Adaptive Test Selection for Deep Neural Networks

Xinyu Gao
xinyugao@smail.nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing 210023, China

Yang Feng[*]
fengyang@nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing 210023, China

Yining Yin
ynyin@smail.nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing 210023, China

Zixi Liu
zxliu@smail.nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing 210023, China

Zhenyu Chen
zychen@nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing 210023, China

Baowen Xu
bwxu@nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing 210023, China

ABSTRACT

Deep neural networks (DNN) have achieved tremendous development in the past decade. While many DNN-driven software applications have been deployed to solve various tasks, they could also produce incorrect behaviors and result in massive losses. To reveal the incorrect behaviors and improve the quality of DNN-driven applications, developers often need rich labeled data for the testing and optimization of DNN models. However, in practice, collecting diverse data from application scenarios and labeling them properly is often a highly expensive and time-consuming task.

In this paper, we proposed an adaptive test selection method, namely ATS, for deep neural networks to alleviate this problem. ATS leverages the difference between the model outputs to measure the behavior diversity of DNN test data. And it aims at selecting a subset with diverse tests from a massive unlabelled dataset. We experiment ATS with four well-designed DNN models and four widely-used datasets in comparison with various kinds of neuron coverage (NC). The results demonstrate that ATS can significantly outperform all test selection methods in assessing both fault detection and model improvement capability of test suites. It is promising to save the data labeling and model retraining costs for deep neural networks.

CCS CONCEPTS

• **Software and its engineering** → *Software testing and debugging*.

KEYWORDS

deep learning testing, deep neural networks, adaptive random testing, test selection

ACM Reference Format:
Xinyu Gao, Yang Feng, Yining Yin, Zixi Liu, Zhenyu Chen, and Baowen Xu. 2022. Adaptive Test Selection for Deep Neural Networks. In *44th International Conference on Software Engineering (ICSE '22), May 21–29, 2022, Pittsburgh, PA, USA*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3510003.3510232

1 INTRODUCTION

Deep neural networks (DNN) have been deployed in many fields to assist in solving various tasks, such as medical image diagnosis [28], autonomous driving [9], customer services [55], machine translations [6] and so on. As the DNN-driven software demonstrates such fantastic performance on well-defined tasks, influencing our daily activities and lives, their quality and reliability have raised wide concerns. DNN-driven software, essentially is one kind of software program, could also suffer from software defects that may cause monetary and even human life losses [1–3]. Therefore, for DNN-driven software, quality assurance techniques have become exceedingly demanded.

However, assuring the quality of DNN-driven software is a challenging task, due to the natural differences between DNN models and conventional software systems. While conventional software systems often rely on developers to construct the business logic manually, DNN models, which form the kernel part and empower the DNN-driven software, employ a data-driven programming paradigm that needs to learn the internal logic from massive data [36]. This feature not only makes the behavior of the DNN model difficult to interpret and analyze but also disables the application of many conventional quality assurance methods. With millions or even billions of neurons, connections, and activation functions, DNN models construct complex nonlinear transformations to map input features into the proper labels. Thus, it is hard for developers to optimize the DNN-driven software by manually tuning the internal parameters of the DNN model. In practice, developers often

[*] Yang Feng is the corresponding author.

ICSE '22, May 21–29, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9221-1/22/05…$15.00
https://doi.org/10.1145/3510003.3510232

retrain the DNN model with rich data [23, 40, 43] to fix the incorrect behaviors and improve the performance of DNN-driven software. In this process, they need to collect a great many of data from application scenarios and hire a large workforce to label them.

Under this situation, identifying and selecting the most representative data become critical for improving the effectiveness and efficiency of quality assurance tasks of DNN-driven software. Inspired by the great success that the structural code coverage criteria achieved in conventional software programs, some prior research has proposed structural neuron coverage to measure the adequacy of DNN testing [32, 41, 49]. These researches have demonstrated the effectiveness of structural neuron coverage on distinguishing the general mutation tests and adversarial samples. However, similar to the conventional code coverage, structural neuron coverage also requires an extremely high overhead in the collection process, thus it is difficult to apply them on the large-scale models [30]. Besides, several independent research groups also reveal that some neuron coverage criteria could rapidly reach the maximum coverage with very few tests, which may restrict their effectiveness as a guidance criterion for test selection [14, 23, 39]. On the other hand, another family of techniques is proposed to prioritize test cases based on some rules [23]. Test prioritization techniques often value test case that has a high probability of detecting DNN incorrect behaviors. These techniques have demonstrated the high effectiveness of collecting a portion of tests from a large size of the dataset; however, they fail to consider the relationship and distribution of selected data, and thus cannot diversify the detected incorrect behaviors. This feature may fundamentally hinder their applications, especially when there are plenty of similar, or duplicate, tests in the candidate set.

This paper extends the above techniques and alleviates their limitations. We explore an alternative solution to assist the test selection of DNN models. We first employ the output vector of a DNN model to represent the model behaviors of the given input. Then, we define the local domain of the DNN model output to describe the fault pattern. By projecting the behavior information into different local domains, we can evaluate the fault pattern of the test cases through extended operation, which values both the model uncertainty and behavior diversity. Further, we design a fitness metric to measure the fault pattern difference between the candidate test and the selected set. Based on the above design, we propose ATS, the first adaptive test selection method for deep neural networks, to select more diverse tests from the candidate set. ATS can select a subset that reveals more different faults in the DNN-driven software and reduces the labeling efforts for the optimization process.

To validate the effectiveness of ATS, we conduct experiments for ATS and baseline methods with four well-designed DNN models and four widely-used datasets. We also realize different pollutions of unfiltered datasets in reality and simulate them in our experiments. The experimental results demonstrate that ATS performs well in defect detecting tasks. Moreover, we also prove that our adaptive test selection can select diverse defects faster than prioritization techniques which blindly select the test cases without considering differences. Finally, we demonstrate that ATS is more effective than both the coverage-guided test selection methods and test prioritization methods for DNN-driven system optimization.

The contributions of this paper could be summarized as follows:

- **Method.** We model the fault pattern and design a metric to evaluate their differences for DNN models. Based on the fault pattern metric, we propose ATS, the first adaptive random test selection method for DNN models.
- **Tool.** We implement ATS into a tool that could help DNN developers to select test cases from massive unlabelled data. We have released the source code of the tool and experimental datasets online [1].
- **Study.** We conduct an extensive experiment to investigate the performance of ATS method, coverage-guided selection methods, and test prioritization methods. The results show that ATS can significantly outperforms other test selection methods and efficiently enhances the DNN model.

2 BACKGROUND

This section includes several basic knowledge of DNN, neuron coverage, and test selection methods.

2.1 The Architecture of Deep Neural Network

The architecture of the deep neural networks (DNN) can be represented as a composite function chain that maps the input data x to the output result y.

$$y = f(x) = f^{(0)}(f^{(1)}(\dots(f^{(D)}(x)))) \tag{1}$$

Take an n-category classifier as an example, the goal of a deep neural network is to approximate y to the theoretical classifier y^*. In other words, by adjusting the inside weights of the DNN model, we have $y \approx y^*$ which gives an output vector $y = f(x)$ close to the one-hot vector (i.e., corresponding ground truth) $l = f^*(x)$. The final output result is calculated through the softmax function. It can be interpreted as a probability (all elements are positive, and the sum is 1). Thus the output domain Y satisfies the constraint that:

$$\mathrm{Y} = \{y|y \in \mathbb{R}^n, \|y\|_1 = 1 \wedge \forall i, y(i) \geq 0\} \tag{2}$$

And the set of all ground truth is denoted as L in this paper:

$$\mathrm{L} = \{l|l \in \mathrm{Y}, \exists i, l(i) = 1\} \tag{3}$$

Among them, the length D of the chain means the depth of the network. The function in the middle of the chain, i.e., $f^{(j)}, j = \{1, 2, \dots, D-1\}$, is the hidden layer of the DNN. Each hidden layer $f^{(j)}$ maps the forward input into a vector, and each element of the vector is an independent unit (called a neuron) representing a vector-to-scalar function. The dimensionality D of the hidden layer $f^{(j)}$ means the width of the network. Inside each layer, all neurons of this layer are independent and act in parallel.

Therefore, for a D layer deep neural network model M with fixed-width W, if an input date x is calculated by the model DNN, we use $\boldsymbol{h}_i \in \mathbb{R}^W$ to represent the i-layer vector value. Combining all hidden layers vector, we could get a hidden layer matrix $H \in \mathbb{R}^{W \times D-1}$, which contains all the hidden neuron values of an input data.

Given a DNN model DNN and an input data x, denote the set $\mathrm{Neuron}(\mathrm{DNN}, x)$ to represent the hidden output values when DNN

[1] https://github.com/SATE-Lab/ATS

is executed with x:

$$\text{Neuron}(\text{DNN}, x) = H = [\boldsymbol{h}_1, \ldots, \boldsymbol{h}_{D-1}] \in \mathbb{R}^{W \times D-1} \quad (4)$$

And the final output result calculated by softmax function is denoted as:

$$\text{Run}(\text{DNN}, x) = y \in \text{Y} \quad (5)$$

2.2 Neuron Coverage Criteria

Inspired by the effectiveness of structural coverage on guiding the testing of conventional software applications, researchers have proposed many testing criteria based on the structural neuron coverage to measure the test adequacy of deep neural network systems. With the guidance of structural neuron coverage, several test generation techniques [26, 49, 57, 59, 63, 67] have been proposed to improve the performance of DNN models. In this section, we briefly introduce those structural neuron coverages.

