Systems and Computing
Volume 2, Issue 1, 72-80, 2026.
https://doi.org/10.64409/sycom.v2.i1.28

Research Article Open Access

Recognition of GUI Web Elements Using the Deep Learning Approach and Yolo Architecture

Helga Prifti [1], **Bekir Karlik** [2]

1. *Department of Computer Engineering, Epoka University, Tirana, Albania*
2. *Department of Electrical and Electronics Engineering, Esenyurt Universitesi, Istanbul, Turkey*

Corresponding author: Bekir Karlik (bekirkarlik@esenyurt.edu.tr)

Manuscript Review Record:

Submitted:
October 2, 2025

Accepted:
December 29, 2025

Published:
January 15, 2026

Cite This:
Helga Prifti & Bekir Karlik. "Recognition of GUI Web Elements Using the Deep Learning Approach and Yolo Architecture". *Systems and Computing*. Volume 2, Issue 1, 72-80, 2026. https://doi.org/10.64409/sycom.v2.i1.28

Abstract- **Context:** The growing complexity of modern web and mobile applications, shaped by rapid UI/UX advancements and agile software development practices, demands efficient and automated recognition of graphical user interface (GUI) elements, as manual testing remains costly and time-consuming. **Objective:** This study aims to evaluate the performance of the YOLOv8 deep learning architecture for accurate recognition of diverse GUI components, such as buttons, fields, headings, links, and images, from real-world application screenshots. **Method:** A novel YOLOv8 model was used on the Roboflow GUI web element dataset, which contains over 1000 labeled website screenshots, employing 35 training epochs, a batch size of 8, and a 640×640 image resolution, while also demonstrating the flaws in the experiment. **Results:** The model achieved measurable improvements, with precision rising from 0.368 to 0.454, recall increasing from 0.296 to 0.425, and mAP50–95 improving from 0.101 to 0.232. Strongest detection was observed for buttons and input fields, while weaker performance was noted for iframes and labels due to their inherent ambiguity. **Conclusions:** The results demonstrate that YOLOv8 has significant potential in automating GUI recognition, reducing reliance on manual testing in agile workflows, and improving interface validation. Further optimization with larger datasets and advanced augmentation methods is recommended to enhance robustness and generalization.

Keywords- Deep learning, GUI web element, YOLOv8, Agile framework, SDLC.

Acknowledgement

The authors declare that no financial support or external assistance was received for this research acknowledged.

1. Introduction

The introduction should provide a clear statement of the problem, by introducing the concepts and the background of Web elements are the fundamental brick in an application-built ranging from simple ones like buttons and input fields to more complex ones like input with dropdowns or dynamic date pickers. In today's application the User Interface is presented as an overall combination of non-abstract and abstract variables, encompassing the user interface, the functionality of it and the reaction time [1-2] In the applications of the 21st century, the rapidness that the Software Development lifecycle is being carried out opts for automatic and rapid GUI component recognition. The continuous integration and continuous deployment pipeline lead to round-the-clock changes in the graphical user interface, while adapting to the customer's and user's need. This type of dynamics calls for continuous and sometimes even extensive testing of the interfaces. This kind of testing is currently only being carried out manually by QA and software testers in agile teams, i.e is almost 100% dependent on manual labor of the responsible professionals.

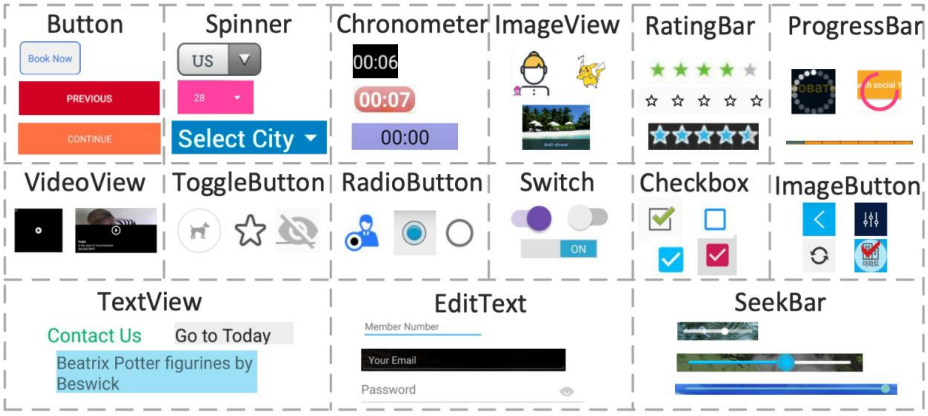

Figure 1. Characteristics of GUI elements.

