Detection and Segmentation of Graphical Elements on GUIs for Mobile Apps Based on Deep Learning

Rui Hu, Mingang Chen(✉), Lizhi Cai, and Wenjie Chen

Shanghai Key Laboratory of Computer Software Testing and Evaluating, Shanghai Development Center of Computer Software Technology, Shanghai, China
cmg@sscenter.sh.cn

Abstract. Recently, mobile devices are more popular than computers. However, mobile apps are not as thoroughly tested as desktop ones, especially for graphical user interface (GUI). In this paper, we study the detection and segmentation of graphical elements on GUIs for mobile apps based on deep learning. It is the preliminary work of GUI testing for mobile apps based on artificial intelligence. We create a dataset, which consists of 2,100 GUI screenshots (or pages) labeled with 42,156 graphic elements in 8 classes. Based on our dataset, we adopt Mask R-CNN to train the detection and segmentation of graphic elements on GUI screenshots. The experimental results show that the mAP value achieves 98%.

Keywords: Mobile apps · GUI dataset · Instance segmentation · Mask R-CNN

1 Introduction

Nowadays, mobile apps are becoming indispensable in our daily life and work. As there are a large number of apps available for users in online stores. It is significant for a competitive app to run smoothly as users expect. Mobile apps that often crash or bug-prone are likely to be quickly abandoned [1] and negatively evaluated [2] by users. Automation testing is one of the most efficient and reliable solutions to ensure the quality, avoiding high cost and low accuracy of manual testing. The GUI is the medium for most mobile apps to interact with users. This makes GUI testing an essential and prominent part of automation testing [3]. Obviously, the importance of GUI testing for mobile apps is self-evident. In this paper, the detection and segmentation technology of graphical elements on GUIs for mobile apps is developed on the basis of deep learning, which is the preliminary work of artificial intelligence-based GUI testing for mobile apps.

The traditional methods of GUI testing usually require testers to manually write a large number of test scripts. These tasks involve a lot of mechanical and repetitive work, and are time-consuming and laborious. Meanwhile, even small changes in GUI can destroy the entire test suite [4] and make the original test script invalid. Due to the

J. Liu et al. (Eds.): MobiCASE 2020, LNICST 341, pp. 187–197, 2020.
https://doi.org/10.1007/978-3-030-64214-3_13

large number of interactions on current GUIs and the increasing complexity of mobile apps, it is practically impossible to automate test generation with sufficient coverage. Muccini et al. [5] emphasized the main challenges of GUI testing for mobile apps: the large number of contextual events and responses, the diversity of devices and screen resolution, and the rapid updating of operating systems.

Today, with the improvement of computing performance, deep learning has made great progress in image detection. Based on Region convolution neural network (R-CNN) [6], the detection technology of graphical elements on GUIs is an end-to-end method. In which, the inputs of the network are the graphic elements on GUI images and their corresponding labels, and the outputs are the results of classification and localization.

In this paper, we apply the deep learning-based detection and segmentation technology to GUI pages. In traditional GUI testing, computers usually cannot recognize graphic elements. We humans can interact with GUIs according to their characteristics. We know that clicking the "Pay" button means paying, dragging the slider means controlling video progress, and sliding the screen means switching app pages. In our approach, the computer can intelligently identify the classification and localization of graphic elements on GUI pages just like human beings and operate the graphic elements without test scripts. To our knowledge, no one has yet applied the deep learning-based detection and segmentation algorithm to the GUI, which means that our work has great innovation.

The main contributions of this paper are summarized as follows:

(1) We create a dataset for detection and segmentation of graphic elements on GUI pages. 2,100 GUI screenshots in our dataset come from the Rico dataset [7] (a large-scale repository of GUI screenshots for mobile apps), Google Play and HUAWEI AppGallery. 42,156 graphic elements are manually labeled from GUI screenshots and classified into 8 classes depending on their types, appearances and operations imposed.
(2) The Mask R-CNN detection and segmentation algorithm is adopted to classify, locate and segment graphic elements on GUIs. Mask R-CNN is a multitask algorithm, which can be applied to train for classification, localization and masking together instead of training in stages. The experimental results show that the mean average precision (mAP) of detection and segmentation for graphic elements on GUI pages achieves 98%, which fully complies with the requirements of artificial intelligence-based GUI testing.

