GUIS2Code: A Computer Vision Tool to Generate Code Automatically from Graphical User Interface Sketches

Zhen Feng, Jiaqi Fang, Bo Cai(✉), and Yingtao Zhang

Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, School of Cyber Science and Engineering, Wuhan University, Wuhan, China
{zhenfeng,fangjiaqi,caib,zhangyingtao}@whu.edu.cn

Abstract. It is a typical task for front-end developers to repetitively transform the graphical user interface model provided by the designer into code. Automatically converting the design draft provided by the designer into code can simplify the task of the front-end engineer and avoid a lot of simple and repetitive work. In this paper, we propose *GUIS2Code* using deep neural network, which is trained on the datasets of the design drafts to detect the UI elements of the input sketches and generate corresponding codes through the UI parser. Our method can generate code for three different platforms (i.e., iOS, Android, and Web). Our experimental results illustrates that *GUIS2Code* achieves an average GUI-component classification accuracy of 95.04% and generates code that can restore the target sketches more accurately while exhibiting reasonable code structure.

Keywords: Graphical user interface · Deep learning · Code generation

1 Introduction

Nowadays, almost all applications are user-oriented with a graphical user interface (GUI), relying on a simple user interface (UI) and intuitive user experience to attract customers [9]. The process of front-end developers implementing the UI based on the graphic user interface design sketches created by the designer is very time-consuming, which will reduce the time they use to implement the actual functions and logic of the software they build [12]. In addition, when the software to be built needs to run on multiple platforms, it will bring a lot of repetitive work. Therefore, the automated conversion of UI design drafts to executable code will greatly improve the efficiency of developers.

Using deep learning to automatically generate code from UI design sketches is a relatively new research field. The key issue is how the machine understands the design sketches and extracts logical information from it, which can be regarded as a computer vision problem. There are some methods that use CNN to extract the visual features of the entire image, such as pix2code [1], HGui2Code [14], etc.,

I. Farkaš et al. (Eds.): ICANN 2021, LNCS 12893, pp. 53–65, 2021.
https://doi.org/10.1007/978-3-030-86365-4_5

which have achieved good results, but they all rely on domain-specific languages (DSL) and are less flexible. The object detection method of real images has always been a key research area of computer vision. Part of our task can be regarded as a object detection task in a specific scene, but it is unknown whether the general object detection approach is suitable for the design drafts we deal with. So, we explored various object detection models, and studied whether their structures and methods are useful for our research.

Finally, in this paper, we proposed *GUIS2Code*, a tool for automatic GUI code generation based on object detection, which can generate corresponding codes only by taking screenshots of UI design drafts as input. We divide the whole task into two steps, the first step is to use object detection methods to classify UI elements into various types (such as buttons, pictures, etc.) and represent them as specific objects; the second step is to generate code for different platforms through the UI parser.

The contribution of work is summarized as follows:

- We develop a deep learning based generative tool: *GUIS2Code* for overcoming the barrier for translating UI images to code.
- Other methods such as pix2code train the entire design draft, but our approach trains each element, so it has higher UI detection accuracy than them and also does not require DSL. Our generative tool combines object detection and text recognition method for learning a crowd-scale knowledge of UI images and component position information from a large number of mobile apps or rendered websites.
- We show our model's robust visual understanding and code generation capability through experiments. Compared with other methods, our model has better performance in terms of accuracy and visual understanding.

2 Related Work

2.1 Object Detection and Text Recognition

The first step of this task actually similar to the object detection problem of computer vision. Object detection can generally be divided into two categories. The first category is a two-stages recognition method represented by Fast R-CNN [6]. The first stage of this structure focuses on proposal extraction, and the second stage performs classification and precise coordinate regression on the extracted proposals. The accuracy of the two-stages structure is higher, but because the second stage needs to classify each proposal separately, the speed is compromised. The second type of structure is a one-stage structure represented by YOLO [15] and SSD [11]. They abandon the process of extracting the proposal and complete the recognition with only one stage. Although the speed is faster, the accuracy rate is far behind two-stages structure. In this paper, we studied various models and whether their structures and methods are useful for our task.

The text recognition process based on deep learning mainly includes text region detection and text sequence recognition. There are already many mature

text recognition networks such as CRNN [16], RARE [17], ESIR [20], etc. In our task, the text is in the component, so the text region detection step can be omitted, and the result of the object detection network can be directly used as the input of the text recognition network. And the recognition network we need does not require a complex structure, so DenseNet [8] is used as a text recognition network in our model to recognize the text of each component.

