Characterizing the Decision Boundary of Deep Neural Networks

Hamid Karimi
Michigan State University
karimiha@msu.edu

Tyler Derr
Michigan State University
derrtyle@msu.edu

Jiliang Tang
Michigan State University
tangjili@msu.edu

ABSTRACT

Deep neural networks and in particular, deep neural classifiers have become an integral part of many modern applications. Despite their practical success, we still have limited knowledge of how they work and the demand for such an understanding is evergrowing. In this regard, one crucial aspect of deep neural network classifiers that can help us deepen our knowledge about their decision-making behavior is to investigate their decision boundaries. Nevertheless, this is contingent upon having access to samples populating the areas near the decision boundary. To achieve this, we propose a novel approach we call **Deep D**ecision boundary **I**nstance **G**eneration (**DeepDIG**). DeepDIG utilizes a method based on adversarial example generation as an effective way of generating samples near the decision boundary of any deep neural network model. Then, we introduce a set of important principled characteristics that take advantage of the generated instances near the decision boundary to provide multifaceted understandings of deep neural networks. We have performed extensive experiments on multiple representative datasets across various deep neural network models and characterized their decision boundaries.

KEYWORDS

Decision boundary, Deep neural networks, Adversarial examples

1 INTRODUCTION

Thanks to available massive data and high-performance computation technologies (e.g. GPUs), deep neural networks (DNNs) have become ubiquitous models in many decision-making systems. Notwithstanding the high performance that DNNs have brought about in many domains [4, 7, 14–19], our understating of them is still very limited and lacking in some respects. This is primarily due to the black-box nature of DNNs where the decisions they are making are opaque and elusive. In this regard, one crucial aspect of DNN classifiers that yet remains fairly unknown is their decision boundaries and its geometrical properties. If we want to continue DNNs' usage for critical applications, understating their decision boundaries and decision regions is essential. This is especially important for safety and security applications such as e.g., self-driving cars [12] whose deep models are vulnerable to erroneous instances near their decision boundaries [30].

Compared to other aspects of DNN e.g., optimization landscape [2], systematic characterization of the decision boundary of DNNs, despite its importance, is still in the early stages of the study. The main challenge hindering in-depth analysis and investigation of decision boundaries of DNNs is generating instances that are simultaneously close to the decision boundary and resemble real instances[1]. We call such instances as *borderline instances*. The difficulty of generating borderline instances stems from the fact that the input space of DNNs is of high dimension e.g., $\mathbb{R}^{784}$ in the case of simple grayscale MNIST images, which makes searching for instances close to the decision boundary a non-trivial and challenging task.

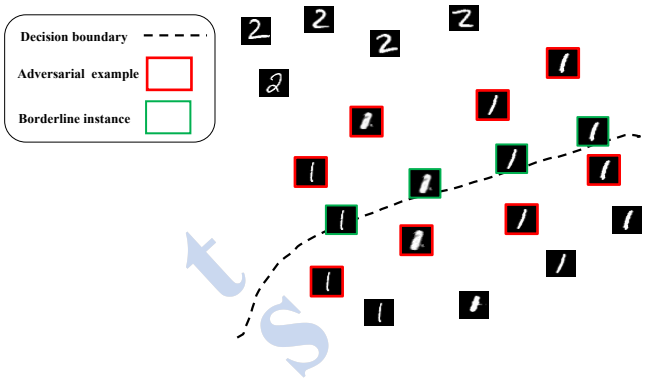


Figure 1: A high-level illustration of Deep Decision boundary Instance Generation (DeepDIG). For a given pre-trained deep neural network model and two classes s and t, DeepDIG tries to find instances as close as possible to the decision boundary between the two classes s and t.

To solve this challenge, we propose a novel framework called Deep Decision boundary Instance Generation (DeepDIG) a high-level illustration of which is shown in Figure 1. Given two classes of samples as well as a pre-trained DNN model, DeepDIG is optimized to generate borderline instances near a decision boundary between two classes i.e., generating instances whose classification probabilities for two classes are as close as possible. DeepDIG utilizes an autoencoder-based method to generate targeted adversarial examples at the two sides of the decision boundary between two classes and further employs a binary search based algorithm to refine and generate borderline instances. Moreover, we leverage the borderlines instances generated by DeepDIG and investigate two notable characteristics concerning the decision boundary of DNNs. First, we measure the complexity of the decision boundary in the input space. To this end, we measure the classification oscillation along the decision boundary between two classes and devise a novel metric offering us a form of *geometrical complexity* of the decision boundary. Second, we investigate the decision boundary in the embedding space learned by a DNN i.e., we measure the complexity of the decision boundary once it is projected in the embedding space. To this end, we take advantage of the linear separability property of DNNs and propose a metric capturing the complexity of the decision boundary in the embedding space. We found consistency between these two complexity measures.

DeepDIG and further characterization of the decision boundary of DNNs are novel with respect to the existing studies [1, 8, 10,

[1]In this paper, we use the terms *instance*, *sample*, and *example* interchangeably.

23, 26, 37] in the following ways. First, the previous work investigated the decision boundary merely through the lens of adversarial examples and considered adversarial examples as a type of borderline instances. However, as we show later, given the definition of the decision boundary, adversarial examples while being close to the decision boundary are not borderline instances. In comparison, DeepDIG, while using adversarial example generation, goes beyond adversarial examples and generates instances that *by design* are ensured to be as close as possible to the decision boundary. Second, we do not make any assumption on the DNNs being investigated and DeepDIG can be applied to any pre-trained DNN classifier. Third, instead of investigating the decision boundary of a DNN from the perspective of a single instance and/or its neighborhood, we characterize a decision boundary between two classes as a whole and shed light on its properties by taking advantage of a collection of instances populating that decision boundary. Through extensive experiments across three datasets, namely MNIST [22], FashionMNIST [35], and CIFAR10 [21], we verify the working of DeepDIG and investigate various pre-trained DNNs.

In summary, our major contributions are as follows.

- We propose a novel framework DeepDIG to generate instances near the decision boundary of a given pre-trained neural network classifier.
- We present several use-cases of DeepDIG to characterize decision boundaries of DNNs which help us to deepen our understating of DNNs.

The rest of the paper is organized as follows. In Section 2, we present the notations and define the problem. In Section 3, we present the proposed framework DeepDIG. Section 4 includes how we can use DeepDIG to characterize the decision boundary of a DNN. Experimental settings and details of investigated DNNs, as well as datasets, will be presented in Section 5. Experimental results and discussions will be presented in Section 6. We review the related work in Section 7 followed by concluding remarks in Section 8.

2 DEFINITIONS AND PROBLEM STATEMENT

In this section, we introduce the basic notations and definitions as well as the problem statement.

Notations. Let $f : \mathbb{R}^D \rightarrow \mathbb{R}^c$ denote a pre-trained c-class deep neural network classifier where D is the dimension of input space. Further, let $\mathcal{F}(x) \in \mathbb{R}^d$ denote the embedding space learned by f where d is the dimension of this space and usually $d \ll D$. We assume that the last layer of f is a $d \times c$ fully connected layer without any non-linear activation function which maps the embeddings to a score vector of size c i.e., $\mathbb{R}^c$. Then, for a sample $x \in \mathbb{R}^D$, the classification outcome is $C(x) = argmax_k f_k(x_i)$ where f_k is the score of k-th class ($1 \leq k \leq c$). We assume scores are calculated by applying the softmax function on the output of last layer of f. In other words, $f_k(x_i)$ denotes the prediction probability of classifying x_i as s. Finally, let $X = \{x_1, x_2 \cdots x_n\}$ denote a dataset of instances $x_i \in \mathbb{R}^D$ associated with ground-truth labels $Y = \{y_1, y_2 \cdots y_n\}$ where $y_i \in [1, c]$.

Decision Region and Decision Boundary. The classifier f partitions the space $\mathbb{R}^D$ into c decision regions $r_1, r_2 \cdots r_c$ where for each $x \in r_i$ we have $C(x) = i$. Now, in line with previous studies [8, 23], the decision boundary between classes s and t ($t, s \in [1, c]$) is defined as $b_{s,t} = \{v \in \mathbb{R}^D : f_s(v) = f_t(v)\}$. In other words, the deep neural network classifier (and as the matter of fact any other classifier) is "confused" about the labels of the instances on the decision boundary between classes s and t.

Problem Statement. *Given a pre-trained deep neural network model $f(.)$, a dataset X, and two classes s and t, we aim to generate instances near the decision boundary between decision regions r_s and r_t. Further, we intend to leverage the borderline instances as well as other generated and original samples to delineate the behavior of model $f(.)$*

3 PROPOSED FRAMEWORK (DEEPDIG)

Given a pre-trained DNN, we intend to generate borderline instances satisfying two important criteria:

(a) They need to be as near as possible to the decision boundary between two classes i.e., their DNN's classification scores (probabilities) be as close as possible. This is basically to follow the definition of decision boundary–Refer to Section 2.
(b) Borderline instances need to be similar to the original (real) instances.

The second criterion is imposed because of two major reasons. First, we are interested in investigating DNNs and their decision boundaries in the presence of realizable and non-random corner cases which in practice can have major safety and security consequences [5, 30, 38]. Second, essentially a DNN carves out decision regions (and decision boundaries) by learning on its *training data* not other random instances in the space $\mathbb{R}^D$. Therefore, random borderline instances (i.e., those which are not similar to real instances) occupy some parts of the input space which are not practically appealing for further decision boundary characterization in Section 4. Note that, in spirit, the second criterion is similar to what that is followed in adversarial example generation [36] where adversarial examples are required to be similar to real (benign) samples.

