OwlEyes-Online: A Fully Automated Platform for Detecting and Localizing UI Display Issues

Yuhui Su[*][†]
Institute of Software Chinese Academy of Sciences, University of Chinese Academy of Sciences
Beijing, China

Zhe Liu[*][†]
Institute of Software Chinese Academy of Sciences, University of Chinese Academy of Sciences
Beijing, China

Chunyang Chen
Monash University
Melbourne, Australia

Junjie Wang[†][‡][§]
Institute of Software Chinese Academy of Sciences, University of Chinese Academy of Sciences
Beijing, China

Qing Wang[†][‡][§]
Institute of Software Chinese Academy of Sciences, University of Chinese Academy of Sciences
Beijing, China

ABSTRACT

GUI provides visual bridges between software apps and end users. However, due to the compatibility of software or hardware, UI display issues such as text overlap, blurred screen, image missing always occur during GUI rendering on different devices. Because these UI display issues can be found directly by human eyes, in this paper, we implement an online UI display issue detection tool `OwlEyes-Online`, which provides a simple and easy-to-use platform for users to realize the automatic detection and localization of UI display issues. The `OwlEyes-Online` can automatically run the app and get its screenshots and XML files, and then detect the existence of issues by analyzing the screenshots. In addition, `OwlEyes-Online` can also find the detailed area of the issue in the given screenshots to further remind developers. Finally, `OwlEyes-Online` will automatically generate test reports with UI display issues detected in app screenshots and send them to users. It was evaluated and proved to be able to accurately detect UI display issues. Tool Link: http://www.owleyes.online:7476. Github Link: https://github.com/franklinbill/owleyes. Demo Video Link: https://youtu.be/002nHZBxtCY.

CCS CONCEPTS

• **Software and its engineering**;

KEYWORDS

UI display, Mobile app, UI testing, Deep learning, Issue detection

[*]Both authors contributed equally to this research.
[†]Also With Laboratory for Internet Software Technologies, Beijing, China
[‡]Corresponding author
[§]Also With State Key Laboratory of Computer Sciences, Beijing, China

ACM Reference Format:
Yuhui Su, Zhe Liu, Chunyang Chen, Junjie Wang, and Qing Wang. 2021. OwlEyes-Online: A Fully Automated Platform for Detecting and Localizing UI Display Issues. In *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '21), August 23–28, 2021, Athens, Greece.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3468264.3473109

1 INTRODUCTION

GUI is widely used among modern mobile apps, making it practical and easy to use. However, with the development of visual effects of GUI, five categories of UI display issues [40] such as *component occlusion, text overlap, missing image, null value* and *blurred screen* always occur during the UI display process, especially on different mobile devices. Detecting those issues is a hard problem because most of those UI display issues are caused by many factors, especially for Android, such as different Android OS versions, device models, and screen resolutions [34]. Nowadays, some practical automated testing tools like Monkey [16, 35], Dynodroid [24] are also widely used in industry. However, these automated tools can only spot critical crash bugs, rather than UI display issues that cannot be captured by common tools. Inspired by the fact that display bugs can be easily spotted by human eyes, we develop an automated online tool `OwlEyes-Online`[1], which provides quick detection and localization of UI display issues from apps or GUI screenshots.

The `OwlEyes-Online` is a user-friendly web app. Developers can upload GUI screenshots or apps and receive accurate UI display issue detection results. When a developer uploads an APK, it will automatically run the app and get its screenshots, and then we use computer vision technologies to detect the UI display issues. `OwlEyes-Online` builds on the CNN to identify the screenshots with issues and Grad CAM to localize the regions with UI display issues in the screenshots for further reminding developers. Finally, it summarizes the detection and localization results, automatically generates the test report and sends it to users. Considering that the CNN needs lots of training data, we adopt a heuristic data generation method to generate the training data.

[1]`OwlEyes-Online` is named as our approach is like the owl's eyes to effectively spot UI display issues. And our model (nocturnal like an owl) can complement conventional automated GUI testing (diurnal like an eagle) for ensuring the robustness of the UI.

