Overview of the 2020 ImageCLEFdrawnUI Task: Detection and Recognition of Hand Drawn Website UIs

Dimitri Fichou[1], Raul Berari[1], Paul Brie[1], Mihai Dogariu[2], Liviu Daniel Ştefan[2], Mihai Gabriel Constantin[2], and Bogdan Ionescu[2]

[1] teleportHQ, Romania, dimitri.fichou@teleporthq.io
[2] University Politehnica of Bucharest, Romania, bogdan.ionescu@upb.ro

Abstract. Nowadays, companies' online presence and user interfaces are critical for their success. However, such user interfaces involve multiple actors. Some of them, like project managers, designers or developers, are hard to recruit and train, making the process slow and prone to errors. There is a need for new tools to facilitate the creation of user interfaces. In this context, the detection and recognition of hand drawn website UIs task was run in its first edition with ImageCLEF 2020. The task challenged participants to provide automatic solutions for annotating different user interfaces elements, e.g., buttons, paragraphs and checkboxes, starting from their hand drawn wireframes. Three teams submitted a total of 18 runs using different object detection techniques and all teams obtained better scores compared to the recommended baseline. The best run in terms of mAP 0.5 IoU obtained a score of 0.793 compared to the baseline score of 0.572. The leading overall precision score was 0.970, compared to the baseline score of 0.947. In this overview working notes paper, we present in detail the task and these results.

1 Introduction

In recent years, there has been a growing interest in data-driven approaches to help user interface (UI) professionals. For instance, Deka et al. [6] collected the RICO data set containing 72,219 layouts mined from 9,722 mobile applications. Its usefulness was proven by implementing an auto-encoder which queried UIs and retrieved similar layouts. This data set was further used to create the SWIRE data set [9], consisting of 3,802 hand drawn UIs. Here, the authors demonstrated the use of nearest neighbour search to retrieve similar UI based on sketch queries. An end-to-end approach was proposed by Bertelami with pix2code [1] and UI2code [5], in both cases the authors used a Convolutional Neural Network (CNN) to encode pixels of a screenshot and a Recurrent Neural Network (RNN) to decode it into a domain specific language translatable into UI

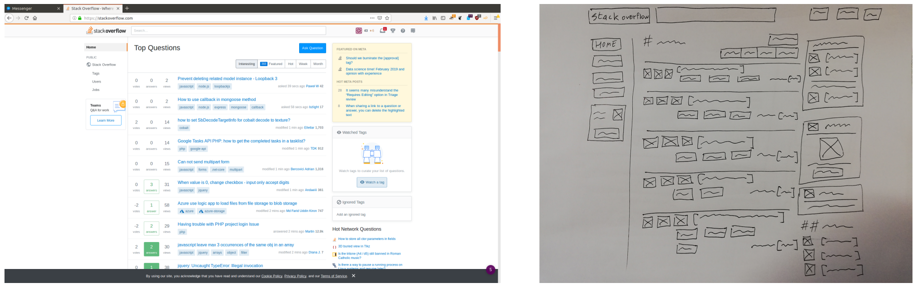

Fig. 1. Original screenshot (left) and drawn wireframe (right).

code. Online recognition of hand drawn gestures and strokes was also explored by different teams [12,13,15] to provide support for both mouse and touchpad inputs.

We proceed in this direction by organizing the first edition of the Detection and Recognition of Hand Drawn Website UIs task, ImageCLEFdrawnUI, with the ImageCLEF[3] benchmarking campaign [11], itself part of CLEF[4] (CROSS Language Evaluation Forum). Following a similar format as the 2019 ImageCLEFcoral annotation task [4], the task requires participants to perform automatic *UI element annotation and localisation* on hand drawn wireframes of websites and applications (Figure 1). In this overview working notes paper, we review the task and the submitted runs.

The rest of the paper is organized as follows: Section 2 presents the data set and how it was collected. Section 3 describes the evaluation methodology. The task results are reported in Section 4. Finally, Section 5 discusses conclusions and future work in this field.