To generate borderline samples satisfying the above criteria, we propose the framework Deep Decision boundary Instance Generation (DeepDIG), which is illustrated in Figure 2. As shown in this figure, DeepDIG includes three major components. In the first component, we utilize an autoencoder-based method to generate targeted adversarial instances from a source class to a target class– See Figure 2 (I). In the second component, we employ another autoencoder-based adversarial instance generation on the first component's adversarial examples and consequently generate new adversarial instances predicted as the source class– See Figure 2 (II). Adversarial samples generated in the first and second components of DeepDIG are at the opposite sides of a decision boundary between a source and a target class, and more importantly, these samples are *by design* close to the decision boundary. Hence, in the third component of DeepDIG, we feed these two sets of adversarial samples to a module named Borderline Instance Refinement which based on a binary search algorithm refines and generates borderline instances being sufficiently close to the decision boundary–See Figure 2 (III). Next, we explain each component in detail.

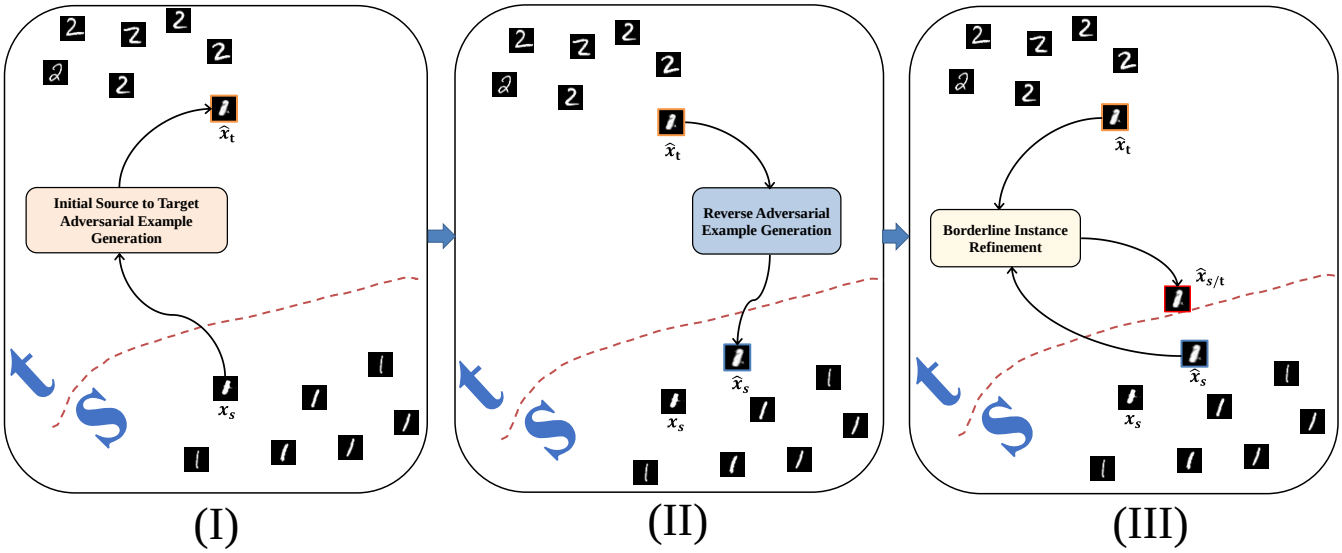


Figure 2: The proposed framework Deep Decision boundary Instance Generation (DeepDIG). It consists of three components. In component (I), targeted adversarial examples of source instances are generated ($\hat{x}_t$). In component (II), from adversarial examples of component (I), a new set of adversarial examples are generated ($\hat{x}_s$) which are classified as s. Finally, in component (III), a binary search based algorithm is employed to refine and identify the borderline instances near the decision boundary.

3.1 Component (I): Initial Source to Target Adversarial Example Generation

One way to obtain samples enjoying the criterion (b) mentioned above is via targeted adversarial examples which are *slightly distorted* versions of real instances and are misclassified by a DNN [36]. As will be discussed shortly, targeted adversarial example generation paves the way to meet the criterion (a) as well. Hence, as the first step towards generating borderline instances, we generate targeted adversarial examples from real instances of class s to be misclassified as class t. To generate such adversarial examples, we utilize a simple yet effective approach using an autoencoder-based method formulated in the following loss function.

$$\mathcal{L}_I = \sum_{\forall x_s} \Big(||x_s - A_1(x_s)||_2^2 + \alpha \times CE\big(f(A_1(x_s)), \vec{t}\big) \Big) \qquad (1)$$

where x_s denotes a sample belonging to class s, $A_1(.)$ is an autoencoder reproducing its input sample (here x_s), $\vec{t}$ is a c-dimensional one-hot vector having its t-th entry equal to 1 and the rest to 0, CE is the class entropy loss function[2], and α is a hyperparamter controlling the trade-off between reconstruction error and adversarial example generation. The loss function $\mathcal{L}_I$ is optimized along with other components of DeepDIG. Also, for convenience, we show the output of $A_1(x_s)$ as $\hat{x}_t$ signifying its mis-classification as class t.

Eq. 1 has two parts. The first part (reconstruction error) ensures keeping the generated adversarial example, $\hat{x}_t$, as close as possible to the real sample x_s i.e., satisfying the criterion (b). The second part of Eq. 1 attempts to misclassify the generated instance i.e., placing it outside the decision region r_s. Therefore, optimizing $\mathcal{L}_I$ makes the generated adversarial examples close to the decision boundary between two classes as has been shown before as well [8, 10, 13]. Nevertheless, given the definition of the decision boundary in Section 2, samples $\hat{x}_t$ are not 'sufficiently' close to the decision boundary between classes s and t and thus criterion (a) is not fully met yet. Hence, DeepDIG is equipped with two other components to generate proper borderline samples.

3.2 Component (II): Reverse Adversarial Example Generation

As mentioned before, an adversarial example $\hat{x}_t$ is outside of the decision region r_s and is near the decision boundary between classes s and t. Aiming at generating samples even closer to the decision boundary, we leverage another targeted adversarial example generation applied on samples $\hat{x}_t$. We call this component *Reverse* Adversarial Example Generation since we generate adversarial examples of the first component's adversarial examples[3]. The loss function is as follows.

$$\mathcal{L}_{II} = \sum_{\forall \hat{x}_t} \Big(||\hat{x}_t - A_2(\hat{x}_t)||_2^2 + \alpha \times CE\big(f(A_2(\hat{x}_t)), \vec{s}\big) \Big) \qquad (2)$$

where $A_2(.)$ is another autoencoder to reproduce its the input sample here (here $\hat{x}_t$)[4], $\vec{s}$ is a c-dimensional one-hot vector having its s-th entry equal to 1 and the rest to 0, and α is a hyperparameter controlling the trade-off between reconstruction error and adversarial

[2]https://en.wikipedia.org/wiki/Cross_entropy

[3]Technically, examples generated in component (II) are not *adversarial* since they are correctly classified as s. However, for the sake of simplicity in the presentation, we abuse the definition and keep referring to them as *adversarial examples*.

[4]Note that autoencoders A_1 ad A_2 has the same architecture while they subscripted here to signify their distinct parameters in components (I) and (II) of DeepDIG, respectively.

example generation. $\mathcal{L}_{II}$ is optimized along with other components of DeepDIG. Again for convenience, we show the output of $A_2(\hat{x}_t)$ as $\hat{x}_s$ signifying its classification as class s. Next, we explain how to we utilize adversarial examples $\hat{x}_t$ and $\hat{x}_s$ to generate borderline instances that are sufficiently near the decision boundary between classes s and t.

3.3 Component (III): Borderline Instance Refinement

As mentioned before, the high dimensional nature of input space in DNNs causes a big challenge for generating instances that are simultaneously near the decision boundary and are similar to real instances i.e., satisfying criteria (a) and (b), respectively. More specifically, randomly generating samples in $\mathbb{R}^D$ even for small D (e.g., 100) has an extremely low chance of producing legitimate and similar-to-real instances let alone yielding those near the decision boundary. Moreover, simply perturbing the real instances aiming at finding borderline samples induces a huge number of directions to consider and is prohibitively infeasible. Nevertheless, thanks to components (I) and (II) of DeepDIG, searching for borderline instances is now facilitated. This is because we generate two sets of adversarial examples (i.e., $\hat{x}_s$ and $\hat{x}_t$ through components (I) and (II), respectively) which are *by design* close to a decision boundary between two classes and populates both sides of the decision boundary. More importantly and again by design, they are similar to real instances. Hence, the Borderline Instance Refinement component of DeepDIG employs a binary search algorithm between the trajectory connecting a pair of samples $\hat{x}_s$ and $\hat{x}_t$ aiming at finding the desired borderline instance. Algorithm 1 shows the proposed approach for borderline instance refinement and is explained in the following.

As input, this algorithm takes generated adversarial examples $\hat{x}_t$ and $\hat{x}_s$ belonging to classes t and s, respectively, i.e., two instances from distinct sides of the decision boundary of the DNN model f–See Figure 2 (III). The algorithm performs a binary search to find a middle point $\hat{x}_m$ whose difference in probabilities belonging to classes t and s is less than a small threshold (e.g., 0.0001) that we denote as β. In the algorithm, this is given by $|f(x_m)_s - f(x_m)_t| < \beta$ (line 10). This thresholding is in line with the definition of decision boundary between two classes where instances should have equal classification probabilities for classes s and t. In other words, the DNN is 'confused' about the class of such instances. We note that Algorithm 1 might fail to find such an instance if the middle point (i.e., x_m) deviates from decision regions of classes s or t– See line 8. Nevertheless, the proposed Algorithm 1 is empirically quite effective at identifying borderline instances as it will be demonstrated in the experiments (Section 5).

