Freeze and Mutate: Abnormal Sample Identification for DL Applications through Model Core Analyses

Huiyan Wang*
State Key Lab for Novel Software Technology, Nanjing University
Nanjing, China
why@nju.edu.cn

Ziqi Chen
State Key Lab for Novel Software Technology, Nanjing University
Nanjing, China
ziqichen@smail.nju.edu.cn

Chang Xu*
State Key Lab for Novel Software Technology, Nanjing University
Nanjing, China
changxu@nju.edu.cn

ABSTRACT

Deep learning (DL) applications, as a representative form of Software 2.0, are gaining increasing popularity by their intelligent services. However, their reliability depends highly on the prediction accuracy of their integrated DL models. In practice, DL models are often observed to suffer from ill predictions for abnormal inputs (e.g., adversarial attacking samples, out-of-distribution (OOD) samples, and etc.), and could lead to unexpected behaviors or even catastrophic consequences. One promising way to defend the reliability is to reveal such abnormal inputs fed to the DL models integrated in the concerned applications. Existing work addressed this problem by either making sample distance comparisons or generating sufficient model mutants for analysis. This caused a restricted focus on samples only, overlooking the DL models themselves, or had to analyze massive mutants, incurring non-negligible costs. In this paper, we propose NetChopper to first conduct a core analysis on the target DL model, and then partition it into two parts, one associating closely with the training knowledge being the model core (expected to be important and thus stable), and the other being the remaining part (expected to be immaterial and thus changeable). Based on them, NetChopper preserves (or freezes) the model core, but mutates the remaining part to produce only a few model mutants. Later, NetChopper would reveal abnormal inputs from normal ones by exploiting these model-relevant and light-weight mutants. We experimentally evaluated NetChopper by widely-used DL subjects (e.g., MNIST+LeNet4, and CIFAR10+VGG16) and typical abnormal inputs (e.g., adversarial and OOD samples). The results reported NetChopper's promising AUROC in revealing the abnormal degrees of inputs, generally outperforming or comparably effective as state-of-the-art techniques (e.g., mMutant, Surprise, and Mahalanobis), and also its high effectiveness and efficiency (with only marginal online overhead).

CCS CONCEPTS

• **Software and its engineering** → *Software testing and debugging*; • **Computing methodologies** → **Neural networks**.

*Corresponding authors

Internetware '22, June 11–12, 2022, Hohhot, China
© 2022 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

KEYWORDS

Deep learning models, abnormal sample identification, model core analyses, freeze and mutate

ACM Reference Format:
Huiyan Wang, Ziqi Chen, and Chang Xu. 2022. Freeze and Mutate: Abnormal Sample Identification for DL Applications through Model Core Analyses. In *Proceedings of the 13th Asia-Pacific Symposium on Internetware (Internetware '22).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Deep learning (DL) applications, as a representative of Software 2.0 [1], are becoming increasingly popular due to their intelligence and adaptability to many fields, e.g., image classification [2], object identification [3], natural language processing [4], autonomous driving [5], and etc. Many DL applications [6–8] have been developed with an integrated DL model for internal decision-making by prediction. However, unlike Software 1.0 [1] (traditional programs) that are written by programmers and have clear logic and specifications, the logic of Software 2.0 is embedded in the weights of trained models, which are abstract and human unfriendly. At present, the reliability of DL models depends highly on their prediction accuracy. Due to statistical characteristics of DL training, DL models are capable of predicting correctly for input samples in most scenarios, but they could still be observed occasionally to suffer from ill predictions for abnormal samples [9], leading to application misbehaviors or even catastrophic consequences [10]. This problem severely undermines DL models' application in safety-critical scenarios like autonomous driving and malware detection.

Existing work has studied this problem from the perspective of adversarial samples as one of the typical abnormal sample sources. Adversarial samples could be generated by adding carefully crafted human-imperceptible perturbations into original clean samples deliberately, aiming to fool DL models' predictions [9]. Many adversarial attackers (e.g., FGSM [9], C&W [11], DF [12], and etc.) have exhibited a high success rate in fooling DL models and making them predict mistakenly. Besides this, another typical abnormal sample source could be out-of-distribution (OOD) samples. OOD samples, although not necessarily for attacking purposes, follow a distribution different from the training samples that are associated with the target DL model, and can also make the model predict mistakenly. Such abnormal samples (including both adversarial and OOD ones) thus call for effective techniques to recognize and isolate them from normal ones to defend the reliability of DL applications.

To address this problem, some existing work chose to measure and compare sample distances to distinguish them [13–15]. However, they could suffer from tedious and time-consuming distance comparisons and lack necessary efficiency [16]. Some other work proposed to dig into input samples' behaviors during their associated predictions, in order to recognize their differences. For example, mMutant [17], one piece of state-of-the-art work along this line, combines mutation analysis and DL testing to reveal abnormal samples' different label changing rates upon prepared mutants. However, it typically requires a large number of mutants and their analysis is indeed time-consuming, limiting its practical usages. Regarding such efficiency requirements, there is also a line of work (e.g., Surprise [18] and Mahalanobis [19]) that examines an input sample's behavior through the original model's prediction from various granularities and inspects evidence for anomaly, being quite efficient. However, to collect such evidence, the existing work has to obtain a certain amount of abnormal samples in advance for preparation, e.g., for training an additional abnormal identification classifier, and this could sometimes be infeasible in practice.

In this paper, we propose a novel approach, named NetChopper, to first conducting a core analysis on the given DL model, and then partitioning it into two parts. One part is supposed to associate closely with the training knowledge, named *model core* (expected to be important and thus stable), and the other is the remaining part (expected to be immaterial and thus changeable). Based on them, NetChopper chooses to freeze the model core and mutate the remaining part, thus producing a few model mutants. This process could be conducted in a model-related and lightweight way, resulting in a set of mutants useful for later analysis to distinguish the behaviors of abnormal samples from those of normal samples by prediction differences. We believe that NetChopper's novel core analysis and core-based mutation could give a generic way for model interpretation, potentially also useful for similar model analysis and testing problems.

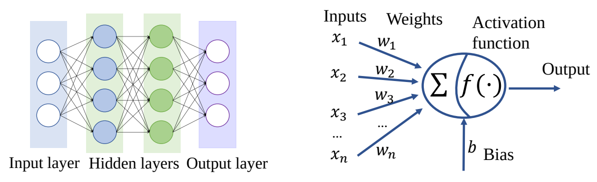
To evaluate NetChopper's performance, we conducted experiments on two widely-used DL subjects (MNIST+LeNet4 and CIFAR10+VGG16), and observed that: (1) NetChopper's reported abnormal scores on abnormal samples and normal samples indeed differed significantly, with AUROC consistently outperforming or comparably effective as existing techniques; (2) NetChopper achieved nice and stable effectiveness in identifying abnormal samples, with an average F_1 score of 0.85 and average accuracy of 0.85, (3) NetChopper also exhibited promising efficiency by taking only 0.11-1.62 minutes during its offline preparation (acceptable) and 0.01–0.97 milliseconds (per sample) during its online sample identification (highly efficient), clearly more satisfactory than existing techniques.

