VisionAPI: An API for Offline and Online Segmentation and Identification of Hand-Sketched Graphical User Interfaces

Paul Brie
teleportHQ
Cluj-Napoca, Romania
paul.brie@teleporthq.io

Nicolas Burny
Université catholique de Louvain
Louvain Research Institute in Management and Organizations
Louvain-la-Neuve, Belgium
nicolas.burny@uclouvain.be

Jean Vanderdonckt
Université catholique de Louvain
Louvain Research Institute in Management and Organizations
Louvain-la-Neuve, Belgium
jean.vanderdonckt@uclouvain.be

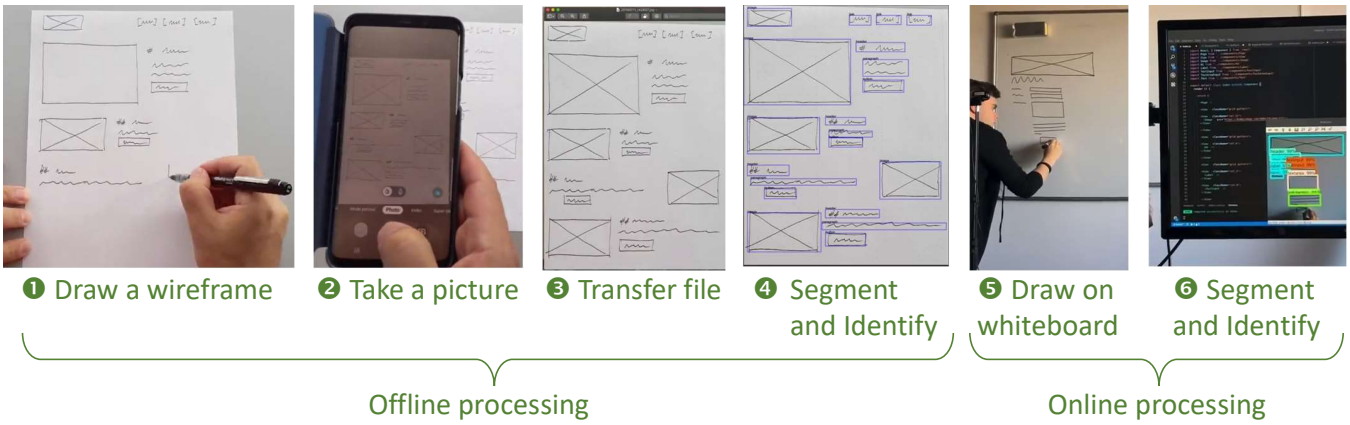

Figure 1: Offline (left) and online (right) processing of a hand-sketched graphical user interface: a practitioner draws a user interface wireframe on a piece of paper ❶, takes a picture of it with a smartphone ❷, and transfers ❸ the picture file to VisionAPI to automatically segment it and identify its elements ❹; a practitioner sketches a user interface with a marker on a whiteboard ❺, which is filmed and transferred in real-time to VisionAPI for immediate segmentation and identification.

ABSTRACT

Segmentation and identification of a graphical user interface consist of detecting the location, dimensions, and arrangement of elements of the user interface, such as controls, labels, images, and icons, and recognizing them, respectively. While these problems have been already addressed for a graphical user interface stored in a file and processed offline, it has received less attention for online processing when the interface evolves and is expressed in different formats, such as a whiteboard drawing or a paper sketch. To overcome these limitations, we present VisionAPI, an application programming interface trained for segmenting and identifying elements of a hand-sketched graphical user interface both offline and online using computer vision. For this purpose, we rely on a software architecture based on Resnet101 to extract features and Faster R-CNN to build boundary boxes to obtain an 85% recognition rate for 21 classes of elements found in graphical user interfaces: paragraph, dropdown list, checkbox, radio button, rating, toggle button, text area, date picker, stepper input, slider, video, label, table, list, header, button, image, linebreak, container, link, and text input.

CCS CONCEPTS

• **Human-centered computing** → **Graphical user interfaces**; *Web-based interaction*; **User interface programming**; **Systems and tools for interaction design**; Wireframes; • **Computing methodologies** → *Video segmentation*; *Computer vision representations*; **Computer vision**; *Object recognition*; *Neural networks*; • **Software and its engineering** → Software architectures.

KEYWORDS

Application Programming Interface; Convolutional Neural Network; Edge detection; Faster R-CNN; Gestural input; GUI element; GUI layout; User Interface segmentation; Widget identification

ACM Reference Format:
Paul Brie, Nicolas Burny, and Jean Vanderdonckt. 2023. VisionAPI: An API for Offline and Online Segmentation and Identification of Hand-Sketched Graphical User Interfaces. In *Companion of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '23 Companion), June 27–30, 2023, Swansea, United Kingdom.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3596454.3597184

EICS '23 Companion, June 27–30, 2023, Swansea, United Kingdom

ACM ISBN 979-8-4007-0206-8/23/06…$15.00
https://doi.org/10.1145/3596454.3597184

1 INTRODUCTION

Segmentation is understood in general as the process of dividing a Graphical User Interface (GUI) into distinct segments based on common properties, such as by task, domain, or user group [3]. More specifically in the context of engineering user interfaces, GUI *segmentation* [6] is hereby referred to as the process of receiving as input a GUI image in a certain format and producing as an output a structure of its GUI segments or regions in common terms of position (*e.g.*, x and y coordinates), dimensions (*e.g.*, length and height in pixels), and arrangement (*e.g.*, layout composition and depth level of elements in the layout hierarchy, such as components in a group box included in a dialog box). Furthermore, the segmentation is intrinsically associated with the complementary *identification*, which is hereby referred to as the process of recognizing each GUI element (*e.g.*, a push button) of the submitted GUI along with any property relevant for further analysis and reasoning about elements (*e.g.*, background and foreground colors, label, icon, activation).

GUI segmentation is typically used for layout testing and analysis [30], computing aesthetic metrics [23, 47], usability metrics [8, 35], adaptivity metrics [13, 16], or evaluation metrics [10, 29, 32], such as complexity [36, 43]. As the formula of these metrics is based on the position, dimensions, and arrangement of GUI elements, their reliable and consistent definition is crucial. For example, Balores [23] estimates the GUI balance based on Ngo's formula [28] by computing the difference between the total weights of the various GUI elements of each part of the vertical and horizontal axes, each element represented by its surface calculated from its dimensions. Since the correctness of the formula depends on the segmented GUI elements, relying on a consistent segmentation method is essential. Bourguet [6] reported that automatic and manual segmentations yield different segmentation results, which produce varying, thus inconsistent, metric measures for the same GUI. Even when participants are instructed to manually segment a GUI according to precise guidelines (*e.g.*, delineate a segment around each push button, but not on its label), the segmented elements may be different with a precision that depends on how the method is supported. To automatically support this process, a dilation Canny edge detector is combined with a special bounding box tool to rearrange areas in a hierarchy [6].