2.2 GUI Code Generation

A lot of work has achieved good results in the field of automatically generating code for UI design drafts. The method REMAUI developed by Nguyen et al. [13] uses computer vision and optical character recognition (OCR) technology to identify user interface elements and further infer Appropriate user interface hierarchy and export it as source code that can be compiled and executed. The pix2code [1] proposed by Beltramell is a deep learning model that can convert UI screenshots into codes for the Web, Android and iOS platforms. Sketch2Code [9] proposed by Jain and other Microsoft researchers consists of a convolutional neural network, which takes a hand-drawn sketch image on a pure white surface and creates an Object representation of the UI, which is read by the UI parser to generate code for the target platform. Moran et al. proposed REDRAW [12], which achieves precise GUI prototyping through the three tasks of detection, classification, and assembly. Chen et al. [3] designed a neural converter. Given an input UI image, CNN extracts a set of different image features through a series of convolution and pooling operations. Then, the RNN encoder and RNN decoder generate a GUI framework form the spatial layout information of these image features. Pang et al. [14] first proposed a model called HGui2Code, which will enable the GUI function of visual attention and the semantic features that support DSL attention Integrated. In addition, they proposed SGui2Code, a novel model that uses the ON-LSTM network to generate syntactically correct DSL codes. The approach proposed by Chen et al. [4] combines the old-fashioned computer vision methods for non-textelement region detection, and deep learning models for region classification and GUI text detection.

3 Approach Description

3.1 Overall Architecture

As mentioned in Sect. 1, we divide the entire task into two steps. The first step includes object detection and text recognition technology, and the second step is the code parser. Modern object detector is usually composed of several parts, a backbone which is pre-trained on ImageNet, as well as the neck that makes better use of the features extracted by the backbone part and a head which is used to predict classes and bounding boxes of objects [2]. We tested the performance of the current mainstream object detection algorithms on our datasets, and finally we chose YOLOv3 as the head of our model. Inspired by YOLOv4 [2], We made a suitable simplification of YOLOv4 for our dataset as the object detection

part of *GUIS2Code*, and our backbone chooses CSPDarkNet65 [18]. For the text recognition module, we choose DenseNet [8] as the recognition network. Next, we developed a Code parser to generate code for our model. The architecture of *GUIS2Code* is shown in Fig. 1, we use CSPDarkNet for feature extraction, and then add the SPP [7] module to increase the receptive field, separates out the most important context features. And we use PANet [10] as the method of parameter aggregation from different backbone levels for different detector levels.

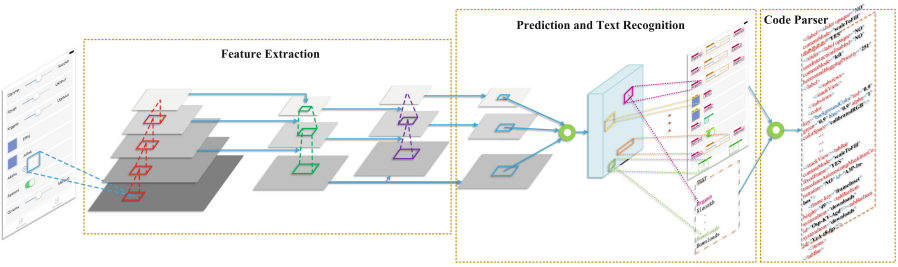


Fig. 1. Overall architecture of our model

3.2 UI Component Detection

For the first sub-problem, to understand the context of the elements present in the image, we have employed a deep neural network based object detection by mapping the input image with a set of classes and generating bounding boxes for the regions where they are present in the image. This mapping allows the training to be done independently of any language or platform restriction. As shown in Fig. 1, given the input UI image, our model extracts the category and location information of the UI element through a series of operations.

The backbone network of *GUIS2Code* is CSPDarknet65, which is based on the Yolov4 backbone network CSPDarknet53 and draws on the experience of Gao et al. [5] to generate the backbone structure, which contains 5 CSPNet modules and two types of modifications to improve the performance of CSPDarknet53, as shown in Fig. 2.

The full name of CSPNet [18] is Cross Stage Paritial Network, which mainly solves the problem of large amount of calculation in reasoning from the perspective of network structure design. The feature map of the base layer is divided into two parts, one of which is directly connected to the end of the stage, and the other part will pass through the res block, as shown in Fig. 3(a), thereby reducing repeated gradient information and computational bottlenecks while ensuring accuracy.