Neuron Activation Coverage($NAC(k)$). As the earliest neuron coverage criterion [49], the primary assumption of $NAC(k)$ built upon is that the more neurons are activated, indicates more states of DNN are explored. The computation process of $NAC(k)$ requires collecting each neuron's output value and counting neurons as covered if their outputs exceed the threshold k. The $NAC(k)$ coverage of a test is computed as the ratio of the number of covered neurons to the total number of neurons.

k-Multisection Neuron Coverage ($KMNC(k)$). Based on the $NAC(k)$ assumption about the DNN states, the researchers further assume that instead of treating the neuron as a value with only two states (activated and inactivated), but treats the output of a neuron as a range of values [41]. In other words, suppose that the output of a neuron o on the training set is in the interval $[low_o, high_o]$, and divide them equally into k segments. The goal of the $KMNC(k)$ criterion is to make the neuron cover each segment of k segments.

Neuron Boundary Coverage (NBC). Different from $KMNC(k)$, neuron boundary coverage (NBC) focuses on whether the corner regions $(-\infty, low_o]$ and $[high_o, \infty)$ are covered by test cases [41].

Strong Neuron Activation Coverage ($SNAC(k)$). Some studies point out that strongly activated neurons may have additional value for DNN, so $SNAC(k)$ was proposed [41]. Strong Neuron Activation Coverage ($SNAC$) is a simplification of NBC, which only collects the ratio of some neurons that cover the upper bound $[high_o, \infty)$ to all neurons.

Top-k Neuron Coverage ($TKNC(k)$). Top-k Neuron Coverage ($TKNC(k)$) focuses on the most active k neurons in each layer [41]. It is computed by the ratio of the total number of top-k neurons on each layer to the total number of neurons in a DNN.

Modified Condition/Decision Coverage (MC/DC). Similar to the concept of conventional software testing, MC/DC criteria of neuron networks [58] models the neuron output value (or sign) as a decision, and all the previous layer connected neurons are modeled as conditions. Neuron network MC/DC consists of four implementations, namely SS, SV, VS, and VV-coverage, and all of the above-mentioned neuron coverage criteria could be regarded as a specific situation of the original MC/DC neuron coverage [58].

2.3 Test case selection methods

In this section, we introduce several widely used test case selection methods. For any test selection method, the goal is to obtain a fixed size (N) subset X_S from the candidate set X_C, i.e., we have $X_S \subseteq X_C$ and $|X_S| = N$.

2.3.1 *Coverage-guided test selection.*

In this section, we introduce a set of test selection methods guided by coverage metrics. In conventional software testing, testers tend to select a test case that covers more different code lines [65]. This kind of selection method follows a basic assumption that early reaching the maximum coverage would lead to the higher capability of defect detection [22].

For any coverage metric, in each iteration, it follows an additional greedy algorithm to select the next test according to the feedback from the previously selected set, i.e., select the test cases that covers the maximum number of uncovered area of the given coverage criterion.

2.3.2 *Prioritization test selection.*

Generally, for a given candidate set X_C, prioritization test selection methods compute a weight p for each test case x in the candidate set. The weight p represents the test significance of test x. In other words, p is the possibility of revealing model errors. It could be denoted as follows:

$$p = \text{Priority}(x) \quad x \in X_C \quad (6)$$

The test case with higher priority p is supposed to have a higher value of detecting faults and enhancing the DNN model. Therefore, the prioritization test selection can be represented as follows:

$$X_S = \mathop{\arg\max}_{X_S \subseteq X_C \wedge |X_S| = N} \sum_{x \in X_S} |\text{Priority}(x)| \quad (7)$$

Simply speaking, the priority method of test case selection is to select the test cases with top-N weights to form a test suite of the required size.

Here we introduce four different prioritization selection methods, which come from different fields of AI software testing and traditional AI research.

DeepGini. Recently, Feng et al. [23] propose a test prioritization technique based on a statistical perspective of DNN, named DeepGini. It takes the use of the Gini coefficient to measure the likelihood of test case x being misclassified.

LSA. Kim et al. [32] propose a test criterion towards DNN testing, called SADL (Surprise Adequacy for Deep Learning). In the research, LSA (**L**ikelihood-based **S**urprise **A**dequacy) is proposed to measure how close to the class boundary the new inputs are. The higher LSA value of the test means it is more surprising to the DNN. Thus it could be regarded as a priority weight for test selection.

CES. Li et al. [40] propose a test selection method based on conditioning, which is to assess the new precision on operational environments. The experiment results show that CES (**C**ross **E**ntropy-based **S**ampling) estimator outperforms random sampling in all experiments.

Max$p$. Max$p$ is a representative uncertainty sampling strategy of active learning [50]. It employs the maximal prediction probability of the classification task to indicate the prediction confidence of the classification model and thus prioritizes the input of the least prediction confidence.

3 METHODOLOGY

The key feature of our selection method is to select test cases with diverse failure directions and higher failure probabilities. First, we introduce a motivating example for adaptive test selection of DNN testing. Then, we propose a mapping relationship, which converts the output domain into a set of intervals to describe the fault pattern of a given test or set. After that, we propose a fitness metric to measure the difference between the candidate test and the selected set. Finally, we propose the adaptive test selection method ATS based on fault pattern and fitness metric.

3.1 Motivation and Inspirations

For the conventional software program, there has been some work discussing the shape and the location of faults from the perspective of the input domain [4, 8, 12, 24, 61]. To achieve an even spread of test cases within the input domain, Chen et al. [15] propose adaptive random testing (ART) based on the analysis of fault patterns. Even though in the past decade, plenty of techniques have been proposed for improving ART and extending its application scenarios [7, 11, 16, 31], it is difficult to apply them to the DNN model testing tasks because of their working nature and input features.

The input data of modern DNN models are often of various types, such as images, point clouds, texts, speech signals, and so on. This variance makes it difficult for us to measure the fault pattern from the input domain. Nevertheless, for a given test input, the DNN model often produces a vector containing the probability of labels and thus determines the final output based on the probability distribution. Thus, based on the probability distribution, selecting the test with higher uncertainty may obtain a higher probability of detecting DNN faults. For a given output vector $y \in \mathbb{R}^n$, the process could be formalized as $\mathsf{Run}(\mathrm{DNN}, x) = y$. For a classification model, we regard an input data x is classified as i-th category, iff the i-th element of y is the maximum element ($i = \arg\max_i y(i)$). If the output probability of the input data is more concentrated ($\max(y(i))$ is closer to 1), we assume that the model has higher confidence for classification results.

Further, the test cases which are different from each other could reveal more diverse faults of DNN models. For multiple test cases with the same degree of uncertainty, a similar test may correspond to the same fault, and the test cases that differ from each other could better reveal different faults of the model. For example, if we have a test set $X = \{x_1, x_2, x_3, x_4\}$. The output vectors are: $y_1 =< 0.9, 0, 0.1 >$, $y_2 =< 0.6, 0, 0.4 >$, $y_3 =< 0.2, 0.4, 0.4 >$ and $y_4 =< 0.59, 0, 0.41 >$. Thus, we can regard the DNN model as more confident for the classification result of x_1, and $\{x_2, x_4\}$ are more likely to be similar. Above observations inspire us to introduce a metric to measure the difference of faults from the perspective of the output domain. Based on this metric, we design and implement ATS to guide the test selection of DNN models.

3.2 Fault Pattern Computation

We design the fault pattern mapping Pattern, which converts the output vector into several intervals to satisfy the insights introduced above. The size and location of the subsets reflect the information of the model's uncertainty and test case x's fault pattern.

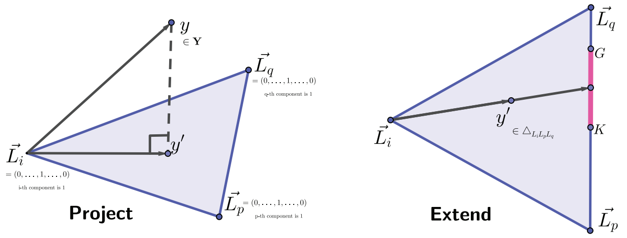


Figure 1: A simplified figure to illustrate Step.3&4. (Project and Extend).

Such a mapping assists us in analyzing and extracting the fault pattern distribution corresponding to each test case from the output domain. On the one hand, we define the uncertainty of test case $x \in T_i$ as $\mathsf{Uncertain}(x) = 1 - \mathsf{Run}(\mathrm{DNN}, x)(i)$. For example, if we have an output vector $y =< 0.3, 0.3, 0.4 >$, then the uncertainty is denoted as $1 - \max(y(i)) = 0.6$. On the other hand, we express the direction as the line from the prediction one-hot vector L_i to the output vector y. For example, for the output vector y denoted above, its prediction vector is denoted as $L_3 =< 0, 0, 1 >$, thus the direction is denoted as $\vec{l_3 y} =< 0.3, 0.3, -0.6 >$. Finally, we design the fault pattern based on both direction and uncertainty.