Manually measuring segments in a screenshot with a rule is error prone [42]. Questihm [47] enables participants to draw a rectangle by direct manipulation over a GUI layout to automatically deduce its position, dimensions, and arrangement, but this mixed-initiative process still relies on the seriousness with which the participant applies segmentation guidelines. Automatic segmentation, such as in [45, 49], usually assumes that the GUI of interest is captured in a file in a certain format (*e.g.*, a PNG file, a MPEG-7 file is only accepted in [44]) and submitted at design time to the software to operate the algorithm, thus making it tedious, if not irrelevant, to apply it at run time, when the GUI changes dynamically and where the capability to store an ongoing GUI in a file does not necessarily exist. In addition to GUI segmentation, GUI identification is a key part of more sophisticated processes, such as widget recognition in sketch-based design [15, 38, 39], GUI accessibility (*e.g.*, to recognize GUI elements so that assistive technologies for people with disabilities can access these elements [12, 48]), GUI reverse engineering [5], and GUI modernization [31]. GUI identification heavily relies on the underlying GUI representation for which the algorithm is often specifically tailored. These representations range from low fidelity (*e.g.*, hand sketched widgets [39]) to high fidelity[38], or continuously between [15].

To overcome these limitations, we present VisionAPI, an Application Programming Interface (API) trained for automatically segmenting and identifying GUI elements at run-time independently of any input format. Section 2 briefly reviews recent efforts achieved in this area and shows that several APIs have been developed to assist designers and developers in the design of GUIs of interactive systems. Section 3 then presents an API for automatic segmentation and identification of hand-sketched graphical user interfaces based on computer vision. Section 4 concludes the paper and presents some future avenues for this work.

2 RELATED WORK

2.1 GUI Segmentation and Identification

In the past, GUI segmentation [3] has used a number of classical methods, often borrowed from the field of pattern matching, such as edge detection, and pattern matching. Originally developed for document segmentation, Vision-based page segmentation (VIPs) [9] is a pioneering algorithm that uses adaptive clustering to detect and segment text, images, and graphics from a scanned document image. Successfully applied to GUIs, it first detects the layout and then clusters pixels with similar characteristics to identify each region. Since VIPs has been shown to perform well on a variety of GUI images, it rapidly became a popular choice for GUI segmentation [31] and has known several improvements, such ExtendedVIPs [4] and V-LBE (VIPs based Layout Block Extraction) [45], which selects all the layout segment candidates whose sizes are above a threshold and then deletes or inserts segments to construct a set of un-overlapped large blocks which cover the whole layout.

Later on, with the advent of artificial intelligence applied to image processing, rich feature hierarchies [19] has become a key component of many state-of-the-art computer vision models for various tasks such as object detection and semantic segmentation. These models use deep neural networks to extract features from images at multiple levels of abstraction, allowing them to capture both low-level details (*e.g.*, position and dimensions) and high-level semantic concepts (*e.g.*, arrangement and element types in their context) to make accurate predictions [11]. By creating a feature hierarchy, these models efficiently process GUI images at different resolutions and scales, allowing them to segment the entire layout with various sizes and shapes with accuracy [49]. Segmentation can be operated manually [42], in a mixed-initiative fashion [15], or completely automatically [45, 49].

By comparing a manual *vs.* automatic segmentation of 10 web pages, [6] concluded that four Ngo metrics [28] and one color metric yield different values, sometimes without any correlation between them. In addition, some metrics, such as balance, symmetry, and proportion, have been demonstrated to be significantly different when segmentation was automatic over manual, although they were based on the same definition. GUI identification is the process of identifying and categorizing different GUI elements or widgets such as buttons, menus, text boxes, and icons that are displayed.

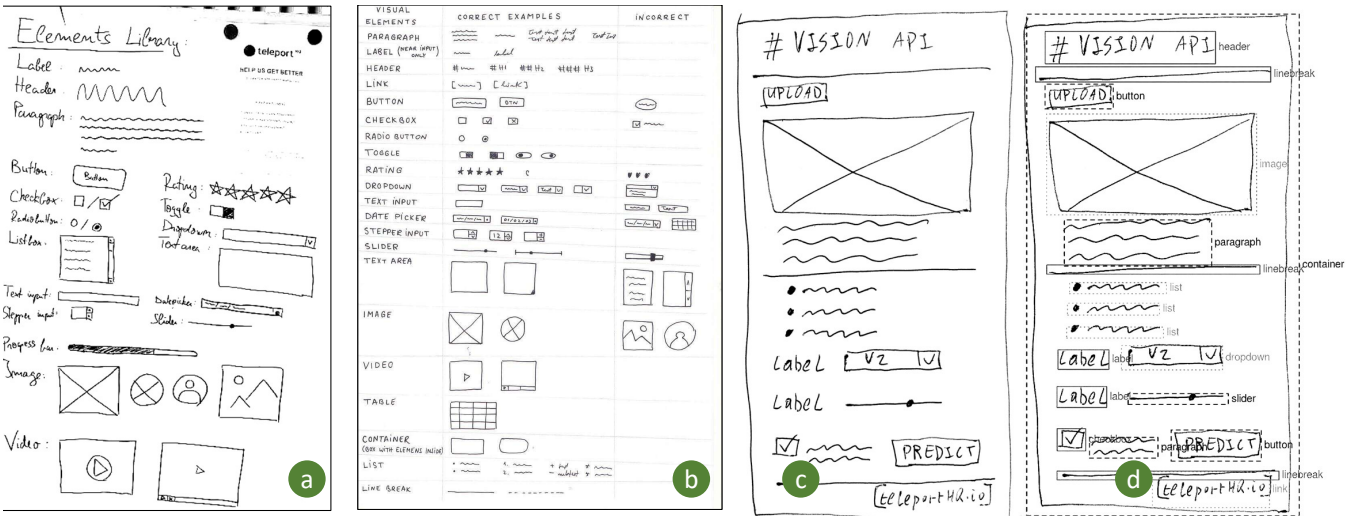

Figure 2: Representation of GUI elements: (a) initial version; (b) final version with admissible *vs.* rejected representations; (c) input example; (d) output example.

