Automated Mobile Apps Testing from Visual Perspective

Feng Xue
School of Computer Science and Engineering
Northwestern Polytechnical University
Xi'an, PR China, 710072
xue.f@mail.nwpu.edu.cn

ABSTRACT

The current implementation of automated mobile apps testing generally relies on internal program information, such as reading code or GUI layout files, capturing event streams. This paper proposes an approach of automated mobile apps testing from a completely visual perspective. It uses computer vision technology to enable computer to judge the internal functions from the external GUI information of mobile apps as we humans do and generates test strategy for execution, which improves the interactivity, flexibility, and authenticity of testing. We believe that this vision-based testing approach will further help alleviate the contradiction between the current huge test requirements of mobile apps and the relatively lack of testers.

CCS CONCEPTS

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

KEYWORDS

Software testing, Test automation, Mobile applications, Computer vision

ACM Reference format:

Feng Xue. 2020. Automated Mobile Apps Testing from Visual Perspective. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'20), July 18-22, 2020, Virtual Event, USA. ACM, New York, NY, USA, 5 pages.* https://doi.org/10.1145/3395363.3402644

1 INTRODUCTION

In recent years, mobile apps have shown a rapid development trend due to benefits from the advancement of mobile devices and mobile technology. As of 2019, the total number of mobile apps on Google Play and Apple App Store had reached 4.4 million [1]. Furthermore, from the advent of mobile apps in 2009 to 2019, the proportion of traffic generated by mobile devices increased from 0.7% to 52.6% of global website traffic [2]. Mobile apps are gradually replacing desktop software in people's daily lives.

However, unlike desktop software, mobile apps are usually large in number, diverse in type, with frequent version iterations, and have more complex operating environment [3, 4]. These features of mobile apps make testing more difficult, and it is even more urgent to seek effective automated testing approaches to solve the current mobile apps testing dilemma. At present, the main automated testing techniques for mobile apps can be divided into the following three levels.

The first level techniques focus on automating test execution.

Script-based testing enables the automatic execution of test cases defined by manually editing scripts. The test automation frameworks, such as Espresso [5], XCTest [6], Appium [7], can convert scripts into event streams and input AUT (App Under Test) to simulate test execution. Different from text-only script, Sikuli [8], Eyeautomate [9] allow to define a script containing visual information (a screenshot of interface element) and implement visual GUI testing through image matching.

Record-replay testing records the manually executed test cases and then plays them back for reuse of test execution. RERAN [10] captures the low-level event stream including GUI events and sensor events, then replays them for test execution. SARA [11] uses the proposed self-replay and adaptive replay mechanisms to achieve a high record-replay success rate for single and cross devices. LIRAT [12] combines image processing technology to achieve cross-platform record-replay testing.

The second level techniques are based on the first level and focus more on automating test case generation and optimization.

Random testing uses a random strategy to explore and test apps. Although random, it automatically generates test inputs. Monkey [13], a native testing tool for Android, sends randomly selected GUI events and system events for testing. More improvements in this approach revolve around increasing the efficiency of effective event generation [14, 15].

Model-based testing automates testing by modeling apps behavior and using model generated test cases. MobiGUITAR [16] dynamically traverses AUT to build model and then generates test cases. AMOGA [17] takes a static-dynamic approach to statically obtain the association between GUI elements and events from the source code, and then dynamically and orderly explores AUT to generate model. These models are usually represented by a finite state machine.

ISSTA'20, July 18-22, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-8008-9/20/07...$15.00
https://doi.org/10.1145/3395363.3402644

Search-based testing leverages heuristic algorithms to generate test cases and optimize test sequences. AGRippin [18] founded by genetic and hill climbing algorithms facilitates efficient and effective test case generation. Sapienz [19] introduces Pareto multi-objective approach to reach better test coverage with fewer test cases.