The remainder of this paper is organized as follows. Section 2 introduces the necessary DL background. Section 3 presents our NetChopper technique for effectively identifying abnormal samples for DL applications. Section 4 experimentally evaluates NetChopper's performance, and compares it to state-of-the-art techniques on the effectiveness and efficiency. Finally, Section 5 discusses the related work, and Section 6 concludes this paper.

2 BACKGROUND

2.1 Deep learning

Deep learning is a sub-field of machine learning, which refers to using deep artificial neural networks (ANN) to learn representations of data. The most commonly used ANN is Deep neural network (DNN). A typical DNN consists of an input layer, an output layer, and multiple hidden layers, as shown in Fig. 1(a). Each layer consists of multiple artificial neurons, following the M-P neuron model, as shown in Fig. 1(b). Typically, deep learning includes training and prediction phases. The former one is basically responsible for assigning the most suitable learning parameters (i.e., weights and biases), typically using the backpropagation algorithm [20]. Then, in the prediction phase, when fed by any input sample from the input layer, each neuron would receive signals (i.e. output values) from the neurons in its previous layer, and carry out weighted summation and activation operations for the final prediction, based on assigned parameters in training.



(a) Basic architecture of a DNN (b) M-P neuron model

Figure 1: A DNN architecture and M-P neuron model.

We emphasize on the widely used convolutional neural networks (CNN), which have been popular in many fields, e.g., computer vision, natural language processing, and so on. In addition to the traditionally fully connected layers, a typical CNN usually has convolutional layers and pooling layers, which are mainly responsible for feature extraction and dimension reduction. The local receptive field and shared weights of the convolutional layers achieve the purpose of extracting features and reducing the number of parameters, and the pooling layer further reduces the feature dimension.

2.2 DL testing & abnormal sample identification

Although DL models perform excellently on many tasks, they are also observed to suffer from uncertainty [21] and non-testability [22] problems. This is because different from traditional software, whose decision logic is clearly specified by developers, the decision logic of DL models is derived from a large number of trained parameters with low interpretability. The poor performance of a DL model in a certain application scenario can be attributed to a combination of factors, e.g., the design of model structure, training data, training parameters, or the application scenario is indeed abnormal to the model.

Therefore, analogous to the definition of machine learning (ML) bugs and ML testing in [23], a *DL bug* can be defined as any imperfection in a DL item that causes a discordance between the existing and the required conditions. *DL testing* can be defined as the activity designed to reveal DL bugs. In DL testing, abnormal sample detection is a popular topic to help reveal DL bugs, since abnormal

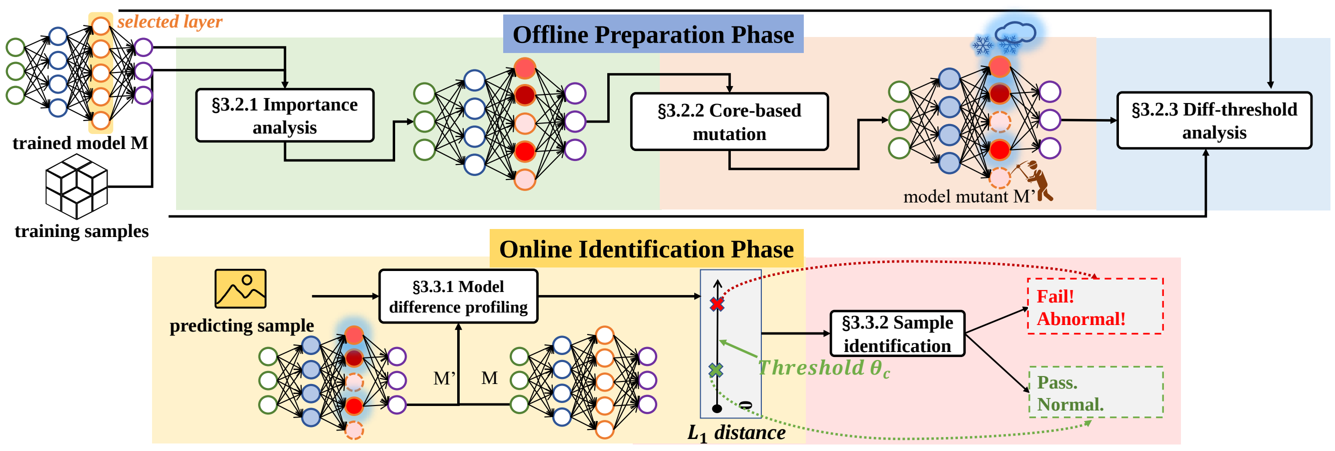

selected layer
Offline Preparation Phase
trained model M
training samples
§3.2.1 Importance analysis
§3.2.2 Core-based mutation
model mutant M'
§3.2.3 Diff-threshold analysis
Online Identification Phase
predicting sample
§3.3.1 Model difference profiling
M'
M
Threshold θ_c
L_1 distance
§3.3.2 Sample identification
Fail! Abnormal!
Pass. Normal.

Figure 2: NetChopper overview.

samples usually work beyond the handling ability of the model. In practice, *adversarial samples* and *out-of-distribution (OOD) samples* are two typical abnormal sample sources. An adversarial example is generated by adding carefully crafted human-imperceptible perturbations into original clean samples deliberately, aiming to fool DL models' predictions. An OOD sample is an input sample following a distribution different from the training samples that are associated with the target DL model, and can also make the model predict mistakenly. Many existing techniques are proposed to effectively identify such abnormal samples from different aspects [19, 24, 25], though there are still limitations on the identification performance.

3 OUR APPROACH

3.1 Approach overview

Our approach consists of two phases, namely, an offline preparation phase and an online identification phase, as shown in Fig. 2. Let a trained model be M and its associated training samples be X.

NetChopper's first phase would: (1) explore model M with samples X to prioritize M's different components (by focusing on the importance of neurons in M), (2) based on the prioritization, partition model M into two parts, one associated closely with the training knowledge (with respect to samples X), named *model core* (expected important and stable), and the other that remains from M (expected immaterial and changeable), we then freeze the model core and mutate the remaining part to produce a model mutant M', and finally (3) calculate the upper bound of training samples X's corresponding prediction differences between M and M' as an appropriate threshold to separate abnormal samples from normal samples.

With such preparations, for any new input sample, NetChopper can then conduct its abnormal sample identification. That is, for a given sample s, NetChopper's second phase would: (1) profile sample s's prediction differences upon M and the mutant M', (2) with the pre-calculated cutting threshold θ_c, determine sample s to be "abnormal" ($> \theta_c$) or "normal" ($\leq \theta_c$).

We elaborate on the two phases in turn below.