This process is essential in UI automation and accessibility testing [11] as it allows the software to simulate and test user behavior or to ensure that they are accessible to people with disabilities. Various widget identification tools and libraries are available for the user interface that use methods such as Optical Character Recognition (OCR), Document Object Model (DOM) parsing for a web page [5], and pattern recognition [15]. For example, UISeg [12] is a unified vision-based algorithm that uses GUI screenshots to estimate the structure using edge detection and widget detection heuristics with a pairwise distance function and a threshold selection algorithm to cluster segmented regions in a hierarchy. IconIntent [46] is an application analysis framework that combines static code program analysis and vision-based icon classification to identify sensitive GUI widgets in Android mobile user interfaces.

2.2 Application Programming Interfaces

The aforementioned algorithms are typically delivered as separate tools and libraries that are sometimes difficult to reuse and integrate into larger processes. We believe that an API, which is a set of defined rules that enable different applications to communicate with each other, could deliver a better and easier service for GUI segmentation and identification. It acts as an intermediary layer that processes the transfer between the GUI source and its resulting segmentation and identification, thus letting designers and developers become eager to reuse it in their GUI development life cycle. Indeed, several APIs have been developed to facilitate the work of designers and developers while addressing usability [26], such as for creating cross-device GUI notifications by scripting [14], for handling full-body gesture interaction [27], for model discovery [18], and for using phones as gateways to prototype GUIs using web scripting [24]. Apigee, Google Cloud's native API management tool, reported that 77% of companies rate APIs "important" to making their systems and data available

3 VISIONAPI, AN API FOR GUI SEGMENTATION AND IDENTIFICATION

VisionAPI consists of a computer vision API specifically trained to segment a GUI captured as a picture or video in real-time and to identify its elements with relevant properties such as colors, fonts, images, and styles. This API can be integrated with any front-end development tool and enables web designers and developers to quickly prototype, design, and build high-fidelity GUIs. Based on a common representation of GUI elements, it uses a software architecture based on Resnet101 for extracting features and Faster R-CNN for bounding-box proposals, and TensorFlow to create and train the machine learning model based on the common representation.

3.1 Towards a Common Element Representation

Since stakeholders involved in the GUI development life cycle have different backgrounds, they invent and manipulate different representations of the same GUI elements, which inevitably poses the problem of coming up with a common element representation. Coyette et al. [15] applied the de Borda count method to collect and classify the most preferred representations in SketchiXML to properly recognize them with a recognition rate of around 90%. Some elements are supported only by one sketch representation while others accommodate up to three different representations. The system enables the end user to customize the representation of any element by drawing 3-5 new templates to train the model. Kieffer et al. [21] show that the level of fidelity of the representation of a GUI element, the number of constraints imposed on the representation, and the visual difference between the representations influence the sketching activity and the recognition rate. This is why Uiskei [40] compared different representations to find the best stroke recognizers to optimize their recognition while sketching. SketchiXML used grammar to associate the visual representation of any GUI element to its internal description in UsiXML.

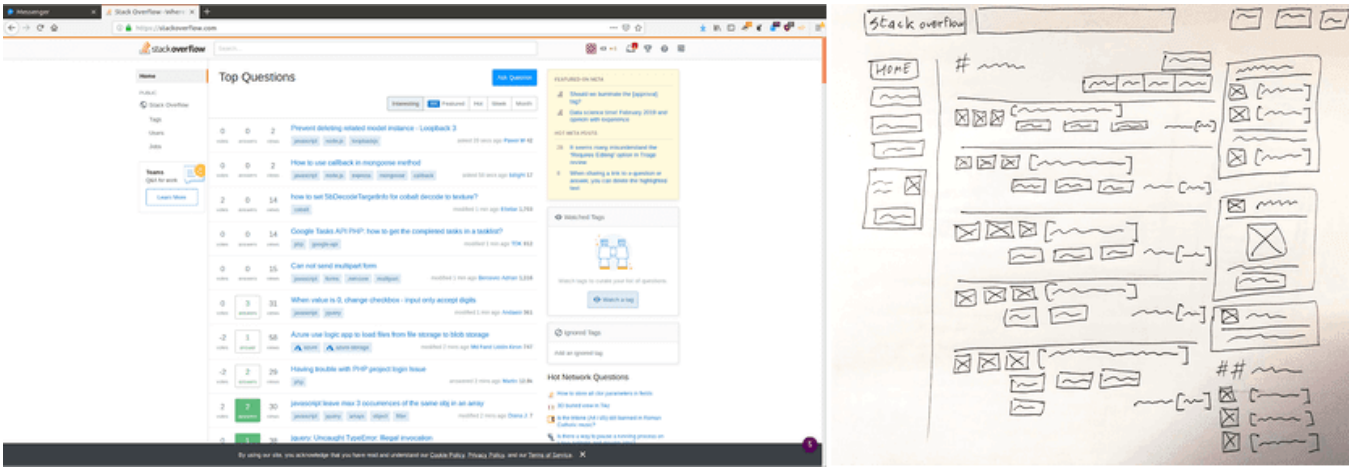

Figure 3: Example of a sketch from its initial GUI.

In the case of VisionAPI, we are faced with the same problem of a common representation of GUI elements. To address this problem, we devised an initial representation of 18 frequent elements (Fig. 2-a): paragraph, label, header, push button, check box, radio button, rating scale, toggle button, dropdown list, list box, text area, text input, date picker, stepper input, slider, progress bar, image, and video. We then asked a small set of practitioners to sketch GUIs of their choice according to this version and annotate the elements. To balance the bias and variance of the training dataset, common representations help in narrowing in on something manageable for the model to learn (thus reducing the annotation bias) while preserving a variety of GUIs drawn by different people, in different colors (thus enhancing the variety). Based on this experience, we decided on a final version (Fig. 2-b) confronting correct, therefore recognizable, representations to incorrect, therefore, not recognizable, representations. We then asked members of a large software development company (>300 employees) to draw and collect 5-10 GUIs found in repositories or of their choice according to this final version. For example, The Webzeitgeist repository [22] was obtained by a crawling seeded with 20 URLs, including the Alexa Top 500 US sites, the Webby Awards gallery, and popular design blogs. To build a diverse dataset, participants were designers, developers, project managers, and graphic artists. We then tested the recognizability of the sketches (Fig. 2-c and -d). Fig. 3 shows an example of a real-world GUI transformed into a sketch that was annotated in the dataset.