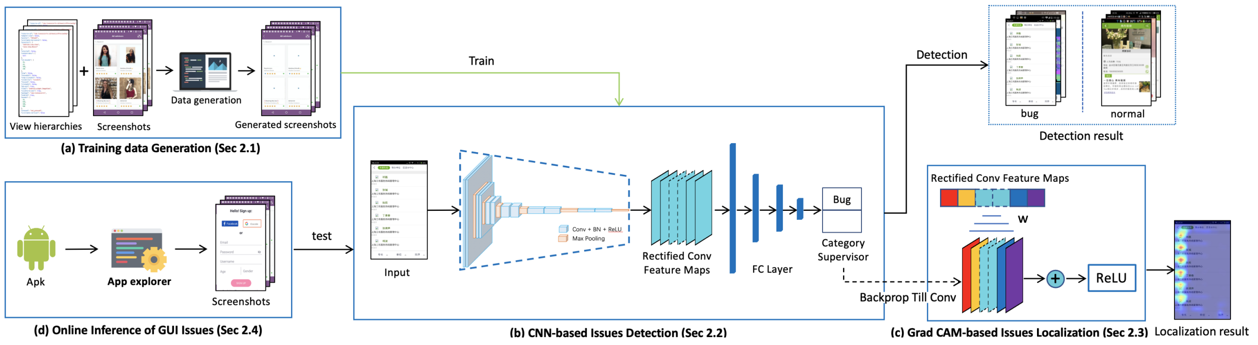


Figure 1: Overview of `OwlEyes-Online`

`OwlEyes-Online` provides a dashboard for users to upload the screenshots or apps. After analyzing an uploaded screenshot, it displays detection results in real-time. As for an app, it automatically generates a test report (issue screenshots, localization, etc.) and sends the report to the user in an email.

This paper makes the following contributions:

- We implement a CNN based issue detection method and a Grad CAM based issue localization method to detect UI display issues from GUI screenshots.
- We develop a fully automated web app. Users only need to upload an APK file, and `OwlEyes-Online` will automatically generate test reports and send them to users. We release the implementation of `OwlEyes-Online` on Github [4].
- An empirical study among professionals proves the value of our UI display issue detection method and `OwlEyes-Online`.

2 OUR FULLY AUTOMATED APPROACH

According to the features of UI display issues, we propose a fully automated UI display issue detection and localization approach. It mainly includes four parts, which are heuristic based data generation, CNN based issues detection, Grad CAM based issues localization, and online inference of GUI issues. As shown in Figure 1, to improve the accuracy of our model, we use the heuristic based data generation method to generate a number of training data. Given an APK, `OwlEyes-Online` automatically runs it and collects screenshots. Then the CNN based model classifies if they relate to any issues via the visual understanding. Once an issue is confirmed, our model can further localize its specific issue position on the UI screenshot by Grad CAM based model to remind the developers.

2.1 Heuristic Based Data Generation

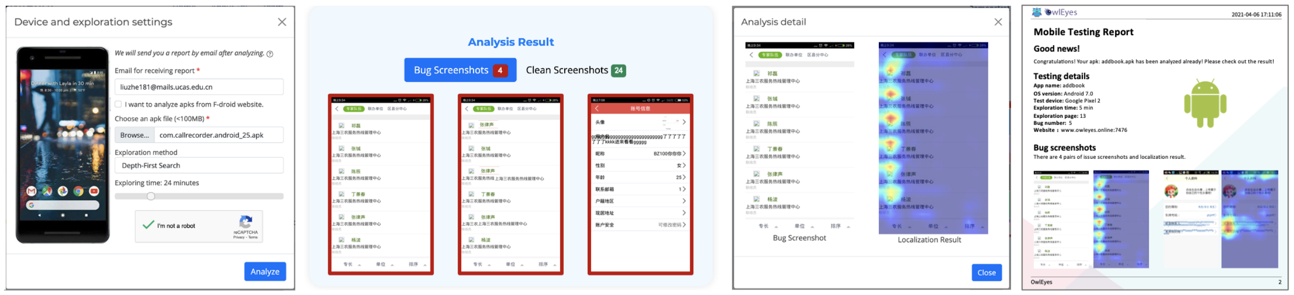
Training our proposed CNN for issues detection requires an abundance of screenshots [17] with UI display issues. However, there is so far no such type of open dataset, and collecting the related buggy screenshots is time- and effort-consuming. Therefore, we develop a heuristic based data generation method for generating UI screenshots with display issues from bug-free UI images in Figure 1(a). The data generation is based on the Rico [15] dataset, which contains more than 66K unique screenshots and their JSON files (i.e., detailed run-time view hierarchy of the screenshot). With the input screenshot and its associated JSON file, we first localize all the TextView and ImageView, then randomly chooses a TextView/ImageView depending on the augmented category. Based on the coordinates and size of the TextView/ImageView, the algorithm then makes its copy and adjusts its location or size according to specific rules to generate the screenshots with corresponding UI display issues.