3.2 Phase 1: Offline preparation phase

3.2.1 Neuron importance analysis. Considering a trained model M, and its associated training samples X, NetChopper would first conduct a neuron importance analysis upon a selected layer. NetChopper uses Layer-wise Relevance Propagation (LRP) [26] method to calculate the importance of neurons. It propagates the prediction result backward in the neural network, by means of purposely designed local propagation rules. There are several different LRP propagation rules, e.g., LRP-0, LRP-ϵ, LRP-γ. NetChopper chooses LRP-ϵ to avoid noises and get sparser analysis results. More elaborately, for a sample x, let j and k be neurons at two consecutive layers of the neural network, neuron j's LRP relevance score R_j^x is given by:

$$R_j^x = \sum_k \frac{a_j^x w_{jk}}{\epsilon + \sum_{0,j} a_j^x w_{jk}} R_k^x, \tag{1}$$

where a_j^x represents the output of neuron j, w_{jk} represents the weight between neuron j and k, ϵ is set to 0.25 as recommended in [26]. For training samples X, NetChopper calculates the sum of absolute values of LRP relevance scores as the neuron importance score $Imp(j)$ of the dataset:

$$Imp(j) = \sum_{x \in X} \left| R_j^x \right| . \tag{2}$$

Except the input and output layer, NetChopper does not restrict which layer to select. We would study the effect of layer selections in our later evaluation.

For the model trained for classification tasks, NetChopper calculates neuron importance scores separately for the subset of each class to conduct a finer analysis. In addition, when the training dataset is large, a randomly sampled subset of the training dataset can replace the entire dataset to improve efficiency. And for some practical cases without available training samples, NetChopper's offline analysis could also be fed with generally normal samples, which are expected to be diverse and representative.

3.2.2 Core-based mutation. In the last step, we have measured neuron importance scores in M. Based on the prioritization of those neurons' importance scores, NetChopper would partition the model into two parts: one part associated closely with the training knowledge, named *model core* (expected to be important and thus stable), and the remaining part (expected to be immaterial and thus changeable). Generally, NetChopper would treat those neurons with high importance scores as "*core neurons*", and extract them into the model core. To do so, NetChopper would control the proportion of neurons in each layer that can be considered as core neurons in practice, by setting a built-in *core percentage* (p_{core}). Then, any neuron belonging to the model core would be "*frozen*", i.e., no further mutation allowed, in order to ensure that the core knowledge learned from training can be preserved during the mutation. In practice, p_{core} can be empirically set to gradually increase until a certain amount of training samples (<1%) no longer hold their original predictions on generated mutants. In our later evaluation, we would also investigate the impacts of different core percentages on NetChopper's performance.

After freezing the model core, NetChopper then arbitrarily mutates neurons belonging to the remaining part, which are believed to be immaterial and thus changeable, and their diverse mutations are expected to help reveal different behaviors of abnormal samples.

One straightforward mutation treatment is to deactivate these neurons, i.e., set their outputs to be zero, suggesting removing their corresponding influences on the prediction. That is:

$$a'_j = \begin{cases} a_j & \textit{j is a core neuron,} \\ 0 & \textit{otherwise. //mutate} \end{cases} \tag{3}$$

In practice, this model mutant can be implemented as inserting a mask layer after the selected layer of the original model, where the outputs of core neurons are multiplied by 1 and the rest by 0.

Note that, since we analyze neuron importance scores of each class for classification models as aforementioned, the number of model mutants should be equal to the number of classes theoretically. To reduce storage cost, when the number of classes is large, we can implement only one mutant and store the different weights of the mask layer, and replace the weights.

3.2.3 Diff-threshold analysis. For the original model M, let one generated model mutant be M'. In order to leverage the expected and unexpected behavioral differences between such two models to help identify abnormal samples, we investigate the prediction differences of obviously normal samples (i.e., training samples with correct predictions). Specifically, we measure the L_1 distance of model outputs between M and M' as the *abnormal score* of a input sample. For a sample x, we have:

$$Diff(x, M, M') = \left\|M(x) - M'(x)\right\|_1 . \tag{4}$$

We also observe that NetChopper's importance calculation method are reasonable, i.e, the neurons we mutate do not have much effect on the normal samples' predictions in a preliminary experiment on a LeNet4 [27] model with MNIST [28] dataset (see 4.2). In this case, when we mutate 10% neurons in turn selected according to the increasing order of neuron's importance sores (from unimportant to important) and construct the corresponding mutants, the average $Diff()$ score of 500 random training samples strictly increases. Especially, when we select to mutate the most important 10% of neurons, there is an obviously huge impact on the average $Diff()$ score than other selections, suggesting only a small amount of neurons might dominate such predictions. This also suggests NetChopper's reasonable ranking for the importance of neurons.

Note that, we do not directly adopt samples' label changing results like existing work [17], since NetChopper's mutation is quite delicate and might not always propagate to the prediction label. After that, for all samples in M's training (i.e., X), we can obtain a series of their $Diff()$ values. We then extract the *upper bound* of $Diff()$ values as an appropriate threshold θ_c to distinguish between normal and abnormal samples. In order not to be affected by outliers, NetChopper refers to the calculation method of upper bound in drawing boxplots [29]. Therefore, instead of using abnormal samples to train an additional classifier like existing work [18, 19], NetChopper obtains the thresholds automatically.

3.3 Phase 2: Online identification phase

After generating model mutant(s) (e.g., M'), and obtaining the corresponding threshold (e.g., θ_c), NetChopper can now conduct the online identification for any collected sample. Suppose an input sample x, and NetChopper would conduct the abnormal sample identification as follows.

3.3.1 Model difference profiling. For sample x, NetChopper can similarly collect its prediction difference i.e., $Diff(x, M, M')$ between the original model M and the model mutant M', as shown in Equation 4. For classification tasks, suppose the model predicts x as class i, the difference is calculated between M and M'_i, i.e., the model mutant whose neuron importance is calculated by training samples from class i. It means that despite being predicted as class i, NetChopper can distinguish between normal class i samples and abnormal ones using the prediction differences. Therefore, unlike existing work [17] that feeds samples into a number of mutants, each sample only needs one model mutant to calculate the difference score in NetChopper.

3.3.2 Sample identification. After that, based on the obtained threshold for M' (i.e., θ_c), and the sample x's prediction difference $Diff(x, M, M')$, NetChopper can easily determine sample s to be "abnormal" ($Diff(x, M, M') > \theta_c$) or "normal" ($Diff(x, M, M') \leq \theta_c$).

4 EVALUATION

In this section, we experimentally evaluate NetChopper and compare it to state-of-the-art techniques (i.e., mMutant, Surprise, and Mahalanobis) for their performance in identifying abnormal samples for DL applications.

4.1 Research questions

We aim to answer the following four research questions:

RQ1 (Distinguishing ability): Can the abnormal scores generated by NetChopper distinguish abnormal samples from normal samples effectively, as compared to existing techniques?

RQ2 (Detection effectiveness): How effective is NetChopper in identifying abnormal samples?