3.2 Implementation of VisionAPI

First, we used **Faster R-CNN** [34] for segmenting each GUI sketch of the dataset into regions. This neural network introduced a Region Proposal Network (RPN), one of the best existing region detection networks that share full-image convolutional features with the detection network, which significantly reduces the segmentation time and complexity. An RPN is a fully convolutional network that simultaneously predicts GUI element edges and objectness scores at each location. The RPN is trained for segmentation, which is then used by Faster R-CNN for real-time detection [34]

Second, we used **ResNet-101** [20] to extract features from each segmented element in the GUI. ResNet-101 was considered appropriate for this task because this convolutional neural network is 101 layers deep [20] and produces a version pre-trained on more than a million images from the ImageNet database, thus resulting in a classification of 1000 object categories, such as keyboard, mouse, pencil, and many animals. Since this network benefits from a rich feature representation for a wide range of images that could be of large size, up to 224×224–, it enables classifying new types of images using the ResNet-101 model according to the GoogLeNet procedure [41]. An important consideration in training our machine learning model is the validity of the dataset in terms of width (all GUI elements should be represented) and depth (each class of GUI elements should be evenly represented). In studying the annotations in our dataset, we found that some classes were underrepresented while others were overrepresented depending on the usual frequency of GUI elements found in layouts, which render some elements more challenging to detect, such as table, rating, list, text area, and stepper. In fact, Sahami Shirazi et al. [37] showed that the most common combination of GUI elements in a mobile layout consists of three nested elements for 5% of all GUI elements, which is more frequent than progress bars and checkboxes, and that the ten most frequent patterns cover 21% of all GUI elements and are more frequent than common widgets such as radio buttons and spinners. When performing the analysis, we also examined the size distribution for each element, a crucial characteristic in object detection datasets. Some elements are smaller, like links and radio buttons, and naturally, these are harder to detect, while larger elements, such as a container or an image, are easier to detect (see Fig. 4 for the distribution of >7k samples in our dataset).

Third and finally, we used **TensorFlow** [1, 2] to create and train the final machine learning, since TensorFlow is an open source software platform for developing and training machine learning models developed by Google Brain Team for a variety of tasks, including recognition of images and speech, natural language processing, and data analysis. It was therefore considered particularly appropriate for our purpose.

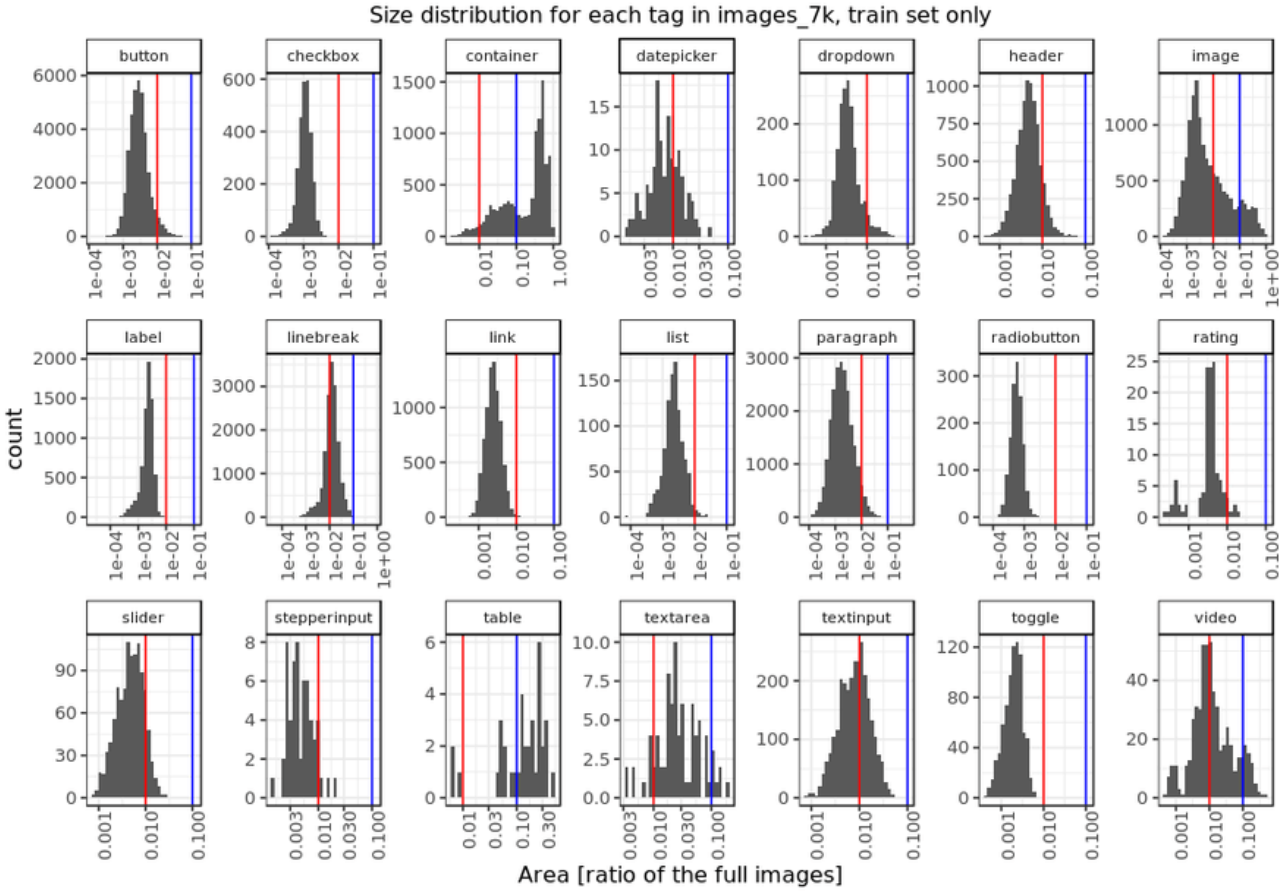

Figure 4: Size distribution of GUI elements in the training set: red and blue lines represent the 1% and 10% thresholds, resp.

