UI Layers Group Detector: Grouping UI Layers via Text Fusion and Box Attention

Shuhong Xiao[1], Tingting Zhou[4], Yunnong Chen[1], Dengming Zhang[2], Liuqing Chen[1,3(✉)], Lingyun Sun[1,3], and Shiyu Yue[4]

[1] Zhejiang University, Hangzhou 310027, China
chenlq@zju.edu.cn
[2] Chongqing University of Posts and Telecommunications, Chongqing 400065, China
[3] Alibaba-Zhejiang University Joint Research Institute of Frontier Technologies, Hangzhou 310027, China
[4] Alibaba Group, Hangzhou 311121, China

Abstract. Graphic User Interface (GUI) is facing great demand with the popularization and prosperity of mobile apps. Automatic UI code generation from UI design draft dramatically simplifies the development process. However, the nesting layer structure in the design draft affects the quality and usability of the generated code. Few existing GUI automated techniques detect and group the nested layers to improve the accessibility of generated code. In this paper, we proposed our UI Layers Group Detector as a vision-based method that automatically detects images (i.e., basic shapes and visual elements) and text layers that present the same semantic meanings. We propose two plug-in components, text fusion and box attention, that utilize text information from design drafts as a priori information for group localization. We construct a large-scale UI dataset for training and testing, and present a data augmentation approach to boost the detection performance. The experiment shows that the proposed method achieves a decent accuracy regarding layers grouping.

Keywords: UI to code · UI layers grouping · Object detection · Multi-modal embedding

1 Introduction

As the central intermediary of human-computer interaction, Graphic User Interface (GUI) is facing great demand with the popularization and prosperity of mobile apps. The traditional process of designing GUI is very long-lasting. It requires investigators to conduct user research, designers to design page materials, and then front-end engineers to code, and it is time-consuming to reach a consensus before multi rounds of back and forth [11].

To achieve faster development and relieve engineers from heavy workloads, some previous researchers applied intelligent methods in automatic GUI generation. Ling et al. [17] considered it a language generation task, and they utilized

L. Fang et al. (Eds.): CICAI 2022, LNAI 13605, pp. 303–314, 2022.
https://doi.org/10.1007/978-3-031-20500-2_25

a language generation method to generate GUI source code from a mixed natural language. REMAUI [20] is the first work to introduce GUI screenshots as materials for code generation. They achieved a reverse process from screenshots to GUI code in seconds. Pix2code [2] then extended the use of images, and they trained an end-to-end deep learning model that generates a source code from a single design input image with 77% accuracy. More recent commercialization cases like Imgcook [12] construct a platform that allows diversified inputs including screenshots, PSDs, and Figma[1] or Sketch[2] files which contain metadata.

(a) An Example of fragmented layer in UI design.

(b) Example of grouping layers.

Fig. 1. (a) The icon with the red bounding box is formed with three basic shape layers. (b) The labeled text and image elements need to be included under the same DOM node using the "group" method by adding #group# to the target containers.

In the practice of automatic code generation, a massive gap between the design draft (created by digital tools like Sketch) and a quality product (Code and its visual presentation) is that some layers form a whole element in the design draft should be included as a single UI component in the code, while the code generated by automation is hard to reach this. In this case, design drafts should be further constrained by some specific rules to achieve UI code generation with high-quality. For instance, the state-of-art solution, Imgcook, highlights "merge" and "group" as two of the most important rules that reorganize the structure of design drafts to bridge the gap we described. The "merge" method integrates multiple fragmented layers representing basic shapes (e.g., rectangle, oval, path) and visual elements(e.g., text and image) into a single image. As illustrated in Fig. 1a, the icon with the red bounding box is formed with three basic shape layers. Without structured merging, these fragmented layers confuse the AI in understanding the semantic meaning of UI components and affect the readability and reusability of generated code. In the other case, the "group" method deals with malposed structures in the design draft. As illustrated in Fig. 1b, to ensure that generated code does not cause element loss or structural redundancy, the labeled text and image elements need to be included under the same DOM node using the "group" method by adding "#group#" to the target containers. However, Imgcook requires front-end engineers to manually locate the layers to