Section 2 gives a survey on the related work containing traditional GUI testing and object detection and segmentation algorithms. Section 3 presents two procedures of our work: the creation and manual labeling of our dataset, and the training and inferring of Mask R-CNN based on ResNet-50 network. Section 4 discusses experimental results and evaluation. Section 5 draws conclusions and prospects the future works.

2 Related Work

The significance of GUI testing for mobile apps is becoming self-evident due to the increasing popularity of mobile devices. Today, there are many approaches to test GUI

for mobile apps. In [8], these techniques can be roughly divided into three categories: random-based testing [9, 10], model-based testing [11–13] and systematic testing [14, 15]. The random-based testing inputs random activity sequences to discover potential defects and bugs. For example, Monkey [9] sends random activities to random locations on the screen without considering the GUI structure. However, it is not suitable for large apps due to the large number of random behavior sequences that need to be processed in memory. Dynodroid [10], which is smarter than Monkey, maintains the input events through a context-sensitive approach. The model-based testing technology [16] builds detailed models of the relationship between events and GUI screens, and then generates test cases automatically according to these models. Based on the established model, MobiGUITAR [11] performs possible events by changing the state of the GUI. The systematic testing technique attempts to input specific test inputs according to pre-determined targets, such as exploring width or depth. A^3E [14] approach can explore apps systematically with Targeted Exploration technique and Depth-first Exploration technique.

Currently, mainstream object detection algorithms include Fast R-CNN [17], Faster R-CNN [18], YOLO [19], SSD [20] and YOLO9000 [21]. Object detection has enormous significance in many fields such as face detection [22] and video surveillance [23]. For a given image, Faster R-CNN returns the class label and bounding box coordinate of each object in the image. Mask R-CNN [24] is an extended instance segmentation algorithm of Faster R-CNN. Therefore, for the given image, Mask R-CNN will return segmentation masks of the object in addition to the class labels and bounding box coordinates. For Faster R-CNN, it uses VGG-16 [25] network to extract the feature map (FM) from the image. The region proposal network (RPN) [18] transmits these FMs and return candidate bounding boxes. Then the merging layer of region of interest (ROI) will be applied to these candidate bounding boxes to make all candidates have the same size.

Similar to the VGG-16 network in Faster R-CNN, from the images, Mask R-CNN uses the ResNet-101 [26] architecture to extract feature maps (FMs), which are regarded as inputs to next layer. Then, these FMs will be implemented to the region proposal network (RPN) to predict whether there are objects in the region. In this step, regions or FMs containing some objects can be predicted and obtained. The regions obtained from RPN may have different shapes, so the pooling layer will be applied to convert all regions to the same shape. These regions will be input to a fully connected layer to predict class labels and bounding boxes. Next, Mask R-CNN will generate segmentation masks [17]. For all predicted regions, the intersection over union (IOU) with the bounding boxes of real data can be calculated as follows:

$$\text{IOU} = \frac{\text{bounding box of candidate regin} \cap \text{bounding box of real data}}{\text{bounding box of candidate regin} \cup \text{bounding box of real data}}. \quad (1)$$

With the ROI based on the IOU value, the mask branch can be added to the existing architecture, which will return the segmentation mask for each region that contains the object.

3 Methodology

3.1 Dataset Creation

2,100 GUI screenshots in our dataset are selected from the Rico dataset, Google Play and HUAWEI AppGallery. The Rico [7] is a large crowd-sourced GUI interaction dataset which contains GUI screenshots and human interactions. Google Play and HUAWEI AppGallery are two online app stores for Android devices. Among our dataset, 1,600 GUI screenshots are provided from Rico dataset, 300 are provided from Google Play and 200 are provided from HUAWEI AppGallery.

Because the label and interaction data in Rico are not designed for our purpose, we first need to reclassify the graphic elements. Rico divides UI components into 25 classes. However, some components, such as Advertisement class, are sub-standard for our requirements. According to the types, appearances and actions applied of graphic elements, we integrate and divide components in Rico into 8 classes: Checkbox, Icon, Input, On/Off Switch, Page Indicator, Radio Button, Slider and Text Button.