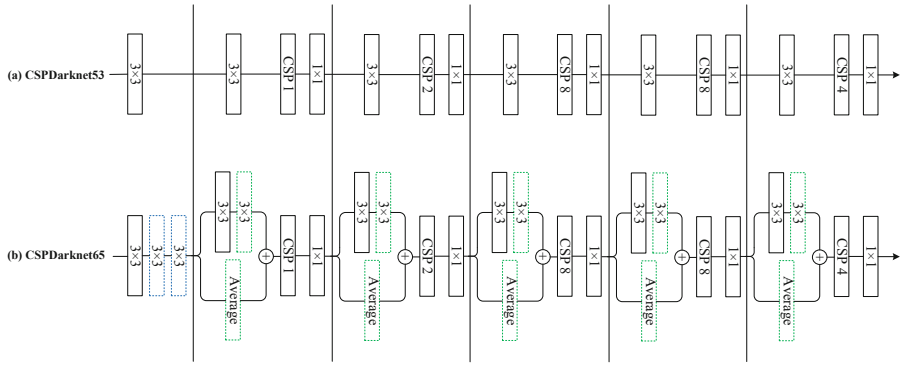


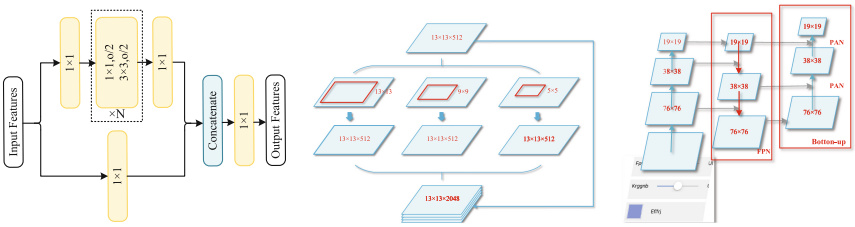
Fig. 2. Illustration of backbone networks. Each rectangle includes Conv, BN and Mish. CSP N, N in {1, 2, 8, 4}, denoted as the residual block repeated N times with CSP structure, as shown in the Fig. 3(a). (a) CSPDarknet53: original structure proposed in [2]. (b) CSPDarknet65: additional residual block (blue block) and substituted downsampling residual block (green block). (Color figure online)

Additional Root Block. In [22], extensive experiments have shown that performance can be improved by using a stack of 3×3 convolution filters. A useful but straightforward scheme is increasing one 3×3 convolution to three 3×3 convolutions. Through a large amount of input, the Root-Stage with three 3×3 convolutions can exploit more local information from the image, so as to extract powerful features for UI component detection. Therefore, an additional block is added at the root stage and is shown as a green block in Fig. 2(b).

Average Pooling Block. The size of the convolution kernel in front of the CSPNet module is 3×3, and the step size is 2, so it can play the role of downsampling. In order to strengthen the gradient propagation in the network, we replace this downsampling layer with Average Pooling block, which is shown as blue blocks in Fig. 2(b). In the projection shortcut path of such block, a 2×2 average pooling layer with a stride of 2 and the 1×1 convolution layer is added to replace downsampling layer, and the stride of 1×1 convolution is set to 1. In comparison with the original downsampling block in CSPDarknet, the improved structure can avoid information loss in projection shortcuts.

We added the SPP block to CSPDarknet65. We improve SPP module to the concatenation of max-pooling outputs with kernel size $k \times k$, where $k = \{1, 5, 9, 13\}$, and stride equals to 1, as shown in Fig. 3(b). Under this design, a relatively large $k \times k$ maxpooling effectively increase the receptive field of backbone feature. We tested the performance of SPP in our datasets in the experiment. Compared with no spp, our model has improved AP after adding SPP, while the computational cost is very small. Then we add PANet [10] to FPN as a parameter aggregation method from different backbone levels for different detector levels. We have added a bottom-up feature pyramid behind the FPN layer, which contains two PAN structures. In this combination of operations, the

FPN layer conveys strong semantic features from the top to the bottom, while the feature pyramid conveys strong positioning features from the bottom to the top. They work together to aggregate different detection layers from different backbone layers. This structure is shown in Fig. 3(c).