3.2.1 Fault Pattern of Test Case. Next, we introduce the main steps of computing the fault pattern. For a test set T, the fault pattern of each test $x \in X$ is calculated by four steps.

Step1. **Test Set Clustering :** For each test case $x \in T$, we cluster the test case into n subset of T based on its prediction category, i.e., we have $x \in T_i$ iff $i = \arg\max_i(\mathsf{Run}(\mathrm{DNN}, x)(i)) = \arg\max_i y(i)$. Clustering based on the prediction category ensures that we can analyze test cases with similar results.

Step2. **Local Domain Determining:** For each test case x in cluster T_i, we construct an index set $\mathrm{Ind}(i) = \{(i, p, q) | p \neq q \neq i \wedge 1 \leq p < q \leq n \wedge p, q \in \mathbb{N}\}$. Each element in $\mathrm{Ind}(i)$ represents a local domain, which is a plane spanned by $L_i L_p$ and $L_i L_q$. By analyzing each local domain one by one, we could extend fault pattern information to tasks with any number of categories ($n \geq 3$).

Step3. **Project Operation:** This step aims to extract local information of test case x's output vector y. Refer to Fig. 1, the goal of Project is to find the $y' \in \triangle_{L_i L_p L_q}$ subject to $(y' - y) \perp \mathsf{span}(\vec{L_i L_p}, \vec{L_i L_q})$. Specific calculation formula can refer to [10, 56]. Through calculation, we could get the local information y' corresponding to index (i, p, q).

Step4. **Extend Operation:** Extend the subspace vector y' determined by (i, p, q) to an interval denoted as $\mathsf{Pattern}_x(i, p, q) = [a, b]$. Refer to Fig. 1, we implement the operation in the triangle $\triangle_{L_i L_p L_q}$. We determine the intersection point by extending $\vec{L_i y'}$ to $L_p L_q$. The intersection point represents the local direction of the test case x. After that, the local fault pattern is designed as a segment GK in line $L_p L_q$. The midpoint of segment GK is the intersection, and the length is determined by the uncertainty of the test case x. Finally, we normalize the $L_p L_q$ to the interval $[0, 1]$, and consider the local fault pattern as the normalized interval $[a, b] \subseteq [0, 1]$

of segment GK. The normalized length of $[a, b]$ is controlled by $\frac{1-y(i)}{\eta}$, where η is a hyperparameter to control the granularity. We record the local fault patterns of test $x \in T_i$ corresponding to index (i, p, q) as $\text{Pattern}_x(i, p, q) = [a, b]$.

Step5. **Local Pattern Gathering:** For each test case x in cluster T_i, we collect all of its local fault patterns $\text{Pattern}_x(i, p, q)$, where (i, p, q) is in $\text{Ind}(i)$, the set of local patterns is regarded as test case x's fault pattern. In other words, we have that $\text{Pattern}_x = \bigcup_{(i,p,q)\in\text{Ind}(i)} \{(i, p, q) : [a, b]\}$.

Table 1: An example to illustrate fault pattern.

Index	1	2	3	4	Inter sect	Local Fault Pattern
$\text{Run}(\text{DNN}, x) = y$	**0.5**	0.2	0.2	0.1	-	-
$\text{Pattern}_x(1, 2, 3)$	0.53	0.23	0.23	0	0.5	[0.45,0.55]
$\text{Pattern}_x(1, 2, 4)$	0.57	0.27	0	0.16	0.64	[0.59,0.69]
$\text{Pattern}_x(1, 3, 4)$	0.57	0	0.27	0.16	0.64	[0.59,0.69]

Example 1. Here we introduce a 4-category classification example to illustrate the fault pattern introduced above. For the test case x and its output result $\text{Run}(\text{DNN}, x) = y =< 0.5, 0.2, 0.2, 0.1 >$. First, we determine $i = \arg\max_i y(i) = 1$. (Step1.) Second, when $n = 4$, the index set of cluster T_i is constructed as $\text{Ind}(i) = \{(1, 2, 3), (1, 2, 4), (1, 3, 4)\}$. (Step2.) After that, the local information y' of output y is calculated (Step3.), which is shown in column 1,2,3,4. Based on the Extend operation (Step4.), the intersection points are shown in Intersect column. The length of local fault pattern is calculated by $(1-y_1)/\eta = 0.1$ ($\eta = 5$ in this example), and the covered interval of (i, p, q) is shown in the last column. Finally, we could gather the fault pattern of test case x (Step5.). $\text{Pattern}_x = \{(1, 2, 3) : [0.45, 0.55], (1, 2, 4) : [0.59, 0.69], (1, 3, 4) : [0.59, 0.69]\}$.

3.2.2 Fault Pattern of Test Set. To obtain the fault pattern of a given test set T, we need to merge the fault patterns calculated by each test case. In other words, for cluster T_i, we take the union of all corresponding local fault patterns under each specific local domain (i, p, q), denoted as $\text{Pattern}_{T_i} = \bigcup_{(i,p,q)\in\text{Ind}(i)} \{(i, p, q) : S\}$. Noted that, S is a subset of normalized segment interval $[0, 1]$, which is obtained by $S = \bigcup_{x\in T_i} \text{Pattern}_x(i, p, q)$.

Finally, we gather together fault patterns of all clusters to get the fault pattern of the given test set T:

$$\text{Pattern}_T = \bigcup_{T_i, i=1,\ldots,n} \bigcup_{(i,p,q)\in\text{Ind}(i)} \{(i, p, q) : S\} \tag{8}$$

3.3 Fault Pattern Fitness Metric Design

With the help of fault pattern, we propose a fitness metric that assesses the difference between test case x and the selected test set S. Based on this metric, we could migrate the basic idea of adaptive random testing (ART) into DNN test case selection.

3.3.1 Fitness Metric. The fitness metric aims to provide a normalized value to quantify the fitness of each candidate x from the candidate set C against the selected set S, denoted as $\text{Fitness}(x, S)$.

When test x belongs to cluster T_i, the fitness metric could be expressed as follows:

$$\text{Fitness}(x, S) = \sum_{(i,p,q)\in\text{Ind}(i)} \frac{|\text{Pattern}_x(i, p, q) \backslash \text{Pattern}_S(i, p, q)|}{|\text{Pattern}_x(i, p, q) \cup \text{Pattern}_S(i, p, q)|} \tag{9}$$

Noted that, the operator $\backslash$ and $\cup$ in Eq. 9 are basic set operators [25], and $|\cdot|$ represent the length of corresponding set. The Fitness metric reflects the difference between the test x and the selected set S.

Algorithm 1: ATS test selection Alg.

```
1  Procedure ATS($DNN, C \subseteq X, N$);
2  collect output vectors: $\text{Run}(\text{DNN}, X)$;
3  determine output vector dimension $n$;
4  cluster $C$ into $n$ subset: $C_i$ $(i = 1, \ldots, n)$;
5  construct index sets: $\text{Ind}(i)$ $(i = 1, \ldots, n)$;
6  for each test $x \in C$, calculate fault pattern: $\text{Pattern}_x$;
7  initial selected subset: $S_1, \ldots, S_n \leftarrow \emptyset$;
8  initial selected test case list: $S \leftarrow \emptyset$;
9  for $j = 1, 2, \ldots, \lfloor \frac{N}{n} \rfloor$ do
       // evenly select from each cluster
10     for $i = 1, 2, \ldots, n$ do
11         if $C_i \neq \emptyset$ then
12             for $x \in C_i$ do
13                 calculate $\text{Fitness}(x, S_i)$;
14             if $\max_x$ $Fitness(x, S_i) > 0$ then
15                 $x = \arg\max_x \text{Fitness}(x, S_i)$;
                   // select test $x$ with maximum fitness
16                 $S.append(x)$;
                   // Added in order
17                 $S_i \leftarrow S_i \cup \{x\}$;
18                 $C_i \leftarrow C_i \backslash \{x\}$;
19 $C \leftarrow \bigcup_{i=1,\ldots,n} C_i$;
20 while $size(S) < N$ do
21     for $x \in C$ do
22         calculate $\text{Fitness}(x, S)$;
23     if $\max(Fitness(x, S)) > 0$ then
24         $x = \arg\max_x \text{Fitness}(x, S)$;
           // select test $x$ with maximum fitness
25     else
26         $x = \arg\max_x (\sum_{(i,p,q)\in\text{Ind}(i)} |\text{Pattern}_x(i, p, q)|$;
           // select test $x$ with higher uncertainty
27     $S.append(x)$;
28     $C \leftarrow C \backslash \{x\}$;
29 return selected list: $S$;
```

3.3.2 Overall procedure. We divide the whole selection procedure into two parts: cluster by cluster selection and total selection. From Line 9 to Line 17, we tend to evenly select test cases which have maximum fitness metric $\text{Fitness}(x, S_i)$ according to the selected subset. Thus, we could select the test case with different fault patterns and higher uncertainty in each cluster. If the candidate set is unbalanced, there may exist a situation that we cannot select enough test cases for some clusters. From Line 20 to Line 28, we select the rest part of the selected set S. First, we prefer test cases with higher fitness metric results. In Lines 25-26, if all test cases in the candidate set have the same fitness scores, then we simply select the next case based on its uncertainty.

