Sketch2Code: Automatic hand-drawn UI elements detection with Faster R-CNN

Aleš Zita[1,2], Lukáš Picek[3,5], and Antonín Říha[4]

[1] Czech Academy of Sciences, Institute of Information Theory and Automation
[2] Faculty of Mathematics and Physics, Charles University
[3] Dept. of Cybernetics, Faculty of Applied Sciences, University of West Bohemia
[4] Faculty of Information Technology, Czech Technical University
[5] PiVa AI

Abstract. Transcription of User Interface (UI) elements hand drawings to the computer code is a tedious and repetitive task. Therefore, a need arose to create a system capable of automating such process. This paper describes a deep learning-based method for hand-drawn user interface elements detection and localization. The proposed method scored 1st place in the ImageCLEFdrawnUI competition while achieving an overall precision of 0.9708. The final method is based on Faster R-CNN object detector framework with ResNet-50 backbone architecture trained with advanced regularization techniques. The code has been made available at: https://github.com/picekl/ImageCLEF2020-DrawnUI.

Keywords: Web Design, Object Detection, Convolutional Neural Networks, Machine Learning, Computer Vision, User Interface, Deep Learning

1 Introduction

The ImageCLEFdrawnUI [3] challenge was organized in connection with the ImageCLEF 2020 evaluation campaign [7] at the Conference and Labs of the Evaluation Forum (CLEF). The Main goal of this competition was to create an algorithm or system which can automatically recognise and localize UI elements on high resolution pictures of their drawings. The desired outcome of the detection process are localized bounding boxes with corresponding classes assignments of the UI elements.

1.1 Motivation

The main motivation for this task is to simplify the process of websites creation by enabling people to create websites by drawing UI elements on a whiteboard or on a piece of paper to make the web page building process more accessible. In this context, the detection and recognition of hand drawn UI elements task addresses the problem of automatically transcribing the UI to computer code.

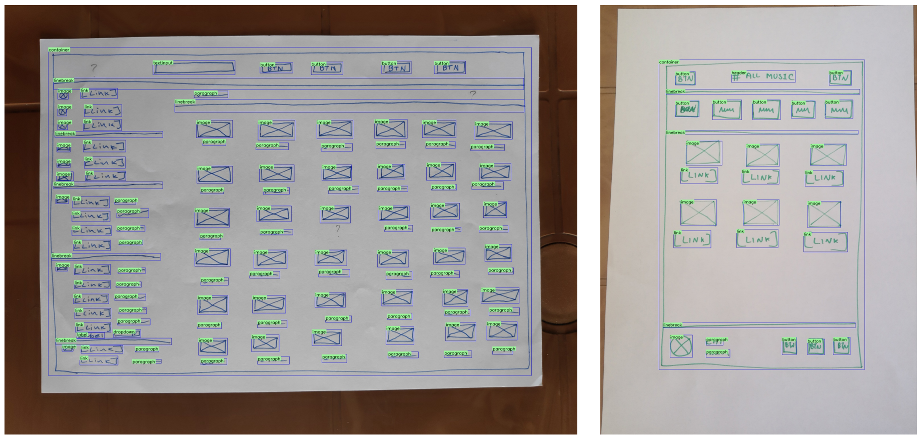

Fig. 1. Example images with annotations from ImageCLEFdrawnUI competition training dataset.

1.2 Dataset

The complete dataset consists of 1,000 hand drawn templates captured multiple times with different cameras, resulting in 2,950 high-resolution images. These data were further randomly split into 2,363 training and 587 test images. The training part includes 65,993 UI elements belonging to 21 classes. All images were annotated with bounding boxes and class labels by human experts. More detailed class distribution description is listed in Table 1. Example images are depicted in Figure 1.