Next, we manually label graphic elements on these 2,100 GUI screenshots. We label a total of 42,156 graphic elements and the label information includes their pixel-level coordinates, bounding boxes and classes. Figure 1 shows these 8 classes of graphic elements. Table 1 shows the quantity distribution of 8 graphic element classes. The number of icon and text button is huge since to their complexity and wide distribution.

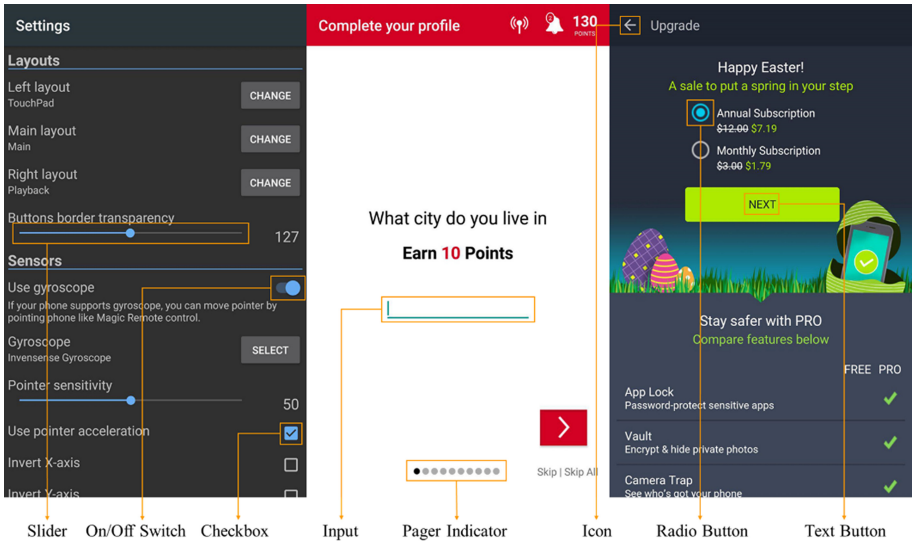


Fig. 1. Graphic elements are divided into 8 classes: Slider, On/Off switch, Checkbox, Input, Pager indicator, Icon, Radio button and Text button.

Table 1. Quantity distribution of 8 graphic element classes in our dataset.

Classification	Number
Checkbox	1,146
Icon	24,804
Input	1,912
On/off switch	1,566
Page indicator	1,032
Radio button	1,726
Slider	1,740
Text button	8,230

3.2 Mask R-CNN for Detection and Segmentation

Previously, the recognition of GUI graphic elements is based on source code. With Mask R-CNN, graphic elements on GUI pages can be identified intelligently by computers without source code.

Input. Our dataset is randomly divided into training (1,800 pages), validation (200 pages) and testing (100 pages) sets. Before training, the training and validation set should be preprocessed. GUI screenshots are converted to grayscale images to eliminate the influence of color on the convolutional neural network and speed up training. Then, the images have one channel and their size is 1440×2560.

Network Configuration. The ResNet-101 structure in Mask R-CNN is too deep to be quickly trained and deployed. Therefore, for our dataset, ResNet-101 is replaced by an agile ResNet-50 network. After considering the number of filtered ROI and the bounding box correlation between candidate region and real data, we choose the threshold value of IOU as 0.7. Due to the limitation of GPU memory, we set the batch size as 4. The momentum constant is set to 0.9 to accelerate the training. The learning rate is set to 0.0005 to prevent over fitting.