Remark. Before introducing the decision boundary characteristics in the next section, we need to clarify a matter. To fully characterize the decision boundary between two classes –say a and b– one needs to generate borderline samples for both a and b. More specifically, following our notations and DeepDIG mechanism demonstrated in Figure 2, once we apply DeepDIG for (s,t)=(a, b) and then (s,t)=(b, a). Hence, to fully characterize the decision boundary between two classes, we obtain two sets of borderline instances.

Algorithm 1: The proposed Borderline Instance Refinement algorithm

Data:
Instances $\hat{x}_s$ and $\hat{x}_t$, threshold α,
pre-trained DNN model f
Initialization:
$x_l = \hat{x}_s$; $x_r = \hat{x}_t$;

```
1  while True do
2      x_m = (x_l + x_r)/2
3      if C(x_m) = s then
4          x_l = x_m
5      else if C(x_m) = t then
6          x_r = x_m
7      else
8          return "Fail"
9      end
10     if |f_s(x_m) − f_t(x_m)| < β then
11         x̂_{s/t} = x_m
12         return x̂_{s/t}
13 end
```

4 DECISION BOUNDARY CHARACTERISTICS

As mentioned before, one of the challenges of principled and in-depth analysis of the decision boundary of DNNs is the inaccessibility of samples close to the decision boundary which would be similar to real samples as well. Nevertheless, DeepDIG addresses this challenge and provides us with a systematic way to generate borderline instances near the decision boundary between two classes. This opens us a door to understand and characterize the decision boundary of DNNs in a better way. To this end, we introduce several metrics informing us about the different characteristics of the decision boundary of a deep neural network. We group the characterization measures into two distinct groups: decision boundary complexity in the input space (Section 4.1) and decision boundary complexity in the embedding space (Section 4.2).

4.1 Decision Boundary Complexity in the Input Space

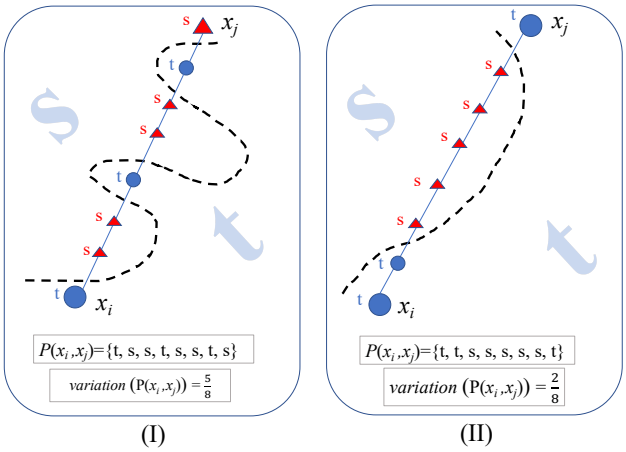
As previously shown [8], DNN classifiers tend to carve out complicated decision regions in the input space to be able to discriminate input samples of different classes. These decision regions are highly non-convex and have highly non-linear decision boundaries. The question is how we can measure the geometrical complexity (non-convexity) of the decision regions in the input space? Thus far, the practical and systematic investigation of the complexity and non-convexity of decision boundaries (and decision regions) of any DNN (regardless of its architecture, model size, etc) has been a challenging task because there has not been an efficient method generating samples populating the decision boundary of a DNN. Fortunately, DeepDIG provides us with a method generating borderline samples quite effectively (will be shown in the experiment section). Hence, we now utilize the borderline samples to measure the degree of complexity (or non-convexity) of a decision boundary

in the input space. To this end, we devise a new metric described as follows.

Let x_i and x_j denote two borderline instances for the decision boundary between classes s and t. Further, we define a trajectory $\mathcal{T}(t)$ between x_i and x_j as $\mathcal{T}(t) = t \times x_i + (1-t) \times x_j$ where $0 \leq t \leq 1$. Then, for m values of t we interpolate m instances along the trajectory $\mathcal{T}(t)$ denoted as $\mathcal{I}_{(x_i,x_j)} = \{x_{p_1}, x_{p_2} \cdots x_{p_m}\}$. We retrieve the DNN's classification outcomes for interpolated samples $\mathcal{I}_{(x_i,x_j)}$ and denote them as $\mathcal{P}_{(x_i,x_j)} = \{C(x_{p_1}), C(x_{p_2}) \cdots C(x_{p_m})\}$. We define the *oscillation* of classification outcomes of interpolated samples $\mathcal{I}_{(x_i,x_j)}$, denoted as $O_{(x_i,x_j)}$, as follows.

$$O_{(x_i,x_j)} = \frac{1}{|\mathcal{I}_{(x_i,x_j)}|} \sum_{k=1}^{m-1} \mathbb{1}(C(x_{p_k}) \neq C(x_{p_{k+1}})) \tag{3}$$

where $\mathbb{1}$ denotes the indicator function[5]. $O_{(x_i,x_j)}$ essentially measures the 'variation' along the trajectory connecting two borderline instances x_i and x_j. A higher value for this metric indicates more alternating between decision regions r_s and r_t and vice versa. This is pictorially illustrated in Figure 3. We can also interpret O as a proxy informing us about the *smoothness* of the decision boundary between two classes.



(I) (II)

Figure 3: An illustration of capturing geometrical complexity of a decision boundary through measuring the *oscillation* between two decision (classification) regions r_s and r_t for samples on a borderline trajectory. Decision boundary in case (I) is geometrically more complex than that of (II).

Now let $B = \{x_1, x_2 \cdots x_n\}$ denote all borderline instances. For each borderline sample $x_i \in B$, we randomly select k other borderline samples whose set is denoted as S_{x_i}. Then, we record the average $O_{(x_i,x_j)}$ for $x_i \in B$ and $x_j \in S_{x_i}$. Eventually we report the average O across all borderline samples x_i as the final value of this measure, which we call **IDC** (**I**nput space **D**ecision boundary **C**omplexity) and is formulated in Eq. 4.

$$\text{IDC} = \frac{1}{n \times k} \sum_{x_i \in B} \sum_{x_j \in S_{x_i}} O_{(x_i,x_j)} \tag{4}$$

[5]https://en.wikipedia.org/wiki/Indicator_function

4.2 Decision Boundary Complexity in the Embedding Space

IDC measure developed in Section 4.1 looks into the decision boundary complexity in the input space. In this part, we focus on the decision boundary in the embedding space as defined next.

Decision boundary in the embedding space. We abuse the definition of the decision boundary in Section 2 and define the decision boundary in the embedding space as $be_{s,t} = \{z \in \mathbb{R}^d : f_s(f^{-1}(z)) = f_t(f^{-1}(z))\}$ where $f^{-1}(z)$ denotes a machinery that returns a sample whose embedding is z. We do not have a direct access to f^{-1}. Rather, in practice, for a collection of samples $v \in \mathbb{R}^D$ (i.e., training and test samples as well as generated borderline instances) we have pairs of (v, z) where through accessing the DNN f we know that $f^{-1}(z) = v$.

Now two interesting questions emerge regarding the decision boundary in an embedding space learned by a DNN. First, if we project borderline samples in the embedding space will they still be in the area separating two classes? In other words, will borderline instances be still near the decision boundary in the embedding space? Second, how we can measure the complexity of the decision boundary in the embedding space? To be more specific, does the decision boundary complexity in the input space manifest itself in the embedding space as well? Aiming at answering these questions, in this part, we propose two measures quantifying decision boundary characterization in the embedding space. To achieve this, we utilize an intriguing property of DNNs described in the following.

One of the fundamental properties of DNNs is their representation power where through a sophisticated combination of layer-wise and non-linear transformations they can map their complicated high dimensional input data to a low-dimension embedding space. It has been shown and will show in Section 5 that in the embedding space data points from different classes can be linearly separated [11, 25]. Not that the capability to learn linear separable embeddings by a DNN is closely related to the generalization power of that DNN [23]. Hence, should a DNN manage to learn linearly separable embeddings on the training set, it is expected to do so on unseen data samples such as borderline instances[6]. With this discussion in mind, we train a linear model on embeddings of training samples of classes s and t and the following measures are considered to characterize the decision boundary in the embedding space. We call these measures **EDC** (**E**mbedding space **D**ecision boundary **C**omplexity).

- **EDC1**. The linear model establishes a hyperplane to separate samples of two classes in the embedding space. This hyperplane acts as a valuable yardstick to characterize the decision boundary in the embedding space. In particular, we measure the average absolute value distance of all borderline instances from the linear model's hyperplane. We call this measure $\text{EDC1}_{\text{Borderline}}$. To contextualize this measure, we also compute it for a held-out test set and denote it as $\text{EDC1}_{\text{Test}}$. If borderline instances are indeed near the decision boundary between two classes in the embedding space, we should expect a higher value for $\text{EDC1}_{\text{Test}}$ than $\text{EDC1}_{\text{Borderline}}$. That is,

[6]Note that DeepDIG treats a model f as a pre-trained model whose parameters have been optimized and learned previously. Thus, as far as model f is concerned, generated borderline samples are still considered unseen data points.

borderline instances should be closer to the decision boundary. Therefore, through EDC1 we should be able to answer the first question asked above.

- **EBC2**. To answer the second question asked before, we record the performance (e.g., accuracy) of the trained linear classifier against borderline samples (denoted as $EDC2_{Borderline}$) as well as a held-out test set (denoted as $EDC2_{Test}$). This measure will complement EDC1 in a sense that allows us to know to what extent samples (borderline samples and an unseen test set) in the embedding space learned by a DNN are linearly separable. Hence, for a more complicated decision boundary in the embedding space, $EDC2_{Borderline}$ will be higher and vice versa.