3.3 Results and Limitations

To evaluate our model, we used the Mean Average Precision at Intersection over Union of 50%, or for short mAP@IOU 0.5. This is a performance metric typically used in object detection tasks to measure the accuracy of a model's predictions [17]. This metric calculates the precision of the model's predictions at a threshold of 0.5 Intersection over Union (IOU) against the ground truth annotations. IOU measures the overlap between the predicted bounding box and the ground truth bounding box. So, a model with a high mAP@IOU 0.5 score indicates that it can accurately identify and localize objects in an image with good precision, with at least 50% overlap between the predicted and ground-truth bounding boxes. Using this metric, the performance obtained with the last version is 85% on the test set. IOU values are typically located in the range of 0.5 and 0.7, which is considered acceptable, for example, for images where large relative areas are covered by foreground objects [17]. Beyond this encouraging result, we observed the following limitations.

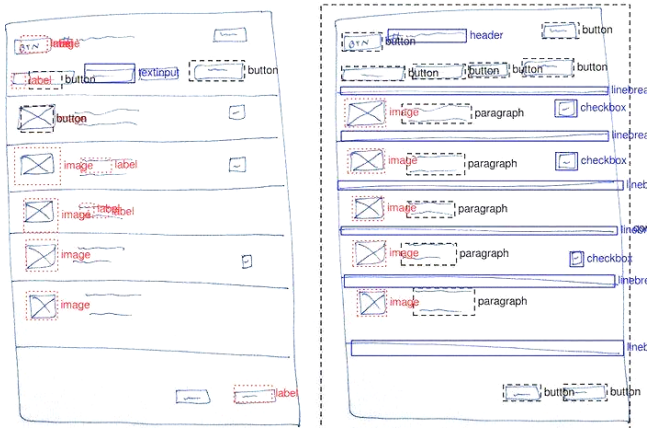

Figure 5: Prediction comparison between the previous (left) and the current version (right).

- The model has difficulty segmenting GUIs decomposed into frames that contain many individual elements or repeated GUI elements, particularly when they are close to each other. This is caused not only by the proximity of existing GUI elements but also by the angle with which the GUI is captured.

For example, Fig. 5 compares a previous version of Vision-API with its current version regarding the segmentation of a GUI consisting of several frames, some of them repeating the same structure. Fig. 6 shows another challenging example containing my repeated sequences of the same elements,

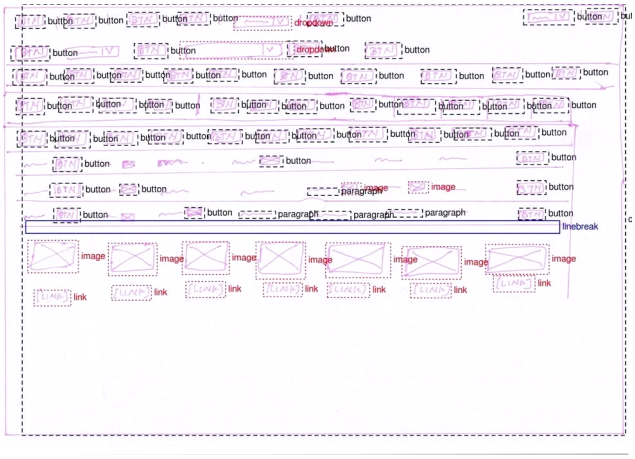

Figure 6: Example of a challenging GUI to segment due to its repeated structure.

where each individual element was correctly segmented but not grouped.

- The uneven dataset poses additional complexity when uncommon classes are used, such as sliders and ratings. For example, a rating bar typically consists of several smaller pieces that are aligned in a row, but hard to estimate as grouped. Rather, individual elements are segmented as separate, unrelated elements. The slider also consists of several elements with different boundary boxes, which makes it difficult to detect.
- The enforcement of using the trained representations is key. Despite the common representation explicitly defined and communicated, deviations from these representations occur rapidly, thus lowering the segmentation prediction. Of course, other representations could be included as in SketchiXML[15] which enables the end user to introduce personalized representations in the recognition process, but at the price of some confusion for those who will use them. Any personalized representation may not work for other end users who are not aware of these new representations.
- Custom widgets or infrequent widgets are not covered. When a GUI includes a custom widget, which is usually programmatically defined on demand and requires additional efforts [33], or infrequent widgets (*e.g.*, a carousel), there is currently no support for recognizing these elements.

4 CONCLUSION

In this paper, we present VisionAPI, an application programming interface trained for segmenting and identifying elements of a graphical user interface at run-time using computer vision by relying on Faster R-CNN for segmenting the layout into elements, Resnet101 for extracting features of each segmented element, and TensorFlow for creating and training the model, of which a prediction rate of 85% has been obtained. This encouraging result, together with other positive results obtained in image segmentation in general, reinforces the impression that the classical segmentation algorithms used so far are being superseded by machine learning algorithms as long as the prediction model is sufficiently well trained. This is largely dependent on the balance between the diversity of the elements subject to training and the representation used. The more different representations there are for the same element, the more possibilities there are to recognize an element accurately or not, but the more confusion there is induced in the end user.

OPEN SCIENCE

The GitHub repository containing related material is publicly accessible at https://github.com/teleporthq/teleport-vision-api. A video demonstrating the online processing of a hand-sketched graphical user interface is accessible at https://www.youtube.com/watch?v=_oet4GOzcRQ. The VisionAPI is also used in teleportHQ, a low-code platform that automatically generates code, in particular for the user interface based on OpenUIDL [25] and in GPT-UI [7].

ACKNOWLEDGMENTS

The authors of this paper are very grateful to the anonymous EICS reviewers and the associate chair for their insightful and constructive comments on earlier versions of this manuscript. We thank Alex Pausan, Dimitri Fichou, and Raul C. Incze for developing and testing the API. Nicolas Burny and Jean Vanderdonckt are supported by the EU EIC Pathfinder-Awareness Inside challenge "Symbiotik" project (1 Oct. 2022-31 Dec. 2026) under Grant no. 101071147.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. https://doi.org/10.48550/ARXIV.1603.04467

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Savannah, GA, USA) *(OSDI'16)*. USENIX Association, USA, 265–283.

