Metamorphic Testing for Plant Identification Mobile Applications Based on Test Contexts

Hongjing Guo[1], Chuanqi Tao[1,2,3(✉)], and Zhiqiu Huang[1,2]

[1] College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211100, China
{guohongjing,taochuanqi,zqhuang}@nuaa.edu.cn
[2] Ministry Key Laboratory for Safety-Critical Software Development and Verification, Nanjing University of Aeronautics and Astronautics, Nanjing 211100, China
[3] National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China

Abstract. With the fast growth of artificial intelligence and big data technologies, AI-based mobile apps are widely used in people's daily life. However, the quality problem of apps is becoming more and more prominent. Many AI-based mobile apps often demonstrate inconsistent behaviors for the same input data when context conditions are changed. Nevertheless, existing work seldom focuses on performing testing and quality validation for AI-based mobile apps under different context conditions. To automatically test AI-based plant identification mobile apps, this paper introduces TestPlantID, a novel metamorphic testing approach based on test contexts. First, TestPlantID constructs seven test contexts for mimicking contextual factors of plant identification usage scenarios. Next, TestPlantID defines test-context-based metamorphic relations for performing metamorphic testing to detect inconsistent behaviors. Then, TestPlantID generates follow-up images with various test contexts for testing by applying image transformations and photographing real-world plants. Moreover, a case study on three plant identification mobile apps shows that TestPlantID could reveal more than five thousand inconsistent behaviors, and differentiate the capability of detecting inconsistent behaviors with different test contexts.

Keywords: Test context · AI-based software · Metamorphic testing

1 Introduction

With the development of the mobile Internet, cloud computing, big data technologies as well as intelligent algorithms, there is a significant increase in

Supported by National Key R&D Program of China (2018YFB1003900), National Natural Science Foundation of China (61602267, 61402229), Open Fund of the State Key Laboratory for Novel Software Technology (KFKT2018B19), Fundamental Research Funds for the Central Universities (NO. NS2019058), and China Postdoctoral Science Foundation Funded Project (No. 2019M651825).

J. Liu et al. (Eds.): MobiCASE 2020, LNICST 341, pp. 209–223, 2020.
https://doi.org/10.1007/978-3-030-64214-3_15

mobile apps. Mobile apps are widely used in people's daily life including business, education, biomedical industry, social media, transportation, etc. Many software products based on artificial intelligence (AI) techniques emerge in the mobile app store, such as object recognition apps, navigation apps, and translation apps. AI-based apps are developed based on advanced Machine Learning (ML) algorithms through large-scale training data, which undoubtedly brings many new difficulties to the quality assurance and verification [1], such as defect analysis, defect prediction, and testing. It also puts forward a large market demand and research demand for quality assurance and verification.

Different from traditional software, AI-based software learns decision its logic from large-scale training data [2]. They are characterized by uncertainty and probabilities, dependence on big data, and constant self-learning from past behaviors [3]. Moreover, AI-based apps usually involve context issues, such as scenario, location [4], time, and stakeholders. Many AI functions of apps generate inconsistent outputs for the same input data when context conditions are changed [5]. For example, Fig. 1 shows two images of the same plant taken by a tester from different angles. For Fig. 1a, it was taken at an approximately 90-degree angle from the plane of the smartphone to the plant. For Fig. 1b, it was taken at an approximately 45-degree angle from the plane of the smartphone to the plant. A plant identification app named PlantSnap, identified Fig. 1a as a Lactuca virosa. However, it identified Fig. 1b as a Hydrangea macrophylla. Interestingly, different photographing angles of the same plant can even result in quite inconsistent behaviors. Therefore, the evaluation of the reliability and robustness of AI-based apps under different context conditions becomes an important task.

(a)  (b) 

Fig. 1. Two images with different photographing angles of a plant

Besides, testing AI-based software has the oracle problem [6]. Currently, metamorphic testing (MT) has been successfully used in alleviating the test oracle problem [7]. Central to MT is a set of metamorphic relations (MRs), which depict the relationships between the results of multiple inputs and their expected output [7]. Specifically, for a plant identification app, the MRs are defined such that no matter how the context conditions are changed, the identification results are supposed to be consistent with the plant images under the original context conditions. Nevertheless, as a typical AI-based mobile app, there is a lack of

approaches focusing on leveraging MRs based on different context conditions to test AI-based plant identification apps.

To address those issues, we propose a novel testing approach, namely TestPlantID, to automatically test AI-based plant identification mobile apps. First, TestPlantID defines a set of test contexts to systematically depict the contextual factors of plant identification app usage scenarios. Next, MRs based on predefined test contexts are defined for plant identification apps. These test-context-based MRs are leveraged for revealing inconsistent behaviors of apps under test. Then, by applying image transformations and photographing real-world plants, TestPlantID generates images with different test contexts for performing MT. Finally, TestPlantID is leveraged to test three real-world plant identification mobile apps and successfully detect inconsistent results.