As for the linear model, in line with previous studies [24, 28] we use a linear Support Vector Machine (SVM) [3]. Our linear SVM seeks to find a hyperplane between learned embeddings of two classes s and t according to Eq. 5.

$$\begin{aligned} &Minimize_{\mathbf{w},b,\epsilon}\ \frac{1}{2}||w||^2 + \gamma(\sum_{i=1}^{n}\epsilon_i) \\ &s.t \begin{cases} y_i(\mathbf{w}\times\mathcal{F}(x_i)+b) \geq 1-\epsilon_i \\ \forall i\ \epsilon_i \geq 0 \end{cases} \end{aligned} \tag{5}$$

where γ is a hyperparmer controlling the error minimization and margin maximization trade-off, $\mathbf{w}$ is the weight vector, b is the bias term, and the ϵ_is are slack variables that allow a sample to be on the separating hyperplane $\mathbf{w}\times\mathcal{F}(x_i)+b$. Recall that $\mathcal{F}(x_i)$ is the embedding vector learned by a DNN for an instance x_i.

5 EXPERIMENTAL SETTINGS

To verify the working and usefulness of DeepDIG, we conduct some experiments. In Section 5.1, we describe the datasets and pre-trained models developed to investigate DeepDIG. Section 5.2 describes the experimental settings for DeepDIG.

5.1 Datasets and Deep Neural Networks

We investigate the proposed framework DeepDIG against three datasets, namely MNIST [22], FashionMNIST [35], and CIFAR10 [21]. For each dataset, we train two models whose description can be found in Table 1. In this table, $CNV(a,b,c)$ denotes the convolution operation with a input channels, b output channels, and kernel size $c\times c$, $ReLU$ is the ReLU activation function [27], $Linear(a,b)$ indicates a fully connected layer with input size a and output size b, and $MaxPool(a)$ denotes max pooling of size $a\times a$. For MNIST and FashionMNIST datasets, we use two simple and distinct models, namely a convolutional neural network (CNN) and a fully connected network (FCN). CIFAR10 is a complicated dataset and we use two well-known deep architectures, namely ResNet [9] and GoogleNet [33]. The building block of the latter is the famous inception network [32]. The third column of Table 1 shows the number of trainable parameters. Also, we have included the accuracy of each DNN against the standard test set of its corresponding dataset. Note that the focus in this work is not having DNNs with state-of-the-art performance. Rather, we focus on analyzing a DNN (regardless of its performance) through the lens of its decision boundaries.

We use the PyTorch package [29] to implement DNNs. Each DNN is trained on its standard training set for 40 epochs and a batch size of 64 samples. We use Adam optimizer [20] to optimize the parameters. The learning rate is set to 0.01 with the decaying rate of 0.99 after every 100 optimization steps. After a model is trained, we save it and utilize it as a pre-trained model for further investigation.

5.2 DeepDIG Experimental Settings

As described in Section 3, component (I) and (II) utilize an autoencoder to generate adversarial examples. Table 2 describes the detail of the utilized autoencoders. Since MNIST and FashionMNIST are similar, we opt for employing the same autoencoder architecture for these two datasets. Each autoencoder consists of two modules: an encoder mapping an input sample to a condensed hidden representation and a decoder mapping back the hidden representation to a reconstruction of the original input sample. Since input samples are images in the pixel range $[0,1]$, we utilize the sigmoid activation function at the end of each decoder[7]. In Table 2, $TCNV$ denotes transposed convolution operation known as deconvolution as well [6]. Next, we explain the training detail of each component.

Component (I). To optimize component (I), we use samples from the standard train set labeled as class s e.g., all training samples labeled as 'Trousers' for FashionMNIST. Out of such samples, we use 80% for training and the rest as the validation set to tune the hyperparameters. Notably, we use the validation set for finding the optimal value of the hyperparameter α in Eq. 1. Two criteria are considered to choose the best value for α: the success rate of adversarial example generation and the quality of generated adversarial examples (examples $\hat{x}_t$ in Figure 2 (I)). To ensure the first criterion, we check the accuracy of adversarial examples against the validation set; the more decline in the accuracy, the better. For the second criterion, we visually inspect the generated examples and ensure they resemble real samples. For all pre-trained DNNs in Table 1, we found $\alpha = 0.8$ as the best choice. We train the adversarial example generation in component (I) for 5000 steps and batch size 128 samples. Adam optimizer [20] is used and the learning rate is set to 0.01 with the decaying rate 0.95 after every 1000 steps.

Component (II). The successfully generated adversarial examples in components (I) (i.e., $\hat{x}_t$ samples whose prediction is t) are used to optimize component (II). Similar to adversarial example generation in component (I), here we check both the accuracy and the quality of generated examples. Note that since we perform the *reverse* adversarial examples generation (i.e., adversarial examples of adversarial examples), the higher the accuracy is, the better the model is performing. Simulation settings are the same with competent (I) including $\alpha = 0.8$ for the loss function in Eq. 2.

Component (III). We run Algorithm 1 for all pairs of successfully generated adversarial samples in component (I) and (II) (i.e., $\{(\hat{x}_t,\hat{x}_s)|C(\hat{x}_t)=t, C(\hat{x}_s)=s\}$) aiming at finding borderline samples. We set the threshold $\beta = 0.0001$. We believe this value is sufficiently small ensuring the criterion (a) discussed in Section 3. As for the criterion (b) –borderline instances being similar to real examples– we visually inspect the borderline samples. In fact, as

[7] https://en.wikipedia.org/wiki/Sigmoid_function

Table 1: Description of investigated DNNs for MNIST, FashionMNIST and CIFAR10 datasets

DNN	Architecture	#Parameters	Test Accuracy
$\text{MNIST}_{\text{CNN}}$	$CNV(1,10,3)$, $MaxPool(2)$, $ReLU$ $CNV(10,10,3)$, $MaxPool(2)$, $ReLU$ $Linear(320,50)$, $ReLU$, $Linear(50,10)$	14,070	98.75
$\text{MNIST}_{\text{FCN}}$	$Linear(784,50)$, $ReLU$, $Linear(50,50)$	42,310	97.57
$\text{FashionMNIST}_{\text{CNN}}$	$CNV(1,10,3)$, $MaxPool(2)$, $ReLU$ $CNV(10,10,3)$, $MaxPool(2)$, $ReLU$ $Linear(320,50)$, $ReLU$, $Linear(50,10)$	14,070	89.11
$\text{FashionMNIST}_{\text{FCN}}$	$Linear(784,50)$, $ReLU$, $Linear(50,50)$	42,310	88.24
$\text{CFIAR10}_{\text{ResNet}}$	ResNet [9]	21,282,122	82.68
$\text{CIFAR10}_{\text{GoogleNet}}$	GoogleNet [33]	6,166,250	84.71

Table 2: Description of autoencoder models used in components (I) and (II) of DeepDIG

Dataset	Encoder	Decoder
MNIST FashionMNIST	$Linear(784,100)$, $ReLU$	$Linear(100,50)$, $ReLU$, $Linear(50,784)$, $Sigmoid$
CIFAR10	$CNV(3,12,4)$, $ReLU$ $CNV(12,24,4)$, $ReLU$ $CNV(24,48,4)$, $ReLU$	$TCNV(48,24,4)$, $ReLU$ $TCNV(24,12,4)$, $ReLU$ $TCNV(12,3,4)$, $Sigmoid$

will be demonstrated in the next section, DeepDIG is capable of meeting both criteria quite effectively.

The entire code is publicly available at https://github.com/hamidkarimi/DeepDIG/.

6 RESULTS AND DISCUSSIONS

In this section, we present the experimental results. First, in Section 6.1, we investigate the working of components (I) and (II) of DeepDIG. In Section 6.2, we compare the performance of DeepDIG with two baseline approaches. Section 6.3 includes the results of characterizing DNNs in Table 1 for a pair of classes. Finally, in Section 6.4, we present the characterization results for all pair-wise classes in the model $\text{MNIST}_{\text{CNN}}$.

6.1 DeepDIG Component Analysis

To recall from Section 3, for a given source class s and target class t, DeepDIG entails two important components including an adversarial example generation from class s to t –See Figure 2 (I), another adversarial example generation model mapping adversarial examples found in component (I) back to the class s region –See Figure 2 (II). Since adversarial example generation methods in components (I) and (II) plays an essential role in generating borderline samples, it is necessary to evaluate and analyze their performance. To this end, we run DeepDIG against models described in Table 1. For DNNs of MNIST, FashionMNIST, and CIFAR10 we investigate the class pairs ('1', '2'), ('Trouser', 'Pullover'), and ('Automobile', 'Bird'), whose results are shown in Tables 3, 4, and 5, respectively. We evaluate the components (I) and (II) in terms of two factors, namely the accuracy of their adversarial example generation and the quality of the generated examples. Hence, results for each DNN include the accuracy score as well as the visualization of some of the generated samples (chosen randomly). Note that since component (I) generates adversarial examples that are miss-classified by a DNN, the smaller value for accuracy means more success. Component (II), however, performs the *reverse* adversarial example generation (i.e., mapping component (I)'s adversarial examples back to their correct classification region) and thus in this component higher value for accuracy means more success. Based on the results presented in Tables 3, 4, and 5, we make the following observation.

- For all DNNs, both components (I) and (II) are shown to be capable of generating samples with very high accuracy[8].
- Overall, for all of DNNs the generated examples have high quality. The colors have been lost in generated examples for $\text{CIFAR10}_{\text{ResNet}}$ and $\text{CIFAR10}_{\text{GoogleNet}}$. However, generated samples are visibly automobiles and birds.

We can conclude that components (I) and (II) of DeepDIG perform as expected and are reliable modules for borderline instance generation.