RQ3 (Controlling factors): How do NetChopper's parameters (e.g., core percentage p_{core} and layer selection) affect its effectiveness?

Table 1: Descriptions for datasets and associated DL models.

Dataset	# samples	DL model	Accuracy
MNIST	60,000/10,000	LeNet4	98.52%
CIFAR10	50,000/10,000	VGG16	92.47%

Table 2: Descriptions for TYPE-I abnormal samples.

Sub.	Attacker	Setups	Attacking success (%)
MNI -ST	FGSM	$\epsilon = 0.3$	98.63% (9,718/9,852)
	JSMA	$\theta = 1, \gamma = 0.1$	100% (9,852/9,852)
	C&W	binary_search_steps=10, max_iter=5000, initial_const=0.01	100% (9,852/9,852)
	DF	overshoot=0.02, max_iter=50	99.42% (9,795/9,852)
	BIM	ϵ = 0.3, eps_step=0.03	100% (9,852/9,852)
CIFAR -10	FGSM	$\epsilon = 0.02$	80.86% (7,477/9,247)
	JSMA	$\theta = 1, \gamma = 0.1$	100% (9,247/9,247)
	C&W	binary_search_steps=10, max_iter=5000, initial_const=0.01	100% (9,247/9,247)
	DF	overshoot=0.02, max_iter=50	99.94% (9,242/9,247)
	BIM	ϵ = 0.01, eps_step=0.005	88.82% (82,13/9,247)

RQ4 (Efficiency): How efficient is NetChopper in identifying abnormal samples (during its offline preparation and online identification), as compared to existing techniques?

4.2 Experimental design and setup

We introduce experimental subjects, design, and implementation in turn below.

Experimental subjects. We used two popular image classification datasets in the DL field as our experimental subjects, namely, MNIST [28] and CIFAR10 [30], each associated with a state-of-the-art DL model, as shown in Table 1. *MNIST* is an image database for hand-writing digit classification (with ten labels), which contains 60,000 training samples and 10,000 predicting samples for testing. Its associated DL model is *LeNet4* [27]. *CIFAR10* is another image database for object recognition (also with ten labels), which contains 50,000 training samples and 10,000 predicting samples. Its associated DL model is *VGG16* [31]. Within the scope of our experiments, we consider the predicting samples with correct predictions from each dataset as "normal" (as contrast to "abnormal" ones with incorrect predictions generated from adversarial attacking or OOD, as explained later soon).

In order to evaluate NetChopper's performance on abnormal sample identification, for each dataset, we prepared two popular types of abnormal samples that have been intensively studied in existing work [16], i.e., adversarial samples (as *TYPE-I*), and out-of-distribution samples (as *TYPE-II*). TYPE-I abnormal samples could be constructed by adding carefully crafted human imperceptible perturbations into original samples deliberately, aiming to fool a DL model's prediction, while TYPE-II abnormal samples refer to those that are clearly under a different distribution from that of a DL model's training samples.

Regarding TYPE-I abnormal samples (adversarial), for each subject, we adopted five popular adversarial attackers upon its predicting samples with correct predictions, in order to deliberately fool the concerned model by changed predictions. These attackers are FGSM [9], JSMA [32], C&W (ℓ_2-norm) [11], DF [12], and BIM [33], used by following their typical settings. These attackers may also need certain setups, e.g., setting an internal attacking step, and we list such attacking details in Table 2. Note that we generate TYPE-I abnormal samples by selecting those samples that can originally be predicted correctly, but later be predicted incorrectly after the attacking. For example, considering the MNIST subject and the FGSM attacker, 9,718 abnormal samples were obtained based on all MNIST's predicting samples with correct predictions (9,852), suggesting an attacking success rate of 98.63%.

Table 3: Descriptions for TYPE-II abnormal samples.

Subject	OOD	# samples	Description
MNIST	FMNIST	10,000	Clothing classification
	EMNIST	10,000	Digit/Letter classification
CIFAR10	CIFAR100	10,000	100 classes object recognition
	SVHN	10,000	Street view recognition

Regarding TYPE-II abnormal samples (OOD), for each subject, we adopted two OOD datasets that had been well studied in existing OOD research [34], namely, FASHION_MNIST [35] (FMNIST for short) and EMNIST [36] for MNIST, and CIFAR100 [30] and SVHN [37] for CIFAR10. Note that we consider all predicting samples from these four OOD datasets as "abnormal", since they were from other datasets completely different from our subjects (designed even for totally different classification tasks). We give these details in Table 3.

Experimental design. We design the following independent variables to control the experiments:

- *Subject.* We used two subjects, each concerning a dataset and a DL model, namely, MNIST + LeNet4, and CIFAR10 + VGG16. When with no ambiguity, we refer to each by the dataset name only.
- *Abnormal type.* For each subject, we generated both TYPE-I and TYPE-II abnormal samples as explained earlier. Concerning TYPE-I abnormal samples, we used five attackers, i.e., FGSM, JSMA, C&W, DF, and BIM. Concerning TYPE-II abnormal samples, we leveraged FMNIST and EMNIST for MNIST, and CIFAR100 and SVHN for CIFAR10, respectively.
- *Core percentage (p_{core}).* To investigate the impacts of NetChopper's different core percentages in identifying model cores, we controlled p_{core} to take a value of 0.05, 0.10, …, or 0.95, with a pace of 0.05.
- *Selected layer.* We investigate the impacts of different layer selections for NetChopper. Note that for the convolution part of a CNN, we take a convolution block as a unit to mutate, i.e., we operate on the outputs of the selected block. For MNIST+LeNet4 which has two convolution blocks and two fully-connected layers, we tried all the block/layers except the input and layer, namely, block1, block2, and fc1. For CIFAR10+VGG16 which has five convolution blocks and three fully-connected layers, we evenly selected three block/layers from different depths, namely, block2, block4, and fc1.
- *Techniques.* We also compare NetChopper with three state-of-the-art abnormal sample identification techniques, namely,

Table 4: Distinguishing ability (AUROC score) comparisons among NetChopper, mMutant, Surprise, and Mahalanobis.