The key contributions of this paper are as follows:

- We present a systematic metamorphic testing approach to automatically test AI-based plant identification mobile apps under different test contexts. We leverage MRs based on test contexts for detecting inconsistent behaviors of apps.
- We perform a case study to indicate the feasibility and effectiveness of the proposed metamorphic testing approach on three real-world plant identification mobile apps. The results show that TestPlantID found more than five thousand inconsistent behaviors across three apps. Besides, we also investigate what extent do different test-context-based MRs reveal inconsistent behaviors.

The remainder of this paper is structured as follows: Sect. 2 gives a general summary of related work. In Sect. 3, the approach of this current study is elaborated. In Sect. 4, we present the case study and the study's validity. Section 5 shows the conclusion and future work.

2 Related Work

2.1 Metamorphic Testing

Metamorphic testing (MT) is a property-based software testing technique, which has been leveraged in many domains for addressing the test oracle problem and test case generation problem [7]. Since Chen et al. [8] introduced MT in 1998, MT has been an attractive research topic in software engineering. It has been used on debugging [9,10], fault localization [11], fault tolerance [12], and program repair [13]. Recently, MT has been proved to be an effective AI-based software testing approach. It has successfully helped detect a large number of real-life faults. Tian et al. [14] proposed a testing tool for Deep Neutral Network (DNN)-based autonomous driving systems called DeepTest. DeepTest has automatically detected potentially fatal behaviors in the system with MT. They simulated the actual driving scenes by applying a variety of image transformations and effect filters for transforming the original driving scene images. Zhang et al.

[15] proposed the first GAN-based MT approach to delivering driving scene-based test generation with various weather conditions for detecting inconsistent behaviors of autonomous driving systems. Zhou et al. [16] found fatal errors in the LiDAR Obstacle Perception system of the Baidu Apollo autonomous driving system by employing MT. In addition to the automatic driving system, MT has also been applied to ML classifiers [17,18], Google map App [19], search engines [20], facial age recognition software [3], and object detection system [21], etc., all of which have achieved good results.

The key element of MT is a set of effective MRs, which are necessary features of the target function or algorithm in relation to multiple inputs and their expected output. Some studies presented various approaches to systemically generate MRs [22–24]. Even though many MRs have been identified for various application domains [7], there is a lack of approaches identifying MRs from the test context perspective. In this paper, TestPlantID leverages an MT approach for plant identification apps, where MRs are defined based on predefined test contexts.

2.2 Testing and Verification of AI-Based Software Systems

Traditional software is implemented by developers with carefully designed specifications and programming logic. It is tested with test cases which are designed based on specific coverage criteria. For traditional software applications, testing is efficient and effective. However, the current practice of testing AI applications lags far behind the maturity of testing traditional software applications [25]. More and more work focused on testing ML-based software, including proposing new testing evaluation criteria, generation of test cases, etc. Pei et al. proposed DeepXplore [26], the first white-box testing framework for real-world Deep Neural Network (DNN) systems. They introduced neuron coverage as s systematic metric for measuring how much of the internal logic of a DNN has been tested. Ma et al. [27] extended the concept of neuron coverage. They proposed a set of multi-granularity test criteria called DeepGauge for the DNN system. Sun et al. [28] proposed four test coverage criteria that are tailored to the distinct features of DNN inspired by the MC/DC coverage criteria. Besides, testing techniques for traditional software have been recently applied for AI-based software systems, including fuzz testing [29–31], mutation testing [32,33], metamorphic testing [34,35], and also symbolic execution [36–38].

As a typical AI-based software, the image recognition system detects and recognizes objects in images by referring to a database of images. However, testing image recognition grave great challenges. Zhu et al. [25] proposed a new method called Datamorphic Testing, which consists of three components: a set of seed test cases, a set of datamorphisms for transforming test cases, and a set of metamorphisms for checking test results. They validated the proposed approach on four real industrial face recognition systems. Tao et al. [3] performed a case study on a realistic facial age recognition provided by Alibaba Company using MT. To the best of our knowledge, TestPlantID is the first work that focuses on testing AI-based plant identification mobile apps.