2.2 CNN Based Issues Detection

As the UI display issues can only be spotted via the visual information [14, 39], we adopt the convolutional neural network (CNN) [20, 21], which has proven to be effective in image classification and recognition in computer vision [17, 29, 31]. Figure 1(b) shows the structure of our model, which links the convolutional layers, batch normalization layers, pooling layers, and fully connected layers. Given the input screenshot, we convert it into a certain image size with fixed width and height as the convolutional layer's parameters consist of a set of learnable filters. After the convolutional layers, the screenshots will be abstracted as a feature graph. In order to improve the performance and stability of CNN, we add Batch Normalization (BN) [19] layers after the convolutional layer and standardize the input layer by adjusting and scaling activation. After the BN layer, we add the Rectified Linear Unit (ReLU) as the activation function of the network. The last several layers are fully connected neural networks (FC) which compile the data extracted by previous layers to form the final output. Finally, we obtain the detection results through softmax [6].

2.3 Grad CAM Based Issues Localization

As shown in Figure 1(c), we adopt the feature visualization method to localize the detailed position of the issues to remind the developers. We apply the Grad CAM model for the localization of UI display issues. Gradient weighted Class Activation Mapping (Grad CAM) is a technique for visualizing the regions of input that are "important" for predictions on CNN based models [26] . First, a screenshot with the UI display issue is input into the trained CNN model, and the category supervisor to which the image belongs is set to 1, while the rest is 0. Then the information is propagated back to the convolutional feature map of interest to obtain the Grad CAM positioning. Through the feedback of global average pooling of the gradient, the weight α_k^b of the importance of neurons is obtained. This weight captures the importance of the feature map K of the target category b (Bug). By performing the weighted combination of the forward activation graph, we can obtain the class-discriminative localization map. Finally, the point multiplication with the backpropagation can obtain the Grad CAM as the result of issue localization.


Device and exploration settings

We will send you a report by email after analyzing.

Email for receiving report *: liuzhe181@mails.ucas.edu.cn

I want to analyze apks from F-droid website.

Choose an apk file (<100MB) *: Browse... com.callrecorder.android_25.apk

Exploration method: Depth-First Search

Exploring time: 24 minutes

I'm not a robot

Analyze

Analysis Result

Bug Screenshots 4 · Clean Screenshots 24

Analysis detail

Bug Screenshot · Localization Result

Close

OwlEyes 2021-04-06 17:11:06

Mobile Testing Report

Good news!

Congratulations! Your apk: addbook.apk has been analyzed already! Please check out the result!

Testing details

App name: addbook
OS version: Android 7.0
Test device: Google Pixel 2
Exploration time: 5 min
Exploration page: 13
Bug number: 5
Website: www.owleyes.online:7476

Bug screenshots

There are 4 pairs of issue screenshots and localization result.

(a) Running the app automatically (b) The Real-time feedback issue detection result (c) The the localization details (d) The issue reports details

Figure 2: Illustration of our `OwlEyes-Online` web application.

2.4 Online Inference of GUI Issues

We use 20,000 screenshots generated in section 2.1 to train our issue detection and localization model. Before the issue detection, we need to preprocess the APK submitted by the user online. As shown in Figure 1(d), the user provides an Android APK, and we use the dynamic analysis method to run the app automatically to obtain the screenshots. In detail, by leveraging the idea of dynamic app GUI testing [8, 16, 22, 30], we adopt an app explorer [22] to automatically explore the pages within an application through interacting with apps using random actions, e.g., clicking, scrolling, and filling in text. We also provide three testing strategies for users to choose from: Depth-First-Search (DFS) [27], Breadth-First-Search (BFS) [5], and random exploration [16].

3 TOOL IMPLEMENTATION AND USAGE

`OwlEyes-Online` is a web app, which provides a convenient tool for users to detect and localize the UI display issues in the GUI screenshots.

3.1 Web Implementation

`OwlEyes-Online` can automatically run applications and generate test reports for users. We customized the deep learning model in PyTorch. The `OwlEyes-Online` consists of two parts: running the application automatically, feeding back the test results in real-time.