Transfer-based testing is dedicated to cross-platform and cross-app use of test cases. TestMig [20] uses similar comparison of GUI controls and events to realize the migration of test cases from IOS to Android. AppTestMigator [21] and CraftDroid [22] enable test cases to run on multiple apps with similar functions through matching the semantically similar of functions.

The third level techniques further introduce the feature of learning.

Traditional model learning-based testing is based on the traditional finite state machine modeling to add a learning mechanism to achieve dynamic construction and generate on-the-fly test inputs. SwiftHand [23] employs an improved L* algorithm to refine model during test execution. DroidBot [24] dynamically updates the model through the adapter module and the brain module.

Deep learning-based testing applies deep neural network technology to train an end-to-end test model. Humanoid [25] designs a deep neural network model to generate human-like inputs by learning user interaction with apps. DL-Droid [26] proposed a deep learning system to detect malicious Android apps through dynamic analysis using stateful input generation.

Reinforcement learning-based testing develops a reinforcement learning mechanism for mobile apps to automatically learn to obtain a test model by a trial-and-error way. [27, 28] exploit Q-learning algorithm to explore AUT and build test model.

All these automated testing techniques have significantly promoted the development of mobile apps testing. However, their implementation is more or less dependent on the internal information of apps, which leads to the limitations in use. Thus, we propose a vision-based testing approach that simulates the user's cognitive and testing process of mobile apps from a totally visual perspective, enabling computer to learn and understand functions contained in mobile app interfaces and associate the correct testing actions. Finally, it can test mobile apps visually like humans without being limited by software.

This approach aims to achieve a more abstract cognition of mobile apps by virtue of the advantages of visual technology (different from most testing techniques that explore and learn mobile apps from a software perspective). Then, it uses the functions of apps as object for modeling to train the computer to learn to cognize the internal function under the external appearance of GUIs among multiple apps (different from most testing techniques use AUT as object for modeling). For example, most apps include a log-in function and the log-in operation is almost similar, yet the log-in GUIs have different styles in different apps. Our approach gives the computer the ability to abstract the representative features of the log-in interface and thus help it to determine the log-in function across apps but not just to identify the log-in of a certain AUT. Subsequently, the computer can associate the appropriate test actions to test certain functions in multiple apps, just as we human testers or users do.

Our expected contributions for this vision-based automated testing approach are followed:

- Recognize apps completely visually like user without resorting to any internal program information.
- Build a more generalized test model with respect to the test model for certain AUT to solve cross-platform and cross-version issues.
- Achieve flexible interactive judgment testing rather than rigid automated execution.

2 VISION-BASED AUTOMATED TESTING OVERVIEW

An overview of the vision-based automated testing we propose is presented in this section. Figure 1 shows the main modules of the testing approach and an overview of the testing process. These modules are (1) *App Interface Recognition* and (2) *Interactive Test Execution* belonging to *Test Execution Environment*, and (3) *Test Self-Learning* and (4) *Test Strategy Generation* belonging to *Test Learning Environment*.

Test Execution Environment completes the test execution of mobile apps in an automated manner and the collection of visual information of apps. In order to reduce the dependence on platform, here we use robotic arm that mimics realistic human actions on mobile apps [29, 30] for test execution and use camera shooting or screen projection to obtain GUI visual information. *Test Learning Environment* implements learning of mobile apps through the GUI visual information, and training to generate corresponding test strategy. A learning-based test model will be built in it.

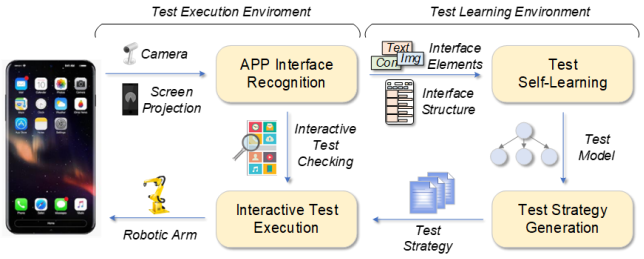

Figure 1: Overview of the vision-based automated testing process

App Interface Recognition. *App Interface Recognition* adopts a purely visual way to recognize apps. Compared with the way of constructing test models by reading configuration files or source code, using vision to cognitive app: on the one hand, from app's external function manifestation rather than internal code logic to explore the app, can achieve a more human-like testing; on the other hand, for the specificity of multi-platform and multi-version of mobile apps, visual way can provide a cross-platform testing advantage and facilitate abstract and flexible judgment. *App Interface Recognition* recognizes the

type and location information of app interface elements and extracts interface features to form GUI abstraction.