3 Approach

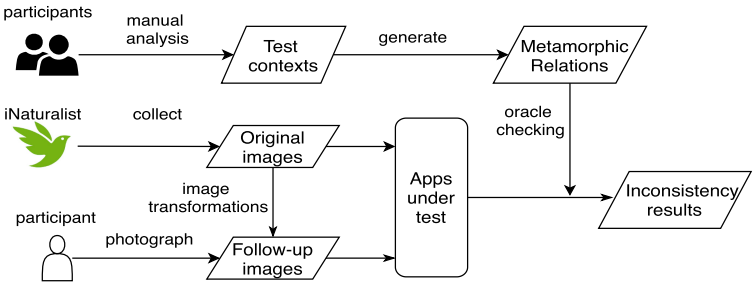

Fig. 2. The framework of TestPlantID

The details of TestPlantID are presented in this section. Figure 2 shows the framework of TestPlantID.

3.1 Test Context Construction

As a typical intelligent software artifact, AI-based plant identification mobile apps allow users to identify diverse plants simply by photographing them or uploading images with users' smartphones. It provides several possible identification results for the uploaded images instantly. This kind of AI-based software is developed based on advanced ML algorithms through large-scale plant images training. As is mentioned before, AI-based mobile apps usually involve contextual factors, such as scenario, location, time, and stakeholders. Many mobile apps with AI functions generated inconsistent behaviors for the same test input when context conditions are changed. Thereby, the relevant real-world contexts such as image rotation, translation, lighting, or the distance between plant and smartphone could be leveraged to test the robustness of AI-based plant identification apps.

It is worth noting that, we only consider the contextual factors relevant from usage scenarios of plant identification apps. In this paper, a test context refers to a major factor or characteristic of an environmental condition when testing AI-based apps. To construct a set of test contexts for the plant identification app, two participants are involved in this process. Both participants are postgraduate students majoring in software engineering and have more than 2 years of experience in conducting research on testing AI software. Inspired by the MRs proposed by the literature [14,39], each participant defines several real-world contexts for plant identification. If they select the same contextual factors, the context condition is considered as a test context. If there are disagreements, they discuss with each other to determine a final judgment. Finally, seven test contexts are determined for testing plant identification apps. Table 1 illustrates test contexts and definitions of plant identification apps.

Table 1. Test contexts and definitions of plant identification apps.

Test context	Definition
Lighting	Change the lighting condition when photographing the plant
Angle	Change the angle at which photograph the plant
Distance	Change the distance between the smartphone and the plant
Background	Change the background scenario of the plant
Position	Change the position of the plant in the image when photographing it
Rotation	The original image is rotated by a specific degree
Image clarity	Change the clarity of the plant image

3.2 Metamorphic Testing for Plant Identification Apps

One of the major challenges in testing AI-based plant identification apps lies in the lack of test oracle, which is also known as "oracle problem". To avoid this issue, TestPlantID adopts MT to test plant identification apps under different test contexts.

The key insight of MT is that even though it is hard to specify the correct behavior of the AI-based plant identification, we can define the relationships between the results of the original image and the corresponding follow-up image. TestPlantID leverages test-context-based MRs for detecting inconsistent behaviors of apps. For example, for the same plant, the output identification results should not change under different angle context conditions, such as 90-degree and 45-degree.

Formally, for an AI-based plant identification app PlantID, given an original image x, X is the database of original images to be identified. TestPlantID defines test context transformations T that simply change the context conditions of plant identification usage scenarios. Let $\tau(x)$ be a follow-up image which is generated by applying a test context transformation t on x. $PlantID(x)$ is the top-k identification results of image x. $PlantID(\tau(x))$ is the top-k identification results of the follow-up image $\tau(x)$. Then, MRs based on test contexts can be defined as follows:

$$\forall x \in X, \forall \tau \in T, PlantID(x) = PlantID(\tau(x)) \tag{1}$$

In this case, test context transformations T can simply change context conditions without impacting the identification results for each plant. One MR is defined that $PlantID(x)$ and $PlantID(\tau(x))$ should be identical when test context t is changed. If $PlantID(x)$ and $PlantID(\tau(x))$ are of significant difference, we can conclude that the plant identification app has wrongly behaved under a specific test context.

3.3 Follow-Up Image Generation

To perform MT on plant identification apps, we need to generate follow-up plant images under different test contexts. The goal of this part is to generate follow-up images under different test context conditions.

In recent studies, DeepTest performs simple image transformations and effect filters on original images to mimic real-world road scenes [14]. In this paper, five different types of simple image transformations (changing brightness, blurring, translation, cropping, rotation) are leveraged to automatically generate follow-up images, in order to implement predefined test-context-based MRs. Changing brightness is a linear transformation, which is performed by adding or subtracting a constant parameter β to each pixel's current value [14]. Cropping transformation is used to select an area of specified size on the original image. Translation and rotation are affine transformations [40], which are the linear mapping between two images that preserve points, straight lines, and planes. It is worth noting that we use crop transformation to mimic the different distances between the smartphone and the plant. By applying blurring effects on original images, the condition of poor image clarity due to camera lens distortions or subjective factors of the photographer can be implemented. Translation transformation is leveraged to simulate the different positions of the plant in the image.

