-engine/. (2018). [Online; accessed 01-Feb-2018].

[24] 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.

[25] Ryan Rifkin and Aldebaro Klautau. 2004. In defense of one-vs-all classification. *Journal of machine learning research* 5, Jan (2004), 101–141.

[26] 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. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y

[27] W. J. Scheirer, A. de Rezende Rocha, A. Sapkota, and T. E. Boult. 2013. Toward Open Set Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35, 7 (July 2013), 1757–1772. https://doi.org/10.1109/TPAMI.2012.256

[28] Gregor PJ Schmitz, Chris Aldrich, and Francois S Gouws. 1999. ANN-DT: an algorithm for extraction of decision trees from artificial neural networks. *IEEE Transactions on Neural Networks* 10, 6 (1999), 1392–1401.

[29] 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.

[30] J Springenberg, Alexey Dosovitskiy, Thomas Brox, and M Riedmiller. 2015. Striving for Simplicity: The All Convolutional Net. In *ICLR (workshop track)*.

[31] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2818–2826.

[32] Tensorflow. 2017. Deep MNIST for Experts. https://www.tensorflow.org/get_started/mnist/pros. (2017). [Online; accessed 07-July-2017].

[33] Csaba D Toth, Joseph O'Rourke, and Jacob E Goodman. 2004. *Handbook of discrete and computational geometry*. CRC press.

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

[35] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. 2015. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579* (2015).

[36] Matthew D Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *European conference on computer vision*. Springer, 818–833.

[37] Daniel Zoran, Balaji Lakshminarayanan, and Charles Blundell. 2017. Learning Deep Nearest Neighbor Representations Using Differentiable Boundary Trees. *arXiv preprint arXiv:1702.08833* (2017).