6.2 Baseline Comparison

To further evaluate the performance of DeepDIG for borderline instance generation, we compare it against several baseline methods described as follows.

- **Random Pair Borderline Search (RPBS).** One may wonder that Algorithm 1 can be applied directly to any two samples as long as they are at the opposite sides of the decision boundary. Hence, in this baseline, we randomly pair up training samples from classes s and t and then apply Algorithm 1. More specifically, the input to Algorithm 1 is $\{(x_i, x_j) | C(x_i) = s, C(x_j) = t\}$ where x_i and x_j belong to the training set and are randomly paired up. The authors in [37] used a similar method to study the decision boundary of DNNs.
- **Embedding-nearest Pair Borderline Search (EPBS).** In this baseline method, instead of randomly pairing up samples at the opposite sides of the decision boundary between two classes, we pair a sample classified as s with its *nearest* sample at the opposite side of the boundary that is classified as t. The distance between the two samples is calculated in the

[8] We found similar results for the f1 score.

Table 3: Experimental results of investigating components (I) and (II) of DeepDIG for MNIST dataset

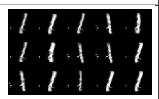
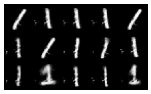
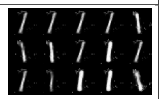
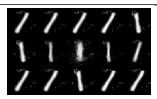
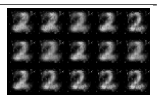
(s, t)	('1', '2')				('2', '1')			
Component	(I)		(II)		(I)		(II)	
Factor / DNN	Acc	Visualisation	Acc	Visualisation	Acc	Visualisation	Acc	Visualisation
$\text{MNIST}_{\text{CNN}}$	0.0		1.0		0.0		1.0	
$\text{MNIST}_{\text{FCN}}$	0.0		0.99		0.0		1.0	

Table 4: Experimental results of investigating components (I) and (II) of DeepDIG for FashionMNIST dataset

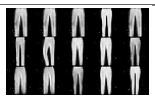
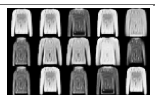
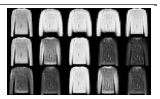
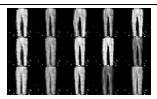
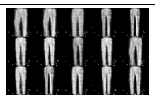
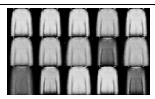
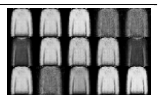
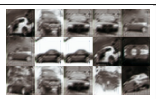
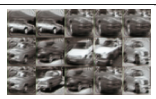
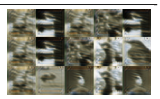
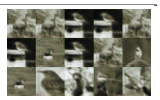
(s, t)	('Trouser', 'Pullover')				('Pullover', 'Trouser')			
Component	(I)		(II)		(I)		(II)	
Factor / DNN	Acc	Visualisation	Acc	Visualisation	Acc	Visualisation	Acc	Visualisation
$\text{FashionMNIST}_{\text{CNN}}$	0.01		1.0		0.0		1.0	
$\text{FashionMNIST}_{\text{FCN}}$	0.0		1.0		0.0		1.0	

Table 5: Experimental results of investigating components (I) and (II) of DeepDIG for CIFAR10 dataset

(s, t)	('Automobile', 'Bird')				('Bird', 'Automobile')			
Component	(I)		(II)		(I)		(II)	
Factor / DNN	Acc	Visualisation	Acc	Visualisation	Acc	Visualisation	Acc	Visualisation
$\text{CIFAR10}_{\text{ResNet}}$	0.00		0.99		0.01		0.99	
$\text{CIFAR10}_{\text{GoogleNet}}$	0.00		0.99		0.01		0.99	

embedding space. More formally, the input to Algorithm 1 is $\{(x_i, x_j)|C(x_i) = s, C(x_j) = t, x_j = min_{x_t} ||\mathcal{F}(x_i) - \mathcal{F}(x_t)||_2^2\}$ where, recalling from Section 2, $\mathcal{F}$ denotes the embedding space learned by a DNN. The reason for including this baseline is that by considering a better-guided trajectory between two samples, EPBS can hopefully generate borderline samples more effectively than RPBS.

Tables 6, 7, and 8 show the results for MNIST, FashionMNIST, and CIFAR10, respectively. Based on the results presented in these tables, we compare DeepDIG with baseline methods in terms of three important factors explained in the following. We should emphasize that an effective method is expected to succeed in all three factors.

(1) As for the first factor, we measure the average absolute difference in a DNN's prediction probabilities of classes s and t for borderline samples. This factor has been shown as $\overline{|f_s(x) - f_t(x)|}$ in Tables 6, 7, and 8. This factor is in line with the definition of the decision boundary (refer to Section 2) and to ensure the criterion (a) discussed in Section 3. We can observe that all methods including DeepDIG succeed in discovering borderline instances whose difference in prediction probabilities for classes s and t is significantly small on average. In other words, a DNN is 'confused' to categorically classify generated borderline instances.

(2) For the second factor, we inspect the quality of the generated borderline samples. We expect a good borderline generation method to generate borderline instances that are visibly similar to real samples. This is in line with the criterion (b) explained in Section 3. Note that unlike baselines methods, DeepDIG approaches the decision boundary between two classes s and t in a two-way fashion i.e., once from s to t

Table 6: Comparing DeepDIG with baseline methods (MNIST dataset)

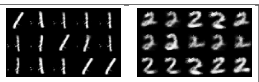
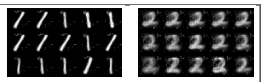
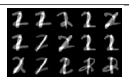
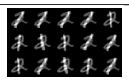
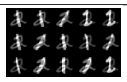
DNN / Factor / Method	$\text{MNIST}_{\text{CNN}}$			$\text{MNIST}_{\text{FCN}}$		
	$\overline{\lvert f_s(x) - f_t(x)\rvert}$	Visualisation	Success Rate	$\overline{\lvert f_s(x) - f_t(x)\rvert}$	Visualisation	Success Rate
DeepDIG	4.41×10^{-5}		98.66	4.43×10^{-5}		99.19
RPBS	4.44×10^{-5}		79.33	4.4×10^{-4}		48.48
EPBS	4.5×10^{-4}		86.48	4.40×10^{-5}		22.50

Table 7: Comparing DeepDIG with baseline methods (FashionMNIST dataset)

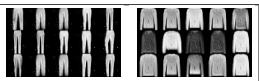
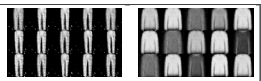
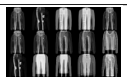
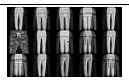
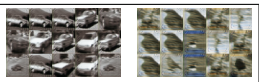
DNN / Factor / Method	$\text{FashionMNIST}_{\text{CNN}}$			$\text{FashionMNIST}_{\text{FCN}}$		
	$\overline{\lvert f_s(x) - f_t(x)\rvert}$	Visualisation	Success Rate	$\overline{\lvert f_s(x) - f_t(x)\rvert}$	Visualisation	Success Rate
DeepDIG	4.38×10^{-5}		98.93	4.44×10^{-5}		93.18
RPBS	4.41×10^{-5}		15.98	4.47×10^{-5}		30.64
EPBS	4.42×10^{-5}		33.48	4.39×10^{-5}		08.40

Table 8: Comparing DeepDIG with baseline methods (CIFAR10 dataset)

DNN / Factor / Method	$\text{CIFAR10}_{\text{ResNet}}$			$\text{CIFAR10}_{\text{GoogleNet}}$		
	$\overline{\lvert f_s(x) - f_t(x)\rvert}$	Visualisation	Success Rate	$\overline{\lvert f_s(x) - f_t(x)\rvert}$	Visualisation	Success Rate
DeepDIG	4.36×10^{-5}		70.60	4.44×10^{-5}		73.65
RPBS	4.46×10^{-5}		12.06	4.32×10^{-5}		13.96
EPBS	4.40×10^{-5}		13.62	4.03×10^{-4}		38.47

and once from t to s as described at the end of Section 3; accordingly, we have included two sets of visualized images. As can be observed in Tables 6, 7, and 8, DeepDIG can generate borderline instances that are indistinguishable from real samples specially for MNIST and FashionMNIST datasets. However, RPBS and EPBS fail to generate similar-to-real samples. A closer look reveals that RPBS and EPBS generate a 'sloppy' combination of the samples of classes s and t e.g., for FashionMNIST the generated borderline instances contain both *trouser* and a *pullover*.

(3) Finally, for the last factor, we expect a method to generate borderline instances with a high *success rate*. To quantify this, we measure the ratio of the number of sample pairs fed to Algorithm 1 to the number of successfully generated borderline instances i.e., those returned in line 12. Remember that the baseline methods RPBS and EPBS still utilize Algorithm 1 to generate borderline methods. As shown in

Tables 6, 7, and 8, DeepDIG significantly outperforms the baseline methods respect to the success rate.

Based on the observations above, we can infer that DeepDIG is an effective method capable of generating borderline instances for different DNNs. Now let's pinpoint the reason for the success of DeepDIG in comparison to baseline approaches. To achieve a good classification performance, DNNs carve out connected decision regions for different classes [8] that are complicatedly intertwined with each other. With this in mind, to generate a borderline sample, RPBS and EPBS form a *simple* trajectory between two samples in the two decision regions r_s and r_t and then perform the greedy binary search (i.e., Algorithm 1). However, given the complexity of the decision regions and their intertwined nature, the binary search along this trajectory is likely to end up in a region other than r_s and r_t i.e., "Fail" in line 8 of Algorithm 1. In contrast, DeepDIG is equipped with two effective components (I) and (II) that make the end-points of its established trajectory very close to the decision boundary between decision regions r_s and r_t. Hence, the binary search for DeepDIG is less prone to end up in a different region other than r_s and r_t and thus higher success rate for DeepDIG is ensued.