2 Data set

The data set consists of a collection of paper-drawn website drawings, i.e., wireframes. Each data point represents an abstracted and simplified version of a real website or mobile application. Figure 1 is an example of such wireframes, corresponding to the desktop version of a website. The data set is split into 2,363 images used for training and 587 used for testing. The average number of UI elements per image is 28, with a minimum of 4 and a maximum of 131 elements.

Creating and labeling such a large number of wireframes would not have been possible in the absence of a common standard. To tackle this, a convention providing 21 of the most common UI elements was provided (Figure 2), ensuring as such that both annotators and drawers followed a single source of truth. The

[3] https://www.imageclef.org/2020/drawnui

[4] http://www.clef-campaign.org/

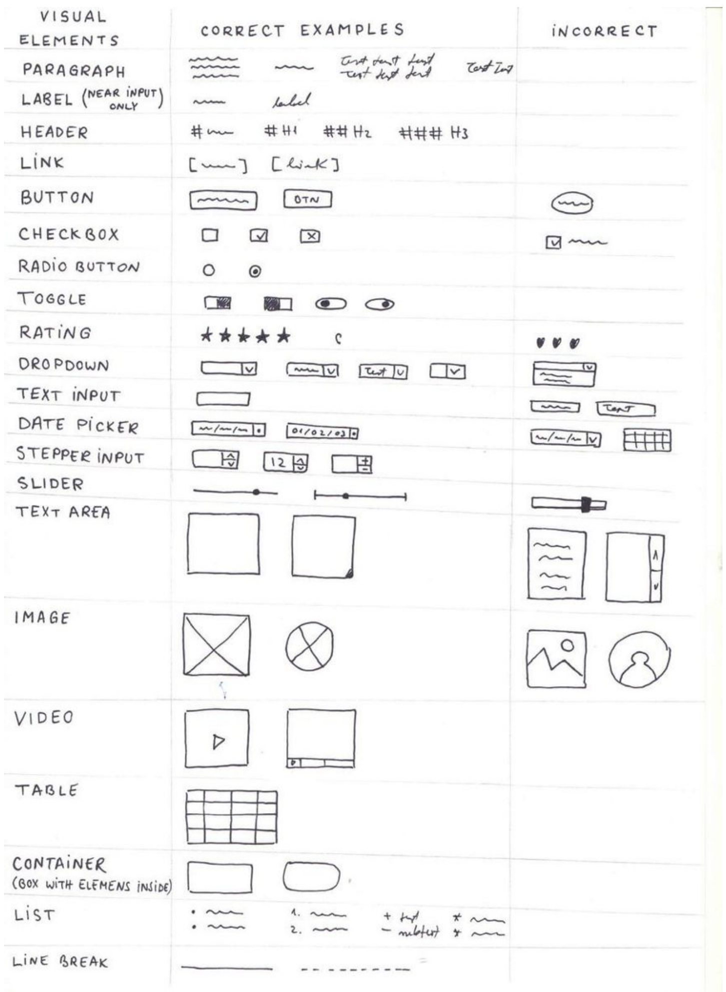

Fig. 2. Ground truth: the 21 visual representations for the UI elements.

21 elements were selected after several discussions with developers and designers, the objective being to cover most use cases.

The convention was designed to minimize the level of ambiguity for the annotators and the machine learning models. However, wireframes require simple representations and creating such a standard will necessarily produce uncertainty in some edge-cases.

In a previous iteration of the data set, a series of assumptions were made about the model capacity for distinguishing between elements based only on their position and size. For example, headers and paragraphs were represented using the same squiggly line, with the former being bigger and at the top of the page, while the latter being smaller and disseminated throughout different sections of a layout. In practice, those assumptions were ambiguous for both the annotators and the model, and a visual distinction between the two was established by placing a hash at the start of a header element.

As a result, the current version of the standard offers a clear representation for each element, while still leaving space for further improvements (i.e. more complicated UI features).