(a) Darknet structure using CSP. (b) Spatial Pyramid Pooling Block (c) FPN and PANet in our model

Fig. 3. Architecture details

Then, for bounding box regression, we use CIoU loss algorithms [21]. About evaluation metric for bounding box regression, Intersection over Union (IoU) is defined as

$$IoU = \frac{|A \bigcap B|}{|A \bigcup B|} \tag{1}$$

where $B = (x^{gt}, y^{gt}, w^{gt}, h^{gt})$is the ground-truth bounding box, and $A = (x, y, w, h)$ is the predicted box [19]. The CIoU loss is proposed by imposing the consistency of aspect ratio, then, the CIoU loss function can be defined as

$$\mathcal{L}_{CIoU} = 1 - IoU + \frac{\rho^2(\mathbf{a}, \mathbf{b})}{c^2} + \frac{\upsilon}{(1 - IoU) + \upsilon}\upsilon \tag{2}$$

where $\mathbf{a}$ and $\mathbf{b}$ denote the central points of A and B, $\rho(\cdot)$ is the Euclidean distance, and c is the diagonal length of the smallest enclosing box covering the two boxes. υ measures the consistency of aspect ratio:

$$\upsilon = \frac{4}{\pi^2}\left(arctan\frac{w^{gt}}{h^{gt}} - arctan\frac{w}{h}\right)^2. \tag{3}$$

In this way, overlapping area factors have higher priority in regression, especially for non-overlapping cases [21].

Finally, we choose CSPDarknet65 backbone network which reduces the amount of calculation while ensuring accuracy, SPP additional module which can improved accuracy and the computational cost is very small, PANet path-aggregation neck with strong positioning features, and YOLOv3 (anchor based) head as the architecture of our model. Through the combination of these approaches, the object detection module of *GUIS2Code* achieved higher accuracy on our UI design draft dataset when compared with other models.

3.3 Text Recognition

At the same time, we considered the text problem of components such as label and text. General text recognition has two parts, text region detection and text recognition. In our method, the result of object detection and recognition can be directly used as the text region to be recognized, so we only used the text recognition network.

DenseNet [8] can achieve good performance in text recognition, so we add trained DenseNet as a text recognition network to our model. In our approach, we label the text area in the GUI element corresponding to the text, use DenseNet-169 as the feature extraction network to extract the features of the text, and train it, and finally get the GUI text detection network.

3.4 Code Parser

The generated token sequence from network mentioned earlier can then be compiled with traditional compilation methods to the desired target language. The code parser is the final step, which converts the UI components in the UI representation object into code that can be executed on the target platform. The generated file is an XML or HTML document containing the UI components which can be run easily. In our experiment, the generated iOS and Android are UIs in the XML format while Web is web-based UIs implemented in HTML/CSS. The algorithm of code parser is shown in Algorithm 1. For space reasons, we have omitted some details.

4 Experimental Results and Analysis

4.1 Dataset and Experiment Setup

The screenshots of the UI interface in our datasets use pictures from the pix2code [1] datasets. The types and numbers of UI interfaces included in this dataset are shown in Fig. 4(a). At the same time, we manually annotated the UI components in each UI interface screenshot, and the dataset annotation format is shown in the Fig. 4(b). Our experimental standards are based on MS COCO, various IoU thresholds are used for more comprehensive calculation. The metric to evaluate detection performance is the mean Average Precision (mAP). The proposed networks are based on the Tensorflow framework. The results in figure are generated on a NVIDIA Tesla V100 16GB GPU with cuDNN (CUDA Deep Neural Network Library) acceleration. The processor used is In-tel(R) Xeon(R) E5-2640 and CentOS 7.5 operating system.

4.2 The Ablation Study and Evaluate

We have verified tricks we have introduced above, the ablation experiments are designed to verify the effects of the network modifications. The results of the ablation experiment are shown in the Table 1. The performance of CSPDarknet53 is

Algorithm 1: Algorithm of code parser.