[3] Simone Agostinelli, Francesco Leotta, and Andrea Marrella. 2021. Interactive Segmentation of User Interface Logs. In *Service-Oriented Computing*, Hakim Hacid, Odej Kao, Massimo Mecella, Naouel Moha, and Hye-young Paik (Eds.). Springer International Publishing, Cham, 65–80. https://doi.org/10.1007/978-3-030-91431-8_5

[4] M. Elgin Akpınar and Yeliz Yesilada. 2013. Vision Based Page Segmentation Algorithm: Extended and Perceived Success. In *Current Trends in Web Engineering*, Quan Z. Sheng and Jesper Kjeldskov (Eds.). Springer International Publishing, Cham, 238–252. https://doi.org/10.1007/978-3-319-04244-2_22

[5] Laurent Bouillon, Quentin Limbourg, Jean Vanderdonckt, and Benjamin Michotte. 2005. Reverse engineering of Web pages based on derivations and transformations. In *Proceedings of the IEEE Third Latin American Web Congress* (Buenos Aires, Argentina) *(LA-WEB 2005)*. IEEE Computer Society Press, Los Alamitos, USA, 11 pp.–. https://doi.org/10.1109/LAWEB.2005.29

[6] Marie-Luce Bourguet. 2018. Metrics-Based Evaluation of Graphical User Interface Aesthetics: The Segmentation Problem. In *Proceedings of the 2018 ACM Companion International Conference on Interactive Surfaces and Spaces* (Tokyo, Japan) *(ISS '18 Companion)*. Association for Computing Machinery, New York, NY, USA, 31–38. https://doi.org/10.1145/3280295.3281747

[7] Paul Brie, Nicolas Burny, Arthur Sluÿters, and Jean Vanderdonckt. 2023. Evaluating a Large Language Model on Searching for Layouts of Graphical User Interfaces. *Proc. ACM Hum.-Comput. Interact.* 7, EICS, Article 178 (jun 2023), 37 pages. https://doi.org/10.1145/3593230

[8] Nicolas Burny and Jean Vanderdonckt. 2021. GUIMETRICS: An Extensible Cloud-based Application for Automatic Computation of GUI Visual Design Measures. In *Proceedings of the 16th International Conference on Software Technologies (ICSOFT 2021)*, Hans-Georg Fill, Marten van Sinderen, and Leszek A. Maciaszek (Eds.). SCITEPRESS, 505–512. https://doi.org/10.5220/0010571605050512

[9] Deng Cai, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma. 2003. *VIPS: a Vision-based Page Segmentation Algorithm.* Technical Report MSR-TR-2003-79. Microsoft Corp. http://www.cad.zju.edu.cn/home/dengcai/VIPS/VIPS_July-2004.pdf

[10] Murilo C. Camargo, Rodolfo M. Barros, and Vanessa T. O. Barros. 2018. Visual Design Checklist for Graphical User Interface (GUI) Evaluation. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing* (Pau, France) *(SAC '18)*. Association for Computing Machinery, New York, NY, USA, 670–672. https://doi.org/10.1145/3167132.3167391

[11] Alex Q. Chen, Simon Harper, Darren Lunn, and Andrew Brown. 2013. Widget Identification: A High-Level Approach to Accessibility. *World Wide Web* 16, 1 (2013), 73–89. https://doi.org/10.1007/s11280-012-0156-6

[12] Yi-An Chen. 2018. *Vision Based User Interface Segmentation Algorithm.* Ph. D. Dissertation. National Taiwan University, Department of Electrical Engineering. https://tdr.lib.ntu.edu.tw/bitstream/123456789/1161/1/ntu-107-1.pdf

[13] Neila Chettaoui and Med Salim Bouhlel. 2018. I2Evaluator: An Aesthetic Metric-Tool for Evaluating the Usability of Adaptive User Interfaces. In *Proceedings of the International Conference on Advanced Intelligent Systems and Informatics 2017*, Aboul Ella Hassanien, Khaled Shaalan, Tarek Gaber, and Mohamed F. Tolba (Eds.). Springer International Publishing, Cham, 374–383.

[14] Fulvio Corno, Luigi De Russis, and Teodoro Montanaro. 2017. XDN: Cross-Device Framework for Custom Notifications Management. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (Lisbon, Portugal) *(EICS '17)*. Association for Computing Machinery, New York, NY, USA, 57–62. https://doi.org/10.1145/3102113.3102127

[15] Adrien Coyette, Sascha Schimke, Jean Vanderdonckt, and Claus Vielhauer. 2007. Trainable Sketch Recognizer for Graphical User Interface Design. In *Human-Computer Interaction - INTERACT 2007, 11th IFIP TC 13 International Conference* (Rio de Janeiro, Brazil) *(Lecture Notes in Computer Science, Vol. 4662)*, Maria Cecília Calani Baranauskas, Philippe A. Palanque, Julio Abascal, and Simone Diniz Junqueira Barbosa (Eds.). Springer, 124–135. https://doi.org/10.1007/978-3-540-74796-3_14

[16] Charles-Eric Dessart, Vivian Genaro Motti, and Jean Vanderdonckt. 2011. Showing User Interface Adaptivity by Animated Transitions. In *Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (Pisa, Italy) *(EICS '11)*. Association for Computing Machinery, New York, NY, USA, 95–104. https://doi.org/10.1145/1996461.1996501

[17] Pritha Ganguly, Nitesh Methani, Mitesh M. Khapra, and Pratyush Kumar. 2020. A Systematic Evaluation of Object Detection Networks for Scientific Plots. https://doi.org/10.48550/ARXIV.2007.02240

[18] Andy Gimblett and Harold Thimbleby. 2010. User Interface Model Discovery: Towards a Generic Approach. In *Proceedings of the 2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (Berlin, Germany) *(EICS '10)*. Association for Computing Machinery, New York, NY, USA, 145–154. https://doi.org/10.1145/1822018.1822041

[19] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2013. Rich feature hierarchies for accurate object detection and semantic segmentation. https://doi.org/10.48550/ARXIV.1311.2524

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Las Vegas, NV, USA) *(CVPR '16)*. IEEE Computer Society Press, Los Alamitos, USA, 770–778. https://doi.org/10.1109/CVPR.2016.90

[21] Suzanne Kieffer, Adrien Coyette, and Jean Vanderdonckt. 2010. User interface design by sketching: a complexity analysis of widget representations. In *Proceedings of the 2nd ACM Symposium on Engineering Interactive Computing Systems* (Berlin, Germany) *(EICS '10)*, Noi Sukaviriya, Jean Vanderdonckt, and Michael Harrison (Eds.). ACM, 57–66. https://doi.org/10.1145/1822018.1822029

[22] Ranjitha Kumar, Arvind Satyanarayan, Cesar Torres, Maxine Lim, Salman Ahmad, Scott R. Klemmer, and Jerry O. Talton. 2013. Webzeitgeist: Design Mining the Web. In *Proceedings of the ACM Conference on Human Factors in Computing Systems* (Paris, France) *(CHI '13)*. Association for Computing Machinery, New York, NY, USA, 3083–3092. https://doi.org/10.1145/2470654.2466420