1.3 Solution

The proposed solution is based on utilization of a standard object detection network architecture and coherent data preparation and augmentation. In particular, the Faster R-CNN [10] framework with the ResNet-50 [5] feature extractor was used. The system was implemented and fine-tuned using TensorFlow Object Detection API[1] [6] from publicly available checkpoints. All networks in our experiments shared the optimizer settings - RMSProp [13] with momentum of 0.9. The initial architecture was based on our work [8] submitted to ImageCLEFcoral competition [2]. This included for instance the data augmentation methods or Accumulated Gradient Normalization technique [4]. During our followup research, we considered and tested several approaches including new data synthesis, different network architectures as well as network ensemble variants.

[1] https://github.com/tensorflow/models/blob/master/research/object_detection

Table 1. Class distribution description including number of UI elements and their number in training and validation set.

Dataset distribution			**Train. / Val. split**		
Class Name	# Boxes	Fraction[%]	Train. Boxes	Val. Boxes	Fraction[%]
button	18,704	28.34	16,841	1,863	9.96%
paragraph	10,367	15.71	9,342	1,025	9.89%
image	7,683	11.64	7,020	663	8.63%
link	6,809	10.32	6,140	669	9.83%
linebreak	5,798	8.786	5,267	531	9.16%
container	4,678	7.089	4,233	445	9.51%
header	4,356	6.601	3,947	409	9.39%
textinput	1,732	2.624	1,577	155	8.95%
label	1,691	2.562	1,539	152	8.99%
dropdown	1,472	2.231	1,350	122	8.29%
list	798	1.209	702	96	12.03%
checkbox	758	1.148	694	64	8.44%
video	360	0.545	323	37	10.28%
radiobutton	279	0.422	246	33	11.83%
toggle	178	0.249	159	19	10.67%
datepicker	91	0.138	83	8	8.79%
rating	75	0.114	62	13	17.33%
slider	75	0.114	65	10	13.33%
textarea	47	0.071	42	5	10.64%
table	29	0.043	25	4	13.79%
stepperinput	13	0.019	10	3	23.08%

2 Methodology

2.1 Data analysis and preparation

Dataset splitting for validation - To create a set for continuous network performance evaluation, the provided dataset needed to be split into training and validation sets. After careful examination of the content, it became apparent that a random split of the dataset could cause discrepancies between the validation and training sets performances. The reason being, that less frequent classes could end up not having comparable representations in both the training and validation sets. Therefore the split had to be carefully engineered and resulted in the final approximate ratio of **11:1** for training and validation sets, respectively.

Data distribution - To better understand the problem at hand, we have performed a frequency analysis on UI element type distribution and concluded, that some of the element types are represented by very few occurrences in the training dataset, namely the *'stepper input'*, *'text area'* or *'table'* (See Table 1). Reviewing the training dataset further revealed that it contains multiple images of the same drawings. This is caused by the fact that the whole dataset (training and testing) consists of 2,950 images of only 1,000 templates, i.e., the templates were each captured by several different cameras. Following the random splitting of the dataset to the training and testing part caused some rarer elements to go to the training set multiple times and others not at all. This worsens the uneven distribution of the UI element classes in such a way that, for example, the rarest element is contained only on two templates (6 images) in the training dataset. For the deep network to learn to recognize such an element, a much higher number of examples is needed.

Synthetic dataset - To compensate for the uneven distribution of the UI element types, we decided to expand the training dataset with synthetic data containing such elements. The data were generated using augmentations of segmented UI elements, which were consequently pasted on random size paper of very light random color. The augmentation consisted mainly of constrained random affine transformations. We have added 500 synthetically generated images with the least frequent classes. Examples of the synthetic data are depicted in Figure 2. UI element classes which were artificially added are: *datepicker*, *rating*, *slider*, *textarea*, *table* and *stepperinput*.