6.3 Inter-model Decision Boundary Characterization

Thus far, we have investigated the components of DeepDIG and made sure of their working. We also compared DeepDIG with baseline methods and showed that it outperforms them. Now it is time to utilize DeepDIG to characterize the decision boundary of DNNs. In Section 4, we developed several measures to characterize the decision boundary between the two classes. We utilize borderline instances generated by DeepDIG and compute those measures for the pre-trained models in Table 1. The results are shown in Table 9. To further illustrate how borderline samples are spatially located in the embedding space learned by a DNN, we visualize them along with training and test samples. Figures 4, 5, and 6 show the visualizations for DNNs trained on MNIST, FashionMNIST, and CIFAR10, respectively. To generate these figures, we project the embeddings learned by a DNN to a 2D space using PCA (Principal Component Analysis) [31]. We fit the PCA on the embeddings of the standard train samples and use it in the inference mode to project the embedding of test instances as well as borderline instances[9]. Note that as explained before, for the decision boundary between two classes s and t, DeepDIG generates two sets of borderline instances, namely ones that are approached from s and consequently their labels are s and ones that are approached from class t and their labels are t – See Tables 6, 7, and 8 for some visualizations of these two sets. However, in projections presented in Figures 4, 5, and 6, we show both sets of borderline instances as 'borderline' to signify their near decision boundary property rather than their labels. Based on the results presented in Table 9 as well as Figures 4, 5, and 6, we make the following observations regarding the characteristics of decision boundaries of investigated DNNs.

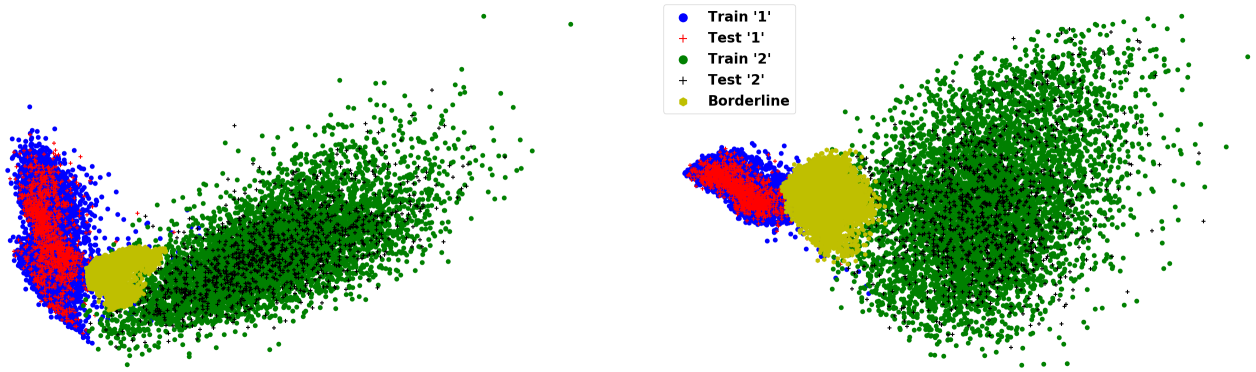
[9]For PCA we use scikit-learn package with `n_components=2` and the other parameter settings as defaults.

Table 9: Results of inter-model decision boundary characterization

	IDC	$EDC1_{Test}$	$EDC1_{Borderline}$	$EDC2_{Test}$	$EDC2_{Borderline}$
$MNIST_{CNN}$	0.037	0.94	0.25	99.18	39.73
$MNIST_{FCN}$	0.018	0.92	0.29	99.72	58.36
$FashionMNIST_{CNN}$	0.035	0.77	0.18	99.40	37.09
$FashionMNIST_{FCN}$	0.017	0.91	0.26	99.01	83.34
$CIFAR10_{ResNet}$	0.035	0.69	0.12	99.45	52.75
$CIFAR10_{GoogleNet}$	0.038	0.55	0.06	99.65	38.51

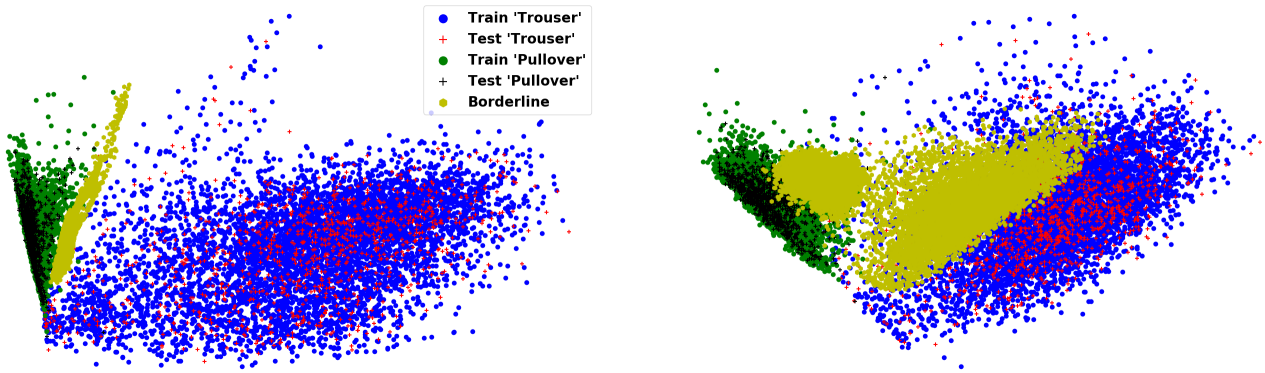
- In Section 4.2, we defined measure EDC1 to determine whether generated borderline instances are near the decision boundary in the embedding space or not. We can observe from Table 9 that borderline samples –compared to unseen test samples– are very close to the separating hyperplane i.e., $EDC1_{Borderline}$ is significantly smaller than $EDC1_{Test}$ for all DNNs. Note that EDC1 is normalized to be in the range $[0, 1]$. This proves our hypothesis that borderline instances are indeed in the separating region between two classes in the embedding space and thus are near the decision boundary in the embedding space. Visualizations in Figures 4, 5, and 6 further corroborate this hypothesis where we can easily observe that borderline samples are between the original samples of two classes. In particular, borderline samples occupy a different region of the embedding space than that of original samples (train and test sets).
- IDC measure is developed to inform us about the complexity of the decision boundary in the input space while EDC2's purpose is the same except in the embedding space. We can observe that these two are not disjoint and there is a strong correlation between these two complexity measures. More specifically, the more complex the decision boundary in the input space is (i.e., a larger value for IDC), the more complex the decision boundary in the embedding space is (i.e., a smaller value for $EDC2_{Borderline}$) and vice versa. To concretely quantify this correlation, we compute the Pearson correlation coefficient[10] between IDC and $EDC2_{Borderline}$. The value is -0.8637 which indicates this is a strong (negative) correlation between IDC and $EDC2_{Borderline}$. To give a frame of reference, Pearson correlation coefficient between IDC and $EDC2_{Test}$ is just 0.1466. Therefore, we reach an important conclusion that *the complexity of the decision boundary formed by a DNN in the input space manifests itself in the embedding space as well.*
- We can observe that a linear model can obtain a perfect accuracy score on the test set ($EDC2_{Test} > 99\%$). The reason is that test samples follow the same distribution with training data as they are surrounded by training data points –See Figures 4, 5, and 6. Borderline samples, in contrast, have a considerably smaller accuracy which is due to again their different distribution than original data. Hence, we can conclude that *the linear separability capability of samples in the*

[10]https://en.wikipedia.org/wiki/Pearson_correlation_coefficient



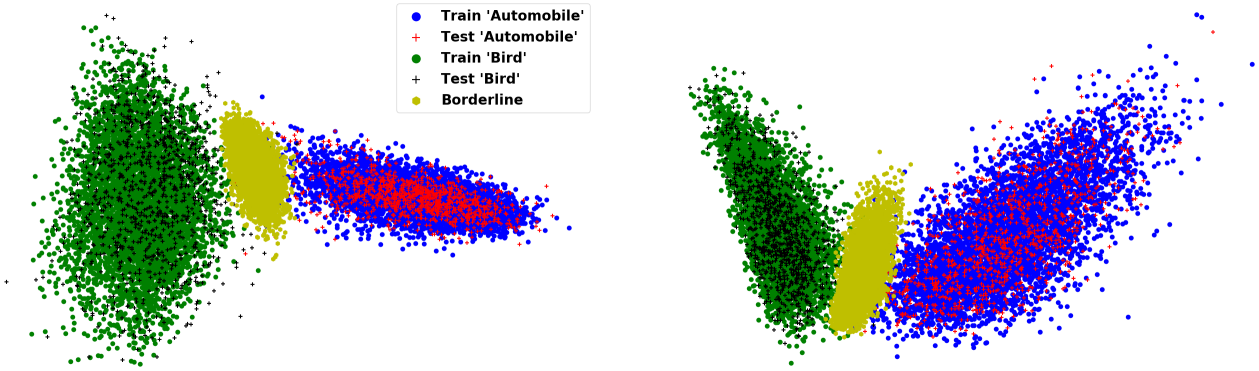
(a) $\text{MNIST}_{\text{FCN}}$ (b) $\text{MNIST}_{\text{CNN}}$

Figure 4: Projection of embeddings of training and test samples as well as borderline instances onto a 2D space (MNIST)



(a) $\text{FashionMNIST}_{\text{FCN}}$ (b) $\text{FashionMNIST}_{\text{CNN}}$

Figure 5: Projection of embeddings of training and test samples as well as borderline instances onto a 2D space (FashionMNIST)



(a) $\text{CIFAR10}_{\text{ResNet}}$ (b) $\text{CIFAR10}_{\text{GoogleNet}}$

Figure 6: Projection of embeddings of training and test samples as well as borderline instances onto a 2D space (CIFAR10)

embedding space learned by a DNN holds as long as samples come from the same distribution with training data.