The following is a set of short descriptions of the 21 UI element classes that can be found throughout the data set:

- **Paragraph:** One or multiple lines of handwritten text or horizontal squiggles.
- **Label:** A single line of text (or a squiggly line), with the added constraint of being in the vicinity of an input (checkbox, radio button, toggle, text input, date picker, stepper input or slider).
- **Heading:** One or multiple hashes (#) succeeded by a text or squiggly line.
- **Link:** A text or squiggly line enclosed in a pair of square brackets.
- **Button:** A small rectangular shape with a single line of text or a squiggly line centered inside its area.
- **Checkbox:** A small square shape, with an X or a tick drawn inside its area.
- **Radio Button:** A small circle shape, optionally with a dot inside its area.
- **Toggle:** A small rectangle with one half of its area shaded. The rectangle can also possess rounded corners.
- **Rating:** A collection of five star shapes aligned horizontally.
- **Dropdown:** A horizontal rectangle, with a V shape drawn in its right-most side. Optionally, the empty space on the left can contain text or a squiggly line.
- **Text Input:** An empty horizontal rectangle.
- **Date Picker:** A horizontal rectangle where a dot shape is present in the right-most side and the space on the left is filled with a date text, which has to provide the slash character as a delimiter. For example, valid date texts include 02/03/04 or 20/12/20.
- **Stepper Input:** A horizontal rectangle where the right-most area consists of two small rectangles distributed vertically, with the one on the top containing a caret (^) and the one on the bottom including a V-like shape. These shapes represent the control over the input, increasing or decreasing it by a predefined step.
- **Slider:** A horizontal line with a small marker (such as a circle) between its ends.
- **Text Area:** An empty rectangle with a small triangle shaded in its bottom-right corner.
- **Image:** A rectangle or a circle with an X spanning its whole area.
- **Video:** A rectangle with a small, right-pointing triangle centered inside its area.
- **Table:** A rectangle subdivided into rectangular areas in a grid-like manner.
- **Container:** A rectangle shape which contains other classes of UI elements.

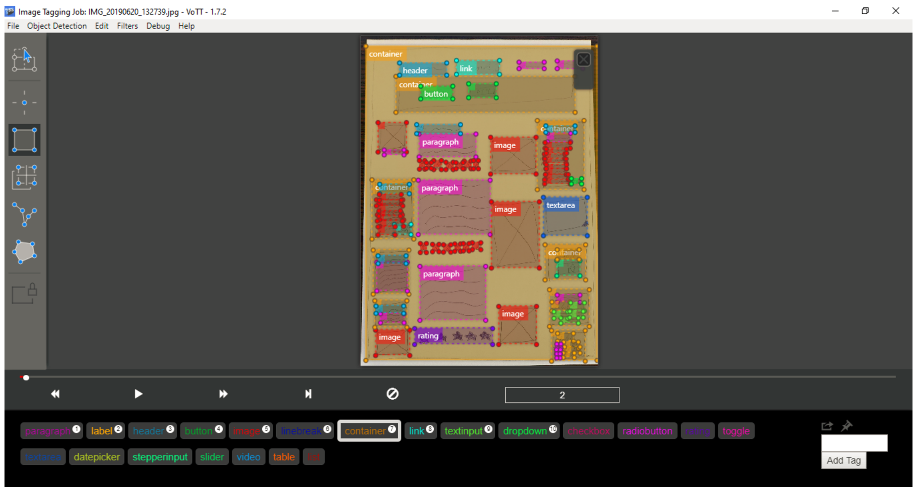

Fig. 3. Wireframe annotation using VoTT

– **List:** A collection of multiple lines (text or squiggly), each being preceded by either a bullet point or a number.
– **Line Break:** A long, horizontal line, usually spanning the whole length of a section.

Once the convention has been specified, the task of creating the data set could be split between two external teams of drawers and annotators, supplemented by overall supervision of the whole process by our team.

2.1 Acquisition

The process was externalized to a team of three professional wireframe drawers. Drawing a wireframe requires recreating an already existing mobile or desktop layout. For the former, the drawers used data points selected from the RICO data set [6], which include automatically-generated screenshots of Android applications. For the desktop layouts, a collection of screenshots was compiled by automatically processing a list of popular sites, using an in-house web parser.