Test Self-Learning. *Test Self-Learning* further cognizes app structure based on app interface elements recognized by *App Interface Recognition*. According to the characteristics of app and the commonalities of app function, two aspects of abstract cognition are respectively conducted: one is the appearance of single static pages, and the other is the dynamic behaviors between multiple pages. Further let computer learn to judge functions of app from the external information and associate accurate test actions. Then, app function-based models are built.

Test Strategy Generation. As the test model generated by *Test Self-Learning* is a function-based model, it cannot be used directly for the AUT. *Test Strategy Generation* transforms function-based models into app type-based models and generates test strategy for AUT.

Interactive Test Execution. *Interactive Test Execution* is performed on the test environment throughout the entire testing process. After executing the test actions of each step, the interactive test execution calls the *App Interface Recognition* to obtain the response interface, check the correctness of the test execution and judge whether the interface shows an abnormality by comparing it with test strategy.

3 IMPLEMENTATION OF VISION-BASED AUTOMATED TESTING

According to the vision-based automated testing process we designed, there are two key issues to break through. First, how to effectively capture GUI information from a visual perspective and achieve an abstract description of GUI; second, how to possess the generalized cognitive ability of app interface and the association ability of appropriate actions like humans. Figure 2 shows the technical approach for the vision-based automated testing, which contains two steps *3.1 GUI Abstraction* and *3.2 GUI Cognition and Action Association* to solve the two issues.

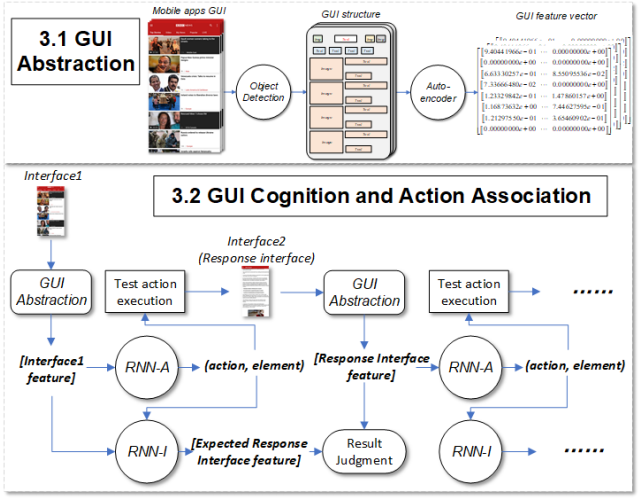

Figure 2: Technical approach for the vision-based automated testing

3.1 GUI Abstraction

For the GUI of mobile apps, we do not care about the styles of the GUI, but the GUI controls it contains. Therefore, an object detection technique is applied to conduct interface element recognition and obtain the abstract GUI structure. In view of the dynamic and real-time nature of testing, a one-stage object detection technique would be more appropriate, such as yolov3 [31] we used in the environment, which allows a balance between speed and precision.

We divide common GUI elements into 12 types (TextView, EditText, ImageView, Button, ImageButton, RadioButton, ToggleButton, CheckBox, Spinner, Switch, ProgressBar, NumberPicker) from a visual point of view and build a mobile apps GUI dataset for one-stage object detection based on RICO [32] a large dataset of mobile apps interfaces. In the training of interface element recognition, we improved the loss function and result output judgment of yolov3 to make it more focused on accurate bounding box output, because the omission of element detection is more fatal to the test than misclassification. Consequently, we extract GUI structure only by visual means without resorting to other platform-dependent tools. In the generated GUI structure, different types of GUI elements will be distinguished by colors. Using GUI structure instead of the original interface image will help the computer to better determine the meaning of the interface elements in app interface and can lead the robotic arm to perform accurate actions on interface.