[23] Salvador González López, Francisco Montero Simarro, and Pascual González López. 2013. *BaLOReS: A Framework for Quantitative User Interface Evaluation.* Springer London, London, 127–143. https://doi.org/10.1007/978-1-4471-5445-7_10

[24] Will McGrath, Mozziyar Etemadi, Shuvo Roy, and Bjoern Hartmann. 2015. Fabryq: Using Phones as Gateways to Prototype Internet of Things Applications Using Web Scripting. In *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (Duisburg, Germany) *(EICS '15)*. Association for Computing Machinery, New York, NY, USA, 164–173. https://doi.org/10.1145/2774225.2774835

[25] Alex Moldovan, Vlad Nicula, Ionut Pasca, Mihai Popa, Jaya Krishna Namburu, Anamaria Oros, and Paul Brie. 2020. OpenUIDL, A User Interface Description Language for Runtime Omni-Channel User Interfaces. *Proc. ACM Hum.-Comput. Interact.* 4, EICS, Article 86 (jun 2020), 52 pages. https://doi.org/10.1145/3397874

[26] Brad A. Myers and Jeffrey Stylos. 2016. Improving API usability. *Commun. ACM* 59, 6 (2016), 62–69. https://doi.org/10.1145/2896587

[27] Michael Nebeling, Elena Teunissen, Maria Husmann, and Moira C. Norrie. 2014. XDKinect: Development Framework for Cross-Device Interaction Using Kinect. In *Proceedings of the 2014 ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (Rome, Italy) *(EICS '14)*. Association for Computing Machinery, New York, NY, USA, 65–74. https://doi.org/10.1145/2607023.2607024

[28] David Chek Ling Ngo, Lian Seng Teo, and John G. Byrne. 2003. Modelling Interface Aesthetics. *Inf. Sci.* 152, 1 (June 2003), 25–46. https://doi.org/10.1016/S0020-0255(02)00404-8

[29] Antti Oulasvirta, Samuli De Pascale, Janin Koch, Thomas Langerak, Jussi Jokinen, Kashyap Todi, Markku Laine, Manoj Kristhombuge, Yuxi Zhu, Aliaksei Miniukovich, Gregorio Palmas, and Tino Weinkauf. 2018. Aalto Interface Metrics (AIM): A Service and Codebase for Computational GUI Evaluation. In *The 31st Annual ACM Symposium on User Interface Software and Technology Adjunct Proceedings* (Berlin, Germany) *(UIST '18 Adjunct)*. ACM, New York, NY, USA, 16–19. https://doi.org/10.1145/3266037.3266087

[30] Irfan Prazina, Šeila Bećirović, Emir Cogo, and Vensada Okanović. 2023. Methods for Automatic Web Page Layout Testing and Analysis: A Review. *IEEE Access* 11 (2023), 13948–13964. https://doi.org/10.1109/ACCESS.2023.3242549

[31] Óscar Sánchez Ramón, Jesús Sánchez Cuadrado, Jesús García Molina, and Jean Vanderdonckt. 2016. A layout inference algorithm for Graphical User Interfaces. *Information Software Technology* 70 (2016), 155–175. https://doi.org/10.1016/j.infsof.2015.10.005

[32] David Raneburger, Roman Popp, and Jean Vanderdonckt. 2012. An automated layout approach for model-driven WIMP-UI generation. In *Proceedings of ACM Symposium on Engineering Interactive Computing Systems* (Copenhagen, Denmark) *(EICS'12)*, Simone Diniz Junqueira Barbosa, José Creissac Campos, Rick Kazman, Philippe A. Palanque, Michael D. Harrison, and Steve Reeves (Eds.). Association for Computing Machinery, New York, NY, USA, 91–100. https://doi.org/10.1145/2305484.2305501

[33] Thomas Rathfux, Roman Popp, and Hermann Kaindl. 2016. Adding custom widgets to model-driven GUI generation. In *Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (Brussels, Belgium) *(EICS 2016)*, Kris Luyten and Philippe A. Palanque (Eds.). ACM, 16–26. https://doi.org/10.1145/2933242.2933251

[34] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. https://doi.org/10.48550/ARXIV.1506.01497

[35] Andreas Riegler and Clemens Holzmann. 2015. UI-CAT: Calculating User Interface Complexity Metrics for Mobile Applications. In *Proceedings of the 14th International Conference on Mobile and Ubiquitous Multimedia* (Linz, Austria) *(MUM '15)*. Association for Computing Machinery, New York, NY, USA, 390–394. https://doi.org/10.1145/2836041.2841214

[36] Andreas Riegler and Clemens Holzmann. 2018. Measuring Visual User Interface Complexity of Mobile Applications With Metrics. *Interacting with Computers* 30, 3 (04 2018), 207–223. https://doi.org/10.1093/iwc/iwy008 arXiv:https://academic.oup.com/iwc/article-pdf/30/3/207/24805365/iwy008.pdf

[37] Alireza Sahami Shirazi, Niels Henze, Albrecht Schmidt, Robin Goldberg, Benjamin Schmidt, and Hansjörg Schmauder. 2013. Insights into Layout Patterns of Mobile User Interfaces by an Automatic Analysis of Android Apps. In *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (London, United Kingdom) *(EICS '13)*. Association for Computing Machinery, New York, NY, USA, 275–284. https://doi.org/10.1145/2494603.2480308

[38] Ugo Braga Sangiorgi, François Beuvens, and Jean Vanderdonckt. 2012. User interface design by collaborative sketching. In *Proceedings of ACM Conference on Designing Interactive Systems* (Newcastle Upon Tyne, United Kingdom) *(DIS '12)*. ACM, 378–387. https://doi.org/10.1145/2317956.2318013

[39] Paul Schmieder, Beryl Plimmer, and Jean Vanderdonckt. 2010. Generating systems from multiple sketched models. *Journal of Visual Languages and Computing* 21, 2 (2010), 98–108. https://doi.org/10.1016/j.jvlc.2009.12.003

[40] Vinícius C. V. B. Segura and Simone D. J. Barbosa. 2012. A combination of stroke manipulation and recognition strategies to support user interface construction and interactive behavior definition through sketching. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing* (Innsbruck, Austria) *(VL/HCC '12)*. IEEE Computer Society Press, Los Alamitos, USA, 45–48. https://doi.org/10.1109/VLHCC.2012.6344479

[41] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2014. Going Deeper with Convolutions. *CoRR* abs/1409.4842 (2014). arXiv:1409.4842 http://arxiv.org/abs/1409.4842

[42] Stefan Trausan-Matu and Brahma Dathan. 2016. Perceived aesthetics of user-modifiable layouts: a comparison between an unspecified design and a GUI. In *Proceedings of 13th International Conference on Human-Computer Interaction* (Iasi, Romania) *(RoCHI 2016)*, Adrian Iftene and Jean Vanderdonckt (Eds.). Matrix Rom, Bucharest, Romania, 22–25.