Input: Bounding box and text pairs $P(B, T)$ of the network output; UI map $M(Class, Code)$;
Output: Generated xml or html file F;

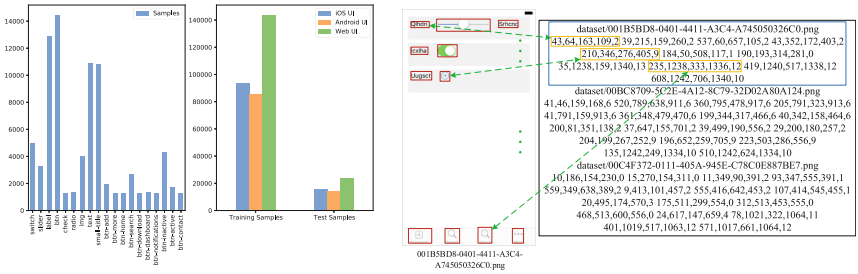
Sort P according to position information
Initialize $parent = Node('start', None), current\_parent = parent$
for $p(b, t) \in P(B, T)$ **do**
    $token = p.b.class$
    if $M.find(token)! = -1$ **then**
        $element = Node(token, p.t)$
        $current\_parent.children.append(element)$
        $current\_parent = current\_parent.children$
    end if
end for
$F =$RENDER$(parent.children, M)$
return F
function RENDER$(parent, mapping)$
    Initialize $content = NULL$
    for $BFS(parent)$ **do**
        $content+ =$REPLACE$(child, mapping)$
    end for
    New a file F
    $F.write(content)$
    return F
end function

81.9%. The second and third rows of Table 1 show that the modifications of the Additional Root block and the Average Pooling blocks improve the performance to 83.1% and 84.4%, respectively. Moreover, with both modifications mentioned above, CSPDarknet65 can achieve 85.3%.

Table 1. Ablation experiment

Backbone	AP (%)	AP_{50} (%)	AP_{75} (%)	Delta (AP)
CSPDarknet53	81.9	99.6	96.9	0
CSPDarknet63 (Average Pooling)	83.1	99.6	98.0	1.2
CSPDarknet55 (Root block)	84.4	99.7	98.3	2.5
CSPDarknet65	**85.3**	**99.7**	**98.6**	**3.4**

The final performance of the UI object detection model reached 85.3% in mAP and 98.6% in AP_{75}. Figure 5 show samples consisting of input GUIs (i.e. ground truth), output GUIs, and generated code (part of the code is omitted). These output GUI screenshots are obtained by sampling code with a trained *GUIS2Code* model, the outputs is then compiled to the appropriate target language producing UI code that can be rendered and captured as an image.



(a) Samples Statistics (b) Example of Label Format

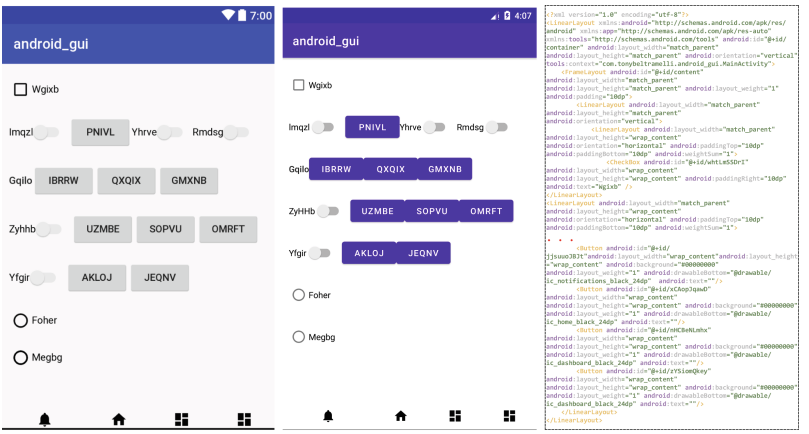
Fig. 4. Dataset details

4.3 Comparison with Other Approaches

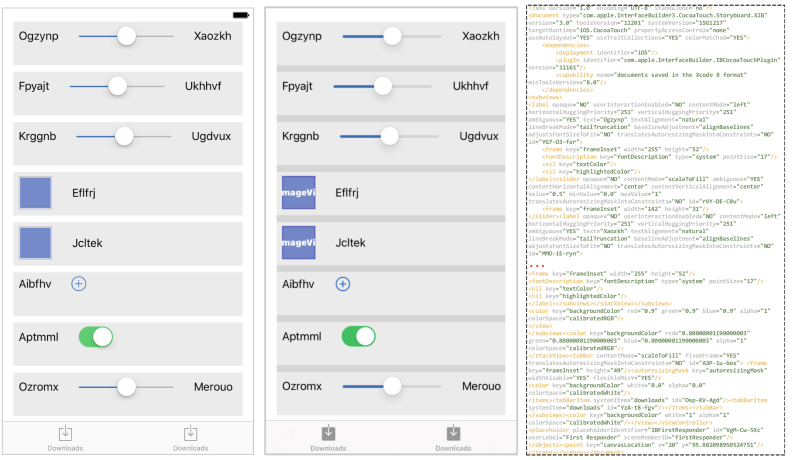
In order to compare with pix2code, we used the code provided by the author of the paper on GitHub and the dataset provided to reproduce the model of pix2code, and compared with our model. We also compared with other models, the comparison experiment results are shown in Table 2. Our model is more accurate than all other models. The evaluation indicator is the accuracy rate of the generated code, which is defined as the ratio of the number of samples with accurate classification to the total number of samples.

