MTGAN: Extending Test Case set for Deep Learning Image Classifier

Erhu LIU††††a, Song HUANG††b, Cheng ZONG††c, Changyou ZHENG†d, Yongming YAO†e, Jing ZHU††f, Shiqi TANG††g, Nonmembers, and Yanqiu WANG††††h, Member

SUMMARY During the recent several years, deep learning has achieved excellent results in image recognition, voice processing, and other research areas, which has set off a new upsurge of research and applications. Internal defects and external malicious attacks may threaten the safe and reliable operation of a deep learning system and even cause unbearing consequences. The technology of testing deep learning systems is still in its infancy. Traditional software testing technology is not applicable to test deep learning systems. In addition, the characteristics of deep learning such as complex application scenarios, the high dimensionality of input data, and poor operation logic bring new challenges to the testing work. This paper focuses on the problem of test case generation and points out that adversarial examples can be used as test cases. Then the paper proposes MTGAN which is a framework to generate test cases for deep learning image classifiers based on Generative Adversarial Network. Finally, this paper evaluates the effectiveness of MTGAN.

key words: test case, deep learning, adversarial example, GAN

1. Introduction

1.1 Application Scenes of Deep Learning

Thanks to the rapid development of artificial neural networks and computer hardware devices, deep learning has subverted traditional techniques and achieved impressive successes in many application areas such as image recognition [1], speech recognition [2], text analysis [3], [4], natural language processing [5], automatic driving [6], industrial robots [7], medical diagnosis [8], computational biology [9] and so on. Image recognition adopted deep learning earlier than other application areas, and benefited from ImageNet Large Scale Visual Recognition Challenge [10], deep learning used for image recognition is more mature than other application areas. At present state-of-the-art image recognition techniques based on deep learning are not limited to 2D images. Guo et al. [11] proposed a method that used a multi-view of 2D images to recognize 3D targets. The speech recognition techniques based on deep learning have also achieved exciting successes. The introduction of different deep learning models such as Convolutional Neural Network (CNN), Long Short Term Memory Network (LSTM) and Feed-forward Sequential Memory Network (FSMN) have greatly improved the accuracy of speech recognition. Besides the mature application areas mentioned above, deep learning is also expanding to other new application areas, such as accelerated optimization [12], network control management [13], distributed Internet of Things [14] and so on. We believe that in the foreseeable future deep learning will further affect our daily lives.

1.2 Significance and Challenges of Testing Deep Learning Systems

Why do we say that deep learning systems need to be tested? Firstly, even though the developers always need to test and verify their deep learning systems before deployments, but this evaluation work is not enough to ensure the correctness, stability, and robustness of deep learning systems. Because developers tend to pay more attention to the system accuracy, and use the test cases sampled from given standard datasets to test and verify the deep learning systems. The diversity of standard data is not enough to meet all the situations encountered in the real world. In this case, the test and verification datasets are usually not sufficient to trigger the defects in the deep learning systems.

Secondly, deep learning systems are vulnerable to malicious attacks, in addition, more and more security-critical areas have adopted deep learning systems. Not only the inner defects but also the external attacks may lead to serious system errors or even catastrophic results. In order to ensure the correctness, stability, and robustness of deep learning systems, software testers need to treat the inner defects and external attacks from the perspective of the test. Furthermore, testers may leverage software test techniques to find defects in the deep learning system as much as possible and help the deep learning system resist external malicious attacks.

Techniques for testing traditional software have been developed over the past several decades. Many useful
methods have been proposed and proved to be effective in practice. This accumulated knowledge provides a basis for traditional software testing, however, the deep learning system is quite different from the traditional software due to some external and internal characteristics.

The external characteristic is that the application scenarios of deep learning systems are complex, and the dimensions of input data are always much higher than traditional software. A higher input dimension leads to a huge input space, and it is difficult for testers to construct a test suite covering the input space. The internal characteristic is that the operational logic of deep learning systems is difficult to interpret. The operational logic and control flow of traditional software are always encoded by the programmers. Different from this, deep learning systems learn operational logic from training data by themselves, and even the developers have no idea about the operational logic.

Because of the external and internal characteristics, existing software techniques can not be applied directly to test deep learning systems, and researchers have to find new effective methods. Since testing deep learning is an emerging research area researchers have a long way to go. After research, we find that the challenges in traditional software testing such as test oracle problem [15], generation of test cases, measurement of test adequacy, and so on become more prominent when come to deep learning system testing. Besides these traditional challenges, deep learning system testing also brings some new challenges [16], [17].

This paper tries to relieve the difficult problem of generating test cases in deep learning system testing. “GAN” is an abbreviation for “Generative Adversarial Network”, and we use GAN instead of Generative Adversarial Network for simplification in the following paragraph. As we all know, GAN is an effective generating framework. A suitable GAN after appropriately trained can be used to generate test cases for deep learning systems. The input of GAN can even be random noise, and this characteristic can reduce the amount of data required for deep learning system testing. So we propose a new generation method based on GAN. In order to evaluate the effectiveness, we focus on the realistic problem: image classification. The deep learning image classifier is a classical deep learning system, and we take deep learning image classifier as the research object.

1.3 Adversarial Examples’ Applicability for Testing Deep Learning Systems

Christian Szegedy et al. [18] proposed an interesting phenomenon about neural networks in 2014. That is a picture that can be misclassified with high confidence and even misclassified into specified categories by adding tiny perturbations to the picture. Figure 1 shows two examples of this phenomenon. The most-left two images of Fig. 1 show original images with label “2” and “snow mountain”, the middle two images are the perturbation that will be added to the original images, and the most-right images are the new images superimposed the original images and the perturbations. The new perturbed images are classified as “6” and “dog”.

The perturbed and misclassified images are named as adversarial examples, and the corresponding original images are called seed inputs. Intuitively, adversarial examples can effectively trigger the internal defects of the deep learning image classifier, and cause the system to make mistakes. Therefore, adversarial examples are deemed to be threats to the deep learning systems, even be used to carry out adversarial attacks to deep learning systems. However, from the perspective of software testers, adversarial examples are regarded as good test cases that can effectively trigger the defects of deep learning systems under test. Even more, adversarial examples can be used to improve system security against adversarial attacks.