[1] https://www.figma.com/.
[2] https://www.sketch.com/.

proceed operate the "merge" or "group" method. Manual identification is time-consuming and prone to omissions because of the numerous layers and diverse nested structures. In this paper, we focus on automatically recognizing "group" problems. More specifically, we try to locate and group images (we define images as all basic shapes and visual elements) and text layers that have the same semantic meaning in the design draft. Therefore, We can optimize the design layout and obtain high-quality UI code.

(a) An Example of semantic consistency. (b) An Example of various image-text group patterns

Fig. 2. (a) The clock image in blue box shares a close semantic meaning with the text "investment schedul", while the text "view it" in the red box represents another meaning. (b) The group strategy is different for the banner and small icon. We keep some background for the banner in order to avoid missing some elements (that maybe invisible) while bound the small icon group as tight as possible.

To address this issue, We proposed our UI Layers Group Detector that utilizes object detection techniques to detect the area to be grouped on real screenshot images. As multi-modal approaches have shown its great power in general UI component detection task [26], we introduce text embedding and box attention mechanism that use text-related information as extra modalities to help improve our Detector. The text layers are used as a local semantic focal, together with the global image feature to benefit generated proposals. Given an arbitrary design draft, we follow semantic consistency as the grouping criterion, i.e., not all adjacent visual elements will be integrated into a single group. As illustrated in Fig. 2a, the clock image in blue box shares a close semantic meaning with the text "investment schedule", while the text "view it" in the red box represents another meaning. Under this condition, precisely predicting the group range is challenging because of the background layer and elements around. Another challenge is that the diversified UI application scenarios result in various group patterns with different sizes and element numbers. For example, as illustrated in Fig. 2b, for the group of banner contains a thumbnail with text information like product name, price, and description, we have the empty background layer inside the bounding box to better retrieve all the banner elements in the design draft. This is because for group like banner which contains abundant elements, elements sometimes partially invisible so that they will be dropped if we apply a tight bound. While for small icon like the group only contains ">" and text

"8 items", we make the bounding box as tight as possible so that no irrelevant element will be covered.

In particular, our method solves the two challenges by following approaches. We adopt a state-of-art object detection model to achieve precise localizing of group objectives. To deploy our Group Detector as a plug-in for UI code generator like Imgcook, we choose to adopt from the Faster-RCNN because it is lightweight, delicate, and easy to modify. We collect our dataset from the Sketch files of the most frequently used mobile apps. The target groups on each image are labeled carefully by professional labelers guided by the semantic consistency we described before. The high-quality dataset allows our data-driven model to make more accurate predictions. We summarize our contributions as follows:

1. We construct a high-quality dataset containing UI screen images from widely used mobile apps. An image augmentation algorithm is introduced to boost the performance of our method.
2. We proposed our UI Layers Group Detector which takes UI screenshots and metadata from design drafts and solve the image-text grouping task. This work is designed to fill in the gap in the automatic UI code generation works.
3. We proposed the text fusion to incorporate features from text layers to the related image region. And the box attention mechanism is introduced as another plug-in component with creating an extra spatial binary image which encodes the position of each text layer inside the UI image.
4. We carry out experiments on the constructed dataset to verify that the UI Layers Group Detector achieves a decent accuracy regarding UI layers grouping.

2 Related Works

2.1 Intelligent UI Code Generation

Automation in UI code generation has become an attractive topic after the machine learning and artificial intelligence boom. Early intelligent automation research was mainly used to replace template-based UI design, where users spend time searching for suitable materials to compose their design. We can further classify these works by the level of the fidelity of the design prototype they use as input [7]. Batuhan et al. [1] use hand-drawn images to identify and generate basic buttons, text, images, and other components. Works as sketch2code [23] utilizes design drafts with more details and achieves the automatic generation of UI structure. Pix2code [2] follows a similar approach to generating textual descriptions from photographs. It takes actual UI screenshots as input and achieves a high accuracy generation. More recently, commercial platforms like Imgcook [12] takes advantage of metadata in professionally designed software and achieve a generation with high quality and reusability.