Training. Our Mask R-CNN network architecture is shown in Fig. 2. The labeled images are entered into ResNet-50 network to obtain the corresponding FMs. Each pixel in FMs has an anchor to obtain multiple candidate ROIs. These candidate ROIs are sent to RPN for binary classification (graphic elements or background) and candidate bounding box regression. Then the candidate ROIs will be filtered by IOU value. The predicted region is considered as ROI only when IOU value is greater than or equal to 0.7, otherwise it will be ignored. The process above is performed on all candidate regions. The remaining ROIs are operated by ROIAlign. Finally, the multitask loss L [17, 24] is applied on these ROIs for classification (8 classes of graphic elements and the background), bounding box regression and segmentation mask generation:

$$L = L_{cls} + L_{box} + L_{mask}. \tag{2}$$

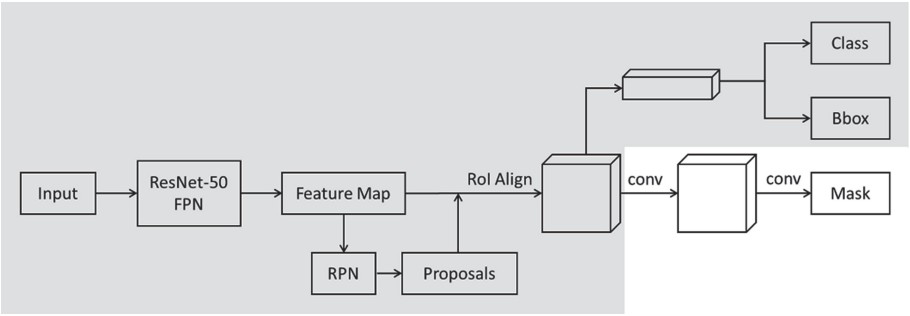


Fig. 2. Mask R-CNN architecture based on ResNet-50, in which the shadow part is Faster R-CNN architecture.

The classification loss L_{cls} is cross-entropy loss for object. The bounding box regression loss L_{box} is

$$L_{box}\left(t_i, t_i^*\right) = \sum\nolimits_{i\in\{x,y,w,h\}} smooth_{L_1}\left(t_i - t_i^*\right), \tag{3}$$

in which

$$smooth_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & otherwise \end{cases}, \tag{4}$$

where t_i is predicted bounding box regression offsets and t_i^* is true offsets, $\{x, y, w, h\}$ is the center coordinate, width and height of the box. Based on the using of a per-pixel sigmoid, the mask loss L_{mask} is defined as average binary cross-entropy loss.

4 Experimental Results

4.1 Training Environment

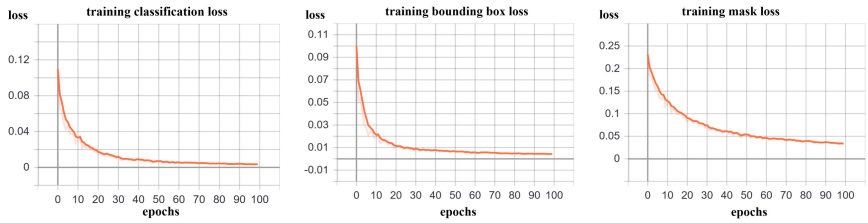
The machine we used to train and test the Mask R-CNN model is a server with an NVidia Tesla P100 PCI-E GPU. The operating system of the machine is Ubuntu 16.04. The model is implemented with TensorFlow.

Since training all the networks will spend too much time and our dataset is small, we used a pre-trained model that was formed on the COCO dataset based on transfer learning. A total of 100 epochs were trained in the experiment, which took three days.

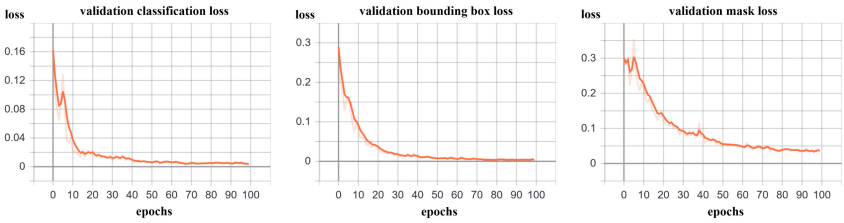
4.2 Evaluation

The used evaluation metric is the mean average precision (mAP), which is the average precision (AP) of all classes. The formula of mAP is as follows:

$$\text{mAP} = \frac{1}{|\text{Q}_\text{R}|} \sum\nolimits_{\text{q}\in\text{Q}_\text{R}} \text{AP(q)}. \tag{5}$$



a. Training loss



b. Validation loss

Fig. 3. The consequences of loss in training and validation processes, including classification loss, bounding box regression loss and mask loss.