We use OpenCV [41] to implement the brightness, cropping, translation, rotation, and blur image transformations. Each transformation has six parameters. The transformations and corresponding parameters are shown in Table 2.

Table 2. Image transformations and parameters for generating follow-up images.

MR	Image transformation	Parameters	Parameter ranges
MR-lighting	Brightness	β	(−60, 60) step 20
MR-distance	Cropping	$(y \cdot \frac{n}{32} : y \cdot \frac{32-n}{32}, x \cdot \frac{n}{32} : x \cdot \frac{32-n}{32})$	n from 1 to 6
MR-position	Translation	(tx, ty)	(60, 60) to (110, 110) step (10, 10)
MR-rotation	Rotation	q (degree)	(30, 80) step 10
MR-image clarity	Blur averaging	Kernel size	5*5, 6*6
	Blur Gaussian	Kernel size	7*7
	Blur Median	Aperture linear size	3,5
	Blur Bilateral	Diameter, sigmaColor, sigmaSpace	9, 75, 75

For MR-background, Python library removebg [42] is leveraged for removing the background of the original image and keep the plant object. Then, we photographed six different images, which are used as new background images. The plant object is inserted into these six background images. Figure 3 illustrates an

Fig. 3. Two images with different background of a plant

original plant image and the corresponding follow-up image after transforming the background.

For MR-angle, one participant was involved in photographing various plants from seven different angles. We use the degree of the angle from the smartphone plane to the plant object to define the seven different angles. The seven different angles of a plant are defined as follows: 45-degree, 75-degree, 180-degree, and four different shooting sides of 90-degree. As is illustrated in Fig. 1, they are two different angles of the same plant photographed by the participant. In the experiment, we use the images photographed from the 45-degree angle as the original image.

4 Case Study

In this section, we perform a case study to indicate the feasibility and effectiveness of the proposed MT approach to plant identification apps under different test contexts. To evaluate the ability of the proposed testing approach, we investigate whether different MRs based on test contexts could trigger inconsistent behaviors of plant identification apps.

4.1 Dataset

We use the dataset from the iNaturalist 2018 Competition [43], which is a part of the FGVC5 workshop at CVPR. iNaturalist is an object identification app, which can identify plants and animals. There are a total of 2917 plant species in the dataset, with 118800 training and 8751 validation images. We selected randomly 200 plant images from validation images as original images. By applying image transformations on original images and photographing real-world plants, a total of 8400 follow-up images were generated.

4.2 Plant Identification Apps Under Test

We have selected three AI-based plant identification mobile apps from Google Play Store as subjects. They are developed by advanced AI algorithms and trained by large-scale plant images.

- PlantSnap [44]: PlantSnap has an average rating of 3.7 in the Google Play Store. It has a database of more than 625000 plants, flowers, and mushrooms. PlantSnap has been widely used over 35 million plant lovers in more than 200 countries. It provides a maximum of 10 possible identification results for each plant image.
- PlantNet [45]: PlantNet has an average rating of 4.6 in the Google Play Store. It is organized in different databases, such as world flora, weeds, useful plants of tropical Africa, etc. In this paper, we choose the world flora database for testing. There are a total of more than 22000 species, with more than 260000 images in the world flora dataset. PlantNet outputs the possible results together with corresponding probabilities.

– PictureThis [46]: PictureThis has an average rating of 4.2 in the Google Play Store. It claimed that it is capable of identifying more than 10000 plant species with an accuracy of 98%, even better than most human experts. PictureThis returns 3 possible results for each plant image.

4.3 Evaluation Metrics

Metrics of Inconsistent Behaviors of Plant Identification Apps: In this paper, we need to evaluate whether different MRs based on test contexts could trigger inconsistent behaviors of plant identification apps. The original images and follow-up images are used to test apps. If the results of the original image and follow-up image violate MRs, it means the app has inconsistent behaviors under specific test contexts. In the case study, we select top-1 and top-3 results of each test image.

For the top-1 result, we compute the number of inconsistent behaviors of original images. For the top-3 results, we compute the dissimilarity of results between the original image and the follow-up image. Given an original image x, its corresponding follow-up image f is generated by applying a test context transformation t. Let $Rx = PlantID(x)$ be the output of original image, $Rf = PlantID(\tau(x))$ be the output of follow-up image. To compute the inconsistency between Rx and Rf, we adopt the Jaccard distance to measure their dissimilarity. It has been used in measuring the inconsistent behaviors of AI software [39]. It is worth noting that we do not consider the order of identification results. The dissimilarity is defined as follows:

$$RD = 1 - \frac{|Rx \bigcap Rf|}{|Rx \bigcup Rf|} \tag{2}$$

Obviously, the higher the dissimilarity is, the lower the similarity between the original image result and the follow-up image result. Therefore, the RD value depicts the inconsistency of results between original and follow-up images.