1.4 Contributions of This Work

Until now, no research has given the benchmark of how many defects should be uncovered in software testing. But when we test a software we all follow a consensus that the more defects detected the better. In the same way, when we test a deep learning system, we hope to construct a test case set with enough ability for defect detection to enhance users’ confidence. At the same time, we should also consider the test overhead, and the scale of the test case set should not be unlimited. As mentioned above, the input space of deep learning systems is always huge, and difficult to be covered by a limited test case set. So at the beginning of test work, we need to construct a test case set with high defect-detecting ability. In order to enhance the ability of defect detection, we can make efforts from the flowing two aspects:

1. Adding adversarial examples to the test case set, within the prescribed scope the more the better. Adversarial examples can help to improve test effectiveness.

2. Choosing adversarial examples with diversity and uniformly distributed in the input space to trigger different kinds of defects of the deep learning system under test. This work can help to improve test efficiency.

It is difficult to meet these two requirements by using
single adversarial examples constructing method. If only a few adversarial examples derived from the same seed input are added to the test case set, collecting enough test cases will cost unacceptable time and resource. On the other hand, if multiple adversarial examples derived from the same seed input are added to the test case set, the test cases are lacking diversity. Is there any method to relieve this situation?

Our research aims at the above two aspects, and make efforts to extend test case set for deep learning image classifier. The extended test case set can get a higher fault-detecting ability. We highlight the contributions of our work as follows:

- We proposed a deep learning image classifier test case generating framework MTGAN based on GAN. MTGAN is short for Metamorphic Transformation Generative Adversarial Networks. The framework MTGAN consists of two GANs with different functions. It takes 100-dimensional random noise as inputs and outputs derived test cases with a high fault-detecting ability for the deep learning system under test.

- We proposed a training strategy for the MTGAN to ensure that the test cases generated by the MTGAN are all labeled automatically. Metamorphic transformation in this paper means keeping the image semantics before and after perturbation. MTGAN makes the metamorphic transformation to inputted images, and the labels of generated images can easily to get. The automatically labeled images will effectively reduce the labor costs in the testing work.

- We performed a series of experiments to evaluate the MTGAN’s effectiveness in generating test cases for deep learning image classifiers. The results show that the framework is effective in generating test cases with high fault-detecting ability.

The rest of this paper is organized as follows. Section 2 is the background and motivation of our work. Section 3 describes the design of our approach in detail. Section 4 and Sect. 5 show the experiment setup and results. Section 6 presents a preliminary study of the related works. Finally, we conclude and plan the next work in Sect. 7.

2. Background and Motivation

2.1 Generating Adversarial Examples for Deep Learning Image Classifiers

We have a consensus that deep learning image classifiers take human’s decisions as benchmarks. If we add small perturbations to a seed input and the change is imperceptible to humans, the classification result of a new image must stay the same with the seed input. Adversarial examples are these perturbed images which are imperceptible to human but cause the deep learning system to misclassify with high confidence. There are two basic criteria for adversarial example: 1. Imperceptible: The image semantics keeps the same to human beings before and after perturbation. 2. Misclassified: The perturbed images may be misclassified with high confidence.

Schematic diagram Fig. 2 shows the process of generating an adversarial example of the handwritten digit. The dashed curve represents the boundary of category “6” and category “2”. The digits in the dots are the manual labels. The grey dots on the left side of the boundary represent the digit images classified as “6” and the yellow dots on the right side of the boundary represent the digit images classified as “2”. The yellow dot with a red border represents a seed input with label “2” and it was correctly classified before perturbed. When the perturbation process starts, the seed input is gradually moved to category “6” under a specific perturbing strategy until it passes through the boundary and be misidentified as “6” by the deep learning classifier. The perturbed seed input becomes an adversarial example to the image classifier.

2.2 Generative Adversarial Networks (GAN)

Generative Adversarial Networks (GAN) was proposed by Ian Goodfellow et al. [19] in 2014 and used as a generation framework. GAN always consists of two deep neural networks. The first one is a generator denoted as G, and the other is a discriminator denoted as D. The generator G attempts to generate realistic images from random noises, and the discriminator D strives to detect the fake images generated by the generator. During the training phase, the losses are fed back to G and D, which will help to adjust the hyperparameters of the two deep neural networks. These two deep neural networks constantly improve themselves by competing with each other during the training phase. Figure 3 shows the basic framework of a GAN.

Formula (1) shows the optimization target of traditional GANs. $G(z)$ represents the fake image generated by G which takes the random noise $z$ as input. $D(x)$ represents the discriminating result of real image $x$, and $D(G(z))$ is the discriminating result of fake image $G(z)$. The values of $D(x)$ and $D(G(z))$ are all between $[0,1]$. When the value is close
to 1, D discriminates the input image as a real image. On the contrary, if the value is close to 0, D discriminates the input image as a fake image. For the discriminator, it is hoped that D(x) is as close to 1 as possible and D(G(z)) is as close to 0 as possible. In this case, the logarithm of D(x) and (1-D(G(z))) will be close to 0, and a zero-sum game is realized. Therefore, when training D, the mathematical expectations of log(D(x)) and log(1-D(G(z))) are maximized by adjusting the hyperparameters in D. For the generator, it is hoped that D(G(z)) is as close to 1 as possible. That means the fake image generated by G is close to the real image. So different from training D, training G is to minimize the mathematical expectation of log(1-D(G(z))) by adjusting the hyperparameters in G.

The training process of GAN is iterated, and during this process, the generator and the discriminator keep to promote each other. As the training process goes in-depth, the competition between the generator and the discriminator reaches equilibrium, and the two competitors gradually become perfect. The images generated by the final generator model are perfectly realistic and difficult to distinguish.

\[ E_X \sim p_{data} [\log(D(x))] + E_Z \sim p_{z} [\log(1 - D(G(z)))] \] (1)

At present, GANs are widely adopted in image translation [20], image super-resolution [21], [22], texture synthesis [23], face synthesis [24], natural language processing [25], etc. In this paper, we used adversarial examples derived from open-source data set to train MTGAN which consists of two GANs. Then we use the trained MTGAN to generate test cases that would be misclassified with a high probability.

2.3 Motivation

Software testing work does not aim at discovering all the possible defects in the software under test, but to find defects out as many as possible. The testing work can give users the confidence to use the tested software. A suitable test case set is the foundation of successful software testing work.

We hope that the constructed test case set owns enough ability to trigger defects. The ability to trigger defects much depends on the number of test cases in the test case set. It is easy to understand that the more test cases there are, the better the defect-trigger ability is. In order to ensure the defect-trigger ability, the test case set must contain a sufficient number of test cases. So the method of constructing plenty of test cases for the deep learning system is needed.