Table 2: A simplified example of ATS.

Candidate test	Case Pattern	Ori Pattern	Round 1 Fitness	Set Pattern	Round 2 Fitness
x_1	[0.45,0.55]	[0.2,0.5]	0.05/0.35	[0.2,0.5]∪ [0.7,0.88]	0.05/0.53
x_2	[0.2,0.4]		0/0.3		0.0/0.48
x_3	[0,0.12]		0.12/0.42		0.12/0.6
x_4	[0.7,0.88]		**0.18/0.48**		-
x_5	[0.46,0.66]		0.16/0.46		**0.16/0.64**

Example 2. In this part, we introduce a simplified example of ATS. We omit the process of clustering (Step1.), local domain determining (Step2.) and local fault patterns gathering (Step5.). Thus only the core part of the selection procedure is introduced. In Round 1, we show the fitness between original fault pattern $[0.2, 0.5]$ and the fault pattern of each test, i.e., $\frac{|\text{Pattern}_x \backslash \text{Pattern}_S|}{|\text{Pattern}_x \cup \text{Pattern}_S|}$. For example, for test case x_1, $\text{Fitness}(x_1, [0.2, 0.5]) = \frac{|[0.45,0.55] \backslash [0.2,0.5]|}{|[0.45,0.55] \cup [0.2,0.5]|} = \frac{|(0.5,0.55]|}{|[0.2,0.55]|} = 0.05/0.35$. Based on Line 15 in Alg. 1, we select out test x_4 in the first round. And in column Set Pattern, we update the fault pattern of the selected set. In the next round (column Round 2), we calculate the fitness between each test in candidate set $\{x_1, x_2, x_3, x_5\}$ and $[0.2, 0.5] \cup [0.7, 0.88]$. Finally, in Round 2, we select out the maximum fitness test case x_5.

3.4 Enhancing the DNN model with ATS

Both testing and enhancing the DNN-driven system need to rely on manually labeled data. Collecting plenty of unlabeled data is usually easy to achieve. However, compared with the cost of data collection, the cost of manual labeling is much greater. For data with strong expertise knowledge, such as medical image data, it is unrealistic to blindly label all collected data. Therefore, the significance of DNN test selection is to reduce the labeling cost. Such an idea is also inspired by *active machine learning* [50], which aims to select data near the decision boundary (uncertainty). Programmers suppose that these uncertain data could be used to optimize the decision boundary of DNN more efficiently.

Besides, ATS also pay attention to the distribution selected data subset. The success of adaptive random testing (ART) proves that the selection strategy based on previously selected data feedback is meaningful. From the perspective of the DNN model, the complexity of the decision boundary makes uncertainty prioritization prefers a specific fault pattern.

To sum up, ATS could not only select test cases to detect model faults more efficiently but also construct a more proper subset for DNN optimization through retraining.

4 EXPERIMENT DESIGN

This section introduces our experimental settings, including the data set and DNN model used in the experiment, select data set generation methods, baseline approaches, and research questions. To conduct the experiments, we implement our approach as well as various NC-guided test select methods upon Keras 2.3.1 with TensorFlow 1.13.1. All experiments are performed on a Ubuntu 18.04.3 LTS server with two NVIDIA Tesla V100 GPU, one 10-core processor "Intel(R) Xeon(R) Gold 6248 CPU @ 2.50GHz", and 120GB memory.

4.1 Datasets and DNN Models

Table 3: Dataset and DNN models.

Dataset	Description	DNN Model	#Neurons	Layers
MNIST	28x28 hand-written digits	LeNet-1	42	5
		LeNet-5	258	7
CIFAR-10	32x32 colored images	ResNet-20	698	20
		VGG-16	7274	21
Fashion	28x28 gray-scale images	LeNet-1	42	5
		ResNet-20	698	20
SVHN	street view numbers	LeNet-5	258	7
		VGG-16	7274	21

In the research, the experiments are constructed on four well-known publicly available DNN datasets: MNIST, CIFAR-10, SVHN, and Fashion. Table 3 presents the statistical details on these datasets. MNIST [64] dataset is a handwritten digits dataset with 10 labels. It contains 70,000 input data in total, of which 60,000 are training data, and 10,000 are test data. CIFAR-10 [33, 34] dataset consists of 60,000 32x32 colour images in 10 classes, with 6,000 images per class. CIFAR-10 is divided into 50,000 training and 10,000 testing images. Fashion [62] is a dataset of Zalando's article images consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image associated with a label from 10 classes. SVHN [47] is a real-world image dataset that can be seen as similar to MNIST. But it incorporates an order of magnitude more labeled data (over 600,000 digit images). It is collected from house numbers in Google Street View images. At the same time, we select four different scale DNN models to ensure the universality of our experiments, which are LeNet-1, LeNet-5 [37], VGG-16 [54] and ResNet-20 [28]. For each data set, we select two different DNN models to ensure the criterion result is stable and excellent on different combinations.

4.2 Candidate Set Construction

To simulate the data mutation in realistic settings, we choose to follow the prior research convention [46, 53] and employ seven well-designed benign perturbations rather than adversarial examples generators for data augmentation. We make this choice is because adversarial examples are often generated from carefully designed algorithms [35, 48], they cannot represent data collected from the real-world application scenario and may lead to unreliable conclusions [39].

For each dataset, we generate the test data based on seven well-used benign perturbations to retain its original label, including shift, zoom, brightness, rotation, shearing, blur, and contrast ratio [46, 53]. When the original test size is N, for each augmentation operator, we generate the same amount of data. We divide the original test set and each generation set into two parts, one for constructing the candidate set and the other for constructing the new test set. For the MNIST dataset, we finally constructed a candidate set and a new test set with the same size 40000, i.e., 5000 original test data, and 5000 generated data for seven augmentation operators.

Furthermore, considering that unfiltered candidate data may include pollution and invalid data. For each generated candidate set, we also design different data pollution scenarios to cover different types of invalid data that may exist in an unfiltered dataset. In addition to the pure candidate set, we construct four polluted candidate sets with 20% additional invalid data, including irrelevant data, meaningless synthetic data, repeat data, and crashed data.

To sum up, we construct five candidate sets for each dataset, including a pure valid data set and four unfiltered datasets with part of invalid data.

4.3 Baseline Approaches

During the test selection, we take the **R**andom **S**ampling as a baseline method naturally. We denote this baseline as **RS**. RS draws samples randomly from the candidate test set according to the target size.

Next, we introduce other baseline approaches used in experiments. The baseline selection methods are divided into two types: coverage-guided test selection and prioritization test selection. We choose four typical techniques for each type.

4.3.1 Coverage-guided Test Selection. To compared with coverage-guided test selection methods, we select 4 well-known DNN neuron coverage criteria (**NAC** [49], **NBC**, **TKNC** and **SNAC** [41]) to guide the test selection procedure introduced in Sec. 2.3.1. For the more specific introduction, refer to Sec. 2.2. Noted that limited by the computation resource, we abandoned KMNC [41] and MC/DC [58] as the baseline because even for a simple DNN model (LeNet-5), they take more than 24 hours to select 10% test cases. The configurable parameters of the neuron coverage criteria follow the authors' suggested settings or employ default settings of the original papers [41, 49].

4.3.2 Prioritization Test Selection. We also choose four prioritization selection methods introduced in Sec. 2.3.2 to compare the effectiveness with widely-used prioritization methods. More specifically, we conduct the experiments with **DeepGini** [23], **LSA** [32], **CES** [40] and **Max*p*** [38]. Noted that for **DeepGini**, we abbreviated it as **Gini** in the rest of the paper.

4.4 Research Questions

ATS is designed to adaptively select the next test case according to the previously selected set. It aims to select an appropriate subset for model faults detection and DNN model optimization. Based on the goals of test selection, we empirically explore the following research questions (RQ).

4.4.1 **RQ1: Fault Detection.** Can ATS detect more faults than baseline approaches?

Similar to conventional software testing, for a given test selection method, a selected set that can trigger more faults means it could reveal more defects in the software. Therefore, we first compare the fault detection capabilities of ATS and baseline approaches. In each DNN model & Dataset, we select 10% size of each candidate set, filter the invalid data, and collect the corresponding fault detection rate. For a selected test set X, the fault detection rate is defined as follows:

$$\text{Fault\_Detection\_Rate}(X) = \frac{|X_{wrong}|}{|X|} \tag{10}$$

where $|X_i|$ denotes the number of test cases being misclassified, and $|X|$ denotes the size of the selected set.

4.4.2 **RQ2: Fault Diversity.** Can ATS select test cases that cover more diverse faults?

For traditional software testing, Chan et al. [15, 18] have observed that failure-causing inputs usually are very dense and close to one another. Such insight could be migrated to DNN model testing, i.e., similar faults may reflect the same defect in DNN. In order to analyze the model more comprehensively, we hope the test selection methods could not only detect more faults but also detect more diverse faults efficiently.