4.4 Case Study Results

Effective MRs are the MRs with a higher chance of revealing failures, which are the key to perform MT. First, we check whether MRs based on test contexts could reveal inconsistent behaviors of the top-1 results between the original and follow-up images. Here, we focus on the number of follow-up images whose output violates MRs. For top-1 results, we compute the number of inconsistent behavior of three apps under different MRs.

Table 3 presents the number of inconsistent behavior of top-1 results across three apps under different MRs. From the table, we can observe that TestPlantID detects 2308, 2605, 1040 inconsistent behaviors of top-1 results for PlantNet, PlantSnap, and PictureThis respectively under all MRs. In total, 5953 inconsistent behaviors are found across all three apps. From Table 3, we can observe that the number of inconsistent behavior of PictureThis is the lowest under all

Table 3. Number of inconsistent behavior of three apps under different MRs

MR based on test context	PlantNet	PlantSnap	PictureThis
MR-angle	335	329	95
MR-background	419	668	159
MR-lighting	367	437	158
MR-image clarity	320	299	163
MR-rotation	218	318	148
MR-distance	325	165	151
MR-position	324	389	166
Total	2308	2605	1040

MRs. PlantSnap shows the worst robustness across three apps since it has the maximum number of inconsistent behaviors. Interestingly, some apps are more prone to inconsistent behaviors for some specific MRs than others. For example, PlantNet produces 325 inconsistent behaviors under MR-distance, while the other two apps produce half of that number. Similarly, we detect 668 inconsistent behaviors of PlantSnap with MR-background, but only 419 and 159 for PlantNet and PictureThis respectively. Besides, PictureThis has the lowest number of inconsistent behavior under MR-angle. We can see from Table 3 that it is feasible and effective to automatically detect inconsistent behaviors of three apps under test by using test-context-based MRs we proposed.

To evaluate whether different test-context-based MRs could trigger inconsistent behaviors of top-3 results, we compute the dissimilarity of top-3 results between the original image and follow-up image. The RD value is computed

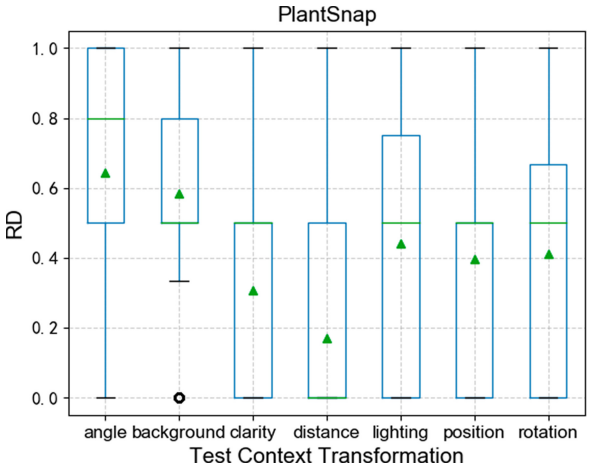

Fig. 4. RD distribution of each MR on PlantSnap

for each pair of test inputs (the original image with its corresponding follow-up image under a specific test context). Figures 4, 5 and 6 show the RD distribution under different MRs for PlantSnap, PlantNet, PictureThis respectively. In these figures, the mean RD value is depicted with a triangle label, and the median RD value is depicted with a solid line. We can observe that PictureThis performs more consistent than the other two apps under almost all seven MRs, with relatively lower median RD values and mean RD values.