However, since the input data of deep learning systems always has high dimensions and input space is actually huge. Let’s take a handwritten digit recognition system with a pixel size of 28*28 as an example, the value of each pixel is between [0,255], so the input space of this recognition system is 256*256*28. This is only a relatively simple deep learning system of many realistic application scenarios. It is impossible to construct a test case set that covered the entire input space. To reduce the overhead of testing work, the scale of the test case set must be limited. Therefore, in order to balance the test overhead and defect-trigger ability, testers should try their best to maximize the effectiveness of test cases. In the black-box test of traditional software, testers usually adopt the equivalence class partitioning method to construct test cases. This method is obviously not applicable for deep learning systems, because the demarcation line of different classes is always indistinct. So we need to find a new method to generate effective and sufficient test cases for deep learning systems. This is the motivation of our work.

3. Methodology

In this section, firstly we will give out some definitions used in this paper, then take an overview of MTGAN. Finally, we will introduce the details of the main components.

3.1 Definitions

In order to ensure the integrity and readability of this paper, we make some definitions used in Fig. 4 and the following paragraphs. As mentioned above, this paper takes the deep learning image classifier as the research object. So the inputs and test cases for the deep learning systems are all in the form of images.

**Definition 1 (Seed inputs).** Seed inputs refer to the images sampled from standard data sets. These images are not perturbed and used to construct original examples.

**Definition 2 (Original adversarial examples).** Original adversarial examples refer to the adversarial examples constructed using traditional methods mentioned in Sect. 2.1. We will adopt multiple algorithms to generate original adversarial examples based on seed inputs. Original examples helped MTGAN learn to perturb images.

**Definition 3 (Follow-up test cases).** Follow-up test cases refer to the perturbed images generated by MTGAN. Follow-up test cases are a beneficial extension to the test case set, and also the main contribution of our work.

**Definition 4 (Test suite).** The test suite can also be called the test case set, which refers to a set of images used to test the image classifier. In this paper, the test suite consists of original adversarial examples and Follow-up test cases. This test suite is considered to have a high
3.2 Work Steps and Overview of MTGAN

The black dotted boxes in Fig. 4 present the main steps of the extension of the test suite using MTGAN. The test suite extension process consists of three major steps at a high level: constructing the original adversarial example set, training GANs, generating follow-up test cases.

**Step 1: Constructing original adversarial example set.**

We use several traditional methods to construct original adversarial examples. The seed inputs are sampled from the standard data set and divided into different sets according to original manual labels. Each set is denoted as \( i \), and \( i \) represents the original manual label. We adopt several algorithms on seed inputs to construct original adversarial examples. These algorithms include FGSM [26], CW attack [27], JSMA [28], and so on. The original adversarial examples are also divided into different sets according to their labels. Each set is an original adversarial example set denoted as \( \theta_i \), which consists of original adversarial examples with the same original label “\( i \)”. \( \theta_i \) includes several subsets denoted as \( \Phi_{i,j} \), where “\( i \)” is the original label and “\( j \)” is the target label (\( i \neq j \)). The target label refers to the class that the original adversarial example misclassified to. The original adversarial examples set \( \theta_i \) can be added to the test suite for the deep learning system test, what’s more they will be used to train the GANs in MTGAN. Step 1 is the basis of other works.

**Step 2: Training the GANs.**

GANs are the key components of MTGAN. There are two GANs in MTGAN. In order to distinguish, they are denoted as GAN1 and GAN2 independently. GAN1 and GAN2 are trained respectively. Training data of GAN1 is randomly selected from the standard image dataset (e.g. MNIST [29], Cifar-10 [30], ...), and training GAN2 we use the original adversarial examples generated in Step 1. Details of the training process will be introduced in Sect. 3.5.

**Step 3: Generating follow-up test cases.**

After GAN1 and GAN2 are trained, we use generators in GAN1 and GAN2 to generate follow-up test cases for the deep learning classifier under test. Generator 1 (generator in GAN1) takes the noise vector as input, and Generator 2 (generator in GAN2) takes the output of Generator 1 as input. Noise vectors are transformed into follow-up test cases by the cascade-connected generators. The follow-up test case set is a good supplement to the test suite.

The blue solid box of Step 3 in Fig. 4 depicts the main generating components of MTGAN. The two series-connected generators play an important role in the framework of MTGAN. Generator 1 is adopted to generate fake images, and these fake images are not perturbed. Generator 1 is composed of multiple different sub-generators (denoted as Generator 1_\( i \), \( i \) represents the index of each category.), each sub-generator is used to generate different images from the perspective of human vision. The training images of Generator 1 are sampled and reorganized according to their original labels so that the fake images generated by Generator 1_\( i \) have the same label “\( i \)”. In other words, the fake images are labeled automatically, and it is very helpful to the testing work.

Generator 2 is trained to generate perturbation added to fake images. The images perturbed by Generator 2 can fool the image classifier in high probability, and we call these images follow-up test cases in this paper. The training data of Generator 2 is an original adversarial example set, which was constructed by different methods in Step 1. Adversarial examples generated by different algorithms make the perturbation generated by Generator 2 diversely distributed. Even though parts of the follow-up test cases might not trigger the defects in the image classifier under test, they are still a good extension of the test suite.

When the training phase is finished, the parameters of generators are fixed. The generating process from noise vectors to follow-up test cases becomes forward propagation calculation and this transformation is more efficient.

3.3 Constructing the Original Adversarial Example Set

In order to ensure diversity, we try to use different algorithms to construct the original adversarial example set. Cleverhans [31] is an open-source tool for generating image adversarial examples. This tool has integrated several classical algorithms and is convenient to use. So we use Cleverhans directly to generate original adversarial examples. Besides the algorithms integrated into Cleverhans, we also implement a white box model to generate adversarial examples. Considering that this paper focuses on the GAN-based follow-up adversarial examples generation, so we just give a brief introduction to our method which is based on gradient calculation. The details of algorithms used in Cleverhans are described in the references.

We have implemented a Convolutional Neural Network (CNN) [29] to generate original adversarial examples for
Algorithm 1: Generating original adversarial examples

**input**: seed input: x, CNN, target output: y,
**output**: adversarial example: fool

1: y ← CNN.eval(x)
2: cross ← reduce_mean(softmax_cross_entropy_with_logits(y, y))
3: x_grad ← gradients(cross, x)
4: while not (y == y):
5: fool_x ← fool_x + eps * x_grad
6: y ← CNN.eval(fool_x)
7: return fool_x

handwritten digit images. There are ten categories of handwritten digit images from "0" to "9" and the images size is 28*28 pixels. The architecture of CNN is shown in Fig. 5. It consists of two convolutional layers, two max-pooling layers, and several fully connected layers.