- CNN architectures are sophisticated methodologies specifically designed to capture salient patterns in images while FCNs consist of simple multi-layer perceptrons. Based on our results, it seems that the capability of extracting complex patterns has caused creating more complex decision boundaries for CNNs compared to FCNs. This has been shown in Table 9 where FCN models have resulted in carving out less complicated decision boundaries than CNNs for MNIST and FashionMNIST datasets. This is particularly evident for FashionMNIST$_{\text{CNN}}$ in Figure 5b wherein borderline instances are complicatedly intertwined with real samples.
- CIFAR10$_{\text{ResNet}}$ forms a less complicated decision boundary than CIFAR10$_{\text{GoogleNet}}$. We speculate this is due to the highly complex structure of GoogleNet [33] and an excessive number of parameters of this model –See Table 1.

Based on the above observations, we make the following conclusion. Although many factors can influence how a DNN establishes a decision boundary e.g., non-linear activation function, regularization, etc, thanks to DeepDIG and further proposed characteristics we can shed light on a DNN and its behavior in a systematic and principled manner. This is particularly useful for the model selection task where it can help us to complement other selection criteria including the common criterion used in this task i.e., the performance on a held-out test set. For instance, while CNN models for MNIST and FashionMNIST (i.e., MNIST$_{\text{CNN}}$ and FashionMNIST$_{\text{CNN}}$, respectively) achieve slightly better performance on the test set than FCN models (i.e., MNIST$_{\text{FCN}}$ and FashionMNIST$_{\text{FCN}}$, respectively) –See Table 1– one might opt to use FCNs due to their simpler decision boundaries. Hence, we believe DeepDIG can help a practitioner/researcher to make a more informed decision regarding developing a deep model.

6.4 Intra-model Decision Boundary Characterization

In Section 6.3 we presented the decision boundary characteristics for several DNNs and a single pair of classes per each dataset. In this part, we investigate the decision boundary characterization for all pairs of classes. To this end, we focus on MNIST$_{\text{CNN}}$ as we achieved similar results for other DNNs. We apply DeepDIG to all pairs of classes in MNIST dataset, namely $\{(s,t)|s,t \in [\text{`0'}, \text{`1'} \cdots \text{`9'}], s \neq t\}$. Note that there are $\binom{10}{2} = 45$ decision boundaries for 10 classes in MNIST dataset. First, we visualize the generated borderline instances to make sure of their quality. Table 10 demonstrates some of the borderline instances (chosen randomly) for all pair-wise classes in model MNIST$_{\text{CNN}}$. As it is evident from this table, DeepDIG manages to generate borderline instances that are visibly very similar to real instances. Now we present the decision boundary characterization results for all pair-wise classes using the measures developed in Section 4. Figure 7 shows the complexity measure IDC for all 45 decision boundaries. Figure 8 shows the results for measure EDC1 i.e., the average distance from the separating hyperplane between two classes in the embedding space for both borderline and test samples. Finally, Figure 9 demonstrates the results for measure EDC2 i.e., the accuracy against the linear SVM in the embedding space for both borderline and test samples. Based on the results presented in these figures, we make the following observations.

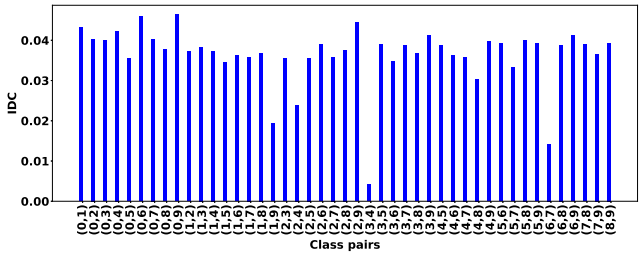


Figure 7: Input space Decision boundary Complexity (IDC) of model MNIST$_{\text{CNN}}$ according to the characteristic measure IDC discussed in Section 4.1

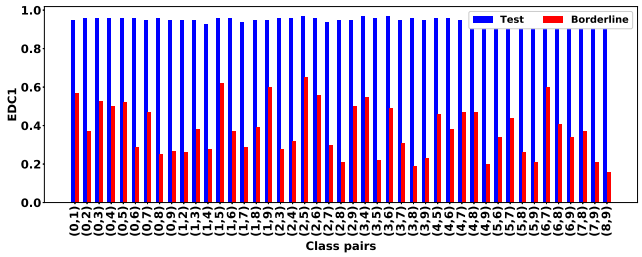


Figure 8: Embedding space Decision boundary Complexity 1 (EDC1) i.e., the distance from the separating hyperplane for all class pairs in model MNIST$_{\text{CNN}}$ according to the characteristic measure EDC1 discussed in Section 4.2

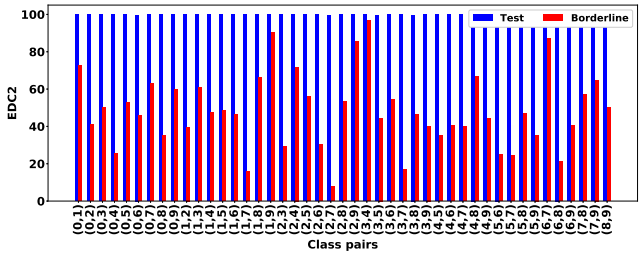
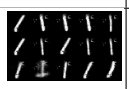
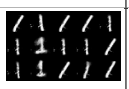
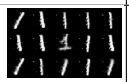
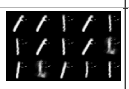
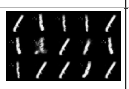
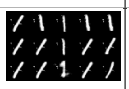
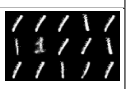
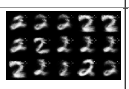
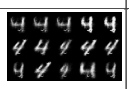
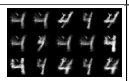
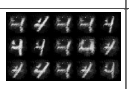


Figure 9: Embedding space Decision boundary Complexity 2 (EDC2) i.e., the accuracy against the linear SVM for all class pairs in model MNIST$_{\text{CNN}}$ according to the characteristic measure EDC2 discussed in Section 4.2

- Similar to what observed in Section 6.4, there is a correlation between IDC and EDC2$_{\text{Borderline}}$. The higher (lower) EDC2$_{\text{Borderline}}$ is, the lower (higher) IDC is. Pearson correlation coefficient between IDC and EDC2$_{\text{Borderline}}$ across all pair-wise classes is -0.5066 whose p-value is 0.000391 and is significant at $p < 0.05$.

Table 10: Illustration of the generated borderline samples for all pair-wise classes in $\text{MNIST}_{\text{CNN}}$

s \ t	'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'9'
'0'	–									
'1'		–								
'2'			–							
'3'				–						
'4'					–					
'5'						–				
'6'							–			
'7'								–		
'8'									–	
'9'										–

- We can observe that different class pairs have different degrees of input space decision boundary complexity (IDC). This depends on how samples are distributed in the input space and how a DNN (here $\text{MNIST}_{\text{CNN}}$) carves out decision regions in this space. Although the exact explanation for this subject (i.e., the sample distribution and model embedding learning) yet to be determined, using the results presented in Figure 7, we can get some insights on how a DNN creates decision regions in the input space. As an example, see the IDC complexity for class '0' against all other classes. The higher value belongs to ('0', '9') while the lowest belongs to the pair ('0', '5'). This seems reasonable since our prior knowledge suggests that digits '0' and '9' have some common patterns e.g., a circle and probably are distributed in a common subspace in the input space whereas image patterns for '5' and '0' are distinct and probably their samples reside in a different subspace.
- Similar reasoning with IDC can be applied to EDC measures as well. The difference in these measures for different class pairs informs us of the degree of difficulty for a DNN to distinguishably project samples of the two classes in the embedding space.

Based on the observations above, we point out a useful use-case of intra-model decision boundary characterization. Usually, deep learning practitioners need to look into the detail of a model and reinforce it against potential future failures. For instance, in some applications, one might be interested to know for which pair of classes a model is more likely to mis-classify samples, so he/she can take actionable measures e.g., adding more samples. In this regard, intra-model decision boundary characterization can provide us with a full profile of the model strengths and weaknesses for all pair-wise classes and potentially can come useful in taking a more guided decision regarding model debugging/reinforcement.

7 RELATED WORK

In recent times, there has been an increasing effort in the machine learning community to propose methods to explain or interpret the results of deep neural networks. We believe one way to better understand DNNs is via taking them out of their "comfort zone" i.e., where there are corner cases for which a DNNs is 'confused' to

make a decisive prediction. In this regard, investigating the decision boundary of deep neural networks is an interesting area. In this section, we review several existing papers that studied the decision boundary of DNNs.