The recent and very rapid advancements in computer vision and deep learning can be exploited to automate this kind of task or at least make it as less human dependent [3-5] Big advances in Convolutional Neural Networks (CNNs) particularly the You Only Look Once (YOLO) family have significantly improved the accuracy and efficiency of automatic GUI recognition. [6-7] Our goal in this paper is to see and evaluate the performance of YoloV8 architecture in the task of recognizing different GUI web elements based on application screenshots.

2. Literature Review and Background Information

Approaches similar to other object detection and recognition have been applied to perform automatic GUI Web Element detection. All the done work can be mainly categorized into two groups: Computer Vision or "Hand-Crafted" approach and the Deep Learning one. Alongside the approaches currently used we will also introduce some of the most used datasets for automatic GUI element detection.

2.1. Datasets

When it comes to GUI element recognition, one of the biggest challenges has to do with the available datasets. Due to its very customizable nature, the datasets for it are not many when compared to other topics like Facial Emotion Recognition. Existing datasets contain labelled data that can only be used for supervised learning.

Most of the available datasets contain mobile application images like RICO [8] which the largest one featuring over 72000 images of mobile application screenshots, and also Visual Assessment of UIs using Neural Networks (VINS) [9] that takes the task even further by trying to analyze the abnormalities in GUI layouts. All these datasets feature tens and hundreds of web elements like sliders, images, input fields etc., some of them depicted in Fig. 1. What we will focus on in this paper are buttons, input fields, headings, images, iframes, labels, links and text.

2.2. Hand-Crafted Approach

The Computer Vision or "Hand-Crafted" Approach is the most traditional type of tackling such object detection and recognition tasks. This method mainly relies on manual definition of algorithms to analyze and process images. This procedure can be broken into three main subprocesses:

2.3. Image preprocessing

This can easily be considered as the genesis of a successful image recognition process. In this step the main goal is to make the image as easily analyzable as possible by omitting redundant information from it, for the features to be extracted easier. In this stage there are several other smaller steps that are executed like normalization and scaling noise reduction and edge detection.

Table 1. Yolo versions throughout the years.

S/N	YOLO Variant	Improvement	Results
1.	YOLOv1 (Redmon et al., 2016)	Single shot detector combines and solves the problem of drawing boundary boxes and class identification	Higher accuracy and speed compared to two-stage object detector such as Faster R-CNN
2.	YOLOv2 (Redmon & Farhadi, 2018)	Iterative improvements on Batch Normalization, higher resolution detection and use of anchor boxes	Reduction in architecture, faster detection and higher accuracy and better detection of high resolution images
3.	YOLOv3 (Redmon & Farhadi, 2018)	Addition of objectness score to bounding box prediction, added connections to the backbone network layers and predictions at three separate granularities.	Improves detection of smaller objects
4.	YOLOv4 (Alexey et al., 2020)	Improved feature aggregations, bag of freebies with mosaic augmentations and use of mish activation	Achieved improved accuracy and ease of training, high quality performance and accessibility
5.	YOLOv5 (Nepal & Eslamiat, 2022)	Reduced network parameters, use of Cross Stage Partial Network (CSPNet) for the head, PANet for the neck of the architecture, residual structure and auto-anchor. It also utilizes mosaic augmentations.	Extremely easy to train, inference on individual, batch images, video feed and webcam ports. Ease of transfer and use of weights. Faster and more lightweight than previous YOLO.
6.	YOLOv6 (Chuyi et al., 2022)	Redesigned network backbone and neck to EfficientRep Backbone and Rep-PAN Neck. The Network head is decoupled separating different features from the final head	Improvement in detecting small objects, anchor free training of model. Less stable and flexible as compared to YOLOv5.
7.	YOLOv7 (Wang et al., 2022)	Layer aggregation using E-ELAN, trainable bag of freebies, 35% fewer network parameters. Model scaling for concatenation-based model	Increase in speed and accuracy, ease of training and inference.

2.4. Feature extraction

In this phase we obtain the important and useful information in the form of symbols or other elements. When it comes to graphical user interface elements, two of the most intuitive methods are to Edge and Contour detection. The Canny Edge detector is a very popular algorithm in such tasks specialized to return edges of an input image. Apart from edge detection, contour detection can also be utilized which comes as a part of the OpenCV library. However, in this case a

binary conversion of the image is needed at first. In both cases the obtained information is then fed to an image classifier to then continue with the image classification.