2.2 UI Design Check

With the popularization of automated UI generation technology, in practical applications, it is necessary to evaluate the generation quality and solve the problems like missing elements or components overlapping caused by hardware or software compatibility. OwlEye [19] detects GUIs with display issues and locates the detailed region of the issue based on the deep learning method. LabelDroid [4] focuses on image-based buttons and achieves a highly accurate prediction of labels by learning from large-scale commercial apps in Google Play. FSMdroid [24] uses the MCMC sampling method to analyze GUI apps dynamically and detect defects that reside on unfrequented trails.

Considering our work as an object detection task, we briefly review the recent advances in this field. Anchor-based approaches like Faster-RCNN [22] define a set of anchor boxes of different sizes and use feature maps from different convolutional layers to classify and regress anchor boxes. The following SSD [18], Yolov2 [21], and RetinaNet [15] continue this idea and achieve a state-of-art performance in natural object detection. Some achievement has also been made in using Object Detection in UI-related areas. Li et al. [13] introduce the CLAY pipeline for UI screen dataset cleaning. They address the mismatches in visual elements and metadata. Zang et al. [26] leverage object detection on recognizing UI icons and achieve good performance.

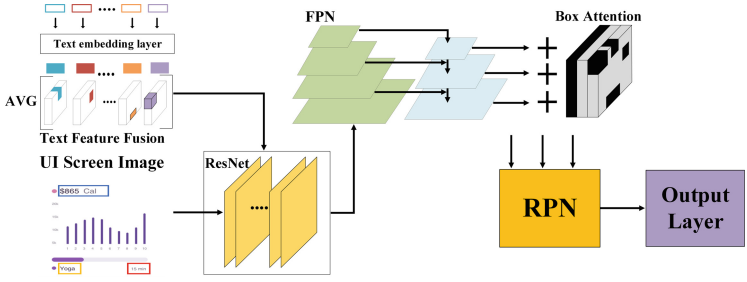


Fig. 3. The overview of proposed method.

3 Methods

In this section, we introduce the proposed approach to our UI Layers Group Detector described in Sect. 1 as shown in Fig. 3. We first introduce our work on collecting UI screen dataset based on Sketch file, which is widely used in software UI design. Image segmentation is applied considering the nature of small and regional target detection (Sect. 3.1). We then introduce our Group Detector via the order of basic setting, text fusion, and box attention. We introduce the strategy of text content and bounding boxes as extra features to help with target

localization. The embedding text features are fused into the early convolution layer of our backbone (Sect. 3.2). We then introduce box attention as another way of utilizing text information. A spatial binary image is created based on text bounding boxes to guide the target detection process by revealing potential image-text group areas. We fuse the box attention with the image feature maps on all FPN output layers and feed them into the RPN for better proposals (Sect. 3.2).

3.1 Dataset

We create a high-quality UI dataset based on UI layouts developed by professional designers using Sketch. Each artboard in a Sketch file represents a UI design for Android or iOS mobile Apps. We eliminate unusual designs that placed the components without artboards as containers and convert the rest as UI screen images. A group of professional labelers was recruited to generate our target group annotation. We adapt our dataset based on COCO-style [6], including images, annotations, categories, and supplementary text consisting of text semantics and position in the images. Considering the nature of the UI screen with a large aspect ratio, while our targets are usually small and regional, image segmentation is applied to all collected UI screens. Given an arbitrary screenshot image, we split it along the long side. To avoid distortion, we keep each piece a square shape, and only bounding boxes that are entirely inside should be recorded. To avoid some bounding boxes being separated into two samples, we leverage a sliding stride to make sure every bounding box will at least appear in one image.