The experiment performed with ResNet-50 backbone and grayscale data with 1000×1000 input size was evaluated over the validation set and showed interesting improvement in all measured scores on RGB images. Specifically, mean average precision with Intersection over Union (IoU) greater than 0.5 (mAP0.5) by **0.0081**, mAP by **0.0222**, and by Recall@100 (Recall calculated using best 100 detections) **0.0315**. Although we were able to flatten the UI elements distribution curve, the overall performance of the original network was marginally better on grayscale images.

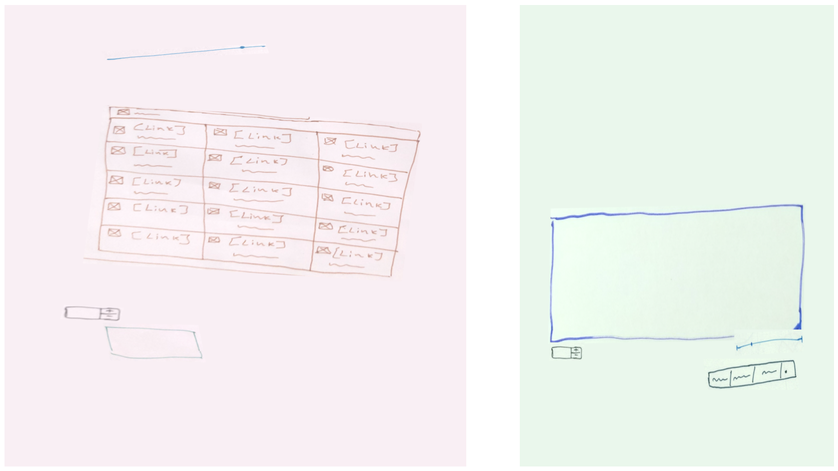

Fig. 2. Synthetic data example images. For classes with very few instances we injected the training set with artificial data containing augmented instances of these classes.

Table 2. Results of input format experiment. Trained for 50 epochs on 1000 × 1000 grayscale images with Faster R-CNN

Input format	mAP0.5	mAP	Recall@100
RGB	**0.9151**	0.5814	0.6594
Grayscale	**0.9151**	**0.5855**	**0.6689**
Gray ++	0.9087	0.5788	0.6705

Data preprocessing - While testing, we experimented with three different approaches to input preprocessing. First, images without any augmentations. Second, images were converted to grayscale. Third, where grayscale images without borders around captured drawing and with dilated lines were used. This effect was achieved by finding contours in the image using OpenCV [1]. These contours were then sorted by area, and the largest of them was used to crop the image. In some cases, this approach did not work correctly, especially where parts of the image were covered with a shadow, so this crop was only applied when the area of the contour was at least 70% of the image. After the crop, we utilized CLAHE [9] for histogram equalization and applied topological erosion to pronounce lines on paper. This approach is described as Gray++ in our results. Refer to Table 2, where it can be seen that Gray++ was worse in our testing, so our submissions mainly used grayscale images.

2.2 Object Detection Networks

There are several network architectures that were taken into the consideration for this task, in particular the Faster R-CNN [10] and EfficientDet [12]. The initial performance test for each particular network architecture was to train these networks with recommended configuration. These tests revealed the overall architecture suitability for the task at hand. The best performance was achieved with the Faster R-CNN architecture that used comparable backbones. Refer to Table 4.

Next, we needed to decide on the object detection network backbone. We have tested several widely used backbone architectures, namely the ResNet-50 [5], ResNet-101 [5], Inception V2 and Inception-ResNet-V2 [11] (Table 3).

Table 3. Results of backbone architecture experiment. Trained for 50 epochs on 1000 × 1000 grayscale images with Faster R-CNN.

Backbone	mAP0.5	mAP	Recall@100
Inception-V2	0.8974	0.5850	0.6689
ResNet-50	0.9151	0.5855	0.6689
ResNet-101	0.9035	0.6035	0.6835
Inception-ResNet-V2	**0.9176**	**0.6095**	**0.6904**

Table 4. Results of detection framework experiment. Trained for 50 epochs on 1000 × 1000 RGB images.