The drawers followed the convention for representing UI elements when creating the wireframes. Omitted elements include the ones which did not fit into any description and those which would have cluttered the general layout. On top of this, the instructions encouraged clearly drawing each element, to ensure that annotation will proceed without difficulty. Drawers were asked to draw only the visible elements of a page. Consequently, UI elements which are concerned with providing the layout (such as containers or line breaks) were represented only when they also provided a clear, visual indication of their presence.

To diversify the data set, the drawers were asked to use different colors when drawing the wireframes. After drawing was done, each wireframe was

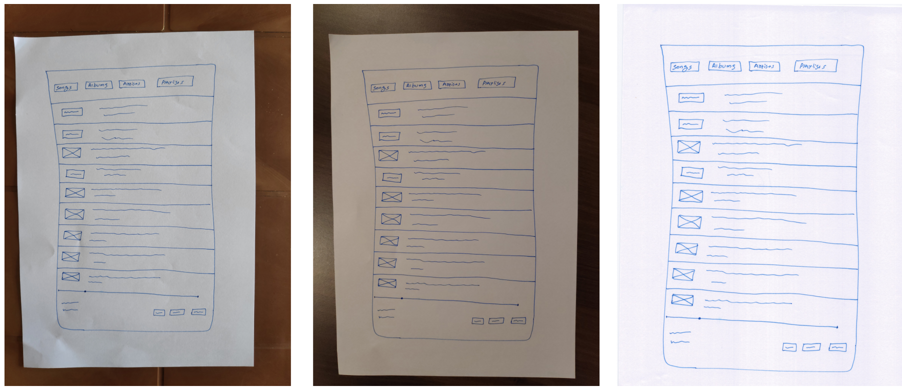

Fig. 4. Bright, dark and scanned versions of the same wireframe.

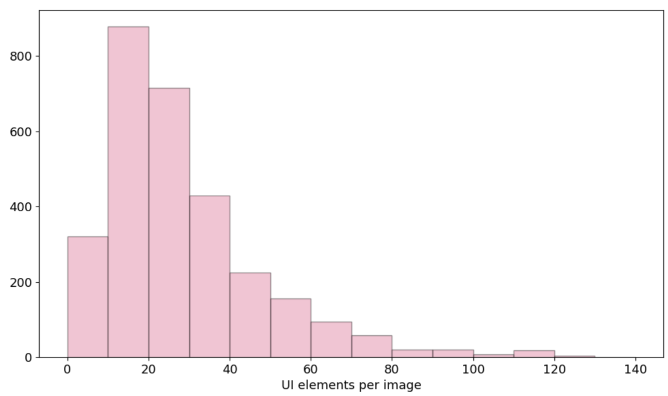

Fig. 5. Distribution of the number of UI elements per image.

photographed in three different lighting conditions: bright, dark and scanned (Figure 4). Bright and dark versions of the wireframes were photographed using a Xiaomi Poco F1 smartphone. The scanned versions were obtained using a Canon MF240 scanner.

2.2 Annotation

Image annotation was provided by a different team of three data annotators, following the same guidelines and using the desktop application VoTT[5] as an interface (Figure 3). Each element is annotated using a rectangle shape which covers the object in its entirety, regardless of potential overlap with other elements. Creating an annotation requires two clicks for drawing the bounding

[5] https://github.com/microsoft/VoTT/

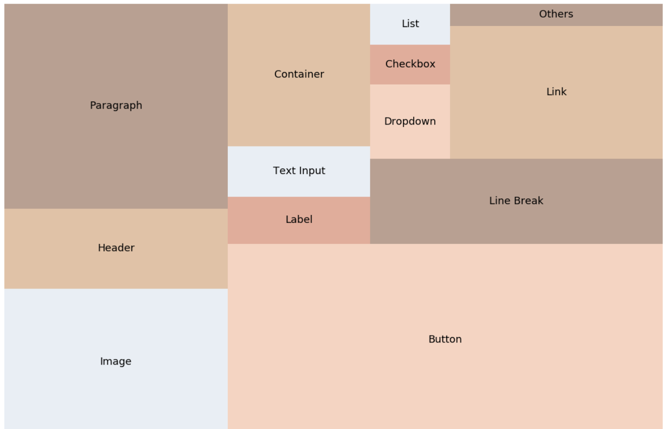

Fig. 6. Tree map with the number of UI elements per class.

box (one in the top-left corner and one in the bottom-right corner), followed by clicking on the corresponding UI element class.