We then split our dataset into training, validation, and test sets for experiments. The split is performed package-wise and image-wise, i.e., artboards in the same Sketch design file and segmented images from the same UI screen image are not shared across different splits. This approach avoids information leakage because UI screen images from the same design might have similar layouts. As a summary, the number of screen images in our dataset increases by a factor of 3 after applying the segmentation. We have 4946 screen images in the training set with 16533 labeled groups and 11617 texts. For the validation set, we have 601 screen images, 2093 labeled groups and 2093 labeled groups. And for the test set, we have 681 screen images, 2410 labeled groups and 1442 labeled groups.

3.2 UI Layers Group Detector

Usually, the placement and size of our target groups vary widely across the different UI designs. Therefore, the most crucial step in our approach is to infer the bounding boxes accurately. To achieve this goal, we utilize ResNet-50 [9] and FPN (Feature Pyramid Networks) [14] as our backbone for extracting feature maps. At the RPN (region proposal network), we use a softmax to determine positive or negative anchors and a bounding box regression to modify anchors to obtain precise proposals. Then we borrow the RoI Align [8] to replace the RoI Pooling for proposal maps extraction based on input feature maps and proposals.

Instead of quantized operations, the RoI Align utilizes linear interpolation to reduce the precision loss. At the classification stage, bounding box regression is introduced again to obtain each proposal's position offset for regressing more accurate target bounding boxes.

Text Fusion. This section introduces the text embedding features as a plug-in component to the Group Detector. As we discussed in Sect. 1, the grouping patterns varies in different application scenarios. While we also find that components with the same functionality are similar in appearance and should further apply the same grouping strategy. In this paper, together with the pixel-based UI screen image data, we further utilize the text layer information in each Sketch design, which contains the text layer position as bounding boxes and text contents. The supplementary text-level knowledge enables us to identify potential pattern-text groups and reveals the function of target groups in the UI screen.

Given a UI screen image in our dataset, let $\{bb_i\}_{i=1}^{N}$ be all text layer bounding boxes inside the image and $\{tt_i\}_{i=1}^{N}$ be all text contents. Let $I \in \mathbb{R}^{3\times H\times W}$ as input image and $C \in \mathbb{R}^{D\times H'\times W'}$ be an intermediate feature map extracted from the first convolution layer $Conv1$ of ResNet-50. Our first step is to construct a feature map $T_i \in \mathbb{R}^{K\times H\times W}$ for every text information with the same height H and width W using the bound box $bb_i = \{x_{min}, y_{min}, x_{max}, y_{max}\}$. To achieve this, we use a text encoder to transform text content tt_i into corresponding text embedding $e_i \in \mathbb{R}^{K\times 1\times 1}$. We let T_i fill with all 0 initially and update the text embedding e_i inside the bounding box range. Specifically, we set $T_i[:, p, q] = e_i$ where $p \in [y_{min} \times H, y_{max} \times H]$ and $q \in [x_{min} \times W, x_{max} \times W]$. In the next step, we calculate $T = avg(T_i)_{i=1}^{N} \in \mathbb{R}^{K\times H\times W}$ and apply the same $Conv1$ and an extra 1×1 convolution layer to get the text feature map T' with the same size of pixel feature map C. In the final step, we calculate a new feature map $F = T' \circ C$, where $\circ$ denotes the element-wise addition, and this new fused feature map is fed into the later convolution layer of Resnet-50.

Box Attention. In this section, we introduce box attention as another way of utilizing text information to improve our Group Detector. We borrow the idea of box attention from Bunian et al [3]. This mechanism was first introduced by Kolesnikov et al [10]. to model object interactions in a visual relationship detection task. Bunian et al. adopted it in their object detection pipeline. The idea of box attention is to create an extra spatial binary image encoding the position of each text layer inside the UI screen. The binary image here, is considered as a position-focal that together with the global image information to understand compositional relationship among elements.