After obtaining the GUI structure, we expect that the computer can further extract the image features of GUI structure. Here, an unsupervised autoencoder neural network [33] is used to obtain the feature vectors of GUI structure. Thus, we have converted the GUI image into feature vectors through two step abstraction, and these vector features will represent the GUI image to participate in the subsequent work of GUI cognition and action association.

3.2 GUI Cognition and Action Association

After acquiring the abstract representation of app static interface, the ability of dynamic behavior cognition is further trained. We propose a joint architecture of double RNN (Recurrent Neural Network) for training to obtain the test model, as shown in Figure 2. In view of the time sequence of test actions and GUI feedback during the testing process, RNN-A is used to process the action sequence, and RNN-I is used to process the GUI sequence.

Here we do not use a single app as the training object, but the app function as the training object. The reason for this is that we expect computer to learn the action intent behind the interface, not just the interface jump relationship. For example, search is a common function in apps, and although it has some appearance differences in different apps, it is still easy to identify by us humans. Then our goal is to allow the computer to cognize the search function in multiple apps through this training, rather than just targeting a certain AUT.

Therefore, in order to allow computer to find common features of functions in different interface appearances, we adopt deep learning techniques that are currently effective means to deal with such problems. In our training, based on the group of same type app, a sequence of test actions is further defined for the common functions of the apps, such as finding the search function and searching for a "The Art of Computer Programming" in different books apps. In continuous training, the computer can extract key features and correctly associate defined actions.

Specifically, in our network architecture, RNN-A is used to receive the feature vectors of the initial GUI generated by GUI abstraction, and output feasible *(action, element)* pairs that reach another target interface. At the same time, the feature vectors of the initial GUI will be delivered to RNN-I as the initial state. Then, the *(action, element)* determined by RNN-A is sent to the test execution environment to actually execute on AUT, and RNN-A obtains the response interface to generate the response interface features again; on the other hand, the *(action, element)* is passed to RNN-I, and the expected interface features are inferred from RNN-I. Finally, the actual interface features are compared with the expected interface features to check and determine the correctness of the test results.

In result judgement, we expect that the computer can judge the similarity between GUIs from a more abstract perspective rather than a simple pixel comparison. Many functions usually have similarities in the representation of GUIs of different apps, although their representation details including the position, size, and style of elements are various, such as log-in or search functions. Since we represent GUI images with feature vectors, the relative entropy [33] of the vectors is calculated to compare the similarity of GUI. Based on similar calculation results, we divide three comparison results by threshold. The first judgment is that the GUIs are almost completely similar. This situation often occurs in the homogeneous interface in the same application. For example, in the product display interface of some shopping apps, the GUI structure is exactly the same, only the product content is different. This judgment is helpful to suppress the GUI path explosion problem in the test. The second judgment is that GUIs are not completely consistent but have similarities, such as login interfaces in multiple apps. The third kind of judgment, GUIs have a big difference. The second and third judgments are conducive to the recognition of the same type of GUI and help us judge the correctness of the GUI.

Different from other model-based testing approach, in this deep learning-based modeling approach, the state of apps is represented by interface features, and the state transition is determined by the RNN-I. The action that triggers the state transition of apps is performed by the RNN-A real-time judgment.

Through the above process, we first train computer to build app function-based models. Each function is divided into different types of apps for the input and output interface and test actions of the function. And app function-based models can be converted into app type-based model for testing of AUT.

During the testing phase, the type of AUT is first obtained, the test model for this type of app applied by training phase is retrieved. Then the test actions are inferred and performed while the GUI recognition due to the test model. The test results are compared to the expected results reflected by the test model. All the test actions and judgment results are recorded to form a test report.