Running the app automatically: Figure 2(a) shows an example of our running the app automatically page. Users can upload the APK or its download link. In addition, we allow users to customize the exploration strategy, select the appropriate device, and some personalization settings to provide a more friendly interactive experience.

The Real-time feedback issue detection results: This page in Figure 2(b) will give real-time feedback test results while running the application automatically. On this page, we implement some functions to provide a more friendly interactive experience, including:

Click to view the localization details: In Figure 2(c), click the screenshot of the UI display issue to view the localization of it in the screenshot (in the form of a thermal graph).

Export test report: In Figure 2(d), users fill in e-mail information, and we will automatically generate test reports and send them to users. The test report includes the number of issues of the application and the screenshots of the issue and the XML corresponding to the screenshots.

3.2 Model Implementation

Our CNN model is composed of 12 convolutional layers with batch normalization, 6 pooling layers, and 4 full connection layers for classifying UI screenshot with display issues. The size of a convolutional kernel in the convolutional layer is 3×3. We set up the number of convolutional kernels as 16 for convolutional layer 1-4, 32 for convolutional layer 5-6, 64 for convolutional layer 7-8, and 128 for convolutional layer 9-12. For the pooling layers, we use the most common-used max-pooling settings [28], i.e., pooling units of size 2×2 applied with a stride [29]. We set the number of neurons in each of the fully connected layers as 4096, 1024, 128, and 2 respectively. For data preprocessing, we rotate some UI of the horizontal screens to vertical, and resize the screens to 768×448. We implement our model based on the PyTorch [1] framework. The model is trained in an NVIDIA GeForce RTX 2060 GPU (16G memory) with 100 epochs for about 8 hours.

3.3 Usage Scenarios

We present several examples to illustrate how developers would interact with `OwlEyes-Online`. In some cases, developers collect a large number of screenshots of applications (such as crowdtesting platform, automatic testing). However, these automated tools can only spot critical crash bugs, rather than UI display issues that cannot be captured by common tools. Developers can upload application screenshots to our `OwlEyes-Online` directly. `OwlEyes-Online` will analyze the screenshots and detect the UI display issue in the screenshots.

For testing whether UI display issues exist in the application, developers can directly upload an APK to our `OwlEyes-Online` , which will automatically explore the application and detect UI display issues. Developers can also customize the exploration method and duration and submit the e-mail information. After the issue detection, `OwlEyes-Online` will automatically generate the issue report and send it to the developer's e-mail. Considering the network delay, developers can also upload an application's download link, and `OwlEyes-Online` will automatically download the APK in the background for testing.

4 EVALUATION

The goal of our study is to evaluate the usefulness of our platform `OwlEyes-Online` in terms of (i) its effectiveness in detecting and localizing UI display issues, and (ii) the usability of our `OwlEyes-Online`.

4.1 Effectiveness Measurement

Given the effectiveness of our OwlEyes-Online for UI display issue detection, we conduct experiments on 8K Android mobile GUI collected by one of the largest crowd-testing platforms [?]. This part is also published in our previous work [40] and and we mainly use evaluation metrics of precision and recall.

Table 1 shows the performance comparison with the baselines. With OwlEyes-Online, the precision is 0.85 and the recall is 0.84. We can see that our proposed OwlEyes-Online is much better than the baselines, i.e., 58% higher in precision and 17% higher in recall compared with the best baseline, Multilayer Perceptron (MLP). This further indicates the effectiveness of OwlEyes-Online. Besides, it also implies that OwlEyes-Online is especially good at hunting for the buggy screenshots from candidate ones, i.e., significant improvement in recall.

Table 1: Performance comparison with baselines

Method	Precision	Recall	F1-score
RF-SIFT	0.458	0.458	0.432
RF-SURF	0.513	0.524	0.519
RF-ORB	0.520	0.528	0.524
MLP	0.537	0.727	0.618
OwlEyes-Online	**0.850**	**0.848**	**0.849**

4.2 Usefulness Measurement

To further assess the usefulness of our approach, we randomly sample 2,000 Android applications from F-Droid [2] and 1,000 applications from Google Play [3]. Note that none of these apps appears in our training dataset. Among the 3,000 collected applications, 59% (1756/3000) applications can be successfully run with OwlEyes-Online. For the remaining 1,756 applications, an average of 8 screenshots is obtained for each application. We then feed those screenshots to our OwlEyes-Online and detect if there are any display issues. Once a display issue is spotted, we create a bug report by describing the issue attached with a buggy UI screenshot. Finally, we report them to the app development team through issue reports or emails. Our OwlEyes-Online has detected 113 UI display issues, among which 35 have been confirmed and 29 have been fixed. These fixed or confirmed bug reports further demonstrate the effectiveness and usefulness of our proposed approach in detecting UI display issues.