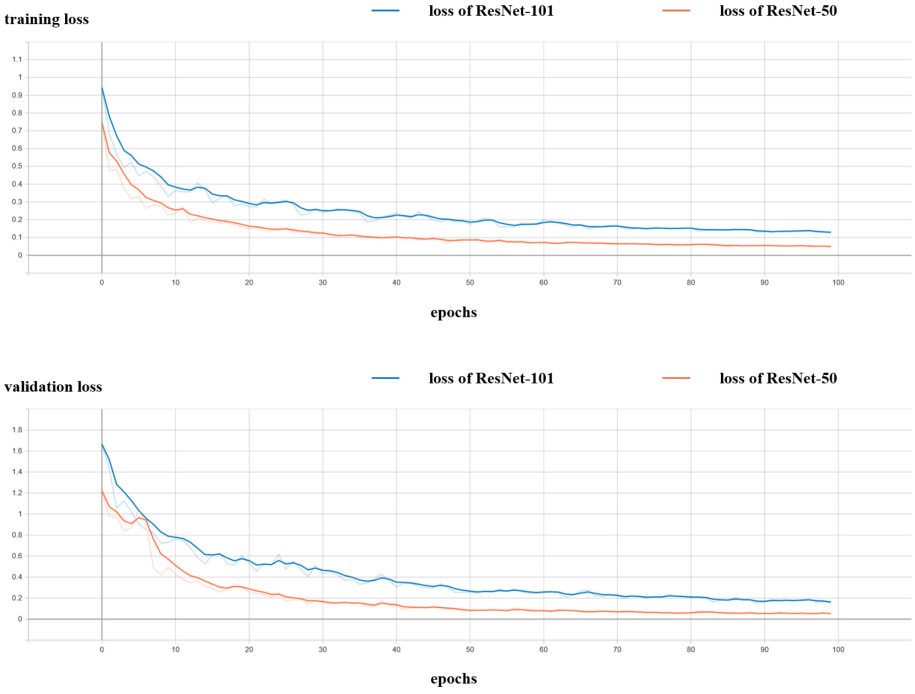


Fig. 4. The training and validation overall loss comparison of Mask R-CNN trained by ResNet-50 and ResNet-101.