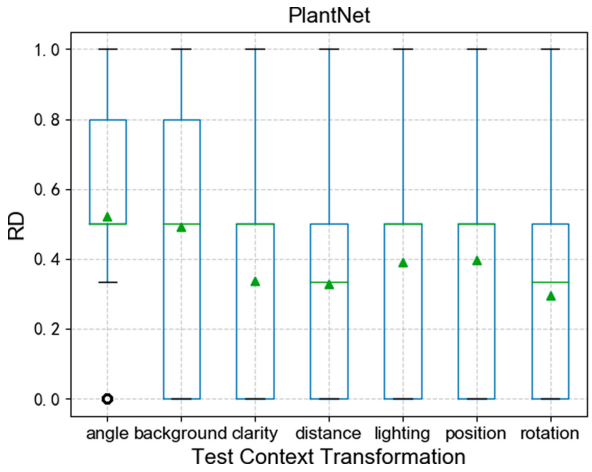

Fig. 5. RD distribution of each MR on PlantNet

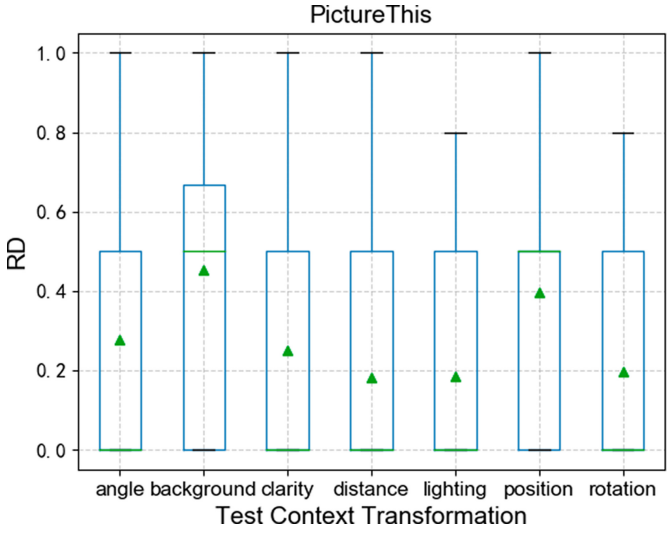

Fig. 6. RD distribution of each MR on PictureThis

MR-lighting, MR-rotation, and MR-distance have the relatively lower capability of revealing inconsistent behaviors on PictureThis. These MRs all have relatively lower median RD values equal to 0 and mean RD values around 0.2. It indicates that PictureThis could stay robust under different lighting, rotation, and distance test contexts. The possible reason is that those image transformations we leveraged to implement MR-lighting, MR-rotation and MR-distance could keep plant features of images (such as fruits, leaves, thorns, buds, or hair on the stem, which are the most characteristic organs), resulting in accurate identification results. Moreover, compared with other MRs, MR-background has the highest median and mean RD value on PictureThis, which indicates that PictureThis has the worst robustness to the background change. Moreover, MR-angle has a high capability of detecting inconsistent behaviors on PlantSnap and PlantNet, with median RD values up to 0.8 and 0.5 respectively. Compared with PlantSnap and PictureThis, MR-distance shows effectiveness in detecting inconsistent behaviors on PlantNet, with a median RD value and a mean RD value all close to 0.4. For PlantNet, MR-rotation has the lowest median RD value and mean RD value compared with other MRs. We also notice that all three apps behave similarly under the position test context condition. They all have a median RD value of 0.5 and a mean RD value of 0.4. MR-lighting and MR-position show similar performance at revealing inconsistent behaviors on PlantSnap and PlantNet. The median RD values are around 0.5 and mean RD values are around 0.4.

To sum up, different MRs based on test contexts not only effectively detects inconsistent behaviors of plant identification functions, but also potentially be useful for the measurement of the robustness of different AI-based apps under diverse context conditions.

4.5 Threats to Validity

In this paper, we generate follow-up plant images by applying simple image transformations and photographing real-world plants. However, even though we have carefully configured the parameters of image transformations, these follow-up images are not sufficient enough to cover all usage scenarios in the real-world. This could affect the chance of revealing inconsistent behaviors of apps. Besides, simple image transformations like changing the background of plants tend to be realistic, while they cannot sophisticatedly synthesize images with different complex usage scenarios. For example, the app user might photograph the plant from a moving car where is a good distance away from the plant. This kind of context scenario cannot be generated by simple image transformations. As the image processing techniques such as GAN become more and more advanced, we do expect that the generated images could be more close to real usage scenarios. Furthermore, We exploited the test contexts defined by two participants with the experience of AI software testing and good English knowledge. In this case, the final determination of test contexts was discussed by two participants, we still cannot avoid subjective factors affecting the construction of test contexts.

Moreover, we validated our approach on a dataset from iNaturalist competition with 200 original plant images. The dataset is relatively small. The limited number of test data could also be a threat to validity. Future work will conduct a large-scale empirical study to address this threat.

5 Conclusion and Future Work

In this paper, we proposed and validated TestPlantID, a metamorphic testing approach to automatically test AI-based plant identification mobile apps under different test contexts. To evaluate the robustness of AI-based plant identification apps, we leverage MRs based on test contexts to detect inconsistent behaviors. By applying image transformations and photographing real-world plants, follow-up images are generated for performing MT. Furthermore, a case study on three plant identification mobile apps is performed to indicate the feasibility and effectiveness of the proposed testing approach.