We use a concept of fault type to answer this question. For a given test case x being misclassified, its fault type is defined as:

$$\text{Fault\_Type}(x) = (\mathsf{Label}(x)^* \rightarrow \mathsf{Label}(x)) \tag{11}$$

where $\mathsf{Label}(x)^*$ denotes the ground-truth label, and $\mathsf{Label}(x)$ denotes the prediction label calculated by DNN model. For example, if a handwritten digits x with true label "7" is misclassified into "1", then the fault type of x is denoted as: $\text{Fault\_Type}(x) = (7 \rightarrow 1)$.

For each candidate set with ten categories, the number of fault types is $10 \times 9 = 90$. Therefore, for each DNN model and dataset combination, we aggregate the fault types of the five candidate sets, the total number of fault types for a specific DNN&Dataset is $90 \times 5 = 450$. We collect the theoretical maximum number of fault types and compare the changing trend of the number of detected fault types.

To quantify the capability of selecting diverse faults, we collect the cumulative sum of the fault types found by specific test selection methods and compute the corresponding RAUC (ratio of area under the curve) between the selection method and theoretical curve.

4.4.3 **RQ3: Optimization Effectiveness.** Does the test cases selected by ATS guide the retraining more effectively?

Different from the traditional software, the DNN model could not be enhanced directly. Therefore, we propose RQ3 to further evaluate the effectiveness of retraining the DNN model with selected test cases. To answer RQ3, we choose four selected set with different sizes (2.5%, 5%, 7.5%, 10%) to add back into the original training set and retrain the DNN model. And the effectiveness of model optimization is compared on the newly constructed test set introduced in Sec. 4.2.

The experiment is repeated twice in all combinations to avoid random errors in the process of retraining. Furthermore, we not only compute the average accuracy improvement but also implement the Wilcoxon rank-sum test [21] to check whether the ATS are statistically significant outperform other baseline approaches at the significance level of 0.05.

5 RESULT ANALYSIS

In this section, we present the results of fault detection (RQ1), fault diversity (RQ2) and further analyze whether ATS could optimize the DNN model more effectively (RQ3).

5.1 Answer to RQ1: Fault Detection

As shown in Table 4, we compare the average fault detection rate between ATS and baseline methods. Limited by the space, we only display two selection ratio results (5% & 10%). Other results follow the same trend.

First, we found that all coverage-guided test selection methods have a poor performance in detecting faults. Such a finding is consistent with the former research [14, 27, 39]. Compared with some existing works that demonstrate the fault detection capability of neuron coverage in the test set with adversarial examples [39, 41], we assume that the faults generated by data mutation in realistic settings are more natural yet challenging to be detected. One of the possible illustrations to the ineffectiveness of neuron coverage-guided test selection may be that, compared to adversarial examples, benign perturbations could not lead to many different neuron activation states. Besides, we also find out that the neuron coverage is easy to reach the maximum during the selection process and no longer be increased, which is also supported by existing researches [23].

Compared with random sampling (RS) baseline, both ATS and most prioritization selection methods show a higher capability of detecting more faults. ATS has the best result in most DNN model & dataset combinations. We conclude that ATS has an outstanding capability of detecting more faults within a limited selection size.

5.2 Answer to RQ2: Fault Diversity

As shown in Fig. 2, for each Dataset & DNN model combination, we draw the cumulative sum curve of the fault types for each selection method. Note that the dark blue curve on the top represents the theoretical maximum number of fault types that could be included in the corresponding subset size. Besides, we also compute the ratio of area under the curve (RAUC) between theoretical and each selection method. The closer the RAUC is to 1, means the corresponding selection method performs better in detecting more diverse faults. The results of RAUC are shown in Tab. 5. Compared with random sampling (RS) baseline, ATS, Gini, Max*p*, and TKNC show better results consistently.

However, different from the results shown in RQ1, ATS achieved the best results under all dataset&DNN model combinations. For example, as for Fashion & LeNet-1, both Gini and Max*p* show a higher fault detection rate than ATS in Tab. 4. In RQ2, the RAUC of Gini and Max*p* are only 79.48% and 81.51%, respectively, while the RAUC of ATS is 91.88%. This phenomenon means that although prioritization test selection methods can sometimes detect more faults, the test cases selected by these methods may have an uneven distribution. In other words, prioritization test selection methods may prefer a specific type of fault. There does not exist an adaptive adjustment step for the priority list to adjust the weight based on selected set feedback. Such selection methods are easy to select a subset with limited fault types, which may have a negative impact on model optimization.

5.3 Answer to RQ3: Optimization Effectiveness

The final goal of DNN testing is to detect faults and optimize the DNN model to improve the generalization of the DNN model. Thus we evaluate the effectiveness of DNN optimization by adding back valid test data into the original training set. We collect the accuracy improvement results of four different selection ratios (2.5%, 5%, 7.5%, 10%). The concrete accuracy improvement results can be found in Tab. 6.

From the perspective of selection methods, the average accuracy improvement results of most selection techniques perform better than random sampling. Although LSA displays the best accuracy improvement under the combination of MNIST &LeNet-5, the retraining results under other combinations guided by LSA are even worse than random sampling the same size test cases for retraining. To evaluate the effectiveness more precisely, we use the Wilcoxon rank-sum test [20] to check whether ATS outperforms other baseline approaches. We implement the one-side Wilcoxon rank-sum test to check whether the baseline method is greater or worse than the ATS. If the p-value is less than 0.05, we reject the null hypothesis H_0 and accept the alternative hypothesis H_1 that one method is stochastically greater than another. Otherwise, we regard there is no significant difference between the two methods. The results show that no baseline exceeds ATS statistically. Only in very few configurations, there are some prioritization methods that show similar performance to ATS.

Furthermore, considering the Fashion & LeNet-1 discussed above, ATS achieves a significant retraining improvement over Gini and Max*p*. Such a result supports our assumption that, instead of selecting more faults with less diversity, ATS could select a subset with enough faults with more fault types, which is more effective to enhance the DNN model.

6 DISCUSSION

This section further discusses the ATS with adaptive random testing and active learning and also demonstrates the threats to the validity of this paper.

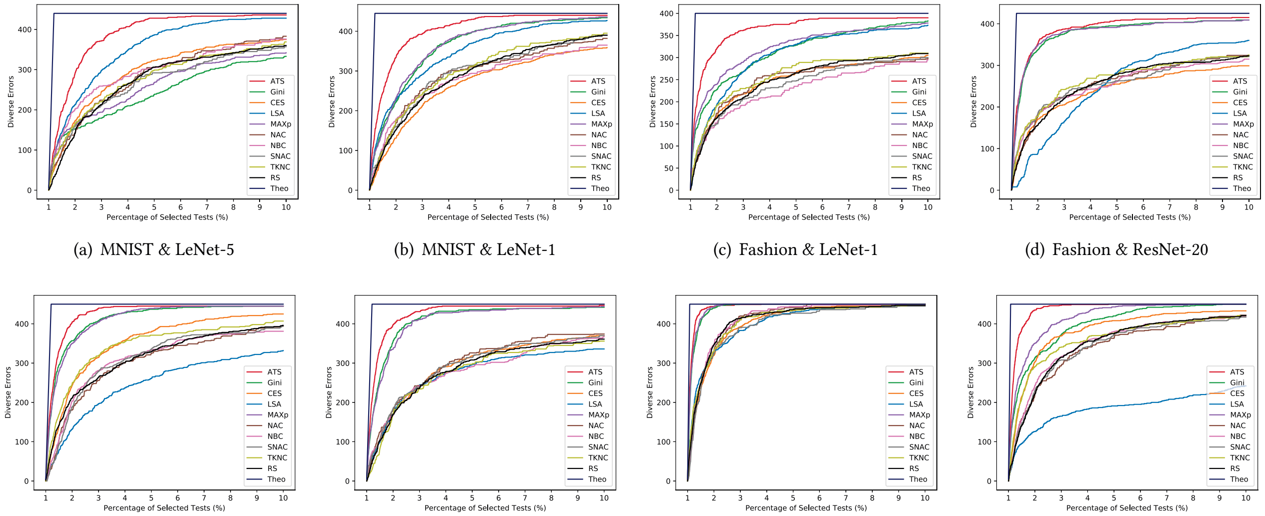
6.1 The Effectiveness of ATS

From all experimental results, ATS is more effective than other existing test selection methods. By analyzing the results of other baselines, we observe that although structural coverage criteria are effective in classic testing applications, neuron coverage criteria seem ineffective to be a guide criterion for DNN test selection. Besides, compared with existing prioritization methods, ATS shows a better and more stable result.

Similar to the idea of classic adaptive random testing (ART) [15, 18], ATS selects the test case based on the fitness between each candidate c from the candidate set C against the executed set E. However, at the same time, ART cannot be directly applied to DNN testing. One reason is that different from the low dimensional input domain discussed in ART, the input data dimension of the DNN model is pretty large [68]. The curse of dimensionality lead to the

Table 4: The average fault detection rate for each configuration.