[43] Alexandre N. Tuch, Eva E. Presslaber, Markus Stöcklin, Klaus Opwis, and Javier A. Bargas-Avila. 2012. The role of visual complexity and prototypicality regarding first impression of websites: Working towards understanding aesthetic judgments. *International Journal of Human-Computer Studies* 70, 11 (2012), 794–811. https://doi.org/10.1016/j.ijhcs.2012.06.003

[44] Silvia Uribe, Federico Álvarez, and José Manuel Menéndez. 2017. User's Web Page Aesthetics Opinion: A Matter of Low-Level Image Descriptors Based on MPEG-7. *ACM Trans. Web* 11, 1, Article 5 (March 2017), 25 pages. https://doi.org/10.1145/3019595

[45] Ou Wu, Yunfei Chen, Bing Li, and Weiming Hu. 2011. Evaluating the Visual Quality of Web Pages Using a Computational Aesthetic Approach. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining* (Hong Kong, China) *(WSDM '11)*. Association for Computing Machinery, New York, NY, USA, 337–346. https://doi.org/10.1145/1935826.1935883

[46] Xusheng Xiao, Xiaoyin Wang, Zhihao Cao, Hanlin Wang, and Peng Gao. 2019. IconIntent: Automatic Identification of Sensitive UI Widgets Based on Icon Classification for Android Apps. In *Proceedings of the IEEE/ACM 41st International Conference on Software Engineering* (Montreal, QC, Canada) *(ICSE '19)*. 257–268. https://doi.org/10.1109/ICSE.2019.00041

[47] Mathieu Zen and Jean Vanderdonckt. 2014. Towards an evaluation of graphical user interfaces aesthetics based on metrics. In *Proceedings of IEEE 8th International Conference on Research Challenges in Information Science* (Marrakech, Morocco) *(RCIS 2014)*, Marko Bajec, Martine Collard, and Rébecca Deneckère (Eds.). IEEE, 1–12. https://doi.org/10.1109/RCIS.2014.6861050

[48] Xiaoyi Zhang, Lilian de Greef, Amanda Swearngin, Samuel White, Kyle Murray, Lisa Yu, Qi Shan, Jeffrey Nichols, Jason Wu, Chris Fleizach, Aaron Everitt, and Jeffrey P Bigham. 2021. Screen Recognition: Creating Accessibility Metadata for Mobile Applications from Pixels. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 275, 15 pages. https://doi.org/10.1145/3411764.3445186

[49] Xianjun Sam Zheng, Ishani Chakraborty, James Jeng-Weei Lin, and Robert Rauschenberger. 2009. Correlating Low-Level Image Statistics with Users - Rapid Aesthetic and Affective Judgments of Web Pages. In *Proceedings of the ACM Conference on Human Factors in Computing Systems* (Boston, MA, USA) *(CHI '09)*. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/1518701.1518703

A HOW TO SEND A REQUEST TO VISIONAPI

To use the API, any request should be sent to the API endpoint at https://api.vision.teleporthq.io/v2/detection For this purpose, the request header should be formatted as follows: a Content-Type key should be added with the value application/json and a Apias-Token key with the provided key. The body of the request is a JSON with two keys: image is a required string parameter that denotes the direct URL to a publicly available JPG, PNG image, or video; threshold is an optional parameter that specifies the confidence threshold below which VisionAPI will not consider and include segmented results in the output, ranging from 0 to 1 with a default value of 0.1. For example, the GUI in Fig. 2-c can be sent as follows:

```
{
    "image": "https://i.imgur.com/HzTWzLS.jpg",
    "threshold": 0.5
}
```

Another request example using Curl is

```
curl
  -X    -X POST https://api.vision.teleporthq.io/v2/detection\
  -H    'Content-Type: application/json' \
  -H    'Apias-Token: your_token' \
  -d    '{
        "image": "https://i.imgur.com/HzTWzLS.jpg",
        "threshold": 0.5
    }'
```

If the request is valid, VisionAPI outputs a JSON file as follows:

```
[   {
            "box": [ y, x, height, width] ,
            "detectionClass": numeric_label,
            "detectionString": string_label,
            "score": confidence_rating
    },...   ]
```

The JSON file contains a list of objects, each corresponding to a detected UI element in the image sent in the request. All keys appear in all elements of the response array as follows:

- box contains the coordinates of the bounding box surrounding the segmented element
- x and y are the coordinates of the top left corner of the box and width and height. All coordinates are normalized between [0, 1] where (0,0) is the top left corner of the element and (1, 1) is the bottom right corner. If one wants the pixel coordinates we have to multiply x and width with the image width and y with the image height.
- detectionClass is the numeric class of the detection.
- detectionString is the human-readable label of the detection.
- score represents how confident the algorithm is that the predicted object is the correct one. It takes values between [0, 1], where 1 represents 100% confidence in its detection.

VisionAPI then creates an output that represents the response to the request as follows:

```
[   {
            "box": [
                                        0.066403992474079,
                                        0.185734212398529,
                                        0.06268358975648,
                                        0.437795639038085
            ],
            "detectionClass": 15,
            "detectionString": "header",
            "score": 0.9958260059356
    },
    {
            "box": [
                                        0.168106362223625,
                                        0.185209602117538,
                                        0.047976151108741,
                                        0.175636291503906
            ],
            "detectionClass": 16,
            "detectionString": "button",
            "score": 0.99246710538864
    },...                               ]
```

The detectionClass to detectionString mapping is done according to this dictionary:

```
{
     1:   'paragraph',
     2:   'dropdown',
     3:   'checkbox',
     4:   'radiobutton',
     5:   'rating',
     6:   'toggle',
     7:   'textarea',
     8:   'datepicker',
     9:   'stepperinput',
    10:   'slider',
    11:   'video',
    12:   'label',
    13:   'table',
    14:   'list',
    15:   'header',
    16:   'button',
    17:   'image',
    18:   'linebreak',
    19:   'container',
    20:   'link',
    21:   'textinput'
}
```

Received 9 March 2023; revised 25 April 2023; accepted 8 May 2023