Table 2. Comparison with other approaches.

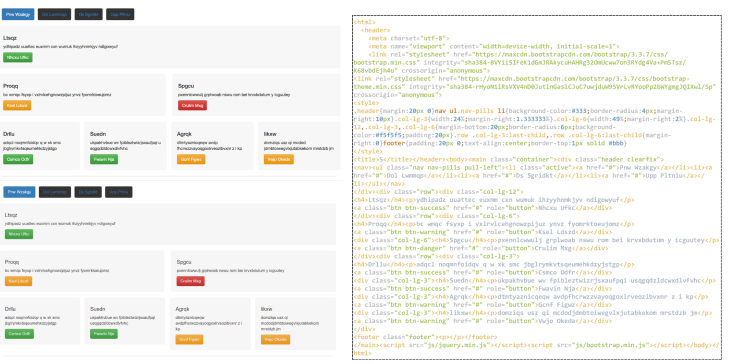
Approaches	Accuracy (%)			
	iOS	Android	Web	Total
Pix2code [1]	77.27	77.66	77.65	77.53
HGui2Code [14]	80.80	81.13	90.40	84.11
AGui2Code [14]	94.00	65.76	64.80	74.85
SGui2Code [14]	77.20	77.71	90.00	81.64
ABHD [23]	81.00	81.35	88.50	83.62
GUIS2Code (Ours)	**95.16**	**94.60**	**95.37**	**95.04**



(a) Android, left to right: Groundtruth, Generated, Code



(b) iOS, left to right: Groundtruth, Generated, Code



(c) Web, top to bottom: Groundtruth, Generated, Code

Fig. 5. Examples of output from *GUIS2Code*.

5 Conclusion

In this paper, we propose an approach based on deep neural network, which uses screenshots of UI design drafts as input to transform and generate corresponding codes in different languages. Our method uses object detection technology, which can improve the recognition rate of UI components. At the same time, we consider the text problem on the component and restore it through text recognition technology. It has succeeded in three different platforms (i.e., iOS, Android and web) Generate code. Through evaluation, our method achieves an accuracy of 95.04%, and the generated code can restore the target model with maximum accuracy. Our dataset comes from pix2code, but it is manually generated by the author, which is different from the actual running application or web page. In fact, we can crawl website screenshots and associated HTML code datasets or Android and iOS. And now there are a large number of web pages and Android and iOS GUIs available on the Internet. Therefore, theoretically, we can obtain almost unlimited training data, thereby enhancing the capabilities of our model, and even generating code from the GUI completely automatically.

References

1. Beltramelli, T.: Pix2Code: generating code from a graphical user interface screenshot. In: Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems, pp. 3:1–3:6. ACM (2018). https://doi.org/10.1145/3220134.3220135
2. Bochkovskiy, A., Wang, C.Y., Liao, H.Y.M.: YOLOv4: optimal speed and accuracy of object detection. arXiv e-prints arXiv:2004.10934, April 2020
3. Chen, C., Su, T., Meng, G., Xing, Z., Liu, Y.: From UI design image to GUI skeleton: a neural machine translator to bootstrap mobile GUI implementation. In: Proceedings of the 40th International Conference on Software Engineering, pp. 665–676. ACM (2018). https://doi.org/10.1145/3180155.3180240
4. Chen, J., et al.: Object detection for graphical user interface: old fashioned or deep learning or a combination? In: ESEC/FSE 2020: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 1202–1214. ACM (2020). https://doi.org/10.1145/3368089.3409691
5. Gao, F., Yang, C., Ge, Y., Lu, S., Shao, Q.: Dense receptive field network: a backbone network for object detection. In: Tetko, I.V., Kůrková, V., Karpov, P., Theis, F. (eds.) ICANN 2019. LNCS, vol. 11729, pp. 105–118. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30508-6_9
6. Girshick, R.B.: Fast R-CNN. In: IEEE International Conference on Computer Vision, Santiago, Chile, pp. 1440–1448. IEEE Computer Society (2015). https://doi.org/10.1109/ICCV.2015.169
7. He, K., Zhang, X., Ren, S., Sun, J.: Spatial pyramid pooling in deep convolutional networks for visual recognition. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014. LNCS, vol. 8691, pp. 346–361. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10578-9_23
8. Huang, G., Liu, Z., van der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 2261–2269. IEEE Computer Society (2017). https://doi.org/10.1109/CVPR.2017.243