Approach	mAP0.5	mAP	Recall@100
EfficientDet-B3	0.583	0.416	0.458
Faster R-CNN ResNet-50	**0.9151**	**0.5814**	**0.6594**

Table 5. Training and network parameters shared among all experiments.

Parameter	Value	Parameter	Value
Optimizer	RMSprop	Gradient Clipping	10.0
Momentum	0.9	Input size	1000×1000
Initial and min LR	0.032 - 0.000032	Feature extractor stride	16
LR decay type	Exponential	Pretrained Checkpoints	COCO
LR decay factor	0.975	Num epochs	50
Batch size	2	Gradient accumulation	12

3 Submissions

In this competition, the AICrowd platform[2] was used to evaluate participants submissions. Each participating team was allowed to submit up to 10 text files with detection bounding-boxes in a specific format for each image. We have created 7 submission using configurations listed below.

Baseline configuration - As a baseline for all our experiments we used Faster R-CNN with ResNet-50 as a backbone. For training we used parameters and augmentations described in Table 5 and [8], respectively. Finally, thresholding was used to select only detection with high confidence.

Submission 1 - Baseline experiment trained on RGB images. Tested on original-size RGB images. Detection confidence threshold was set to 0.8.

Submission 2 - Submission 1 trained and tested on the grayscale images.

Submission 3 - Submission 2 trained on whole training set with no data for validation with confidence threshold of 0.95.

Submission 4 - Submission 3 trained for 80 epochs.

Submission 5 - Submission 1 with Inception-ResNet-V2 as backbone. Trained and tested on grayscale images with confidence threshold of 0.8.

Submission 6 - Voting ensemble created by combining models used in Submissions 2, 3 and 5 with confidence threshold of 0.8.

Submission 7 - Submission 6 with confidence threshold of 0.45.

4 Competition Results and Discusion

The official ImageCLEFdrawnUI competition results are displayed in Figure 3. The proposed system achieved the best Overall Precision score of **0.9709** and outperformed 2 other participating teams as well as the baseline solution proposed by organizers. The best scoring submission was produced by Mask R-CNN model with ResNet-50 backbone architecture and input resolution of 1000×1000 trained for 80 epochs with parameters and augmentations described in Table 5

[2] https://www.aicrowd.com

Table 6. Submission results achieved over test set.

Submission	1	2	3	4	5	6	7
Overall Precision	0.939	0.956	**0.971**	0.944	0.956	0.956	0.942
mAP@0.5	0.688	0.676	0.583	0.647	0.695	0.694	**0.755**
Recall@0.5	0.536	0.517	0.445	0.472	0.520	0.519	**0.555**
Run ID	67733	67814	67816	67991	68003	68014	68015

and [8], respectively. The resulting predictions were filtered with confidence threshold of 0.95 to maximize the official metric of mAP.

In our opinion, the winning submission is not the best of our submissions. According to the widely accepted performance metrics (mAP@0.5 and Recall@0.5), our Submission 5 (run ID 68003), which scored 3rd place overall, is superior to the winning submission. It diminishes ImageCLEF Overall Precision only by **0.0144**, while it increases mAP@0.5 by **0.111** and Recall@0.5 by **0.074**.

5 Conclusion

In this paper, we have presented a system for automatic hand-drawn UI element detection and localization. To achieve this goal, we had to gain a deep understanding of the provided dataset and perform many experiments to craft the best data preprocessing and augmentation methods, as well as objectively adjust the network parameters.

The final methods were based on the Faster R-CNN detection network with ResNet-50 used as a backbone architecture. The presented method scored first place in ImageCLEFdrawnUI competition, with an overall precision of **0.9708**.