He et al. [10], similar to our approach, utilized adversarial examples and investigated the decision boundary of DNNs. They considered a large neighborhood around adversarial examples and benign samples and then discovered that such neighborhoods have distinct proprieties e.g., in terms of the distance to the decision boundary. In an attempt to bridge theoretical properties and practical power of DNNs, the authors in [23] studied the decision boundary of DNNs. They proved and empirically showed that the last layer of a DNN behaves like a linear SVM. In Section 4.2, we took advantage of this property of DNNs along with their linear separability property and characterized the decision boundary of DNNs in the embedding space. Fawzi et al. [8] studied topology and geometry of DNNs and showed that DNNs carve out complicated and *connected* classification/decision regions. Furthermore, they investigated the curvature of the decision boundary through which they proposed a method to distinguish benign samples from adversarial ones. Authors of [26] made a connection between the adversarial training and decision boundary and further demonstrated that adversarial training helps in decreasing the curvature of the decision boundary. Yousefzadeh and O'Leary [37] conducted a study to investigate the decision boundary of DNNs. Similar to our algorithm in component (III) (i.e., Algorithm 1) they drew a trajectory between two samples at the opposite sides of the decision boundary and then tried to determine what they call "flip points" i.e., borderline instances. Then they analyzed different patterns that emerge from connecting different points at the two sides of the decision boundary. In spirit, their method is similar to the first baseline method we considered in Section 6.2 where we demonstrated that its success rate of generating proper borderline instances is very low for complex multi-class classification problems considered in our experiments. In a recent study, the authors in [1] introduced tropical geometry as a new perspective to study the decision boundary of DNNs. They used their mathematical findings of decision boundary of DNNs for two applications, namely adversarial examples generation and network pruning.

8 CONCLUSION

Although novel DNN architectures are continuously being developed to achieve better and better performance, the understanding of these models has primarily been ignored. One crucial aspect of DNNs that can help us deepen our understanding of their decision-making behavior is their decision boundaries. However, this is fairly unexplored in the machine learning literature, and thus in this work, we embarked upon a research inquiry to study the decision boundary of DNNs and investigated their behaviors through the lens of their decision boundaries. To make this feasible, we proposed a new framework called **Deep D**ecision boundary **I**nstance **G**eneration (**DeepDIG**). DeepDIG utilized an approach based on adversarial example generation and generates two sets of adversarial examples at the opposite sides and near the decision boundary between two classes. Then, aiming at refining and discovering borderline samples, we proposed a method based on the binary search along a trajectory between the two sets of generated adversarial samples. To show the usefulness of DeepDIG, we utilized borderline instances and defined several important measures determining the complexity of the decision boundary between two classes in both input and embedding spaces.

We conducted extensive experiments and demonstrated the working of DeepDIG. First, we showed that DeepDIG –with very high performance– can generate borderline instances that are sufficiently close to the decision boundary. Moreover, we experimented on three datasets and two representative DNNs for each dataset and determined the behavior of different DNNs through the characterization of their decision boundaries. Notably, we bridged between the decision boundary in the input space and the embedding space learned by a DNN. Untimely, we applied DeepDIG on the full range of pair-wise classes of MNIST dataset for a DNN and showed how decision boundaries between different pairs of classes differ. There exist several important directions to follow up in the future:

- DeepDIG while being effective does not approach generating borderline instances in an end-to-end manner. We plan to formulate the borderline instance generation problem in a unified and end-to-end fashion. In particular, we intend to merge the optimization of DeepDIG into a single optimization formulation.
- How to improve the robustness of DNNs against adversarial examples is an emerging and interesting research direction. In this regard, one can integrate borderline instances in the model training e.g., similar to the adversarial training method of [34] and then investigate if the model is getting robust or not. Moreover, comparing decision boundary characteristics of a robust and non-robust model can potentially provide us with insights about the causes of non-robustness.
- DeepDIG was optimized and tested against a continuous data type e.g., images. DeepDIG can be extended to discrete data types as well e.g., texts, graphs, and so on. This is a challenge and needs deliberated considerations. For instance, the distance metric capturing similarity of two samples –see Eq. 1 and Eq. 2– should change appropriately to properly quantify the *similarity* between two discrete data instances. Adapting ideas from adversarial examples generation methods for texts [39] seems like a proper avenue to extend DeepDIG to discrete data types.

REFERENCES

[1] Motasem Alfarra, Adel Bibi, Hasan Hammoud, Mohamed Gaafar, and Bernard Ghanem. 2020. On the Decision Boundaries of Deep Neural Networks: A Tropical Geometry Perspective. https://openreview.net/forum?id=BylldnNFwS

[2] Léon Bottou, Frank E Curtis, and Jorge Nocedal. 2018. Optimization methods for large-scale machine learning. *Siam Review* 60, 2 (2018), 223–311.

[3] Nello Cristianini, John Shawe-Taylor, et al. 2000. *An introduction to support vector machines and other kernel-based learning methods.* Cambridge university press.

[4] Tyler Derr, Hamid Karimi, Xiaorui Liu, Jiejun Xu, and Jiliang Tang. 2019. Deep Adversarial Network Alignment. *arXiv preprint arXiv:1902.10307* (2019).

[5] Simant Dube. 2018. High Dimensional Spaces, Deep Learning and Adversarial Examples. *arXiv preprint arXiv:1801.00634* (2018).

[6] Vincent Dumoulin and Francesco Visin. 2016. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285* (2016).

[7] Wenqi Fan, Tyler Derr, Yao Ma, Jianping Wang, Jiliang Tang, and Qing Li. 2019. Deep Adversarial Social Recommendation. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI'19).* AAAI Press, 1351–1357. http://dl.acm.org/citation.cfm?id=3367032.3367224

[8] Alhussein Fawzi, Seyed-Mohsen Moosavi-Dezfooli, Pascal Frossard, and Stefano Soatto. 2018. Empirical study of the topology and geometry of deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3762–3770.

[9] 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.

[10] Warren He, Bo Li, and Dawn Xiaodong Song. 2018. Decision Boundary Analysis of Adversarial Examples. In *ICLR*.

[11] Tin Kam Ho and Mitra Basu. 2002. Complexity measures of supervised classification problems. *IEEE Transactions on Pattern Analysis & Machine Intelligence* 3 (2002), 289–300.

[12] Brody Huval, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, Mykhaylo Andriluka, Pranav Rajpurkar, Toki Migimatsu, Royce Cheng-Yue, et al. 2015. An empirical evaluation of deep learning on highway driving. *arXiv preprint arXiv:1504.01716* (2015).

[13] Adel Jaouen and Erwan Le Merrer. 2018. zoNNscan: a boundary-entropy index for zone inspection of neural models. *arXiv preprint arXiv:1808.06797* (2018).

[14] Hamid Karimi*, Tyler Derr*, Aaron Brookhouse, and Jiliang Tang. 2019. Multi-Factor Congressional Vote Prediction. In *Advances in Social Networks Analysis and Mining (ASONAM), 2019 IEEE/ACM International Conference on*. IEEE.

[15] H Karimi, T Derr, K Torphy, K Frank, and J Tang. 2019. A Roadmap for Incorporating Online Social Media in Educational Research. *Teachers College Record Year Book* 2019 (2019).

[16] Hamid Karimi, Proteek Roy, Sari Saba-Sadiya, and Jiliang Tang. 2018. Multi-Source Multi-Class Fake News Detection. In *Proceedings of the 27th International Conference on Computational Linguistics*. Association for Computational Linguistics, Santa Fe, New Mexico, USA, 1546–1557. https://www.aclweb.org/anthology/C18-1131

[17] Hamid Karimi and Jiliang Tang. 2019. Learning Hierarchical Discourse-level Structure for Fake News Detection. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 3432–3442. https://doi.org/10.18653/v1/N19-1347

[18] H. Karimi, J. Tang, and Y. Li. 2018. Toward End-to-End Deception Detection in Videos. In *2018 IEEE International Conference on Big Data (Big Data)*. 1278–1283. https://doi.org/10.1109/BigData.2018.8621909

[19] H. Karimi, C. VanDam, L. Ye, and J. Tang. 2018. End-to-End Compromised Account Detection. In *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. 314–321. https://doi.org/10.1109/ASONAM.2018.8508296

[20] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[21] Alex Krizhevsky. 2009. *Learning multiple layers of features from tiny images*. Technical Report. Citeseer.

[22] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[23] Yu Li, Peter Richtarik, Lizhong Ding, and Xin Gao. 2018. On the decision boundary of deep neural networks. *arXiv preprint arXiv:1808.05385* (2018).

[24] Ana C Lorena, Luís PF Garcia, Jens Lehmann, Marcilio CP Souto, and Tin K Ho. 2018. How Complex is your classification problem? A survey on measuring classification complexity. *arXiv preprint arXiv:1808.03591* (2018).

[25] Stéphane Mallat. 2016. Understanding deep convolutional networks. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 374, 2065 (2016), 20150203.

[26] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Jonathan Uesato, and Pascal Frossard. 2019. Robustness via curvature regularization, and vice versa. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 9078–9086.

[27] Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*. 807–814.

[28] Albert Orriols-Puig, Núria Macia, and Tin Kam Ho. 2010. Documentation for the data complexity library in C++. *Universitat Ramon Llull, La Salle* 196 (2010), 1–40.

[29] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*. 8024–8035.

[30] 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.

[31] Jonathon Shlens. 2014. A tutorial on principal component analysis. *arXiv preprint arXiv:1404.1100* (2014).

[32] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. 2017. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*.

[33] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1–9.

[34] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[35] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747* (2017).

[36] Han Xu, Yao Ma, Haochen Liu, Debayan Deb, Hui Liu, Jiliang Tang, and Anil Jain. 2019. Adversarial attacks and defenses in images, graphs and text: A review. *arXiv preprint arXiv:1909.08072* (2019).

[37] Roozbeh Yousefzadeh and Dianne P O'Leary. 2019. Investigating Decision Boundaries of Trained Neural Networks. *arXiv preprint arXiv:1908.02802* (2019).

[38] Xiaoyong Yuan, Pan He, Qile Zhu, Rajendra Rana Bhat, and Xiaolin Li. 2017. Adversarial examples: Attacks and defenses for deep learning. *arXiv preprint arXiv:1712.07107* (2017).

[39] Wei Emma Zhang, Quan Z Sheng, and Ahoud Abdulrahmn F Alhazmi. 2019. Generating Textual Adversarial Examples for Deep Learning Models: A Survey. *arXiv preprint arXiv:1901.06796* (2019).