9. Jain, V., Agrawal, P., Banga, S., Kapoor, R., Gulyani, S.: Sketch2Code: transformation of sketches to UI in real-time using deep neural network. arXiv e-prints arXiv:1910.08930, October 2019
10. Liu, S., Qi, L., Qin, H., Shi, J., Jia, J.: Path aggregation network for instance segmentation. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 8759–8768. IEEE Computer Society (2018). https://doi.org/10.1109/CVPR.2018.00913
11. Liu, W., et al.: SSD: single shot multibox detector. In: 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
12. Moran, K., Bernal-Cárdenas, C., Curcio, M., Bonett, R., Poshyvanyk, D.: Machine learning-based prototyping of graphical user interfaces for mobile apps. IEEE Trans. Softw. Eng. **46**(2), 196–221 (2020). https://doi.org/10.1109/TSE.2018.2844788
13. Nguyen, T.A., Csallner, C.: Reverse engineering mobile application user interfaces with REMAUI (T). In: 30th IEEE/ACM International Conference on Automated Software Engineering, pp. 248–259. IEEE Computer Society (2015). https://doi.org/10.1109/ASE.2015.32
14. Pang, X.W., Zhou, Y., Li, P., Lin, W., Wu, W., Wang, J.Z.: A novel syntax-aware automatic graphics code generation with attention-based deep neural network. J. Netw. Comput. Appl. **161**, 102636 (2020). https://doi.org/10.1016/j.jnca.2020.102636
15. Redmon, J., Divvala, S.K., Girshick, R.B., Farhadi, A.: You only look once: unified, real-time object detection. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 779–788. IEEE Computer Society (2016). https://doi.org/10.1109/CVPR.2016.91
16. Shi, B., Bai, X., Yao, C.: An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. IEEE Trans. Pattern Anal. Mach. Intell. **39**(11), 2298–2304 (2017). https://doi.org/10.1109/TPAMI.2016.2646371
17. Shi, B., Wang, X., Lyu, P., Yao, C., Bai, X.: Robust scene text recognition with automatic rectification. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 4168–4176. IEEE Computer Society (2016). https://doi.org/10.1109/CVPR.2016.452
18. Wang, C., Liao, H.M., Wu, Y., Chen, P., Hsieh, J., Yeh, I.: CSPNet: a new backbone that can enhance learning capability of CNN. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2020, pp. 1571–1580. IEEE (2020). https://doi.org/10.1109/CVPRW50498.2020.00203
19. Yu, J., Jiang, Y., Wang, Z., Cao, Z., Huang, T.: UnitBox: an advanced object detection network. In: Proceedings of the 24th ACM International Conference on Multimedia, pp. 516–520. Association for Computing Machinery, New York (2016). https://doi.org/10.1145/2964284.2967274
20. Zhan, F., Lu, S.: ESIR: end-to-end scene text recognition via iterative image rectification. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 2059–2068. Computer Vision Foundation/IEEE (2019). https://doi.org/10.1109/CVPR.2019.00216
21. Zheng, Z., Wang, P., Liu, W., Li, J., Ye, R., Ren, D.: Distance-IoU loss: faster and better learning for bounding box regression. In: The Thirty-Fourth AAAI Conference on Artificial Intelligence, pp. 12993–13000. AAAI Press (2020). https://doi.org/10.1609/aaai.v34i07.6999

22. Zhu, R., et al.: ScratchDet: training single-shot object detectors from scratch. arXiv e-prints arXiv:1810.08425, October 2018
23. Zhu, Z., Xue, Z., Yuan, Z.: Automatic graphics program generation using attention-based hierarchical decoder. In: Jawahar, C.V., Li, H., Mori, G., Schindler, K. (eds.) ACCV 2018. LNCS, vol. 11366, pp. 181–196. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-20876-9_12