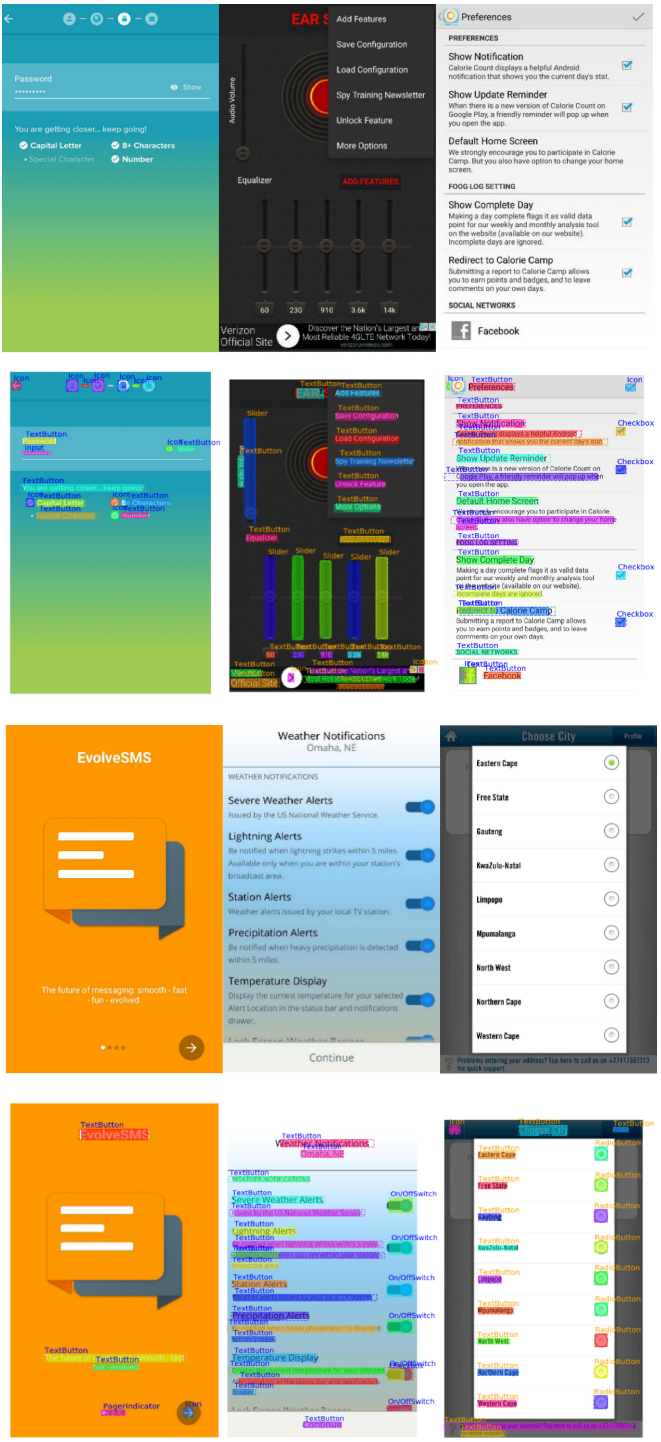


Fig. 5. Detection and segmentation results for some images. The label classes, bounding boxes and segmentation masks for graphic elements are shown.

Where AP(q) is the average precision of each class and Q_R is the number of classifications. AP value is the area under the curve of precision and recall. For testing, the experimental results demonstrate that the mAP value of ResNet-50 is 98%. The training and validation loss are shown in Fig. 3. In addition, on the basis of using our dataset, we compare two backbones networks of Mask R-CNN, ResNet-50 and ResNet-101. The overall loss is the sum of classification loss, bounding box loss and mask loss. ResNet-50 achieves 0.047 overall loss on training dataset and 0.049 on validation dataset. While the overall loss of ResNet-101 on training dataset is 0.13 and that on validation dataset is 0.15. Figure 4 shows the comparison of overall loss trained by ResNet-50 and ResNet-101.

Figure 5 shows the results of some images that contain the label classes, bounding boxes and segmentation masks of graphic elements. The results show that the Mask R-CNN based on ResNet-50 can brilliantly recognize graphic elements on GUIs.

5 Conclusion

In this paper, we create a dataset consists of 2,100 GUI screenshots for detection and segmentation tasks of graphical elements on GUIs. 1,600 screenshots are selected from Rico dataset, 300 are selected from Google Play and 200 are selected from HUAWEI AppGallery. The GUI screenshots are manually labeled a total of 42,156 graphic elements, which are divided into 8 classes according to their types, appearances and actions applied. In addition, Mask R-CNN based on ResNet-50 is applied to the detection and segmentation of graphic elements on GUIs. The mAP value achieves 98%, showing the brilliant application effect. As for future work, on the basis of this paper, we will study GUI testing based on artificial intelligence. We will also improve our method to support more complex events such as context events and capturing GUI bugs.

Acknowledgement. This work is funded by National Key R&D Program of China (No. 2018YFB1403400), and Science and Technology Commission of Shanghai Municipality Program, China. (Nos. 17411952800, 18DZ2203700, 18DZ1113400).

References

1. Moran, K., Linares-Vasquez, M., Bernal-Cardenas, C.: Automatically discovering, reporting and reproducing android application crashes. In: ICST, pp. 33–44. IEEE Computer Society, Los Alamitos (2016)
2. Khalid, H., Shihab, E., Nagappan, M.: What do mobile app users complain about? IEEE Softw. **32**(3), 70–77 (2015)
3. Kaur, A.: Review of mobile applications testing with automated techniques. Int. J. Adv. Res. Comput. Commun. Eng. **4**(10), 503–507 (2015)
4. Coppola, R., Raffero, E., Torchiano, M.: Automated mobile UI test fragility: an exploratory assessment study on Android. In: INTUITEST 2016: Proceedings of the 2nd International Workshop on User Interface Test Automation, pp. 11–20. ACM, New York (2016)
5. Muccini, H., Francesco, A.D., Esposito, P.: Software testing of mobile applications: challenges and future research directions. In: International Workshop on Automation of Software Test (AST), pp. 29–35. IEEE Computer Society, Los Alamitos (2012)

6. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: CVPR, pp. 580–587. IEEE Computer Society, Los Alamitos (2014)
7. Deka, B., Huang, Z., Franzen, C.: Rico: a mobile app dataset for building data-driven design applications. In: UIST '17: Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology, pp. 845–854. ACM, New York (2017)
8. Choudhary, S., Gorla, A., Orso, A.: Automated test input generation for android: are we there yet? In: 30th IEEE/ACM International Conference on Automated Software Engineering, pp. 429–440. IEEE Computer Society, Los Alamitos (2015)
9. UI/Application Exerciser Monkey. https://developer.android.com/studio/test/monkey.html Accessed 27 July 2020
10. Machiry, A., Tahiliani, R., Naik, M.: Dynodroid: an input generation system for android apps. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, pp. 224–234. ACM, New York (2013)
11. Amalfitano, D., Fasolino, A., Tramontana, P., Ta, B., Memon, A.: MobiGUITAR: automated model-based testing of mobile apps. IEEE Softw. **32**(5), 53–59 (2015)
12. Yang, W., Prasad, M.R., Xie, T.: A grey-box approach for automated GUI-model generation of mobile applications. In: Cortellessa, V., Varró, D. (eds.) FASE 2013. LNCS, vol. 7793, pp. 250–265. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37057-1_19
13. Amalfitano, D., Fasolino, A., Tramontana, P., De Carmine, S., Memon, A.: Using GUI ripping for automated testing of android applications. In: Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, pp. 258–261. ACM, New York (2012)
14. Azim, T., Neamtiu, I.: Targeted and depth-first exploration for systematic testing of android apps. In: Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications, pp. 641–660. ACM, New York (2013)
15. Bhoraskar, R., Han, S., Jeon, J.: Brahmastra: driving apps to test the security of third-party components. In: Proceedings of the 23rd USENIX Conference on Security Symposium, pp. 1021–1036. USENIX Association, San Diego (2014)
16. Pretschner, A., Prenninger, W., Wagner, S.: One evaluation of model-based testing and its automation. In: Proceedings of the 27th International Conference on Software Engineering, pp. 392–401. ACM, New York (2015)
17. Girshick, R.: Fast R-CNN. In: ICCV, pp. 1440–1448. IEEE Computer Society, Los Alamitos (2015)
18. Ren, S.Q., He, K.M., Girshick, R.B.: Faster R-CNN: towards real-time object detection with region proposal networks. In: NIPS, pp. 91–99. MIT Press, Cambridge (2015)
19. Redmon, J., Divvala, S., Girshick, R.: You only look once: unified, real-time object detection. In: CVPR, pp. 779–788. IEEE Computer Society, Los Alamitos (2016)
20. Liu, W., et al.: SSD: single shot multibox detector. In: Xie, T., Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9905, pp. 21–37. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46448-0_2
21. Redmon, J., Farhadi, A.: YOLO9000: better, faster, stronger. In: CVPR, pp. 6517–6525. IEEE Computer Society, Los Alamitos (2017)
22. Wei, Z.Y., Wen, C., Xie, K.: Real-time face detection for mobile devices with optical flow estimation. J. Comput. Appl. **38**(4), 1146–1150 (2018)
23. Li, J.W., Zhou, X.L., Chan, S.X.: A novel video target tracking method based on adaptive convolutional neural network feature. J. Comput. Aided Des. Comput. Graph. **30**(2), 273–281 (2018)
24. He, K.M., Gkioxari, G., Dollar, P.: Mask R-CNN. In: ICCV, pp. 2980–2988. IEEE Computer Society, Los Alamitos (2017)

25. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv.org, https://arxiv.org/abs/1409.1556 Accessed 3 July 2020
26. He, K.M., Zhang, X.Y., Ren, S.Q.: Deep residual learning for image recognition. In: CVPR, pp. 770–778. IEEE Computer Society, Los Alamitos (2016)