2.5. GUI element classification

In this last step the GUI web element is predicted using the obtained information from the previous steps. For this part classical object detection classifiers are used like SVM (Support Vector Machine) or Bayesian Algorithms. At the same time studies have been conducted and custom algorithms like UIED [10] have been developed to classify web elements.

2.6. Deep Learning Approach

Like the Hand-Crafted approach, we can again define three main processes in the deep learning method: preprocessing, pattern learning and the classification at the end. In this approach, preprocessing takes an even bigger significance as the images need to be the same size and the GUI components should be well aligned for the purpose of minimizing classification error later in the process. This is a very delicate step as the generated data needs to belong to a broad spectrum but at the same time not too generalized so that the model can still pick up a pattern.

After all the images have been tailored to suit the model, the pattern learning process takes place which essentially substitutes all the subprocesses in the hand-crafted method. There are many methods to fulfill the task we need such as Generative Adversarial Networks (GANs) or Deep Belief Networks (DBNs), but in this paper we will focus on Convolutional Neural Networks (CNNs). When it comes to GUI element recognition there are three main CNN architectures that take the lead in terms of performance: Region Based CNN (R-CNN) [11], Faster R-CNN [12] and YOLO CNNs [13]. In our paper we have observed the results and performance of YoloV8 model.

3. Materials and Methodologies

3.1. YOLO Family

The Yolo architecture model belongs to “One-Stage” [14-15] object detectors, which stand out in terms of processing speed in comparison to other detectors. In our task such speed is essential due to the mentioned fast paced processes in the agile environment SDLC. These models instead of firstly dealing selected areas of interest, make an initial prediction on the object’s class as well as the bounding boxes of the image in a single run.

Yolo’s independence from image size input makes it a great choice for GUI element detection as there are a variety of devices’ sizes that are designed, developed and tested. Despite the versatility in the architecture backbone model, Yolo has undergone a series of improvements, leading to more accurate and fast models. In Table 1 we are presenting a short summary of the first seven Yolo models and their characteristics.

In this paper as mentioned we will be utilizing the YoloV8 version of the architecture launched by Ultralytics and combining it with the Roboflow GUI element detection dataset [16-17]. The dataset contains website screenshots from over 1000 very popular websites that are automatically annotated, Figure 2. by the company with the following classes: button, heading, link, field, iframe, image, label and text. This dataset was split based on the classic ratio for dataset splitting when it comes to Deep Learning approaches. 70% of the dataset was used for Table 1. Yolo versions throughout the years Figure 2. Train annotated web screenshot training, 20% for validation and the remaining 10% was utilized for the model’s testing phase.