Specifically, given an image feature map $F_i \in \mathbb{R}^{D\times H_i\times W_i}$ and text bounding box $\{bb_j\}_{j=1}^{N}$ where $bb_j = \{x_{min}, y_{min}, x_{max}, y_{max}\}$ corresponding to input UI screen image height H and width W, we first transform the bounding box based on H_i and W_i and create the bounding box map $B_{i,j} \in \mathbb{R}^{3\times H_i\times W_i}$. We set $B_{i,j}[0, p, q] = 1$ where $p \in [y_{min} \times H_i, y_{max} \times H_i]$ and $q \in [x_{min} \times W_i, x_{max} \times W_i]$, $B_{i,j}[1, :, :] = 0$ and $B_{i,j}[2, :, :] = 1$. In the next step, we calculate the overall box

attention map B_i for the i^{th} image feature map as $B_i = avg(B_{i,j})_{j=1}^{N}$ and an extra 1×1 convolution layer is applied on it to match the image feature map dimension D. In the final step, we add the box attention map with each FPN output image feature map F_i to get the fused feature map $M_i = F_i \circ B_i$ for further proposal generating.

4 Experiments

4.1 Implementation Details

We implemente our model using MMDetection [5] codebase. We use ResNet-50 pre-trained on ImageNet together with the FPN as our backbone network. We set the anchor size [32,64,128,256,512] with anchor ratio [0.5,1.0,2.0,4.0,8.0] for the potential tiny and nonsquare target. The input UI screen images are resized into size in [800,1300] before being fed into Resnet-50. The output features of stage1-4 of ResNet-50 are fed into FPN, together with a further MaxPooling layer, giving us the five feature maps with different scales. For FPN settings, we follow the standard settings as in [14]. Proposals are computed from all five pyramid feature maps, and RoI Align is performed with bilinear interpolation.

For text embedding fusion, we first encode the text contents into vectors with length K using pre-trained transformers [25]. We set the parameter K to 16, considering the average text length of 11.4. After generating the text feature map $T \in \mathbb{R}^{K \times H' \times W'}$, we increase the channel size from K to D which is the channel size of feature map at Conv1 of stage 0 in ResNet-50. For box attention, we adopt five spatial binary images with the same size as the pyramid feature maps.

For the training details, we train our model with a mini-batch of 2 for 72 epochs using SGD optimizer with a momentum update of 0.9 and a weight decay of 0.0005; and set the initial learning rate 0.01 with a decay of factor 0.1 every 10 epochs during training. We train our model on an NVIDIA GeForce RTX 3080Ti GPU and it takes about 9 h for the model to converge.

4.2 Evaluation Metrics

In this paper, we report performance metrics used in the COCO detection evaluation criterion [16] and provide mean Average Precision (AP) across various IoU thresholds i.e. IoU=0.50:0.95,0.50,0.75 and various scales: small, medium and large. Without further specified, we refer mAP[0.50:0.95]to as AP.

4.3 Results

Detection Performance After Image Segmentation. We first test the benefits of our image segmentation algorithm. As shown in Table 1, the image segmentation contributes to a performance boost of AP by about 20%. In the UI design draft, a potential group like a banner or an icon only takes up a small

area of the overall screen. Even if we have modified the anchor's size and aspect ratio accordingly, finding all targets is still challenging. With segmentation, the target becomes more prominent in the background. Furthermore, we also get an augmentation effect with an overlapping slide window that applies multiple locations for each target in image slices.

Table 1. Detection performance after image segmentation.

Method	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
No segmentation	0.428	0.585	0.474	0.367	0.408	0.446
Apply segmentation	**0.625**	**0.799**	**0.699**	**0.550**	**0.593**	**0.648**

Detection Models Comparison. We then compare the detection performance using recent state-of-art object detection models as shown in Table 2. All the results are tested using the segmented dataset. To deploy our Group Detector as a plug-in for UI code generator like Imgcook, we expect it to be lightweight while achieving high accuracy. "UIGD-TF" denotes our UI Layers Group Detector with the text fusion and "UIGD-BA" denotes our UI Layers Group Detector with the box attention. Here we could see that "UIGD-TF" achieves the highest AP performance of 0.658 among all the model we experimented. And "UIGD-BA" takes the second position with AP performance of 0.650. They are also lighter than other models. For example, "UIGD-TF" is 66.9% lighter on model parameters than the Sparse-RCNN while 3.7% higher on accuracy.