Regarding the user experience of our OwlEyes-Online, we create an online survey on 20 professional developers, testers, and researchers, all of whom major in computer science with more than 3 years of app testing or developing experience. 10 of them are from the industry with practical working experience[2]. We ask them to use our OwlEyes-Online and ask them about the usefulness of the OwlEyes-Online for their work, as well as its potential and scalability in the future. In the end, participants fill in the System Usability Scale (SUS) questionnaire [7] (5-point Likert scale [25] from 1 (strongly disagree) to 5 (strongly agree)). The questionnaire also asks participants to select the TechLand system features that they deem most useful or least useful for the tasks.

Figure 3 summarizes the participants' ratings of the 10 system design and usability questions in the System Usability Scale questionnaire. The upper half of figure 3 shows that participants agree

[2]Some testers are from NVIDIA, Citibank, Sony, Baidu, Alibaba, Three Fast Online, and ByteDance.

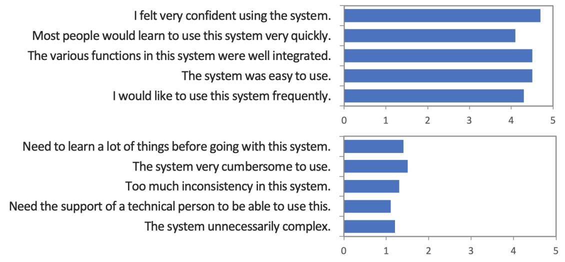


Figure 3: Average score of SUS results

or strongly agree that our system is easy to use and the features of the OwlEyes-Online system are well-devised. The lower half of figure 3 further confirms the simplicity and consistency of our OwlEyes-Online system. Furthermore, the average helpfulness of the OwlEyes-Online system for the tasks is 4.42, which indicates that participants appreciate the help of the OwlEyes-Online system in the tasks. All participants indicated that OwlEyes-Online has a good UI display issue detection effect. Among these professionals, 10 of them are working on app testing. They think OwlEyes-Online can help them localize the UI display issues more quickly. 7 Android developers said that our issue localization model helps them better localize the issue on the UI interface so that they can better repair the issue later. Among them, 4 developers hope we can further give the possible repair methods and causes of these issues. The other 3 participants who are studying GUI testing also indicated that they hope we can analyze the cause of issue in the next stage. They think that using the visual information of application screenshots is a very helpful and engaging work.

5 CONCLUSION

Improving the quality of mobile applications, especially in a proactive way, is of great value and always encouraged. In this demo, we show OwlEyes-Online, a fully automated UI display issue detection and localization tool. We use dynamic analysis to explore the application automatically and get its screenshots. And users can customize the exploration time, exploration strategy and so on. Then we can complete the detection and localization of UI display issues based on CNN and Grad CAM. Finally, we automatically generate test reports and send them to users. We evaluate it from two aspects of detection accuracy and tool practicability. The OwlEyes-Online is proven to be effective in real-world practice, i.e., 64 confirmed or fixed previously undetected UI display issues from popular Android apps. It also achieves boosts of more than 17% and 23% in recall and precision compared with the best baseline. The evaluation shows that OwlEyes-Online is a good starting point for UI display issue detection.

In the future, we will further study the root cause of UI display issue. Finally, according to the issue category, we will devise a set of tools for recommending patches to developers to fix the UI display issues.

ACKNOWLEDGMENTS

This work is supported by the National Key Research and Development Program of China under grant No.2018YFB1403400, National Natural Science Foundation of China under Grant No. 62072442, No. 62002348.