Subject	Description	NetChopper	mMutant-GF	mMutant-NAI	Surprise-LSA	Mahalanobis
MNIST	I (FGSM)	0.9961	0.9840	0.8322	0.8659	0.6819
	I (JSMA)	0.9649	0.8508	0.9993	0.9986	0.8756
	I (DF)	0.9988	0.7380	0.9973	0.9967	0.6776
	I (C&W)	0.9966	0.7147	0.9985	0.9976	0.6926
	I (BIM)	0.9678	0.9597	0.4395	0.5502	0.9728
	II (EMNIST)	0.9446	0.8114	0.8931	0.8618	0.9147
	II (FMNIST)	0.9975	0.9953	0.8958	0.8746	0.9823
CIFAR10	I (FGSM)	0.8321	0.8243	0.7493	0.8262	0.8460
	I (JSMA)	0.9827	0.9659	0.9968	0.9939	0.9870
	I (DF)	0.9928	0.9997	0.9600	0.9517	0.9987
	I (C&W)	0.9771	0.9187	0.9992	0.9913	0.9554
	I (BIM)	0.5865	0.6654	0.6433	0.5981	0.6120
	II (CIFAR100)	0.9026	0.8562	0.8408	0.9129	0.9251
	II (SVHN)	0.9124	0.8745	0.8145	0.9503	0.9517

[1] the closer the color of a grid is to red, the higher the AUROC score, and the closer it is to green, the lower the score.

mMutant [17], Surprise [18], and Mahalanobis [19]. Concerning *mMutant*, which originally has four variants, we adopted two of its variants (mMutant-NAI and mMutant-GF), as they exhibited the best performance [17]. Concerning *Surprise*, which originally has two variants (Surprise-LSA and Surprise-DSA), we adopted Surprise-LSA since both variants exhibited comparable performance [18]. *Mahalanobis* [19] was configured to use its original setting [19]. We used original implementations of these three techniques, or slightly adapted them to identify samples into "abnormal" and "normal" two categories for experimental purposes.

The four techniques need some setups: (1) NetChopper needs to select layers for its model mutation. We selected block1 for MNIST and block4 for CIFAR10 to generate core-based model mutants. Besides, by default, we set p_{core} to be 0.95 for both subjects. We also investigated these factors' impacts in RQ3. (2) mMutant needs to configure a mutation degree. This parameter was set to 0.05 for MNIST and 0.005 for CIFAR10 as suggested [17]. For each mMutant variant, 200 mutants were generated, and its parameters for MNIST and CIFAR10 on classifying abnormal and normal samples were as suggested [17]. (3) Surprise [18] needs to set a variance threshold for removing neurons. It was set to 10^{-5} for MNIST as suggested [17]. For CIFAR10, the suggested value 10^{-4} would cause "NaN" when calculating kernel density estimation score, so we had to expand this value to 10^{-3}. The original paper does not clarify which kind of layer LSA is more suitable, so we similarly chose the same layer as NetChopper for its analysis. (4) Mahalanobis [19] needs to configure a noise magnitude for input enhancement. This parameter was set to 0 for simplicity. For layer selection, we chose all blocks' outputs to calculate Mahalanobis scores as suggested. (5) Both Surprise [18] and Mahalanobis [19] additionally need normal and abnormal samples in hand for training their internal classifiers, and thus we randomly selected 1,000 samples (around 10% as suggested [18, 19]) for this purpose.

As mentioned earlier, we regard the samples with correct predictions and from original MNIST/CIFAR10's predicting samples as "normal", and those generated by TYPE-I/TYPE-II treatments as "abnormal". We mixed normal and abnormal samples together for experiments. For example, concerning the MNIST subject and the FGSM type, we mixed all normal samples (9,852) from MNIST's predicting samples with correct predictions and all generated TYPE-I abnormal samples with wrong predictions (9,718) by FGSM together.

Then, for evaluating these techniques' performance (effectiveness and efficiency), we design the following metrics.

For the effectiveness, we used the *area under the receiver operating characteristic* (AUROC) to measure the distinguishing ability of sample scores generated by convention. And we measured the detection effectiveness of NetChopper by *precision*, *recall*, F_1*-score* and *prediction accuracy*, which are commonly used in machine learning field to measure the effectiveness of binary classifiers.

For the efficiency, we recorded the time spent by each technique on its offline preparation overhead and online identification overhead. For example, for NetChopper, its offline preparation overhead refers to the time spent on its model mutant preparation, and its online identification overhead refers to the time spent on its runtime sample analysis and identification based on the prepared model mutants.

Implementation. We conducted all experiments on a Linux server with two Intel(R) Xeon(R) Gold 5118 CPU @2.30GHz, 10 GeForce RTX 2080Ti GPUs, and 384GB RAMs, running Ubuntu 16.04.

4.3 Experimental results and analyses

4.3.1 RQ1 (Distinguishing ability). This question studies the ability to discriminate between abnormal samples and normal samples of the abnormal scores reported by NetChopper (i.e., $Diff()$ scores), as compared to existing techniques. We calculated the abnormal scores on each mixed dataset using NetChopper and three existing techniques. We list the corresponding AUROC results in Table 4.

From the table, we observed and analyzed that: (1) in most cases, the NetChopper' s abnormal scores had a strong distinguishing ability towards abnormal samples, leading to an average AUROC of 0.9809 for MNIST and 0.8837 for CIFAR10. (2) compared to the other three techniques, on MNIST, NetChopper had a more consistent performance with AUROC all greater than 0.94, while the other methods performed extremely well on some datasets (e.g., mMutant-NAI got AUROC of 0.9993 on JSMA), but their worst values are as low as around 0.5. On CIFAR10, although overall Mahalanobis slightly outperformed us on most datasets, note that its abnormal scores were derived from the logistic regression classifier trained with abnormal samples, while NetChopper did not use

Table 5: Detection effectiveness of NetChopper.

Subject	Description	precision	recall	F_1	accuracy
MNIST	I (FGSM)	0.84	1.00	0.91	0.91
	I (JSMA)	0.84	0.97	0.90	0.89
	I (DF)	0.84	1.00	0.91	0.91
	I (C&W)	0.84	1.00	0.91	0.91
	I (BIM)	0.84	0.99	0.91	0.90
	II (EMNIST)	0.83	0.89	0.86	0.85
	II (FMNIST)	0.84	1.00	0.92	0.91
CIFAR10	I (FGSM)	0.77	0.71	0.74	0.75
	I (JSMA)	0.82	1.00	0.90	0.89
	I (DF)	0.82	1.00	0.90	0.89
	I (C&W)	0.82	1.00	0.90	0.89
	I (BIM)	0.57	0.29	0.39	0.53
	II (CIFAR100)	0.82	0.88	0.85	0.83
	II (SVHN)	0.83	0.95	0.88	0.87

any abnormal samples in advance. mMutant and Surprise also did pretty well in some data sets (e.g., JSMA and C&W), but considering their relatively large overhead as studied later, they may not be that reliable. Therefore, NetChopper had comparable performance with these comparison techniques on CIFAR10. (3) it is worth mentioning that all methods performed poorly on BIM samples with CIFAR10 model. This may be related to the structure of VGG16 and the attack intensity of BIM attack, which is worth further study.

Therefore, we answer RQ1 as follows: *NetChopper's measured abnormal scores on input samples can clearly distinguish abnormal and normal ones, and generally outperformed the existing techniques or had a comparable performance to them, supporting NetChopper's application to abnormal sample identification.*

4.3.2 RQ2 (Detection effectiveness). Based on abnormal scores, NetChopper automatically calculates a threshold (θ_c) to determine whether an input sample is "abnormal" or "normal", as described in 3.2.3. This research question studies how effective NetChopper is in identifying abnormal samples. We measured it by calculating the precision, recall, F_1 score, and prediction accuracy of NetChopper on mixed datasets, as shown in Table 5.