Table 2. Detection performance comparison.

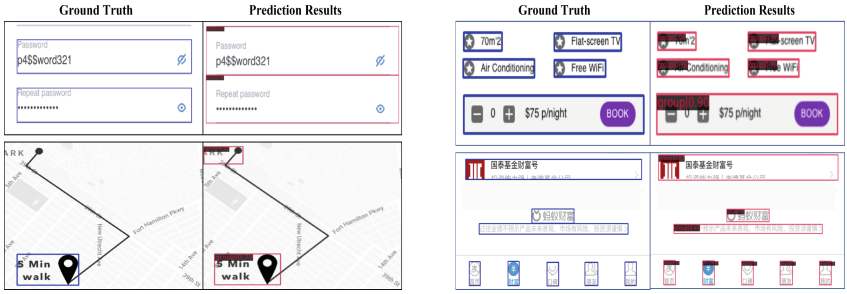
Method	Parameters ↓	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Deformable DETR	49.76M	0.603	0.751	0.670	0.439	0.558	0.635
YoloX	54.15M	0.592	0.788	0.654	0.448	0.517	0.634
Sparse-RCNN	125.21M	0.621	0.759	0.683	0.500	0.566	0.657
Faster-RCNN	41.35M	0.625	0.799	0.699	0.550	0.593	0.648
UIGD-TF	41.30M	**0.658**	**0.853**	**0.729**	**0.572**	0.643	**0.708**
UIGD-BA	41.29M	0.650	0.825	0.715	0.568	**0.651**	0.678

Performance with Text Fusion and Box Attention. We then investigate the contribution of our proposed text fusion component. As shown in Table 2, compare with the Fatser-RCNN, which got the highest accuracy in the rest of the model, we get an 3.3% AP improvement brought by the text fusion strategy suggests the necessity of the text semantic and location prior knowledge, which enriches the feature representation. The box attention also gives a 2.5%

AP improvement, indicating that concentrating on text location help captures potential group targets. Comparing the model parameters, it can be ssen that apply the text fusion and the box attention only increase the model parameters by 0.48% and 0.24% while the accuracy is significantly increased by 5.28% and 4.00%. All the results show that our proposed text fusion and box attention work effectively, and achieve our requirement of lightweight.

Detection Cases Analysis. Finally, we present some of our detection results. Figure 4a shows two cases that our Group Detector successfully localize all the target groups with highly matched bounding boxes. In these two examples, it can be seen that our model achieve a high accuracy of predicting diversified UI components with different patterns and elements number as we discussed in Sect. 1.

We also present some cases where our model fails. Figure 4b shows two typical examples that fully embody the existing shortcomings. The picture above shows that all group predictions are correct but not perfect fits. With an empty background around, our model fails to determine the boundaries of the target groups. Although this does not affect the localization of the layers in design drafts (no extra layers are included by error), the model performance is underestimated because of the low IoU. Another challenge is that our model shows weakness in distinguishing between foreground and background. As the bottom picture shows, our model mistakenly groups the address "ARK" in the background layer with the black dots in the foreground. The classic expression of this problem is that our model produces the wrong groups when faced with complex multi-level structures.



(a) Example of good cases. (b) Example of failure cases.

Fig. 4. (a) Our Group Detector successfully localize all the target groups with highly matched bounding boxes. (b) Our Group Detector show weakness on perfectly fit the ground truth bounding boxes on empty background around, and produces the wrong groups when faced with complex multi-level structures.

5 Conclusion

This paper investigates a novel issue about layers grouping in an automatic design draft to UI view code process, which can decrease the quality of generated code. To solve this issue, we propose our UI Layers Group Detector to locate the group accurately. By dataset segmentation, we achieve about a 20% boost in detection AP. We also propose two plug-in components to help increase the detection performance. The Text fusion introduces text semantic and location prior knowledge and achieves about a 3.3% increase in detection AP. For the box attention, the spatial binary images encoding potential text location give us a 2.5% increase in detection AP.