REFERENCES

[1] 2021. https://pytorch.org/.
[2] 2021. http://f-droid.org/.
[3] 2021. http://play.google.com/store/apps/.
[4] 2021. Github Link. https://github.com/franklinbill/owleyes/.
[5] Scott Beamer, Krste Asanovic, and David Patterson. 2012. Direction-optimizing breadth-first search. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–10. https://doi.org/10.1109/SC.2012.50
[6] Christopher M Bishop. 2006. *Pattern recognition and machine learning*. springer.
[7] John Brooke et al. 1996. SUS-A quick and dirty usability scale. *Usability evaluation in industry* 189, 194 (1996), 4–7. https://doi.org/10.1201/9781498710411-35
[8] Tianqin Cai, Zhao Zhang, and Ping Yang. 2020. Fastbot: A Multi-Agent Model-Based Test Generation System Beijing Bytedance Network Technology Co., Ltd.. In *Proceedings of the IEEE/ACM 1st International Conference on Automation of Software Test*. 93–96. https://doi.org/10.1145/3387903.3389308
[9] Kaibo Cao, Chunyang Chen, Sebastian Baltes, Christoph Treude, and Xiang Chen. 2021. Automated Query Reformulation for Efficient Search based on Query Logs From Stack Overflow. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 1273–1285. https://doi.org/10.1109/ICSE43902.2021.00116
[10] Chunyang Chen, Sidong Feng, Zhengyang Liu, Zhenchang Xing, and Shengdong Zhao. 2020. From Lost to Found: Discover Missing UI Design Semantics through Recovering Missing Tags. *Proc. ACM Hum. Comput. Interact.* 4, CSCW2 (2020), 123:1–123:22. https://doi.org/10.1145/3415194
[11] Chunyang Chen, Zhenchang Xing, Yang Liu, and Kent Ong Long Xiong. 2021. Mining Likely Analogical APIs Across Third-Party Libraries via Large-Scale Unsupervised API Semantics Embedding. *IEEE Trans. Software Eng.* 47, 3 (2021), 432–447. https://doi.org/10.1109/TSE.2019.2896123
[12] Jieshan Chen, Chunyang Chen, Zhenchang Xing, Xin Xia, Liming Zhu, John C. Grundy, and Jinshui Wang. 2020. Wireframe-based UI Design Search through Image Autoencoder. *ACM Trans. Softw. Eng. Methodol.* 29, 3 (2020), 19:1–19:31. https://doi.org/10.1145/3391613
[13] Jieshan Chen, Chunyang Chen, Zhenchang Xing, Xiwei Xu, Liming Zhu, Guoqiang Li, and Jinshui Wang. 2020. Unblind your apps: predicting natural-language labels for mobile GUI components by deep learning. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 322–334. https://doi.org/10.1145/3377811.3380327
[14] Qiuyuan Chen, Chunyang Chen, Safwat Hassan, Zhengchang Xing, Xin Xia, and Ahmed E. Hassan. 2021. How Should I Improve the UI of My App?: A Study of User Reviews of Popular Apps in the Google Play. *ACM Trans. Softw. Eng. Methodol.* 30, 3 (2021), 37:1–37:38. https://doi.org/10.1145/3447808
[15] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 2017. Rico: A Mobile App Dataset for Building Data-Driven Design Applications. In *Proceedings of the 30th Annual Symposium on User Interface Software and Technology (UIST '17)*. https://doi.org/10.1145/3126594.3126651
[16] Android Developers. 2012. Ui/application exerciser monkey.
[17] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 770–778. https://doi.org/10.1109/CVPR.2016.90
[18] Yujin Huang, Han Hu, and Chunyang Chen. 2021. Robustness of on-Device Models: Adversarial Attack to Deep Learning Models on Android Apps. In *43rd IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2021, Madrid, Spain, May 25-28, 2021*. IEEE, 101–110. https://doi.org/10.1109/ICSE-SEIP52600.2021.00019
[19] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015 (JMLR Workshop and Conference Proceedings, Vol. 37)*. 448–456.
[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105. https://doi.org/10.1145/3065386
[21] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324. https://doi.org/10.1109/5.726791
[22] Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen. 2017. Droidbot: a lightweight ui-guided test input generator for android. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 23–26. https://doi.org/10.1109/ICSE-C.2017.8
[23] Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen. 2017. DroidBot: A Lightweight UI-Guided Test Input Generator for Android. In *Proceedings of the 39th International Conference on Software Engineering Companion* (Buenos Aires, Argentina) *(ICSE-C '17)*. IEEE Press, 23–26. https://doi.org/10.1109/ICSE-C.2017.8