4 EVALUATION PLAN

The validation and evaluation of our vision-based automated testing will be planned from two aspects.

Firstly, evaluate the accuracy of the algorithms we introduced: (1) the accuracy of interface elements recognition through object detection; (2) the correct abstraction and judgment of interface through autoencoder and relative entropy; (3) the correctness of inferring the feasible actions and the response interface by RNN. Accuracy, precision, recall, F1-score, mAP and other evaluation metrics will be employed.

Secondly, evaluate the effectiveness of the proposed vision-based automated testing approach for mobile apps. We will use real apps (market apps from Google Play and Apple App Store, open-source apps from F-Droid and AndroTest) as test objects to compare with the state-of-the-art testing approaches, especially some highly related works [21, 25, 34, 35]. We will evaluate the performance of the following test features under different testing techniques: (1) platform dependency, whether it can effectively support cross-platform or cross-version testing; (2) permission requirement, whether it requires a higher privilege to conduct testing, such as developer privilege that may not be obtained under certain test conditions; (3) efficiency of testing, whether it has an efficient performance in the test preparation phase and execution phase; (4) supported types of testing, whether it can support various types of testing, such as functionality testing, quality of service testing, usability testing, security testing; (5) quality of testing, whether it has relatively good performance in bug detection and test coverage.

5 CONCLUSIONS AND FUTURE RESEARCH

Based on computer vision technology, we propose an automated mobile apps testing approach from user perspective, which provides a new idea for the current mobile apps testing automation.

At present, we have completed the preliminary realization of *3.1 GUI Abstraction*, and *3.2 GUI Cognition and Action Association* will be the focus of the next step. And in the follow-up, we will conduct a full experimental comparison with the current state-of-the-art testing techniques to verify and optimize our approach.

We believe that this vision-based testing approach can provide a test capability similar to manual testing to a certain extent, thereby alleviating the contradiction between the huge test requirements of mobile apps and the relatively lack of testers.

REFERENCES

[1] Statista. 2019. Number of apps available in leading app stores. Retrieved March 16, 2020 from https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

[2] Statista. 2019. Percentage of mobile device website traffic worldwide. Retrieved March 16, 2020 from https://www.statista.com/statistics/277125/share-of-website-traffic-coming-from-mobile-devices

[3] Tramontana Porfirio, Amalfitano Domenico, Amatucci Nicola and Fasolino Anna Rita. 2019. Automated functional testing of mobile applications: a systematic mapping study. *Software Quality Journal* 27, 1 (March. 2019), 149-201. https://doi.org/10.1007/s11219-018-9418-6

[4] Kong Pingfan, Li Li, Gao Jun, Liu Kui, Bissyande Tegawende F. and Klein Jacques. 2019. Automated Testing of Android Apps: A Systematic Literature Review. *IEEE Transactions on Reliability* 68, 1 (March 2019), 45-66. https://doi.org/10.1109/TR.2018.2865733

[5] Espresso. Use Espresso to write concise, beautiful, and reliable Android UI tests. Retrieved March 16, 2020 from https://developer.android.com/training/testing/espresso

[6] Xctest. Create and run unit tests, performance tests, and UI tests for your Xcode project. Retrieved March 16, 2020 from https://developer.apple.com/documentation/xctest

[7] Appium. Automation for Apps. Retrieved March 16, 2020 from http://appium.io

[8] Yeh Tom, Chang Tsung-Hsiang and Miller Robert C. 2009. Sikuli: using GUI screenshots for search and automation. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology (UIST '09)*, Victoria, BC, Canada. Association for Computing Machinery, 183–192. https://doi.org/10.1145/1622176.1622213

[9] Eyeautomate. Visual Script Runner. Retrieved March 16, 2020 from https://eyeautomate.com/eyeautomate/