Fault	Detect(%)	Selecting 5% tests										Selecting 10% tests									
		RS	**Coverage-Guided**				**Prioritization**				**ATS**	**RS**	**Coverage-Guided**				**Prioritization**				**ATS**
Data	Model		NAC	NBC	SNAC	TKNC	Gini	CES	LSA	Max*p*			NAC	NBC	SNAC	TKNC	Gini	CES	LSA	Max*p*	
MNIST	LeNet5	8.4	8.8	9.4	8.4	8.6	23.9	10.6	**40.8**	24.8	42.5	8.5	8.7	9.2	8.6	8.5	23.7	10.6	40.4	24.7	**40.9**
	LeNet1	9.9	10.1	9.7	9.7	10.0	36.6	8.8	32.7	38.0	**48.7**	9.8	9.9	9.7	9.8	9.7	36.4	8.7	32.6	37.8	**47.0**
Fashion	LeNet1	18.7	18.7	18.4	18.5	20.9	53.2	16.1	31.3	**54.8**	53.2	18.6	18.6	18.6	18.3	20.3	52.5	16.0	31.3	**53.9**	51.5
	ResNet20	18.8	18.9	19.0	17.6	19.6	53.1	17.9	23.9	52.1	**57.4**	18.6	18.7	19.1	18.0	19.3	52.1	18.0	24.6	51.4	**55.7**
CIFAR	VGG16	19.2	17.0	19.2	18.5	22.3	50.4	25.9	13.5	51.0	**56.4**	19.3	17.4	19.0	18.5	21.6	49.6	25.8	13.6	50.1	**55.0**
	ResNet20	13.5	13.8	14.1	13.6	14.1	43.3	15.1	13.8	43.8	**52.4**	13.3	13.8	14.0	13.5	13.6	42.6	15.1	13.7	43.4	**51.2**
SVHN	LeNet5	16.4	15.7	16.3	16.2	15.9	53.8	14.9	24.4	**54.3**	53.1	16.3	15.7	16.1	15.9	15.9	52.7	14.8	24.1	**53.0**	49.4
	VGG16	7.1	7.3	7.5	7.2	8.4	28.5	12.5	6.5	30.0	**44.3**	7.4	7.5	7.8	7.5	8.6	28.9	13.2	6.5	30.9	**40.1**



(a) MNIST & LeNet-5 (b) MNIST & LeNet-1 (c) Fashion & LeNet-1 (d) Fashion & ResNet-20

(e) CIFAR & VGG-16 (f) CIFAR & ResNet-20 (g) SVHN & LeNet-5 (h) SVHN & VGG-16

Figure 2: The cumulative sum of the fault types found by specific test selection methods.

Table 5: When selecting 10% tests, the ratio of area under the curve between each selection method to theoretical.

RAUC	MNIST		Fashion		CIFAR		SVHN	
(%)	LeNet5	LeNet1	LeNet1	ResNet20	VGG16	ResNet20	LeNet5	VGG16
RS	62.97	65.97	62.87	61.25	70.41	63.90	91.34	77.67
NAC	65.13	66.39	62.83	61.38	68.12	67.38	92.16	75.84
NBC	67.74	63.91	57.53	60.52	70.05	62.97	92.12	78.15
SNAC	61.92	67.06	60.41	61.38	70.08	66.05	88.87	76.42
TKNC	63.95	68.15	65.58	63.42	77.20	62.70	91.03	80.57
Gini	55.73	82.57	79.48	89.97	92.66	92.00	98.18	88.94
CES	67.82	60.84	62.58	57.41	79.45	65.57	90.19	85.37
LSA	80.92	79.18	77.01	60.69	55.60	61.51	89.72	41.15
Max*p*	59.75	83.00	81.51	90.06	92.50	91.34	98.43	92.71
ATS	**89.32**	**91.48**	**91.88**	**92.20**	**95.60**	**95.43**	**98.80**	**97.52**

invalidation of fitness metric. Thus we choose to design ATS in the output domain. The other reason is that the target between random testing and DNN testing is different. ART aims to reduce the resource cost of constructing test cases randomly. It means that only if the computational cost of ART is much lighter than RT then we can consider ART is an effective and efficient test selection method [5, 60]. However, for DNN testing, the cost of manual labeling is often extremely expensive, especially when the labeling task requires professional knowledge [51]. In this case, the controversy of computational overheads is no longer essential for DNN test selection because the resource cost in test execution and data selection is much cheaper in comparison with the consumption of manually labeling.

To further discuss the significance of ATS for DNN testing, on the basis of RQ3, we obtain the accuracy improvement when retraining the model with the whole candidate set. First, in Row **Ori Acc**, we give the original accuracy of each model on the newly

Table 6: The DNNs' accuracy improvement value after retraining with the selected tests.

	Dataset	Model	ATS	Coverage-Guided				Prioritization				RS		ATS	Coverage-Guided				Prioritization				RS
				NAC	NBC	SNAC	TKNC	Gini	CES	LSA	Max$p$				NAC	NBC	SNAC	TKNC	Gini	CES	LSA	Max$p$	
Select 2.5%	MNIST	LeNet5	2.38	1.39	1.83	1.60	1.63	1.42	1.65	**2.60**	1.52	1.42	Select 7.5%	4.09	2.80	2.97	2.70	2.94	2.46	2.73	**4.16**	2.68	2.55
		LeNet1	**2.37**	1.57	1.67	1.55	1.65	1.78	1.08	2.07	2.01	1.43		**3.97**	2.97	2.85	2.90	2.81	3.46	2.00	3.31	3.66	2.51
	Fashion	LeNet1	**1.53**	0.92	0.79	0.82	0.91	1.14	0.70	1.11	1.19	0.84		**2.82**	1.93	1.66	1.89	1.95	2.32	1.40	1.92	2.45	1.72
		ResNet20	**3.34**	2.86	2.86	2.83	2.75	3.22	2.57	2.23	3.24	2.66		**4.96**	4.17	4.30	4.08	4.15	4.86	3.88	3.57	4.94	4.04
	CIFAR	VGG16	**2.36**	1.85	1.83	1.84	1.98	2.27	1.95	1.34	2.21	1.86		**3.73**	2.61	2.67	2.67	2.78	3.48	2.75	1.79	3.52	2.65
		ResNet20	**0.99**	0.45	0.42	0.50	0.31	0.88	0.40	0.15	0.88	0.38		**1.68**	0.96	0.95	1.07	1.03	1.52	1.00	0.55	1.49	0.94
	SVHN	LeNet5	**1.74**	0.90	0.95	0.93	0.84	1.49	0.99	1.26	1.50	1.14		**3.17**	1.73	1.90	1.77	1.81	2.90	1.71	2.03	2.96	1.90
		VGG16	**2.09**	1.12	1.17	1.10	1.23	1.52	1.37	0.75	1.52	1.09		**2.95**	1.71	1.82	1.75	1.83	2.32	2.12	0.95	2.48	1.73
Select 5%	MNIST	LeNet5	3.52	2.28	2.43	2.29	2.46	1.96	2.26	**3.59**	2.16	2.08	Select 10%	4.53	3.20	3.31	3.16	3.38	2.99	3.05	**4.57**	3.13	2.94
		LeNet1	**3.46**	2.42	2.28	2.34	2.31	2.74	1.58	2.88	2.94	2.05		**4.30**	3.38	3.19	3.38	3.30	3.95	2.27	3.70	4.14	2.87
	Fashion	LeNet1	**2.40**	1.48	1.46	1.44	1.63	1.71	1.07	1.57	1.96	1.41		**3.15**	2.24	2.19	2.29	2.33	2.75	1.75	2.16	2.87	2.11
		ResNet20	**4.36**	3.68	3.66	3.76	3.74	4.25	3.41	2.96	4.23	3.51		5.30	4.62	4.78	4.49	4.72	**5.37**	4.37	3.79	**5.37**	4.36
	CIFAR	VGG16	**3.08**	2.25	2.29	2.32	2.42	2.88	2.42	1.58	2.93	2.28		**4.13**	2.94	2.91	2.93	3.16	3.99	3.08	2.29	3.95	2.95
		ResNet20	**1.44**	0.75	0.72	0.91	0.79	1.30	0.79	0.40	1.30	0.72		**1.83**	1.21	1.09	1.21	1.15	1.69	1.19	0.70	1.70	1.11
	SVHN	LeNet5	**2.64**	1.33	1.41	1.36	1.35	2.31	1.36	1.67	2.41	1.47		**3.59**	2.16	2.24	2.07	2.27	3.47	1.98	2.38	3.56	2.21
		VGG16	**2.72**	1.47	1.54	1.55	1.58	1.97	1.82	0.88	2.10	1.49		**3.18**	1.94	2.02	1.99	1.97	2.61	2.32	1.09	2.80	1.95

[1] Colored baseline blocks represent the accuracy improvement between ATS and baseline is similar statistically, and uncolored blocks represent ATS is greater than baseline in corresponding configurations.

[2] For each selection ratio, we **bold** the maximum accuracy improvement value of each DNN & dataset combination.

Table 7: Model optimization effect compared with retraining with all candidate tests.