The specific calculation process is as shown in Algorithm 1. Gradient calculation method is used to generate the adversarial examples, and the optimization goal is to minimize the difference between classifier operation results and the target operation results. The deep learning classifier can be represented by $y = f(x)$, where x represents the image input and y is the operation result of the classifier. We use $y_\ast$ to represent the result of the target result. The optimization goal can be expressed as $\text{Minimize}(y - y_\ast)$ (Line 2). After the iterative training phase, the deep learning model has fixed weights and offset values. The input x is used as an independent variable to calculate the gradient and the result of $\frac{\partial(y - y_\ast)}{\partial(x)}$ is the correction amount of x (Line 5). The input image x will be changed step by step until it can fool the classifier. At this time the changed image x becomes the adversarial example.

3.4 Structure of GANs in MTGAN

There are two GANs in MTGAN. The structure of the GANs should be adjusted according to the input image size. Figure 6 depicts the structure of GAN1 and GAN2 for the handwritten digit images with a size of 28*28 pixels. We adopt modified convolutional neural networks as GANs’ main components and this method was first proposed by Radford et al. [32].

The discriminators of GAN1 and GAN2 have the same architecture. They all take images with a size of 28*28 pixels as input, and output one-dimensional data which indicates whether the input images to be true or false. During the training phase, Generator 1 is trained to generate digit images for the deep learning system using noise vectors, and Generator 2 is trained to generate perturbation for the fake images generated by Generator 1. The two generators’ architectures only have a slight difference in the size of the input data as depicted in Fig. 6 A. The inputs of Generator 1 are 100-dimensional noise vectors, and the inputs of Generator 2 are reshaped digit image data in the size of 784.

Different from the traditional CNN, we remove the pooling layers in the CNN models. In the discriminators, the stride convolution is used instead of the pooling layer, and in the generator, the fractional-stride convolution is used. The advantage of these structures is that the networks could learn to spatial downsample and upsample by themselves so that discriminators and generators could have the same abilities. We also remove the full connected layers to reduce the parameters and enhance operation efficiency.

3.5 Training the GANs

3.5.1 Training Data for the Generators

The selection of training data for Generator 1 and Generator 2 depends on their different functions. Generator 1 is trained to generate fake images, and Generator 2 is trained to perturb the fake images. Due to the different tasks, the
training data comes from different datasets. For Generator 1, we choose data from the standard dataset (e.g. MNIST, Cifar-10, …), and divide them into different sub-datasets according to their original labels. Then we use these sub-datasets to train the corresponding generators in Generator 1. All the generators of Generator 1 are trained independently, and during each training epoch batch size of images are fed into the generator randomly.

Generator 2 is trained to perturb the fake images generated by Generator 1. The perturbation information is extracted from original adversarial examples and with the purpose to confuse the image classifier under test. We use the original adversarial example set to train Generator 2. We deploy three methods integrated into Cleverhans (i.e. FGSM, CW attack, and JSMA) and our gradient calculation method introduced above to construct the original adversarial examples set.

3.5.2 Loss Functions

Generator 1 is designed to transform 100-dimensional noise vectors into specific images. The loss function of Generator 1 is depicted in Formula (2).

\[ L_{G_1} = \frac{1}{m} \sum_{i=1}^{m} \log(1-D_1(G_1(z_i))) \]  

(2)

\( D_1 \) and \( G_1 \) are the mathematical function forms of Discriminator 1 and Generator 1 respectively. \( z_i \) represents random noise vectors, and \( m \) is the batch size of the training process. When training Generator 1, the parameters in Discriminator 1 are fixed, and the loss \( L_{G1} \) forces the fake images generated by Generator 1 to be as real as possible.

Generator 2 is designed to perturb the fake images generated by Generator1. The total loss \( L_{G2} \) is composed of two parts as depicted in Formula 3. \( D_2 \) and \( G_2 \) are the mathematical function forms of Discriminator 2 and Generator 2 respectively. As same as \( L_{G1} \), the first part of \( L_{G2} \) forces the outputs of Generator 2 to fool Discriminator 2. In other words, by minimizing the first part of \( L_{G2} \), \([G_2(x_i)+x_i]\) will become adversarial examples to the deep learning image classifier with high probability. The second part of \( L_{G2} \) is a 2-norm of the perturbation added to the input images. It ensures that the perturbation is slight and imperceptible, but can be misclassified by the deep learning image classifiers with high probability. The constant \( \lambda \geq 0 \) is selected experimentally, and it is taken to balance the two parts of the loss function. If the selected is large, the training process will focus on reducing the perturbation, and the perturbed images can not effectively fool the classifiers. On the other hand, if we select a small \( \lambda \), the perturbation may even be obvious to humans. In this case, the generated images may be too far away from the boundary of different classes, so that the labels of generated follow-up adversarial examples may not be consistent with the images inputted to Generator 2. We set the value of \( \lambda \) to 0.01 according to experience.

\[ L_{G2} = \frac{1}{n} \sum_{i=1}^{n} \log(1-D_2(G_2(x_i)+x_i))+\lambda \frac{1}{n} \sum_{i=1}^{n} \|G_2(x_i)\|_2 \]  

(3)

4. Experiments Setup

We design several experiments to evaluate the effectiveness of MTGAN. In this section, we will propose several research questions and introduce the setup of the corresponding experiment.

4.1 Experimental Preparation and Research Questions

We use python (ver.3.5) to implement MTGAN based on the deep learning frameworks Tensorflow (ver.1.5.0) and Keras (ver.2.2.4). All the experiments are carried out on a Windows laptop with 8 cores Intel i7-9700 3.0 GHz processor, 8 GB of memory, and an NVIDIA GeForce GTX 1050 Ti GPU.

MTGAN is designed for generating test cases for deep learning image classifiers. In order to evaluate MTGAN, we propose the following research questions and tried to find the answers from several experiments.

- **RQ1**: Can MTGAN be used to generate test cases for deep learning image classifiers?
- **RQ2**: Can the test cases generated by MTGAN trigger the defects in the deep learning image classifiers under test?
- **RQ3**: Whether the images generated by MTGAN are diverse or not?
- **RQ4**: Can MTGAN be used in testing deep learning image classifiers with other datasets?