Acknowledgement. This research is supported by Alibaba-Zhejiang University Joint Research Institute of Frontier Technologies.

References

1. Aşıroğlu, B., et al.: Automatic html code generation from mock-up images using machine learning techniques. In: 2019 Scientific Meeting on Electrical-Electronics & Biomedical Engineering and Computer Science (EBBT), pp. 1–4 (2019)
2. Beltramelli, T.: pix2code: generating code from a graphical user interface screenshot. In: Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems, pp. 1–6 (2018)
3. Bunian, S., et al.: Visual search for mobile user interface design. In: Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, pp. 1–14 (2021)
4. Chen, J., et al.: Unblind your apps: Predicting natural-language labels for mobile GUI components by deep learning. In: 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE), pp. 322–334. IEEE (2020)
5. Chen, K., et al.: MMDetection: Open MMLAB detection toolbox and benchmark. arXiv preprint arXiv:1906.07155 (2019)
6. Chen, X., et al.: Microsoft coco captions: data collection and evaluation server. arXiv preprint arXiv:1504.00325 (2015)
7. Dave, H., Sonje, S., Pardeshi, J., Chaudhari, S., Raundale, P.: A survey on artificial intelligence based techniques to convert user interface design mock-ups to code. In: 2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS), pp. 28–33 (2021)
8. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask r-cnn. In: Proceedings of the IEEE international conference on computer vision, pp. 2961–2969 (2017)
9. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
10. Kolesnikov, A., Kuznetsova, A., Lampert, C., Ferrari, V.: Detecting visual relationships using box attention. In: Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops (2019)
11. Kumar, A.: Automated front-end development using deep learning. https://blog.insightdatascience.com/automated-front-end-development-using-deep-learning-3169dd086e82/ (2018)

12. lab, A.: Intelligent code generation for design drafts (2021). http://www.imgcook.com/
13. Li, G., Baechler, G., Tragut, M., Li, Y.: Learning to denoise raw mobile UI layouts for improving datasets at scale. In: CHI Conference on Human Factors in Computing Systems, pp. 1–13 (2022),
14. Lin, T.Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S.: Feature pyramid networks for object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2117–2125 (2017)
15. Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollár, P.: Focal loss for dense object detection. In: 2017 IEEE International Conference on Computer Vision (ICCV), pp. 2999–3007 (2017)
16. Lin, T., et al.: Microsoft COCO: common objects in context. CoRR (2014), arXiv:1405.0312
17. Ling, W., et al.: Latent predictor networks for code generation. arXiv preprint arXiv:1603.06744 (2016)
18. 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
19. Liu, Z., Chen, C., Wang, J., Huang, Y., Hu, J., Wang, Q.: Owl eyes: spotting UI display issues via visual understanding. In: 2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 398–409. IEEE (2020)
20. Nguyen, T.A., Csallner, C.: Reverse engineering mobile application user interfaces with remaui (t). In: 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 248–259 (2015)
21. Redmon, J., Farhadi, A.: Yolo9000: better, faster, stronger. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 7263–7271 (2017)
22. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards real-time object detection with region proposal networks. IEEE Trans. Pattern Anal. Mach. Intell. **39**(6), 1137–1149 (2017)
23. Robinson, A.: Sketch2code: generating a website from a paper mockup. arXiv preprint arXiv:1905.13750 (2019)
24. Su, T.: Fsmdroid: guided GUI testing of android apps. In: 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), pp. 689–691 (2016)
25. Wolf, T., et al.: Transformers: state-of-the-art natural language processing. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pp. 38–45 (2020)
26. Zang, X., Xu, Y., Chen, J.: Multimodal icon annotation for mobile applications. In: Proceedings of the 23rd International Conference on Mobile Human-Computer Interaction, pp. 1–11 (2021)