Dataset	MNIST		Fashion		CIFAR		SVHN	
Model	LeNet5	LeNet1	LeNet1	ResNet20	VGG16	ResNet20	LeNet5	VGG16
Ori Acc	91.01	89.95	79.40	79.78	78.32	85.35	82.39	92.30
100% tests	6.48	6.30	5.15	8.42	8.59	2.76	7.25	4.40
10% ATS	4.53	4.30	3.15	5.30	4.13	1.83	3.59	3.18
Imp%	**69.9%**	**68.3%**	**61.2%**	**62.9%**	**48.1%**	**66.2%**	**49.5%**	**72.1%**

constructed test set. The accuracy improvement is shown in Tab. 7 Row **100% tests**. Refer to Tab. 6, we show the accuracy improvement when use ATS select 10% test cases from the candidate set, denoted as **10% ATS**. And in the last row, we calculated the improvement ratio of 10% ATS to 100% tests. The results show that for DNN testing, an appropriate test case selection method could optimize the DNN model with a much lighter labeling cost.

Based on the experimental results and discussion, we draw the conclusion that the adaptive test selection method ATS could select test cases from a massive unlabeled dataset automatically. The test set selected by ATS contains numerous faults with diverse types. From the results of RQ3, we found that such a test set could enhance the DNN effectively and efficiently.

6.2 Comparison with Active Learning

Active learning (AL) is a subfield of machine learning (ML) in which a learning algorithm aims to achieve good accuracy with fewer training samples by interactively querying the oracles to label new data points [50, 66]. Thus, both AL and test case selection attempt to overcome the labeling bottleneck by selecting data from an unlabeled candidate set.

Although the design motivation of the two technologies overlaps slightly, it is worth emphasizing that there are inherent differences between test selection and active learning. The kernel purpose of AL is to obtain a model of better performance with less ground truth query efforts. However, from the perspective of DNN-driven system testing, ATS focuses on the testing and debugging process of a pre-trained model, which aims to expose unpredicted behaviors and to enhance the model. To this end, we implement the research experiments of faults detection rate and faults diversity in RQ1&RQ2.

Furthermore, in evaluation, we choose a representative AL method, namely Max$p$, as one of the test prioritization baseline [19, 52]. It is almost the most commonly applied uncertainty-sampling strategy in active learning [66], which employs the same way of ATS to measure input uncertainty. The experimental results show that compared with random sampling, Max-p is effective as a selection method. Meanwhile, ATS outperforms Max$p$ under most of the model-dataset combinations because we design an adaptive selection strategy to overcome the weakness of priority-based selection.

6.3 Threats to Validity

Subject selection. The selection of training datasets and DNN models could be a threat to validity. We alleviate this threat by employing large-scale datasets and four well-designed DNN models in the experiment. Further, for each studied dataset, we employed two DNN models with different numbers of neurons and architecture to evaluate the performance of ATS. However, some of the experiment results may not be perfectly generalized to other datasets and DNN models.

Data simulation. Further, we employ augmented data to simulate the unseen inputs for DNNs, which may cause another threat. Although the augmented operators are common data noises in the virtual environment, it is impossible to guarantee that the distribution of the real unseen input is the same as our simulation. Additional experiments based on real unseen inputs needs to be conducted in future work.

7 RELATED WORKS

This section introduces the related works on two aspects: the testing of deep learning systems and adaptive random testing.

7.1 The Testing of Deep Learning Systems

To measure the test adequacy of deep learning systems, several test criteria have been proposed. Pei et al. [49] proposed the first white-box testing framework DeepXplore, which aims to identify and generate the corner-case inputs that may induce different behaviors over different DNNs. Ma et al. [41] further refined the idea of coverage and proposed fine-grained multi-layer testing criteria to guarantee the quality of the model entirely. Based on the mutation test, Ma et al. [29, 42] proposed DeepMutation and DeepMutation++ to evaluate the quality of datasets with model-level mutation operators.

Based on the above coverage criteria, different DNN test application techniques are proposed. Tian et al. [59] presented DeepTest to generate test cases by maximizing the number of activated neurons. Similarly, Zhang et al. [67] give an adversarial network to generate different weather conditions driving scenes to increase the diversity of datasets.

However, as pointed out in the introduction section, many existing testing techniques give less consideration to the characteristic of the DNN-driven system, which results in the difference between the improvement of the DNN system and the repair of the traditional software system. This is why recent studies [23, 27] argue the guidance of existing testing criteria, especially the neuron coverage criteria. In addition, other test case generation techniques or prioritization techniques are difficult to give a suitable testing criterion or are based on a low-guidance (neuron coverage) test adequacy criterion. In this paper, the method ATS is intended to break out of the scope of neuron coverage design and start from the sight of the DNN-driven software system to identify and select test suites of high model improvement capability.

7.2 Adaptive Random Testing

Test selection is a classic research topic of software testing. There are plenty of techniques and methods are proposed for conventional software systems.

Among them, one of the well-known test selection method is Adaptive Random Testing (ART). Chen et al. [15] proposed the first specific algorithm of this method (FSCS-ART). The core idea is to choose a new test case, k candidates are randomly generated. For each candidate c_i, the closest previously executed test is located, and the distance d_i is determined. The candidate with the largest d_i is selected, and the other candidates are discarded. Besides, another test selection technique, Antirandom testing [44] is also based on the concept of distance to distribute test cases. It is almost deterministic, which means it requires the number of test cases to be chosen in advance. The core of the technology to improve random testing efficiency is to achieve even spread test case distribution. Therefore, a number of different methods using different intuitions to achieve this goal have been investigated in the literature. An example is Restricted Random Testing (RRT) [13], which is based on the notion of adaptive exclusion, and the goal of this method is to select the test case outside of the exclusion zones. ART by Partitioning[18] uses a rather different intuition, partitioning the input domain in essence, and allocating test cases evenly to partitions, achieves even spread. Besides, other attempts to take advantage of failure region contiguity, but using various other intuitions to achieve the even spreading of test cases, including Quasi-Random Testing [17], and Lattice-Based ART [45].

In the testing of DNN-driven software systems, it is easy to get enough unlabeled test cases. However, choosing proper data to label manually is always difficult. Our work aims to help testers select valuable tests from massive unlabeled tests efficiently.

8 CONCLUSION

In this paper, we propose ATS an adaptive test selection method for Deep Neural Networks. We design a fitness computation method to adaptively determine which test in the candidate set is more suitable to be labeled manually. The experimental results in this paper demonstrate that ATS can effectively help testers choose more valuable test cases to improve the quality of the model. Different from current test selection methods, ATS is guided by fault pattern design and candidate fitness metric for test selection of Deep Neural Networks. We provide an alternative view for identifying and selecting the tests. We expect that ATS can inspire testers to select test suites with enough diversity and faults detection ability, which can efficiently improve the system.

ACKNOWLEDGEMENT

We would like to thank anonymous reviewers for their insightful and constructive comments. This project was partially funded by the National Natural Science Foundation of China under Grant Nos. 62002158, 61832009, and 61932012, and the Science, Technology and Innovation Commission of Shenzhen Municipality (No. CJGJZD20200617103001003).

REFERENCES

[1] [n.d.]. Amazon promises fix for creepy Alexa laugh - BBC News. https://www.bbc.com/news/technology-43325230. (Accessed on 08/23/2021).

[2] [n.d.]. A Google self-driving car caused a crash for the first time - The Verge. https://www.theverge.com/2016/2/29/11134344/google-self-driving-car-crash-report. (Accessed on 09/04/2021).

[3] [n.d.]. Tesla' s Latest Autopilot Death Looks Just Like a Prior Crash | WIRED. https://www.wired.com/story/teslas-latest-autopilot-death-looks-like-prior-crash/. (Accessed on 09/04/2021).

[4] Paul Eric Ammann and John C Knight. 1988. Data diversity: An approach to software fault tolerance. *Ieee transactions on computers* 37, 4 (1988), 418–425.

[5] Saswat Anand, Edmund K Burke, Tsong Yueh Chen, John Clark, Myra B Cohen, Wolfgang Grieskamp, Mark Harman, Mary Jean Harrold, Phil McMinn, Antonia Bertolino, et al. 2013. An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software* 86, 8 (2013), 1978–2001.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[7] Arlinta C Barus, Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, Robert Merkel, and Gregg Rothermel. 2016. A cost-effective random testing method for programs with non-numeric inputs. *IEEE Trans. Comput.* 65, 12 (2016), 3509–3523.

[8] Peter G Bishop. 1993. The variation of software survival time for different operational input profiles (or why you can wait a long time for a big bug to fail). In

FTCS-23 The Twenty-Third International Symposium on Fault-Tolerant Computing. IEEE, 98–107.

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

[10] Otto Bretscher. 1997. *Linear algebra with applications.* Prentice Hall Eaglewood Cliffs, NJ.

[11] Paulo MS Bueno, Mario Jino, and W Eric Wong. 2014. Diversity oriented test data generation using metaheuristic search techniques. *Information Sciences* 259 (2014), 490–509.