Generating image test cases is the basic function of MTGAN, and RQ1 is designed for evaluating the availability of this function. Wong et al. [33] pointed out that the effectiveness of test cases depends on the ability to uncover defects in the software. RQ2 is designed for evaluating the effectiveness of generated test cases. Testing deep learning systems is still a new research field, and there is no standard to measure the diversity of defects in the software under test. Therefore, we design RQ3 i.e. diversity of generated images to measure the quality of test cases like other researches [34], [35]. RQ4 is used to evaluate the generalization ability of MTGAN.

4.2 Dataset and Image Classifier Models

All the experiments are carried out based on the well-known handwritten digits dataset MNIST. To evaluate MTGAN we select three pre-trained deep learning image classifier models as experiment objects. One is the LeNet-5 [29], and the other two models are trained by ourselves.

- **Dataset.** MNIST contains a training set with 60000 digit images. We divide these images into ten groups according to the labels. Each group comprises more than 5000 digit images with the same label and will be used to train the corresponding generator in Generator 1. All the generators
of Generator 1 are trained independently, and during each training epoch batch size of images are fed into the generator randomly.

The original adversarial example set is used to train Generator 2. We adopt three methods integrated into Cleverhans i.e. FGSM [26], CW attack [27], and JSMA [28], and our gradient calculation method mentioned above to construct the original adversarial examples. First, we randomly selected 100 digit images from MNIST for each seed input set \( \Psi_i \) \( (i=0, 1, \ldots, 9) \). Then the four methods all take \( \Psi_i \) as input, towards each target label \( j \) \( (j \neq i, j=0, 1, \ldots, 9) \) to construct original adversarial examples. The original adversarial example set \( \theta_i \) contains 3600 \((100*9*4)\) images with the same label. Finally, the ten original adversarial example sets \( \theta_i \) \( (i=0, 1, \ldots, 9) \) are combined to train Generator 2.

Models. Three pre-trained image classifier models are prepared to evaluate the effectiveness of MTGAN. LeNet-5 is a convolutional neuron network with 6 hidden layers and was proposed by Lecun Yan et al. in 1998. M-3 and M-6 are fully-connected deep neuron networks trained by ourselves. Table 1 gives out the details of the three models.

4.3 Implementation of MTGAN

MTGAN composes of two main components: Generator 1 and Generator 2. Before we use MTGAN to generate follow-up test cases for deep learning image classifiers, we need to train the GANs depending on the application scenario. For the handwritten digit recognition, Generator 1 has ten sub-generators to train i.e. from Generator 1\_0 to Generator 1\_9. The generators are trained in pairs with corresponding discriminators for 50 epochs. The training losses of GAN 1\_0 are depicted on the left of Fig. 7. The training losses of other sub-generators are similar to GAN 1\_0, so they are not depicted here. After GAN 1 is trained, we use the original adversarial example set to train GAN 2 for 200 epochs. The training loss of GAN 2 is depicted in the right of Fig. 7. By observing the training loss curve, we can conclude that the models of GAN 1 and GAN 2 are all convergent.

5. Experimental Results

In this section, we evaluate the effectiveness of MTGAN with the three typical deep learning image classifiers on the dataset MNIST and provide further insight on the ability of follow-up test case generation.

5.1 RQ1: Evaluation on the Ability of Test Cases Generation

Each generator of GAN 1 output the digits with the same label even after the perturbation of GAN 2. In order to evaluate this performance, we randomly select a scale of 1000 100-dimension noise vectors for each category from 0 to 9. These noise vectors are fed to MTGAN, then 10000 \((1000*10)\) perturbed digit images are generated. Part of the digit images are exhibited in Fig. 8.

Human cognition is the benchmark of deep learning image classifier, in order to evaluate the quality of the generated test cases, we check all the 10000 digit images manually. The misclassified digit images are highlighted by a red circle as shown in Fig. 8 and we can find that these images are distorted or incomplete. In order to analyze the cause of this problem, we also check the digit images before perturbation manually and make a comparison.

Comparing the ten pairs of digit images we can find that the perturbation only adds specific noise onto the fake digit images, and has little influence on semantics. The comparison indicates that GAN 1 plays a more important role than GAN 2 on the misclassifications. We make a statistic that during all the 10000 digit images generated by MTGAN, 33 images can not be classified correctly manually. Obviously, the probability of this is very small. In addition, during the manual identification process, we follow stricter standards. If we lower the standard, almost all the...
Fig. 9  Comparison of digit images before and after perturbation. The five digits in the top two columns are classified manually, and the five digits in the bottom two columns are misclassified.

Table 2  Misclassified digit images by classifier models

| Category | Number of digit images | Number of Misclassified digit images |
|----------|------------------------|--------------------------------------|
|          |                        | LeNet-5 | M-3 | M-6 |
| 0        | 999                    | 81      | 152 | 67  |
| 1        | 997                    | 28      | 99  | 64  |
| 2        | 998                    | 50      | 124 | 203 |
| 3        | 998                    | 39      | 144 | 46  |
| 4        | 991                    | 85      | 83  | 54  |
| 5        | 997                    | 42      | 75  | 181 |
| 6        | 997                    | 32      | 82  | 58  |
| 7        | 998                    | 14      | 64  | 45  |
| 8        | 994                    | 199     | 178 | 126 |
| 9        | 998                    | 84      | 93  | 59  |
| SUM      | 9967                   | 654(6.56%) | 1094(10.97%) | 903(9.06%) |

10000 images could be classified correctly manually. Therefore, in practical application, MTGAN can label the generated test cases automatically.

To address RQ1: MTGAN has a good ability to generate test cases for deep learning image classifiers. The labels of these test cases are all tagged automatically, and this will effectively alleviate the test oracle problem[15].

5.2 RQ2: Evaluation on the Ability to Trigger Defects

We hope that the generated test cases can trigger as many defects as possible. In order to measure the ability of trigger defects, we feed the 9967 digit images which can be classified correctly by the human to the pre-trained classifier models. Table 2 gives out the summary of prediction results by the three classifier models. The right three columns present the number of each category misclassified digit images. The total misclassification rates of the three models are 6.56%, 10.97%, and 9.06%. From Table 1 we can see that the three classifiers are all well trained, and the lowest test accuracy is 98.6%. However, when we use the test cases generated by MTGAN to test the classifiers, the test accuracy is obviously reduced. These misclassified digit images are regarded as good test cases to the classifiers and should be added to the test suite.