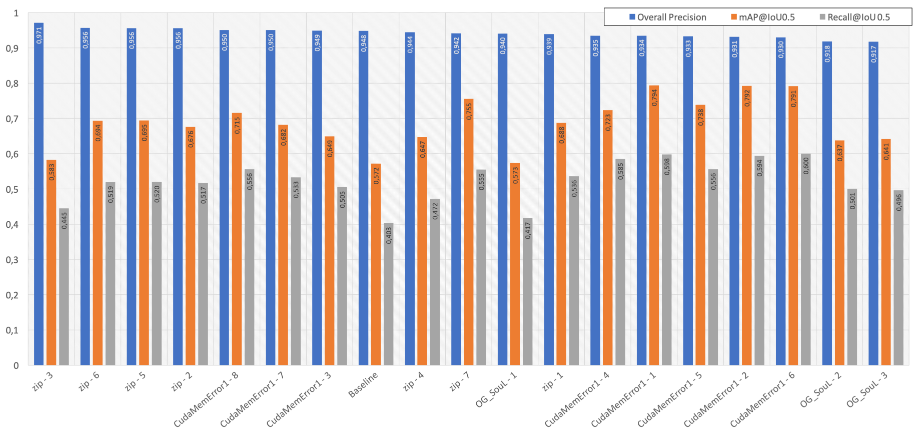

Fig. 3. Results for all runs submitted by the competition participants. Including additional metrics e.g. mAP@IoU0.5 and Recall@IoU0.5.

6 Acknowledgements

Lukáš Picek was supported by the Ministry of Education, Youth and Sports of the Czech Republic project No. LO1506, and by the grant of the UWB project No. SGS-2019-027.

References

1. Bradski, G.: The OpenCV Library. Dr. Dobb's Journal of Software Tools (2000)
2. Chamberlain, J., Campello, A., Wright, J.P., Clift, L.G., Clark, A., García Seco de Herrera, A.: Overview of the ImageCLEFcoral 2020 task: Automated coral reef image annotation. In: CLEF2020 Working Notes. CEUR Workshop Proceedings, CEUR-WS.org <http://ceur-ws.org> (2020)
3. Fichou, D., Berari, R., Brie, P., Dogariu, M., Ştefan, L.D., Constantin, M.G., Ionescu, B.: Overview of ImageCLEFdrawnUI 2020: The Detection and Recognition of Hand Drawn Website UIs Task. In: CLEF2020 Working Notes. CEUR Workshop Proceedings, CEUR-WS.org <http://ceur-ws.org>, Thessaloniki, Greece (September 22-25 2020)
4. Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., He, K.: Accurate, large minibatch sgd: Training imagenet in 1 hour. arXiv preprint arXiv:1706.02677 (2017)
5. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2016)
6. Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S., et al.: Speed/accuracy trade-offs for modern convolutional object detectors. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 7310–7311 (2017)
7. 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 lifelogging, medical, 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)
8. Picek, L., Říha, A., Zita, A.: Coral reef annotation, localisation and pixel-wise classification using mask-rcnn and bag of tricks. In: CLEF (Working Notes). CEUR-WS.org <http://ceur-ws.org>, Thessaloniki, Greece (September 22-25 2020)
9. Pizer, S.M., Amburn, E.P., Austin, J.D., Cromartie, R., Geselowitz, A., Greer, T., ter Haar Romeny, B., Zimmerman, J.B., Zuiderveld, K.: Adaptive histogram equalization and its variations. Computer vision, graphics, and image processing **39**(3), 355–368 (1987)
10. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems 28, pp. 91–99. Curran Associates, Inc. (2015)

11. Szegedy, C., Ioffe, S., Vanhoucke, V., Alemi, A.A.: Inception-v4, inception-resnet and the impact of residual connections on learning. In: Thirty-first AAAI conference on artificial intelligence (2017)
12. Tan, M., Pang, R., Le, Q.V.: Efficientdet: Scalable and efficient object detection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 10781–10790 (2020)
13. Tieleman, T., Hinton, G.: Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning **4**(2), 26–31 (2012)