From Table 5, we observed that NetChopper achieved a nice balance between the sample identification precision and recall, leading to a satisfactory F_1 score of 0.86–0.92 and accuracy of 0.85–0.91 for MNIST. For CIFAR10, NetChopper also performed well on 6 out of 7 datasets with a F_1 score of 0.73–0.92 and accuracy of 0.75–0.89. NetChopper only had a poor performance on BIM dataset, which was consistent with the result of RQ1 and has been explained. Generally, the detection results of NetChopper were consistent with the distinguishing ability in RQ1, which indicated that NetChopper's threshold analysis method is effective.

Therefore, we answer RQ2 as follows: *NetChopper was generally effective in identifying abnormal samples from normal ones, and achieved an average F_1 score of 0.85 and average accuracy of 0.85.*

4.3.3 RQ3 (Controlling factors). We next study how NetChopper's effectiveness could be affected by its controlling factors (i.e., core percentage p_{core} and layer selection for mutation).

Core percentage. We controlled to set different values to NetChopper's internal core percentage p_{core}, which denotes the proportion of core neurons preserved in the selected layer of the model, as well as affecting how many neurons could be left for the mutation to

Table 6: Effectiveness results of layer selections (MNIST).

Description	block1	block2	fc1
	$p/r/F_1/a$	$p/r/F_1/a$	$p/r/F_1/a$
I (FGSM)	0.84/1.00/**0.91**/**0.91**	0.84/0.94/0.89/0.88	0.64/0.41/0.50/0.59
I (JSMA)	0.84/0.97/0.90/0.89	0.85/1.00/**0.92**/**0.91**	0.71/0.56/0.63/0.67
I (DF)	0.84/1.00/0.91/**0.91**	0.85/1.00/**0.92**/**0.91**	0.70/0.53/0.60/0.65
I (C&W)	0.84/1.00/0.91/**0.91**	0.85/1.00/**0.92**/**0.91**	0.69/0.50/0.58/0.64
I (BIM)	0.84/0.99/**0.91**/**0.90**	0.73/0.49/0.59/0.66	0.52/0.25/0.34/0.51
II (EMNIST)	0.83/0.89/**0.86**/**0.85**	0.83/0.87/0.85/0.84	0.65/0.42/0.51/0.59
II (FMNIST)	0.84/1.00/**0.92**/**0.91**	0.85/0.99/**0.92**/**0.91**	0.64/0.39/0.49/0.58

[1] $p/r/F_1/a$ refers to $precision/recall/F_1/accuracy$ values.

Table 7: Effectiveness results of layer selections (CIFAR10).

Description	block2	block4	fc1
	$p/r/F_1/a$	$p/r/F_1/a$	$p/r/F_1/a$
I (FGSM)	0.77/0.70/**0.73**/**0.74**	0.77/0.70/**0.73**/**0.74**	0.78/0.69/**0.73**/**0.74**
I (JSMA)	0.82/1.00/0.90/0.89	0.82/1.00/0.90/0.89	0.83/1.00/**0.91**/**0.90**
I (DF)	0.82/1.00/0.90/0.89	0.82/1.00/0.90/0.89	0.83/1.00/**0.91**/**0.90**
I (C&W)	0.82/0.99/0.90/0.89	0.82/0.99/0.90/0.89	0.83/1.00/**0.91**/**0.90**
I (BIM)	0.57/0.28/**0.38**/**0.53**	0.57/0.28/**0.38**/**0.53**	0.57/0.26/0.36/**0.53**
II (CIFAR100)	0.82/0.88/**0.84**/**0.83**	0.82/0.88/**0.84**/**0.83**	0.82/0.85/**0.84**/**0.83**
II (SVHN)	0.83/0.95/0.88/**0.87**	0.83/0.95/0.88/**0.87**	0.84/0.94/**0.89**/**0.87**

generate new model mutants. To save space, we only show two representative abnormal datasets for each subject (FGSM from TYPE-I, FMNIST and SVHN from TYPE-II), and the results of other datasets were similar. We list the results in Fig 3.

From the figure, we observed that in most cases, when p_{core} increased, suggesting that more model neurons belonging to "core" and would not be mutated, the corresponding effectiveness metrics generally increased steeply at first then became stable. This is understandable since when p_{core} was set to a small value, NetChopper's mutation could happen to almost any neuron in the model, and this would bring uncontrollable impact to the model. Then when the p_{core} value increased, more core neurons were preserved, and correspondingly more core knowledge learned from training was preserved. This contributed to a controllable mutation and stable result. As such, the impact when the p_{core} value was small could be a little unpredictable. Still, when the p_{core} value was set over 60%, its trend started to behave similarly as expected. From this figure, we thus set 95% as the default p_{core} value for our both subjects.

Layer selection. Next, we consider different layers in DL models for selection in NetChopper. Table 6 and Table 7 show the results of representative layers from different depths with respect to subject MNIST and CIFAR10. From these tables, we observe that: (1) on MNIST, the effectiveness of NetChopper was sensitive to the layer selections. For example, the F_1 score and accuracy on layer fc1 was significantly lower than those on block1 and block2. For FGSM and BIM, this sensitivity was even more obvious. (2) on CIFAR10, however, the effectiveness of NetChopper seemed not sensitive to the layer selections, because the results were almost the same at different depths of layers in Table 7. This may be related to the more robust architecture of VGG16. (3) generally, for simple neural network architectures, a lower layer is preferred. For complicated neural network architectures, NetChopper does not restrict the layer selections.

As a summary, we answer RQ3 as follows: *NetChopper's internal factors could affect its effectiveness, and although they sometimes affect NetChopper's stableness, its effectiveness generally holds.*

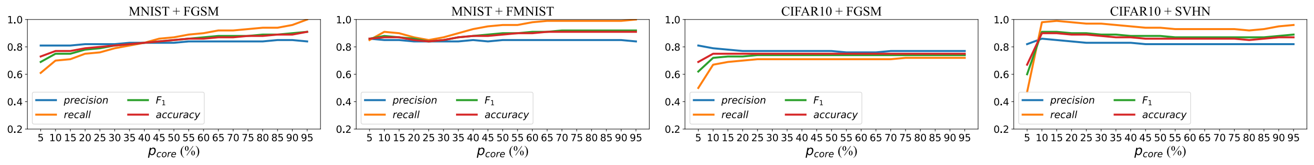


Figure 3: Effectiveness results of different p_{core} values.

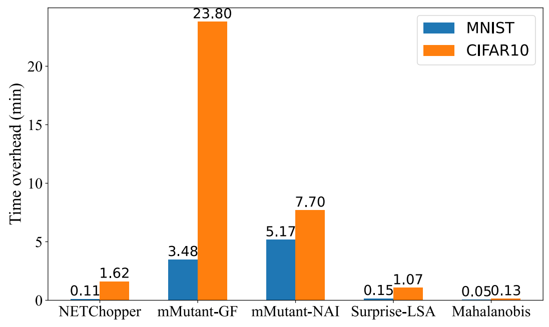


Figure 4: Comparisons on offline overhead (min).