[10] Gomez Lorenzo, Neamtiu Iulian, Azim Tanzirul and Millstein Todd. 2013. Reran: Timing-and touch-sensitive record and replay for android. In *Proceedings of the 2013 35th International Conference on Software Engineering (ICSE '13)*, San Francisco, CA, USA. IEEE, 72-81. https://doi.org/10.1109/ICSE.2013.6606553

[11] Guo Jiaqi, Li Shuyue, Lou Jian-Guang, Yang Zijiang and Liu Ting. 2019. Sara: self-replay augmented record and replay for Android in industrial cases. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '19)*, Beijing, China. Association for Computing Machinery, 90–100. https://doi.org/10.1145/3293882.3330557

[12] Yu Shengcheng, Fang Chunrong, Feng Yang, Zhao Wenyuan and Chen Zhenyu. 2019. LIRAT: Layout and Image Recognition Driving Automated Mobile Testing of Cross-Platform. In *Proceedings of the 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE '19)*, San Diego, CA, USA. IEEE, 1066-1069. https://doi.org/10.1109/ASE.2019.00103

[13] Monkey. UI/Application Exerciser Monkey. Retrieved March 16, 2020 from https://developer.android.com/studio/test/monkey

[14] Osman Mohamed S and Wasmi Hiba Ayyed. 2019. Improved Monkey Tool for Random Testing in Mobile Applications. In *Proceedings of the 2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT '19)*, Amman, Jordan, Jordan. IEEE, 658-662. https://doi.org/10.1109/JEEIT.2019.8717506

[15] Machiry Aravind, Tahiliani Rohan and Naik Mayur. 2013. Dynodroid: an input generation system for Android apps. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE '13)*, Saint Petersburg, Russia. Association for Computing Machinery, 224–234. https://doi.org/10.1145/2491411.2491450

[16] Amalfitano Domenico, Fasolino Anna Rita, Tramontana Porfirio, Ta Bryan Dzung and Memon Atif M. 2014. MobiGUITAR: Automated model-based testing of mobile apps. *Ieee Software* 32, 5 (April 2014), 53-59. https://doi.org/10.1109/MS.2014.55

[17] Salihu Ibrahim-Anka, Ibrahim Rosziati, Ahmed Bestoun S, Zamli Kamal Z and Usman Asmau. 2019. AMOGA: a static-dynamic model generation strategy for mobile apps testing. *IEEE Access* 7 (January 2019), 17158-17173. https://doi.org/10.1109/ACCESS.2019.2895504

[18] Amalfitano Domenico, Amatucci Nicola, Fasolino Anna Rita and Tramontana Porfirio. 2015. AGRippin: a novel search based testing technique for Android applications. In *Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile (DeMobile '15)*, Bergamo, Italy. Association for Computing Machinery, 5–12. https://doi.org/10.1145/2804345.2804348

[19] Mao Ke, Harman Mark and Jia Yue. 2016. Sapienz: multi-objective automated testing for Android applications. In *Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA '16)*, Saarbrücken, Germany. Association for Computing Machinery, 94–105. https://doi.org/10.1145/2931037.2931054

[20] Qin Xue, Zhong Hao and Wang Xiaoyin. 2019. TestMig: migrating GUI test cases from iOS to Android. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '19)*, Beijing, China. Association for Computing Machinery, 284–295. https://doi.org/10.1145/3293882.3330575

[21] Behrang Farnaz and Orso Alessandro. 2019. Test migration between mobile apps with similar functionality. In *Proceedings of the 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE '19)*, San Diego, CA, USA, USA IEEE, 54-65. https://doi.org/10.1109/ASE.2019.00016

[22] Lin Jun-Wei, Jabbarvand Reyhaneh and Malek Sam. 2019. Test Transfer Across Mobile Apps Through Semantic Mapping. In *Proceedings of the 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE '19)*, San Diego, CA, USA, USA. IEEE, 42-53. https://doi.org/10.1109/ASE.2019.00015

[23] Choi Wontae, Necula George and Sen Koushik. 2013. Guided GUI testing of android apps with minimal restart and approximate learning. *SIGPLAN Not.* 48, 10 (October 2013), 623–640. https://doi.org/10.1145/2544173.2509552