From Table 2 we can see that most of the digit images are still classified correctly by the classifiers under test. What shall we do with the rest of the digit images? Are they useless for testing? For further analysis, we compare the Euclidean distance (Ed) between the predictions of standard digit images and follow-up digit images generated by MTGAN with respect to their manual labels. We use X to represent a digit image, and use $\text{Softmax}(X) = (x_1, x_2, \ldots, x_{10})$ to represent the output of SoftMax-layer for X. $L(X) = (l_1, l_2, \ldots, l_{10})$ is the one-hot code of manual label. The Ed of X is defined as follows. Intuitively, the higher the value of Ed($\text{Softmax}(X)$) is, the more likely X could trigger defects of the classifiers.

$$Ed(\text{Softmax}(X)) = \sqrt{\sum_{i=1}^{10} (x_i - l_i)^2}$$

We select 1000 digit images from every category of MNIST training dataset, and similarly, we also select 1000 digit images generated by MTGAN which are classified correctly by three classifiers. Figure 10 depicts the comparison of Euclidean distance between standard digit images and correctly classified follow-up digit images with respect to a one-hot code of manual labels.

In order to further evaluate MTGAN’s ability to trigger defects, we compare it with three classic deep learning testing tools Deepxplore[36], DLFuzz [35], and DeepHunter[37] through experiments. The principles of these three tools are all based on test coverage criteria, so we select two different test coverage criteria for comparison. One criterion is Neuron Coverage (NC)[36] which represents the ratio of activated neurons. The other criterion
is K-multisection Neuron Coverage (KMNC) \cite{38} which calculates the ratio of covered k-multisections of neurons.

Deepxplore’s generation of test cases strongly relies on Neuron Coverage, as a result, Deepxplore is only compared under the test coverage criteria of NC. DLFuzz and DeepHunter only use the calculation result of test coverage to guide the mutations of test cases, so they are compared under both of the two coverage criteria. The three test coverage based tools all take the test coverage as the test termination condition, different from this, the test process of MTGAN needs the tester’s manual termination. In the comparative experiments, we set 500 test cases as the termination condition. Then we compare the numbers of test cases that trigger the defects of three models. The experimental results are shown in Table 3.

Obvious, all three test-coverage-based tools have achieved higher test coverage than MTGAN. Although the performance on test coverage is not as good as other tools, MTGAN still obtains 100% neuron coverage for all the deep learning models under test. Deepxplore takes triggering defects and test coverage as the optimization objectives, and its performance in triggering defects is similar to MTGAN. However, Deepxplore can only conduct test work under the NC test coverage criteria. DLFuzz and DeepHunter mutate the randomly selected test cases under the guidance of test coverage criterion, and the generated test cases trigger fewer defects than MTGAN and Deepxplore.

To address RQ2: Different from other adversarial example methods, only part of images generated by MTGAN can trigger defects of image classifiers under test. However, the probability of successfully triggering defects is much higher than standard images and test cases generated by other deep learning test tools.

5.3 RQ3: Evaluation on the Diversity of Images Generated by MTGAN

We evaluate the diversity of digit images generated by MTGAN from the perspective of SoftMax-layer output for a set of samples. A new metric is proposed which uses Mean Squared Error (MSE) of $Ed(\text{SoftMax}(X))$ comparing with digit adversarial examples generated by other methods. This metric for measuring diversity is defined as follows. Ave represents the average value of $Ed(\text{Softmax}(X))$.

$$
\text{Metric}_{\text{MSE}} = \frac{1}{n} \sum_{i=1}^{n} (Ed(\text{Softmax}(X_i)) - \text{Ave})^2
$$

We compute the value of $\text{Metric}_{\text{MSE}}(X_{\text{MTGAN}})$ and $\text{Metric}_{\text{MSE}}(X_{\text{other-method}})$ and comparing the results. If $\text{Metric}_{\text{MSE}}(X_{\text{MTGAN}}) > \text{Metric}_{\text{MSE}}(X_{\text{other-method}})$ means the diversity of digit images generated by MTGAN is higher than other methods. A comparison between MTGAN and the other three methods i.e. FGSM, CW, JSMA for generating adversarial examples is conducted. In addition, we compute the $\text{Metric}_{\text{MSE}}$ of standard data sampled from MNIST as a reference. We randomly select 1000 digit images from 5 data sets and feed them to the three image classifiers. The results of $\text{Metric}_{\text{MSE}}$ are presented in Table 4.

Images selected from MNIST are almost classified correctly, and images from three adversarial example sets are misclassified. The selected images generated by MTGAN are sampled from all the 10000 images, including classified correctly and misclassified. The range of $Ed(\text{Softmax}(X))$ is larger than the result shown in Fig. 9. From Table 3 we can learn that the $\text{Metric}_{\text{MSE}}$ of MTGAN is obviously higher than images selected from other data sets on all the three classifier models.

To address RQ3: MTGAN shows a more prominent performance in generating test cases with high diversity. This feature enables the test suite to cover input space more adequately.

5.4 RQ4: Evaluation on the Generalization Ability of MTGAN

Although MNIST is a classic image classification dataset, MNIST is simple and rudimentary. To evaluate the generalization ability of MTGAN, we have designed experiments on deep learning classifiers based on Cifar-10 \cite{30}. Cifar-10 comprises of 50000 training data and 10000 test data in ten classes. Each image of Cifar-10 has three channels, and each channel is 32x32 in pixel size. The images depicted in the left column of Fig. 11 are sampled from Cifar-10. Obviously, the classification take of Cifar-10 is complex than MNIST. In the evaluation experiments, we take

| Models | MTGAN | Deepxplore | DLFuzz | DeepHunter |
|--------|--------|------------|--------|------------|
| LeNet-5| 100%/78% | 100%/--  | 100%/98% | 100%/97% |
| M-3   | 100%/62% | 100%/--  | 100%/94% | 100%/98% |
| M-6   | 100%/85% | 100%/--  | 100%/95% | 100%/98% |
| SUM   | 162/147 | 157/--   | 67/50  | 98/66      |

| Models | MTGAN | Deepxplore | DLFuzz | DeepHunter |
|--------|--------|------------|--------|------------|
| LeNet-5| 37/42  | 45/--      | 16/16  | 25/24      |
| M-3   | 71/67  | 75/--      | 23/15  | 34/17      |
| M-6   | 54/38  | 37/--      | 28/19  | 39/25      |

| Models | MTGAN | Deepxplore | DLFuzz | DeepHunter |
|--------|--------|------------|--------|------------|
| LeNet-5| 0.00089 | 0.00102    | 0.00092 |
| M-3   | 0.09891 | 0.10031    | 0.08453 |
| M-6   | 0.00467 | 0.00564    | 0.00401 |
| C&W   | 0.00681 | 0.00787    | 0.06432 |
| JSMA  | 0.00549 | 0.00746    | 0.00597 |

| Models | MTGAN | Deepxplore | DLFuzz | DeepHunter |
|--------|--------|------------|--------|------------|
| LeNet-5| 0.00089 | 0.00102    | 0.00092 |
| M-3   | 0.09891 | 0.10031    | 0.08453 |
| M-6   | 0.00467 | 0.00564    | 0.00401 |
| C&W   | 0.00681 | 0.00787    | 0.06432 |
| JSMA  | 0.00549 | 0.00746    | 0.00597 |