In terms of quality control, the annotations were thoroughly verified and corrected by a single member of our team to ensure consistency. The process consisted in checking the alignment of the bounding boxes and rectifying erroneous labels. A small number of inconsistencies were removed as a result.

2.3 Data set analysis

As shown in Figures 5 and 6, the number of certain types of elements drawn in wireframes reflect the different use-cases solved by user interfaces. Firstly, buttons and links represent the most common ways of navigating throughout different pages of an interface. Then, elements such as paragraphs, headers or images indicate the content-based nature of UIs. Lastly, elements which describe the layout of the page (such as containers and line breaks) represent 15% of the total number of elements found inside the data set. It must be noted that the distribution of those elements is difficult to compare with the distribution found in real websites and mobile applications, as there are multiple ways to program what appears in the screenshots.

Consequently, a single wireframe will contain information about website layouts (and implicitly, hierarchies), navigation and content by using a simple, abstracted representation.

3 Evaluation methodology

Evaluation was performed using three methods, the overall precision, mean average precision (mAP) and recall [7]. For all these metrics, each detection is required to be higher than 0.5 IoU to register as a true positive when compared to the ground truth. For mAP and recall, and adapted version of the cocoAPI was used[6].

- **Overall Precision:** It refers to the number of true positive predictions out of the total sum of true positives and false negatives.
- **mAP@0.5 IoU:** The localised Mean Average Precision for each submission.
- **Recall@0.5 IoU:** The localised mean recall for each submission,

Throughout the contest, overall precision was used as the sole metric for creating the leader board, to be consistent with the other ImageCLEF challenges such as ImageCLEF Coral. However, several discussions with the participants during the competition showed that Recall and Mean Average Precision are also necessary to properly judge the results, so they have been added afterwards.

4 Results

Three teams submitted a total of 18 runs. The task had a submission limit of 10 runs per team. Table 4 displays the overall precision, mean average precision and recall at 0.5 IoU for each run.

The baseline score was obtained by training a Faster R-CNN [14] model (with a resnet101 backbone), using Tensorflow's object detection API [10] on an Amazon Web Services EC2 instance. The instance was equipped with an nVidia K80 GPU, CUDA 10.0 and Python 3.6. In terms of hyperparameters, the batch size was set to 1, the number of steps to 100,000 and the resizing range was set between 600 and 800 pixels. Data augmentation was provided by the built-in techniques of random cropping, horizontal flips, color distortion, and conversion to gray-scale.

All the participants made use of deep neural networks specifically tailored for object detection, such as Mask R-CNN [8], Cascade R-CNN [3] or YOLOv4 [2]. The baseline overall precision was surpassed by two of the teams. Furthermore, each team had at least one run which outperformed the baseline mAP and Recall scores. This confirms that this type of network remains the preferred standard for object detection and could be used for larger and more difficult projects in the domain of UI.

With regards to model settings, team OG_SouL improved their results by implementing a novel multi-pass inference technique for detecting smaller elements. After running the image through Mask R-CNN, the detected bounding boxes were filled with the color white and the image was passed once again to the model, essentially pressuring it into detecting some of the remaining elements. This method improved their mAP score from 0.573 to 0.641.

Data analysis played an important role in improving the accuracy of the results. The winning team, zip, performed a distribution analysis on the data set and split it according to an 11:1 ratio between train and validation, ensuring that

[6] `https://github.com/philferriere/cocoapi/`

Table 1. Task results: computed overall precision, mAP@0.5 IoU and Recall@0.5 IoU for each run. The baselines and best values for each metric are in bold.