[24] Li Yuanchun, Yang Ziyue, Guo Yao and Chen Xiangqun. 2017. DroidBot: a lightweight UI-guided test input generator for Android. In *Proceedings of the 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, Buenos Aires, Argentina. IEEE, 23-26. https://doi.org/10.1109/ICSE-C.2017.8

[25] Li Yuanchun, Yang Ziyue, Guo Yao and Chen Xiangqun. 2019. Humanoid: A Deep Learning-Based Approach to Automated Black-box Android App Testing. In *Proceedings of the 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE '19)*, San Diego, CA, USA. IEEE, 1070-1073. https://doi.org/10.1109/ASE.2019.00104

[26] Alzaylaee Mohammed K, Yerima Suleiman Y and Sezer Sakir. 2020. DL-Droid: Deep learning based android malware detection using real devices. *Computers & Security* 89 (February 2020), 101663. https://doi.org/10.1016/j.cose.2019.101663

[27] Koroglu Yavuz, Sen Alper, Muslu Ozlem, Mete Yunus, Ulker Ceyda, Tanriverdi Tolga and Donmez Yunus. 2018. QBE: QLearning-based exploration of android applications. In *Proceedings of the 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST '18)*, Vasteras, Sweden. IEEE, 105-115. https://doi.org/10.1109/ICST.2018.00020

[28] Vuong Thi Anh Tuyet and Takada Shingo. 2018. A reinforcement learning based approach to automated testing of Android applications. In *Proceedings of the 9th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation (A-TEST '19)*, Lake Buena Vista, FL, USA. Association for Computing Machinery, 31–37. https://doi.org/10.1145/3278186.3278191

[29] Craciunescu Mihai, Mocanu Stefan, Dobre Cristian and Dobrescu Radu. 2018. Robot based automated testing procedure dedicated to mobile devices. In *Proceedings of the 2018 25th International Conference on Systems, Signals and Image Processing (IWSSIP '18)*, Maribor, Slovenia. IEEE, 1-4. https://doi.org/10.1109/IWSSIP.2018.8439614

[30] Mao Ke, Harman Mark and Jia Yue. 2017. Robotic testing of mobile apps for truly black-box automation. *Ieee Software* 34, 2 (March 2017), 11-16. https://doi.org/10.1109/MS.2017.49

[31] Redmon Joseph and Farhadi Ali. 2018. Yolov3: An incremental improvement. arXiv:1804.02767. Retrieved from https://arxiv.org/abs/1804.02767

[32] Deka Biplab, Huang Zifeng, Franzen Chad, Hibschman Joshua, Afergan Daniel, Li Yang, Nichols Jeffrey and Kumar Ranjitha. 2017. Rico: A Mobile App Dataset for Building Data-Driven Design Applications. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST '17)*, Québec City, QC, Canada. Association for Computing Machinery, 845–854. https://doi.org/10.1145/3126594.3126651

[33] Dizaji Kamran Ghasedi, Herandi Amirhossein, Deng Cheng, Cai Weidong and Huang Heng. 2017. Deep Clustering via Joint Convolutional Autoencoder Embedding and Relative Entropy Minimization. In *Proceedings of the 2017 IEEE International Conference on Computer Vision (ICCV '17)*, Venice, Italy. IEEE, 5747-5756. https://doi.org/10.1109/ICCV.2017.612

[34] Degott Christian, Jr. Nataniel P. Borges and Zeller Andreas. 2019. Learning user interface element interactions. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA ' 19)*, Beijing, China. Association for Computing Machinery, 296–306. https://doi.org/10.1145/3293882.3330569

[35] Borges Nataniel P., Gómez Maria and Zeller Andreas. 2018. Guiding app testing with mined interaction models. In *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft '18)*, Gothenburg, Sweden. Association for Computing Machinery, 133–143. https://doi.org/10.1145/3197231.3197243