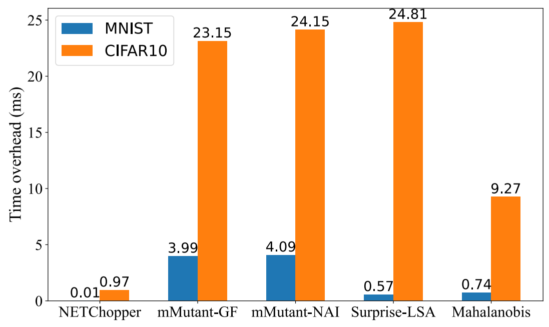


Figure 5: Comparisons on online overhead (ms).

4.3.4 RQ4 (Efficiency). We finally evaluate NetChopper's efficiency in identifying abnormal samples by focusing on its time overheads in the offline mutation preparation and online identification, and compare them to those of the other three techniques. We give the results in Fig. 4 and Fig. 5.

From the figures, we observe that: (1) regarding the offline overhead, two mMutant variants took significantly more time (3.48 and 5.17 minutes for MNIST, and 23.80 and 7.70 minutes for CIFAR10) than the other three techniques (e.g., 0.11 and 1.62 minutes for NetChopper, 0.15 and 1.07 minutes for Surprise, and 0.05 and 0.13 minutes for Mahalanobis, on MINST and CIFAR10, respectively); (2) regarding the online overhead (more importantly), NetChopper performed the best by costing significantly less time than the other three techniques, which only took 0.01 and 0.97 milliseconds for MNIST and CIFAR10 per sample. This is because for each sample, NetChopper only needs to feed it into two models (i.e., the original model and the model mutant and obtains their prediction difference to compare with the threshold θ_c. However, mMutant needs to feed it into hundreds of models. Surprise and Mahalanobis need to obtain and operate the intermediate outputs of the model. Therefore, they consumed more time than NetChopper. (3) summed up, NetChopper could be considered to both exhibit excellent efficiency (acceptable offline overhead and marginal online overhead); Therefore, we answer RQ4 as follows: *NetChopper was very efficient with acceptable offline overhead and only marginal online overhead, exhibiting promising potential among studied techniques.*

4.4 Threat Analysis and Discussion

Our experiments used only two datasets, namely, MNIST and CIFAR10, and this might threaten the external validity of our experimental conclusions. We alleviated this threat as follows. First, our selected datasets have been typically used in relevant research [34, 38], and for each dataset, we generated rich abnormal sample types, including five adversarial and two OOD ones, trying to make them diverse and representative. For comparisons, we selected three latest and highly related techniques (mMutant, Surprise, and Mahalanobis) for experimental effectiveness comparisons, representative as mutation analysis, distribution analysis, and distance analysis techniques (from both the SE and AI communities). We believe that our experimental conclusions should generally hold, and will test on diverse datasets and compare to more techniques in future.

5 RELATED WORK

Our work mainly relates to work from three aspects, namely, DL testing, abnormal sample identification, and model comprehension and interpretation.

DL testing. Testing has exhibited its unique importance on ensuring applications' reliability. Concerning DL testing, existing work mainly focuses on proposing diverse test adequacy criteria and generating effective test inputs. Inspired by traditional code coverage criteria, Pei et al. [39] first proposed neuron coverage (NC) in DL to measure the activated neurons by test inputs. Later, Ma et al. [40] proposed DeepGuage, a more fine-grained testing criterion, to both measure a DL model's functionality and corner cases in neuron and layer level. Kim et al. [18] then introduced surprise adequacy to calculate how surprising a test input is to a DL model with respect to its training dataset through kernel density estimation and distance-based methods. Gerasimou et al. [41] proposed an importance-driven test adequacy criterion (IDC) to evaluate the semantic diversity of a test set. Along this line, some work proposes to effectively generate inputs to meet a higher coverage. Pei et al. [39] used joint optimization to maximize NC while also exposing as many differences between multiple similar DL systems. Subsequent work [42–44] used coverage-guided fuzzing to generate inputs with various heuristic strategies. However, such inputs generated by these methods could be found unnatural [45]. However, there is also some debate on whether a higher coverage denotes better model reliability or robustness. For example, Li et al. [46] pointed out that the structural coverage criteria could sometimes be misleading due to no strong correlation observed between misclassified inputs in a test set and their corresponding structural coverage metrics.

Abnormal sample identification. There is also a line of work, that emphasizes on identifying abnormal sample effectiveness, so as to ensure DL applications' ability at runtime. Early methods [47–49] focused on empirical differences between clean and perturbed samples, which are easy to bypass by new attacks [50]. Some other work chose to train abnormal sample detectors directly [25, 51, 52], therefore they usually required abnormal samples in advance for training. Some other work [24, 53] tried to transform inputs to eliminate some impact of attacks, which could be model-independent and neglect the ability of a certain DL model. Lee et al. [19] proposed to use confidence scores based on Mahalanobis distance from different layers, which was shown to be robust to typical abnormal samples, i.e., adversarial and OOD examples. However, its weighted averaging process still needs to train logistic regression detectors using both in-distribution and OOD samples. Wang et al. [17] first proposed to use model mutation testing to detect adversarial samples, while it could suffer from huge time and space overhead. Our work also works along this line, trying to seek for a great balance on identification effectiveness and efficiency. These identification results could greatly contribute to the reliability maintenance for DL applications.

Model comprehension and interpretation. Our work also relates to DNN models' comprehension and interpretation. Existing work typically analyzed internal units' specific contributions of DNN models from various perspectives. Some work used gradient-based [54, 55] or up-convolutional techniques [56, 57] to visualize hidden neurons as meaningful images to help users understand the role of individual neurons. Some other work [58–60] conducted sensitivity analysis on models, i.e., extract the important input regions or hidden neurons that are highly sensitive to the model outputs. Some other work [61–63] uses sensitivity-based method to prune unimportant parts of DNN models while maintaining the performance for acceleration. Our work also contributes to this line of research by adopting LRP [26], an explanation technique for DNN which operates by propagating the prediction backward in the neural network according to local propagation rules, to calculate the importance scores of neurons. Based on the scores, we could further derive DNN models' core part that is associated closely to the training knowledge and model it as our *model core*. By doing so, NetChopper then presents a freeze-and-mutate mechanism by first combining model mutation and our model core analysis to effectively identify abnormal samples. We believe that NetChopper's core analysis and core-based mutation could also suggest a generic way for model comprehension and interpretation.