Team	Run ID	Method	Overall Precision	mAP@0.5 IoU	R@0.5 IoU
zip	67816	resnet50 Faster R-CNN, full-size, grayscale	**0.970**	0.582	0.445
zip	68014	inception resnet v2 Faster R-CNN, full-size, merging	0.956	0.693	0.519
zip	68003	inception resnet v2 Faster R-CNN, full-size, grayscale	0.956	0.694	0.520
zip	67814	resnet50 Faster R-CNN, 12MP, grayscale	0.955	0.675	0.517
CudaMemError1	67972	Cascade R-CNN	0.950	0.715	0.556
CudaMemError1	67833	obj wise 2	0.950	0.681	0.533
CudaMemError1	67710	resnet101	0.949	0.649	0.505
dimitri.fichou	67413	baseline: Faster R-CNN, data augmentation	**0.947**	**0.572**	**0.403**
zip	67991	resnet50 Faster R-CNN, full-size, all data	0.944	0.647	0.472
zip	68015	inception resnet v2 Faster R-CNN, full-size, merging	0.941	0.755	0.555
OG_SouL	67391	Transfer Learning using Mask R-CNN pre-trained with COCO	0.940	0.573	0.417
zip	67733	-	0.939	0.687	0.536
CudaMemError1	67722	resnet101	0.934	0.723	0.585
CudaMemError1	67706	YOLOv4	0.934	**0.793**	**0.598**
CudaMemError1	67829	obj fusion	0.932	0.738	0.556
CudaMemError1	67707	-	0.931	0.792	0.594
CudaMemError1	67831	image wise fusion	0.929	0.791	0.600
OG_SouL	67699	Mask R-CNN, multi-pass inference, grayscale	0.918	0.637	0.501
OG_SouL	67712	Mask R-CNN, multi-pass inference	0.917	0.641	0.496

the least common elements have not been omitted. The team OG_SouL computed a similarity score between pictures of the same UI layout found throughout the data set and removed them from training to prevent over-fitting.

Data set augmentation was also provided through a variety of methods. Team CudaMemError1 used YOLOv4 techniques such as CutMix or Mosaic, where different areas of the pictures are cropped and combined with others to produce synthetic data points. Team zip generated 500 new images containing the least common UI elements by cropping them from the original data set, applying affine transformations and pasting them on randomly sized papers which used a light color as background.

Finally, two of the teams used conversion to black and white or gray scale, taking advantage of the fact that drawn UI elements are agnostic to the color of the instrument used for representing them on paper. OG_SouL showcased an OpenCV RGB to BW pipeline, applying brightness refining, erosion, Otsu's binarization algorithm, Gaussian thresholding and noise removal.

5 Discussion and conclusions

An overall precision as high as 0.97 may implicate that the task has already been solved, but this metric does not properly account for a high number of false negatives or poor results on the rarer elements from the data set. Mean Average Precision and Recall are better ways of representing the performance of a model in this case. Accordingly, the best run achieved only 0.79 on the mAP@0.5 IoU measure, indicating that the techniques could still be further improved.

Since the drawn wireframes have been modeled after real applications and websites, the data set was skewed from the start towards certain classes of UI elements. This meant that the teams went at great lengths to compensate for the lack of uncommon elements. As a result, synthetic data was created either through YOLOv4's 'cut-and-paste' techniques or by cropping the least common elements and applying affine transformations to them. Conversion to gray-scale or black and white proved to be an efficient method for reducing the data set file size and improving the detection score.

The results and techniques presented by the teams are encouraging and show the untapped potential provided by combining machine learning with user interfaces. For the next editions of the task, we plan to tackle two, more difficult problems regarding UI element detection and processing.

The first problem consists in predicting the nested structure of a UI based on a wireframe drawing. While the current challenge assumed that UI elements' locations are identified by their absolute positioning, without any hierarchical relationship between them, real-life scenarios presuppose relative positioning and a hierarchy built out of different classes of elements. This particularly challenging task may be solved through a mix of natural language processing and computer vision.

The second problem would necessitate performing object detection on real screenshots instead of wireframes. In practice, designers often attach screenshots of similar layouts along with the wireframes, with the intention of giving the developer a more refined idea of the task at hand. A data set similar to the current one can be generated by parsing a number of websites, analysing their respective DOM trees and then screen-capturing the visible area. However, due to the nature of most websites found throughout the web, cleaning the whole data set can become a laborious and time-consuming problem. Consequently, manual cleaning will be provided only for the test set, while the training set will be offered in its raw form. The task will require the participants to filter through the data set on their own.