Table 3 Comparison with other tools

Table 4 Comparison of MSE value between different methods
three well-known classification models (i.e. ResNet-20 [39], VGG-16 [40], and VGG-19 [40]) as the deep learning systems under test. All three models are well trained, and the test accuracy is higher than 90%.

Like the experiments on MNIST, we repeat the workflow for Cifar-10. We use original adversarial examples to train MTGAN, after that the trained models are used to generate 1000 follow-up test cases for each class. The original adversarial examples and follow-up test cases are depicted in the middle and right columns in Fig. 11. Considering that in the above MNIST experiments, the probability of successfully generating automatically-labeled follow-up test cases is nearly 100%. In addition, we randomly sample 10% of the generated follow-up test cases to check the rate of the successful generation. The labels of sampled images are consistent with manual identification. As a result, we use the 10000 follow-up test cases to test the three deep learning image classifiers. The result is presented in Table 5.

Table 5 shows that among all the follow-up test cases, 1834 images have triggered defects in ResNet-20, 1764 images have triggered defects in VGG-16, and 1647 images have triggered defects in VGG-19. It proves that MTGAN has good generalization ability on Cifar-10. Comparing with MNIST, MTGAN has generated more images which could trigger defects in the deep learning classifiers. Each sample of MNIST has 784-dimensional (28x28) features, and each sample of Cifar-10 has 3072-dimensional (32x32x3) features. For higher-dimensional images, it is more difficult to analyze the distribution of all the features, and the classification boundary of the corresponding deep learning classifier is more complex. Therefore the unstable input space of the higher-dimensional model is larger than the lower-dimensional model. We believe that larger unstable input space is the reason why MTGAN can generate more effective follow-up test cases.

To address RQ4: MTGAN has good generalization ability on other datasets and can be used to test different deep learning images classifiers.

6. Related Work

The major contribution of this paper is proposing a test case generation framework for testing deep learning image classifiers. Before us, many researchers have proposed a lot of different methods for generating test cases. Therefore, in the related work section, we will introduce their research work first. In addition to verifying the normal function, testers also have to design test cases to uncover defects of the software under test. As a result, testers prefer test cases that would cause the software to make mistakes. The adversarial example attack is to attack the deep neuron networks by constructing adversarial examples. Although this attack may be malicious, adversarial examples are regarded as test cases with a high ability to reveal defects of deep learning systems under test. This paper uses adversarial example attack technologies to construct an original adversarial example set, so several different research on adversarial example attacks will be introduced as related work too.

6.1 Generating Test Cases for Deep Learning

Tian et al. [41] proposed a test framework named DeepTest generate test cases for deep learning systems based on metamorphic relations [42]. Wicker et al. [34] tried to systematically change the dimension of inputs and enumerate the input space to construct a test case set. DLFuzz [35] adopted a fuzzy test to guide test case generation for deep learning tests. TensorFuzz [43] proposed a coverage guided fuzzy test technology for deep learning test, which generated new test cases by randomly changing the seed test cases. DeepHunter [37] was also a coverage guided fuzzy test framework. Different from TensorFuzz, DeepHunter was a grey box method and based on several coverage criteria proposed by Ma Lei et al. [38].

6.2 Adversarial Example Attack

Since the concept of adversarial example was proposed in 2014, many different methods of constructing adversarial
examples have been proposed. Goodfellow et al. [26] proposed a fast gradient sign method (FGSM), which could effectively calculate adversarial perturbation added to the input example. Kurakin et al. [44] proposed a multi-step attack method based on FGSM. But the multi-step method was weaker than the single-step method on the characteristic of the transfer. After that Kurakin et al. [45] proposed Least-Likely-Class Iterative Method in another paper. Papernot et al. [28] proposed a Jacobian-based Saliency Map Attack method (JSMA). Su et al. [46] proposed a more extreme attack method which only perturbed one pixel to attack. Carlini et al. [27] proposed three attack methods extreme attack method which only perturbed one pixel to was weaker than the single-step method on the character-attack method based on FGSM. But the multi-step method was more robust. Sarkar et al. [49] was to construct an anti perturbation for a single target. Baluja et al. [50] proposed Adversarial Training Nets (ATN) which took the joint optimization objective. The generated adversarial examples could be used to attack the deep neuron networks.

7. Conclusion and Future Work

Deep learning has achieved impressive progress, however, test techniques for deep learning still stay at the early stage. Among all the challenges of testing deep learning images classifiers, this paper focuses on the test case generation and proposes a framework MTGAN to generate test cases. MTGAN consists of two generative adversarial networks, and after the training stage, the two cascaded generators can transfer random noise vectors to image test cases. We also evaluate the quality of the generated test cases based on real application scenarios i.e. handwritten digit recognition. Through the evaluation, we find that the images generated by MTGAN have a good performance on diversity and are good at triggering defects of image classifiers under test. In addition, the generation efficiency is higher than the traditional method.

Limited by the computation resource, we only evaluate MTGAN by a simple application scenario. We will further expand the function of MTGAN for more scenarios. Moreover, we will integrate the deep learning system test coverage criterion into MTGAN, and use the coverage criterion-based Greedy algorithm to guide the generated test case selection.

Acknowledgments

This work is supported by the National Key R&D Program of China (No. 2018YFB1403400).