6 CONCLUSION

DL applications' reliability is gaining popularity, and effectively identifying abnormal inputs for the DL models deployed in DL applications is a promising way towards the application reliability. In this paper, we propose NetChopper to identify such abnormal inputs in an effective and efficient way, by generating a small number of valuable mutants via a novel core analysis and model mutation. The key insight is to isolate a model's core functionalities from marginal supports, and exploit such isolation to distinguish abnormal inputs from normal ones based on their behavioral differences, which also gives a generic way for model interpretation, potentially also useful for similar model analysis and testing problems. The experimental evaluation also confirmed NetChopper's promising performance and stability, generally outperforming or comparably effective as existing techniques. In future, we consider to further strengthen NetChopper's effectiveness by investigating more NetChopper's variants and considering more complex application scenarios.

REFERENCES

[1] Andrej Karpathy. Software 2.0. https://karpathy.medium.com/software-2-0-a64152b37c35, 2017.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016)*, pages 770–778, 2016.

[3] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. Deep neural networks for object detection. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 2553–2561, 2013.

[4] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*, 2010.

[5] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[6] Apple. About face id advanced technology. [EB/OL]. https://support.apple.com/en-us/HT208108.

[7] Google. Google translate. [EB/OL]. https://translate.google.cn/.

[8] Tesla. Autopilot. [EB/OL]. https://www.tesla.com/autopilotAI.

[9] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[10] Hussein Dia. 'self-driving' cars are still a long way off. here are three reasons why. [EB/OL]. https://theconversation.com/self-driving-cars-are-still-a-long-way-off-here-are-three-reasons-why-159234 Accessed April 22, 2021.

[11] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *Proceedings of the 38th IEEE Symposium on Security and Privacy (SP 2017)*, pages 39–57. IEEE, 2017.

[12] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. DeepFool: a simple and accurate method to fool deep neural networks. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016)*, pages 2574–2582, 2016.

[13] Fabio Carrara, Fabrizio Falchi, Roberto Caldelli, Giuseppe Amato, Roberta Fumarola, and Rudy Becarelli. Detecting adversarial example attacks to deep neural networks. In *Proceedings of the 15th International Workshop on Content-Based Multimedia Indexing*, pages 1–7, 2017.

[14] Chawin Sitawarin and David Wagner. Defending against adversarial examples with k-nearest neighbor. *arXiv e-prints*, pages arXiv–1906, 2019.

[15] Gilad Cohen, Guillermo Sapiro, and Raja Giryes. Detecting adversarial samples using influence functions and nearest neighbors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14453–14462, 2020.

[16] Saikiran Bulusu, Bhavya Kailkhura, Bo Li, P Varshney, and Dawn Song. Anomalous instance detection in deep learning: A survey. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2020.

[17] Jingyi Wang, Guoliang Dong, Jun Sun, Xinyu Wang, and Peixin Zhang. Adversarial sample detection for deep neural network through model mutation testing. In *Proceedings of the 41st ACM/IEEE International Conference on Software Engineering (ICSE 2019), forthcoming, arXiv preprint arXiv:1812.05793*, May 2019.

[18] Jinhan Kim, Robert Feldt, and Shin Yoo. Guiding deep learning system testing using surprise adequacy. In *Proceedings of the 41th ACM/IEEE International Conference on Software Engineering (ICSE 2019), forthcoming, arXiv preprint arXiv:1808.08444*, May 2019.

[19] Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31 (NIPS 2018)*, pages 7167–7177. Curran Associates, Inc., 2018.

[20] DE Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back propagating errors. *Nature*, 323(6088):533–536, 1986.

[21] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

[22] Chris Murphy, Gail E. Kaiser, and Marta Arias. An approach to software testing of machine learning applications. In *Proceedings of the Nineteenth International Conference on Software Engineering & Knowledge Engineering (SEKE'2007), Boston, Massachusetts, USA, July 9-11, 2007*, page 167. Knowledge Systems Institute Graduate School, 2007.

[23] Jie M. Zhang, Mark Harman, Lei Ma, and Yang Liu. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering*, pages 1–1, 2020.

[24] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018.

[25] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[26] Grégoire Montavon, Alexander Binder, Sebastian Lapuschkin, Wojciech Samek, and Klaus-Robert Müller. Layer-wise relevance propagation: an overview. *Explainable AI: interpreting, explaining and visualizing deep learning*, pages 193–209, 2019.

[27] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[28] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[29] Michael Frigge, David C Hoaglin, and Boris Iglewicz. Some implementations of the boxplot. *The American Statistician*, 43(1):50–54, 1989.

[30] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[31] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

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

[33] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial machine learning at scale. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[34] Papers with code. Out-of-distribution detection. [EB/OL]. https://paperswithcode.com/task/out-of-distribution-detection.

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

[36] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2921–2926. IEEE, 2017.

[37] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[38] Papers with code. Adversarial defense. [EB/OL]. https://paperswithcode.com/task/adversarial-defense.

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

[40] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, et al. DeepGauge: Multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE 2018)*, pages 120–131. ACM, 2018.

[41] Simos Gerasimou, Hasan Ferit Eniser, Alper Sen, and Alper Cakan. Importance-driven deep learning system testing. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 702–713. IEEE, 2020.

[42] Jianmin Guo, Yu Jiang, Yue Zhao, Quan Chen, and Jiaguang Sun. DLFuzz: differential fuzzing testing of deep learning systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018)*, pages 739–743. ACM, 2018.

[43] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th International Conference on Software Engineering (ICSE 2018)*, pages 303–314. ACM, 2018.

[44] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. Deephunter: a coverage-guided fuzz testing framework for deep neural networks. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 146–157, 2019.

[45] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE 2018)*, pages 132–142. ACM, 2018.

[46] Zenan Li, Xiaoxing Ma, Chang Xu, and Chun Cao. Structural coverage criteria for neural networks could be misleading. In *Proceedings of the 41th ACM/IEEE International Conference on Software Engineering (ICSE 2019 NIER), forthcoming*, May 2019.

[47] D. Hendrycks and K. Gimpel. Visible progress on adversarial images and a new saliency map. 2016.

[48] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[49] Shiyu Liang, Yixuan Li, and R. Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks. 2018.

[50] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 3–14, 2017.

[51] Zhitao Gong, Wenlu Wang, and Wei-Shinn Ku. Adversarial and clean data are not twins. *arXiv preprint arXiv:1704.04960*, 2017.

[52] Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410*, 2017.

[53] Chuan Guo, Mayank Rana, Moustapha Cissé, and Laurens van der Maaten. Countering adversarial images using input transformations. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

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

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

[56] Alexey Dosovitskiy and Thomas Brox. Inverting visual representations with convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4829–4837, 2016.

[57] Anh Nguyen, Jeff Clune, Yoshua Bengio, Alexey Dosovitskiy, and Jason Yosinski. Plug & play generative networks: Conditional iterative generation of images in latent space. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4467–4477, 2017.

[58] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[59] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[60] Ruth C Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *Proceedings of the IEEE international conference on computer vision*, pages 3429–3437, 2017.

[61] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[62] Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*, pages 293–299. IEEE, 1993.

[63] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.