[12] FT Chan, Tsong Yueh Chen, IK Mak, and Yuen-Tak Yu. 1996. Proportional sampling strategy: guidelines for software testing practitioners. *Information and Software Technology* 38, 12 (1996), 775–782.

[13] Kp Chan, Ty Chen, and D Towey. 2006. Restricted random testing: Adaptive random testing by exclusion. *International Journal of Software Engineering & Knowledge Engineering* 16, 4 (2006), 553–584.

[14] Junjie Chen, Ming Yan, Zan Wang, Yuning Kang, and Zhuo Wu. 2020. Deep neural network test coverage: How far are we? *arXiv preprint arXiv:2010.04946* (2020).

[15] Tsong Yueh Chen, Hing Leung, and I. K. Mak. 2004. Adaptive Random Testing. (2004).

[16] Tsong Yueh Chen, R Merkel, PK Wong, and G Eddy. 2004. Adaptive random testing through dynamic partitioning. In *Fourth International Conference onQuality Software, 2004. QSIC 2004. Proceedings.* IEEE, 79–86.

[17] Tsong Yueh Chen and Robert G. Merkel. 2007. Quasi-Random Testing. *IEEE Transactions on Reliability* 56 (2007), 562–568.

[18] T. Y. Chen, R. G. Merkel, P. K. Wong, and G. Eddy. 2004. Adaptive random testing through dynamic partitioning. In *Quality Software, 2004. QSIC 2004. Proceedings. Fourth International Conference on.*

[19] Aron Culotta and Andrew McCallum. 2005. Reducing labeling effort for structured prediction tasks. In *AAAI*, Vol. 5. 746–751.

[20] Jack Cuzick. 1985. A Wilcoxon-type test for trend. *Statistics in medicine* 4, 1 (1985), 87–90.

[21] L De Capitani and D De Martini. 2011. On stochastic orderings of the Wilcoxon rank sum test statistic—with applications to reproducibility probability estimation testing. *Statistics & probability letters* 81, 8 (2011), 937–946.

[22] Daniel Di Nardo, Nadia Alshahwan, Lionel Briand, and Yvan Labiche. 2013. Coverage-based test case prioritisation: An industrial case study. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation.* IEEE, 302–311.

[23] Yang Feng, Qingkai Shi, Xinyu Gao, Jun Wan, Chunrong Fang, and Zhenyu Chen. 2020. DeepGini: prioritizing massive tests to enhance the robustness of deep neural networks. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis.* 177–188.

[24] George B Finelli. 1991. NASA software failure characterization experiments. *Reliability Engineering & System Safety* 32, 1-2 (1991), 155–169.

[25] Abraham Adolf Fraenkel, Yehoshua Bar-Hillel, and Azriel Levy. 1973. *Foundations of set theory.* Elsevier.

[26] Jianmin Guo, Yu Jiang, Yue Zhao, Quan Chen, and Jiaguang Sun. 2018. 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.* 739–743.

[27] Fabrice Harel-Canada, Lingxiao Wang, Muhammad Ali Gulzar, Quanquan Gu, and Miryung Kim. 2020. Is neuron coverage a meaningful measure for testing deep neural networks?. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* 851–862.

[28] 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.

[29] Qiang Hu, Lei Ma, Xiaofei Xie, Bing Yu, Yang Liu, and Jianjun Zhao. 2019. DeepMutation++: A Mutation Testing Framework for Deep Learning Systems. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE).* IEEE, 1158–1161.

[30] Gunel Jahangirova and Paolo Tonella. 2020. An Empirical Evaluation of Mutation Operators for Deep Learning Systems. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST).*

[31] Bo Jiang, Zhenyu Zhang, Wing Kwong Chan, and TH Tse. 2009. Adaptive random test case prioritization. In *2009 IEEE/ACM International Conference on Automated Software Engineering.* IEEE, 233–244.

[32] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding Deep Learning System Testing Using Surprise Adequacy. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE).*

[33] A. Krizhevsky and G. Hinton. 2009. Learning multiple layers of features from tiny images. *Handbook of Systemic Autoimmune Diseases* 1, 4 (2009).

[34] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. 2009. CIFAR10. https://www.cs.toronto.edu/~kriz/cifar.html.

[35] Alexey Kurakin, Ian Goodfellow, Samy Bengio, et al. 2016. Adversarial examples in the physical world.

[36] Hugo Larochelle, Yoshua Bengio, Jérôme Louradour, and Pascal Lamblin. 2009. Exploring strategies for training deep neural networks. *Journal of machine learning research* 10, 1 (2009).

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

[38] David D Lewis and William A Gale. 1994. A sequential algorithm for training text classifiers. In *SIGIR' 94.* Springer, 3–12.

[39] Zenan Li, Xiaoxing Ma, Chang Xu, and Chun Cao. 2019. Structural coverage criteria for neural networks could be misleading. In *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER).* IEEE, 89–92.

[40] Zenan Li, Xiaoxing Ma, Chang Xu, Chun Cao, Jingwei Xu, and Jian Lü. 2019. Boosting operational DNN testing efficiency through conditioning. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* 499–509.

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

[42] Lei Ma, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Felix Juefei-Xu, Chao Xie, Li Li, Yang Liu, Jianjun Zhao, et al. 2018. Deepmutation: Mutation testing of deep learning systems. In *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE).* IEEE, 100–111.

[43] Wei Ma, Mike Papadakis, Anestis Tsakmalis, Maxime Cordy, and Yves Le Traon. 2021. Test selection for deep learning systems. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30, 2 (2021), 1–22.

[44] Y. K. Malaiya. 1995. Antirandom testing: getting the most out of black-box testing. In *International Symposium on Software Reliability Engineering.*

[45] Johannes Mayer. 2005. Lattice-based adaptive random testing. In *IEEE/ACM International Conference on Automated Software Engineering.*

[46] Agnieszka Mikołajczyk and Michał Grochowski. 2018. Data augmentation for improving deep learning in image classification problem. In *2018 international interdisciplinary PhD workshop (IIPhDW).* IEEE, 117–122.

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

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

[49] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. *Getmobile Mobile Computing & Communications* 22, 3 (2017).

[50] Burr Settles. 2009. Active learning literature survey. (2009).

[51] Burr Settles. 2011. From Theories to Queries: Active Learning in Practice. In *Active Learning and Experimental Design workshop In conjunction with AISTATS 2010 (Proceedings of Machine Learning Research),* Isabelle Guyon, Gavin Cawley, Gideon Dror, Vincent Lemaire, and Alexander Statnikov (Eds.), Vol. 16. PMLR, Sardinia, Italy, 1–18. https://proceedings.mlr.press/v16/settles11a.html

[52] Burr Settles and Mark Craven. 2008. An analysis of active learning strategies for sequence labeling tasks. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing.* 1070–1079.

[53] Connor Shorten and Taghi M Khoshgoftaar. 2019. A survey on image data augmentation for deep learning. *Journal of Big Data* 6, 1 (2019), 1–48.

[54] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[55] Satinder P Singh, Michael J Kearns, Diane J Litman, and Marilyn A Walker. 2000. Reinforcement learning for spoken dialogue systems. In *Advances in Neural Information Processing Systems.* 956–962.

[56] Gilbert Strang, Gilbert Strang, Gilbert Strang, and Gilbert Strang. 1993. *Introduction to linear algebra.* Vol. 3. Wellesley-Cambridge Press Wellesley, MA.

[57] Youcheng Sun, Xiaowei Huang, Daniel Kroening, James Sharp, Matthew Hill, and Rob Ashmore. 2019. DeepConcolic: Testing and debugging deep neural networks. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion).* IEEE, 111–114.

[58] Youcheng Sun, Xiaowei Huang, Daniel Kroening, James Sharp, Matthew Hill, and Rob Ashmore. 2019. Testing Deep Neural Networks. arXiv:cs.LG/1803.04792

[59] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2017. DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars. (2017).

[60] Dave Towey. 2007. Adaptive random testing; ubiquitous testing to support ubiquitous computing. In *Proceedings of the Korea Society of Information Technology Applications Conference.* The Korea Society of Information Technology Applications, 138–138.

[61] Lee J White and Edward I Cohen. 1980. A domain strategy for computer program testing. *IEEE transactions on software engineering* 3 (1980), 247–257.

[62] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. arXiv:cs.LG/cs.LG/1708.07747

[63] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. 2019. 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*. 146–157.

[64] Christopher J.C. Burges Yann LeCun, Corinna Cortes. 1998. MNIST. http://yann.lecun.com/exdb/mnist/.

[65] Shin Yoo and Mark Harman. 2012. Regression testing minimization, selection and prioritization: a survey. *Software testing, verification and reliability* 22, 2 (2012), 67–120.

[66] Xueying Zhan, Huan Liu, Qing Li, and Antoni B Chan. [n.d.]. A Comparative Survey: Benchmarking for Pool-based Active Learning. ([n. d.]).

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

[68] Arthur Zimek, Erich Schubert, and Hans-Peter Kriegel. 2012. A survey on unsupervised outlier detection in high-dimensional numerical data. *Statistical Analysis and Data Mining: The ASA Data Science Journal* 5, 5 (2012), 363–387.