For future work, we will evaluate TestPlantID on more plant identification mobile apps. A large-scale empirical study with more datasets will be conducted. Meanwhile, we plan to implement an automatic testing tool for detecting inconsistent behaviors.

References

1. Zhang, J., Harman, M., Ma, L., Liu, Y.: Machine learning testing: survey, landscapes and horizons. arXiv preprint arXiv:1906.10742 (2019)
2. Amershi, S., et al.: Software engineering for machine learning: a case study. In: Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), Montreal, QC, Canada, pp. 291–300 (2019)
3. Tao, C., Gao, J., Wang, T.: Testing and quality validation for AI software-perspectives, issues, and practices. IEEE Access **7**, 120164–120175 (2019)
4. Yin, Y., Chen, L., Xu, Y., Wan, J.: Location-aware service recommendation with enhanced probabilistic matrix factorization. IEEE Access **6**, 62815–62825 (2018)
5. Gao, J., Tao, C., Jie, D., Lu, S.: Invited paper: what is AI software testing? and why. In: IEEE International Conference on Service-Oriented System Engineering (SOSE), San Francisco East Bay, CA, USA, pp. 27–2709 (2019)
6. Barr, E., Harman, M., McMinn, P., Shahbaz, M., Yoo, S.: The oracle problem in software testing: a survey. IEEE Trans. Softw. Eng. **41**(5), 507–525 (2015)
7. Chen, T.Y., et al.: Metamorphic testing: a review of challenges and opportunities. ACM Comput. Surv. **51**(1), 4:1–4:27 (2018)
8. Chen, T.Y., Cheung, S., Yiu, S.: Metamorphic testing: a new approach for generating next test cases. Technical report HKUST-CS98-01. Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong (1998)
9. Chen, T.Y., Tse, T., Zhou, Z.: Semi-proving: an integrated method for program proving, testing, and debugging. IEEE Trans. Softw. Eng. **37**(1), 109–125 (2011)
10. Jin, H., Jiang, Y., Liu, N., Xu, C., Ma, X., Lu, J.: Concolic metamorphic debugging. In: Proceedings of the IEEE 39th Annual Computer Software and Applications Conference (COMPSAC), Los Alamitos, CA, pp. 232–241 (2015)

11. Xie, X., Wong, W.E., Chen, T.Y., Xu, B.: Spectrum-based fault localization: testing oracles are no longer mandatory. In: Proceedings of the 11th International Conference on Quality Software (QSIC), Los Alamitos, CA, pp. 1–10 (2011)
12. Liu, H., Yusuf, I.I., Schmidt, H.W., Chen, T.Y.: Metamorphic fault tolerance: an automated and systematic methodology for fault tolerance in the absence of test oracle. In: Companion Proceedings of the 36th International Conference on Software Engineering (ICSE Companion), New York, NY, pp. 420–423 (2014)
13. Jiang, M., Chen, T.Y., Kuo, F.C., Towey, D., Ding, Z.: A metamorphic testing approach for supporting program repair without the need for a test oracle. J. Syst. Softw. **126**, 127–140 (2017)
14. Tian, Y., Pei, K., Jana, S., Ray, B.: DeepTest: automated testing of deep-neural-network-driven autonomous cars. In: Proceedings of the 40th International Conference on Software Engineering (ICSE), Gothenburg, Sweden, pp. 303–314 (2018)
15. Zhang, M., Zhang, Y., Zhang, L., Liu, C., Khurshid, S.: DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems. In: Proceedings of the 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE), Montpellier, France, pp. 132–142 (2018)
16. Zhou, Z., Sun, L.: Metamorphic testing of driverless cars. Commun. ACM **62**(3), 61–67 (2019)
17. Murphy, C., Kaiser, G.E., Hu, L., Wu, L.: Properties of machine learning applications for use in metamorphic testing. In: Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE), San Francisco, CA, USA, pp. 867–872 (2008)
18. Xie, X., Ho, J.W., Murphy, C., Kaiser, G., Xu, B., Chen, T.Y.: Testing and validating machine learning classifiers by metamorphic testing. J. Syst. Softw. **84**(4), 544–558 (2011)
19. Brown, J., Zhou, Z. Chow, Y.: Metamorphic testing of navigation software: a pilot study with google maps. In: 51st Hawaii International Conference on System Sciences (HICSS), Hilton Waikoloa Village, Hawaii, USA, pp. 1–10 (2018)
20. Zhou, Z., Xiang, S., Chen, T.Y.: Metamorphic testing for software quality assessment: a study of search engines. IEEE Trans. Softw. Eng. **42**(3), 264–284 (2016)
21. Wang, S., Su, Z.: Metamorphic testing for object detection systems. arXiv preprint arXiv:1912.12162 (2019)
22. Chen, T.Y., Poon, P., Xie, X.: METRIC: METamorphic Relation Identification based on the Category-choice framework. J. Syst. Softw. **116**, 177–190 (2016)
23. Zhang, J., et al.: Search-based inference of polynomial metamorphic relations. In: Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE), New York, pp. 701–712 (2014)
24. Zhu, H.: A tool for automated Java unit testing based on data mutation and metamorphic testing methods. In: Proceedings of the 2nd International Conference on Trustworthy Systems and Their Applications (TSA), Los Alamitos, CA, pp. 8–15 (2015)
25. Zhu, H., Liu, D., Bayley, I., Harrison, R., Cuzzolin, F.: Datamorphic testing: a method for testing intelligent applications. In: IEEE International Conference On Artificial Intelligence Testing (AITest), Newark, CA, USA, pp. 149–156 (2019)
26. Pei, K., Cao, Y., Yang, J., Jana, S.: Deepxplore: Automated whitebox testing of deep learning systems. In: Proceedings of the 26th Symposium on Operating Systems Principles (SOSP), pp. 1–18. Shanghai, China (2017)
27. Ma, L., et al.: DeepGauge: multi-granularity testing criteria for deep learning systems. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE), Montpellier, France, pp. 120–131 (2018)