[24] Aravind Machiry, Rohan Tahiliani, and Mayur Naik. 2013. Dynodroid: An Input Generation System for Android Apps. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering* (Saint Petersburg, Russia) *(ESEC/FSE 2013)*. Association for Computing Machinery, New York, NY, USA, 224–234. https://doi.org/10.1145/2491411.2491450
[25] Yusuke Oda, Hiroyuki Fudaba, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. 2015. Learning to generate pseudo-code from source code using statistical machine translation (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 574–584. https://doi.org/10.1109/ASE.2015.36
[26] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. 2017. Grad-CAM: Visual Explanations From Deep Networks via Gradient-Based Localization. In *The IEEE International Conference on Computer Vision (ICCV)*. https://doi.org/10.1109/ICCV.2017.74
[27] Safaa H Shwail, Alia Karim, and Scott Turner. 2013. Probabilistic multi robot path planning in dynamic environments: A comparison between A* and DFS. *International Journal of Computer Applications* 975 (2013), 8887.
[28] Patrice Y. Simard, Dave Steinkraus, and John C. Platt. 2003. Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition - Volume 2 (ICDAR '03)*. IEEE Computer Society, USA, 958. https://doi.org/10.1109/ICDAR.2003.1227801
[29] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.).
[30] Ting Su, Guozhu Meng, Yuting Chen, Ke Wu, Weiming Yang, Yao Yao, Geguang Pu, Yang Liu, and Zhendong Su. 2017. Guided, stochastic model-based GUI testing of Android apps. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM, 245–256. https://doi.org/10.1145/3106237.3106298
[31] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2818–2826. https://doi.org/10.1109/CVPR.2016.308
[32] Junjie Wang, Song Wang, Jianfeng Chen, Tim Menzies, Qiang Cui, Miao Xie, and Qing Wang. 2021. Characterizing Crowds to Better Optimize Worker Recommendation in Crowdsourced Testing. *IEEE Trans. Software Eng.* 47, 6 (2021), 1259–1276. https://doi.org/10.1109/TSE.2019.2918520
[33] Junjie Wang, Ye Yang, Rahul Krishna, Tim Menzies, and Qing Wang. 2019. iSENSE: Completion-Aware Crowdtesting Management. In *ICSE'2019*. 932–943. https://doi.org/10.1109/ICSE.2019.00097
[34] Lili Wei, Yepang Liu, and Shing-Chi Cheung. 2016. Taming Android fragmentation: Characterizing and detecting compatibility issues for Android apps. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. 226–237. https://doi.org/10.1145/2970276.2970312
[35] Thomas Wetzlmaier and Rudolf Ramler. 2017. Hybrid Monkey Testing: Enhancing Automated GUI Tests with Random Test Generation. In *Proceedings of the 8th ACM SIGSOFT International Workshop on Automated Software Testing* (Paderborn, Germany) *(A-TEST 2017)*. Association for Computing Machinery, New York, NY, USA, 5–10. https://doi.org/10.1145/3121245.3121247
[36] Mulong Xie, Sidong Feng, Zhenchang Xing, Jieshan Chen, and Chunyang Chen. 2020. UIED: a hybrid tool for GUI element detection. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann (Eds.). ACM, 1655–1659. https://doi.org/10.1145/3368089.3417940
[37] Bo Yang, Zhenchang Xing, Xin Xia, Chunyang Chen, Deheng Ye, and Shanping Li. 2021. Don't Do That! Hunting Down Visual Design Smells in Complex UIs against Design Guidelines. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 761–772. https://doi.org/10.1109/ICSE43902.2021.00075
[38] Dehai Zhao, Zhenchang Xing, Chunyang Chen, Xiwei Xu, Liming Zhu, Guoqiang Li, and Jinshui Wang. 2020. Seenomaly: vision-based linting of GUI animation effects against design-don't guidelines. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 1286–1297. https://doi.org/10.1145/3377811.3380411
[39] Tianming Zhao, Chunyang Chen, Yuanning Liu, and Xiaodong Zhu. 2021. GUIGAN: Learning to Generate GUI Designs Using Generative Adversarial Networks. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 748–760. https://doi.org/10.1109/ICSE43902.2021.00074
[40] Liu Zhe, Chen Chunyang, Wang Junjie, Huang Yuekai, Hu Jun, and Wang Qing. 2020. Owl Eyes: Spotting UI Display Issues via Visual Understanding. In *2020 35rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE. https://doi.org/10.1145/3324884.3416547