References

1. Beltramelli, T.: pix2code : Generating Code from a Graphical User Interface Screenshot. Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems pp. 1–9 (2018)
2. Bochkovskiy, A., Wang, C.Y., Liao, H.Y.M.: Yolov4: Optimal speed and accuracy of object detection. ArXiv **abs/2004.10934** (2020)
3. Cai, Z., Vasconcelos, N.: Cascade R-CNN: Delving into High Quality Object Detection. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition pp. 6154–6162 (2018). https://doi.org/10.1109/CVPR.2018.00644
4. Chamberlain, J., Campello, A., Wright, J., Clift, L., Clark, A., Seco De Herrera, A.G.: Overview of ImageCLEFcoral 2019 task. CEUR Workshop Proceedings **2380**, 9–12 (2019)
5. 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. International Conference on Software Engineering **6** (2018)
6. Deka, B., Huang, Z., Franzen, C., Hibschman, J., Afergan, D., Li, Y., Nichols, J., Kumar, R.: Rico: A mobile app dataset for building data-driven design applications. In: UIST 2017 - Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology. pp. 845–854 (2017). https://doi.org/10.1145/3126594.3126651
7. Everingham, M., Gool, L.V., Williams, C.K.I., Winn, J., Zisserman, A.: The PASCAL Visual Object Classes (VOC) Challenge. Int J Comput Vis **88**, 303–338 (2010). https://doi.org/10.1007/s11263-009-0275-4
8. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask R-CNN. IEEE Transactions on Pattern Analysis and Machine Intelligence **42**(2), 386–397 (2020). https://doi.org/10.1109/TPAMI.2018.2844175
9. Huang, F., Canny, J.F., Nichols, J.: Swire: Sketch-based User Interface Retrieval. pp. 1–10. CHI '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3290605.3300334, `https://doi.org/10.1145/3290605.3300334`
10. Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S., Murphy, K.: Speed/accuracy trade-offs for modern convolutional object detectors. In: Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017. vol. 2017-January, pp. 3296–3305 (2017). https://doi.org/10.1109/CVPR.2017.351
11. Ionescu, B., Müller, H., Péteri, R., Abacha, A.B., Datla, V., Hasan, S.A., Demner-Fushman, D., Kozlovski, S., Liauchuk, V., Cid, Y.D., Kovalev, V., Pelka, O., Friedrich, C.M., de Herrera, A.G.S., Ninh, V.T., Le, T.K., Zhou, L., Piras, L., Riegler, M., l Halvorsen, P., Tran, M.T., Lux, M., Gurrin, C., Dang-Nguyen, D.T., Chamberlain, J., Clark, A., Campello, A., Fichou, D., Berari, R., Brie, P., Dogariu, M., Ştefan, L.D., Constantin, M.G.: Overview of the ImageCLEF 2020: Multimedia retrieval in medical, lifelogging, nature, and internet applications. In: Experimental IR Meets Multilinguality, Multimodality, and Interaction. Proceedings of the 11th International Conference of the CLEF Association (CLEF 2020), vol. 12260. LNCS Lecture Notes in Computer Science, Springer, Thessaloniki, Greece (September 22-25 2020)
12. Kieffer, S., Coyette, A., Vanderdonckt, J.: User interface design by sketching: A Complexity Analysis of Widget Representations. EICS p. 57 (2010). https://doi.org/10.1145/1822018.1822029

13. Mohian, S., Csallner, C.: Doodle2App: Native app code by freehand UI sketching. In: Proc. 7th IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft), Tool Demos and Mobile Apps Track. ACM (May 2020), to appear.
14. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In: Advances in Neural Information Processing Systems (2015)
15. Vanderdonckt, J., Roselli, P., Pérez-Medina, J.L.: !FTL, an Articulation-Invariant Stroke Gesture Recognizer with Controllable Position, Scale, and Rotation Invariances. 20th ACM International Conference on Multimodal Interaction pp. 125–134 (2018). https://doi.org/10.1145/3242969.3243032