References

[1] L. Chang, C. Yu, L. Yan, G. Chen, V. Vokkarane, and Y. Ma, “Deep-Food: Deep learning-based food image recognition for computer-aided dietary assessment,” 2016.
[2] T.N. Sainath, R.J. Weiss, K.W. Wilson, L. Bo, and C. Kim, “Multi-channel signal processing with deep neural networks for automatic speech recognition,” IEEE/ACM Trans. Audio, Speech, Language Process., vol.25, pp.965–979, 2017.
[3] A. Severyn and A. Moschitti, “Learning to Rank Short Text Pairs with Convolutional Deep Neural Networks,” International Acm Sigir Conference, 2015.
[4] S.H.C. Zong, E. Liu, Y. Yao, and S. Tang, “Nowhere to Hide Methodology: Application of Clustering Fault Diagnosis in the Nuclear Power Industry,” IEEE Access, vol.7, 2019.
[5] T. Young, D. Hazarika, S. Poria, and E. Cambria, “Recent Trends in Deep Learning Based Natural Language Processing [Review Article],” IEEE Comput. Intell. Mag., vol.13, no.3, pp.55–75, 2018.
[6] G. Kahn, A. Villafor, B. Ding, P. Abbeel, S. Levine, G. Kahn, et al., “Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation,” 2018.
[7] S. Gu, E. Holly, T. Lillicrap, and S. Levine, “Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates,” IEEE International Conference on Robotics Automation, 2017.
[8] V. Jindal, “Integrating mobile and cloud for PPG signal selection to monitor heart rate during intensive physical exercise,” International Conference on Mobile Software Engineering Systems, 2016.
[9] C. Angermueller, T. Parnamaa, L. Part, and O. Steglo, “Deep learning for computational biology,” Molecular Systems Biology, vol.12, p.878, 2016.
[10] A.K.I.S.G. Hinton, “ImageNet classification with deep convolutional neural networks,” Proc. NIPS, vol.1, pp.1097–1105, 2012.
[11] H. Guo, J. Wang, Y. Gao, J. Li, and H. Lu, “Multi-View 3D Object Retrieval with Deep Embedding Network,” IEEE Trans. Image Process., vol.25, no.12, pp.5526–5537, 2016.
[12] R. Morcel, H. Akkary, H. Hajj, M. Saghir, A. Keshavamurthy, R. Khamma, and H. Artail, “Minimalist Design for Accelerating Convolutional Neural Networks for Low-End FPGA Platforms,” IEEE International Symposium on Field-programmable Custom Computing Machines, 2017.
[13] J. Zhu, Y. Song, D. Jiang, and H. Song, “A new deep-Q-learning-based transmission scheduling mechanism for the cognitive internet of things,” IEEE Internet Things J.
[14] M. Mohammadi, A. Al-Fuqaha, M. Guizani, and J.S. Oh, “Semi-supervised deep reinforcement learning in support of IoT and smart city services,” IEEE Internet Things J., vol.PP, pp.1–1, 2017.
[15] S. Nakajima and H.N. Bui, “Dataset Coverage for Testing Machine Learning Computer Programs,” 2016 23rd Asia-Pacific Software Engineering Conference, A. Potanin, G.C. Murphy, S. Reeves, and J. Dietrich, Eds., ed New York: Ieee, 2016, pp.297–304.
[16] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, “DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems,” pp.132–142, 2018.
[17] L.E.-h.H. Song, H. Zhan-wei, T. Shi-qi, and Z. Suo-juan, “Chal-}
[18] L.E.-h.H. Song, H. Zhan-wei, T. Shi-qi, and Z. Suo-juan, “Chal-

References
ERHU LIU was born in Xuzhou, Jiangsu Province, China in 1986. He received his Bachelor's degree and a Master’s degree from Southeast University in 2008 and 2013 respectively. He is currently pursuing Ph.D. degree in software engineering at Army Engineering University of PLA. His research interests are in the areas of AI testing and metamorphic testing.

SONG HUANG was born in Huaian, Anhui Province, China in 1970. He received Ph.D degree from PLA university of Science and Technology. He is a member of CCF and ACM. He is currently a professor of software engineering at Software Testing and Evaluation Center at Army Engineering University of PLA. He is currently a professor of software engineering at Software Testing and Evaluation Center at Army Engineering University of PLA. He received his Bachelo's degree from PLA university of Science and Technology in 1992, and a Ph.D degree from PLA university of Science and Technology in 1998. His research interests are in the areas of software testing, quality assurance, data mining and empirical software engineering. Contact him at hs0317@163.com.

CHENG ZONG was born in Yangzhou, Jiangsu Province, China in 1986. He received the B.S degree in electric engineering from Southeast University, Nanjing, China in 2008, and M.S degrees in control engineering from Southeast University, Nanjing, China in 2014. He is currently pursuing Ph.D. degree in software engineering at Army Engineering University of PLA. His research of interests are in the area of data mining, industrial control and failure analysis. He is currently working in Jiangsu Nuclear Power Corporation.

CHANGYOU ZHENG was born in China, 1986. He received the Ph. D degree in military in formation from PLA University of Science and Technology in 2013. He is currently a teacher at Army Engineering University of PLA. His research interests are in the area of software testing, blockchains.
Yongming Yao was born in Yangzhou, Jiangsu Province, China in 1987. He received the B.S. degree in communication engineering from Nanjing University of Posts and Telecommunications in 2010 and the M.S. degree in computer system architecture from Xi’an University of Posts and Telecommunications in 2013. He is currently pursuing the Ph.D. degree in software engineering at Army Engineering University of PLA. Since 2013, he has been an Assistant professor with the software engineering department, Tongda College, Nanjing University of Posts and Telecommunications. His research interests are in the area of crowdsourced software testing and android permissions detection.

Jing Zhu received the B.E. degree from Jiangsu University of Science and Technology in 2006, and the M.E. degree from Navy Command College in 2011. He is a Ph.D candidate in Software Engineering at Army Engineering University of PLA. He joined the Navy Command College as a lecturer in 2013, and his research interests include software engineering, pattern recognition and intelligent systems.

Shiqi Tang was born in Hengyang, Hunan Province, China in 1990. He received the B.S. degree in civil engineering from University of South China in 2012 and the M.S. degree in Software Engineering from Guilin University Of Electronic Technology in 2017. He is currently pursuing the Ph.D. degree in software engineering at Army Engineering University of PLA. His research interests are in the area of the fault localization, defect detection and machine learning.

Yanqiu Wang received the B.E. degree in 1982. She had been a lecturer in the institute before retiring, teaching Information Fusion, C++, Machine Learning, and she also has been a private teacher who teaches relevant courses in another private institution. She has now been rehired to continue her research, which includes fuzzy reasoning, Asian language studies, and so on.