28. Sun, Y., Huang, X., Kroening, D.: Testing deep neural networks. arXiv preprint arXiv:1803.04792 (2019)
29. Guo, J., Jiang, Y. Zhao, Y. Chen, Q. Sun, J.: DLFuzz: differential fuzzing testing of deep learning systems. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/SIGSOFT), Lake Buena Vista, FL, USA, pp. 739–743 (2018)
30. Odena, A., Olsson, C., Andersen, D., Goodfellow, I.: TensorFuzz: debugging neural networks with coverage-guided fuzzing. In: Proceedings of the 36th International Conference on Machine Learning (ICML), Long Beach, California, USA, pp. 4901–4911 (2019)
31. Xie, X., Chen, H., Li, Y., Ma, L., Liu, Y., Zhao, J.: Coverage-guided fuzzing for feedforward neural networks. In: 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), San Diego, CA, USA, pp. 1162–1165 (2019)
32. Ma, L., et al.: DeepMutation: mutation testing of deep learning systems. In: Proceedings of the 29th IEEE International Symposium on Software Reliability Engineering (ISSRE), Memphis, TN, pp. 100–111 (2018)
33. Shen, W., Wan, J., Chen, Z.: MuNN: mutation analysis of neural networks. In: IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), Lisbon, pp. 108–115 (2018)
34. Ding, J., Kang, X., Hu, X.: Validating a deep learning framework by metamorphic testing. In: 2017 IEEE/ACM 2nd International Workshop on Metamorphic Testing (MET), Buenos Aires, pp. 28–34 (2017)
35. Murphy, C., Shen, K., Kaiser, G.: Using JML runtime assertion checking to automate metamorphic testing in applications without test oracles. In: International Conference on Software Testing Verification and Validation (ICST), Denver, Colorado, USA, pp. 436–445 (2009)
36. Sun, Y., Wu, M., Ruan, W., Huang, X., Kwiatkowska, M., Kroening, D.: Concolic testing for deep neural networks. In: 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE), Montpellier, France, pp. 109–119 (2018)
37. Gopinath, D., Wang, K., Zhang, M., Pasareanu, C., Khurshid, S.: Symbolic execution for deep neural networks. arXiv preprint arXiv:1807.10439 (2018)
38. Gopinath, D., Zhang, M., Wang, K., Kadron, B., Pasareanu, C., Khurshid, S.: Symbolic execution for importance analysis and adversarial generation in neural networks. In: IEEE 30th International Symposium on Software Reliability Engineering (ISSRE), Berlin, Germany, pp. 313–322 (2019)
39. Zhang, Z., Xie, X.: On the investigation of essential diversities for deep learning testing criteria. In: IEEE 19th International Conference on Software Quality, Reliability and Security (QRS), Sofia, Bulgaria, pp. 394-405 (2019)
40. Affine Transformation (2015). https://www.mathworks.com/discovery/affine-transformation.html
41. Open Source Computer Vision Library (2015). https://github.com/itseez/opencv
42. Removebg (2019). https://github.com/remove-bg
43. iNaturalist 2018 Competition (2018). https://github.com/visipedia/inat_comp/tree/master/2018
44. PlantSnap. https://play.google.com/store/apps/details?id=com.fws.plantsnap2
45. PlantNet. https://play.google.com/store/apps/details?id=org.plantnet
46. PictureThis. https://play.google.com/store/apps/details?id=cn.danatech.xingseus