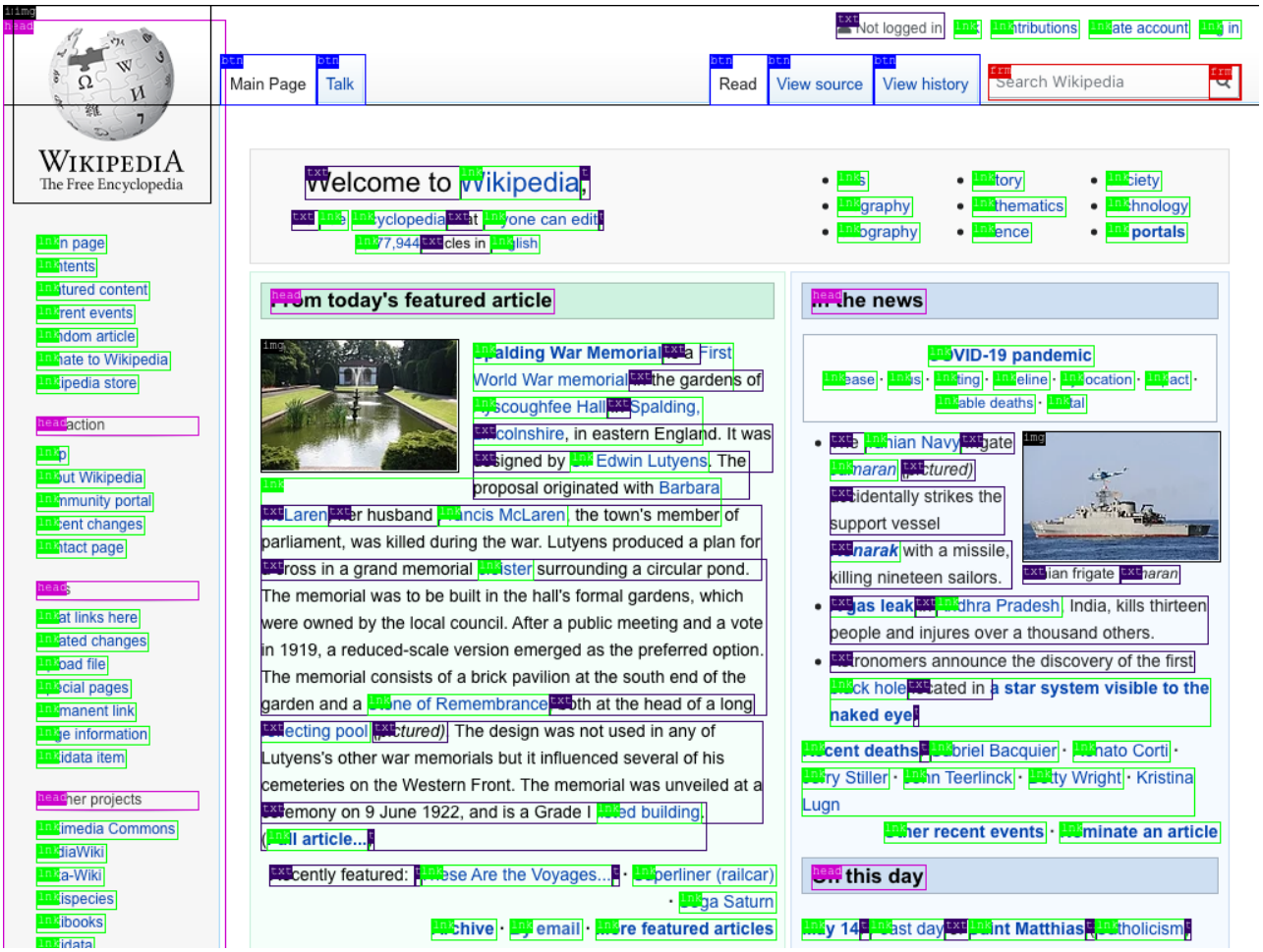

Figure 1. Train annotated web screenshot.

3.2. YoloV8

Due to lack of an official paper on the YoloV8 architecture, we will be analyzing the provided repo by the company to introduce the enhancements made to it compared to previous versions. This model features an anchor-free functionality that directly predicts the object's center unlike its predecessors that rely on predetermined anchor boxes. With the anchor-free detection the Non-Maximum Suppression (NMS) is simplified thus providing more efficient false positives removal.

The new introduced convolution structure features a swap between the 6x6 kernel in the stem with a 3x3 one and a substitution of C3 with two C2f in the backbone which concatenate the output of all bottleneck modules instead of using the output of the last one [17].

Mosaic data augmentation is another added feature that enhances the model's ability to "build knowledge" by mixing four different images to provide it with more information context. Different from the older versions, the augmentation process stops after the first 10 epochs in the training process.

Our experiment relies on the use of the YoloV8s-OBB model, which is the second smallest model in its family due to its experimental nature and the small dataset used. The model has nearly 9.7 million parameters, with an mAP50 of approximately 79.5%, a result based on a standard benchmark of the COCO dataset.

4. Results and Discussion

We trained a small YoloV8 model for 35 epochs with a batch size of 8 using images of size 640x640 which is the default Yolo model images size. This study was executed in Google Collaboration on a single NVIDIA GPU. Our model contained 144 layers where the last one was an Oriented Bounding Box (OBB) detection layer. After the training and validation processes, we notice a significant drop in classification loss throughout the epochs during both training and validation Figure 3, however the loss had a steadier drop during training process and we notice some fluctuations of it during validation. This could come because of the complexity that the website screenshots contain due to the variations that web elements have in each website, as well as a relatively large number of classes defined for this experiment.

Figure 2. Precision over Epochs graph.

Our model reached a precision rate of 0.454 after starting with a rate of 0.368 in the first epoch. This incline showed pleasant results as it can be improved if the model gets trained for a longer period using more epochs and a larger batch size. There was a significant drop in the 19th epoch Figure 4, showing signs of underfitting. If a larger and more diverse dataset was provided for the model, the learning rate would have been better hence improving the model's ability to better generalize the features. At the same time, more data augmentation might be needed to result in steadier precision incline.

Our model predicted our Web Elements by giving the predicted class/label of the element associating it with a probability rate to give a certainty for the predicted element. The bounding boxed were color coded for each class, apart from the word labelling and the certainty percentage. Figure 5.

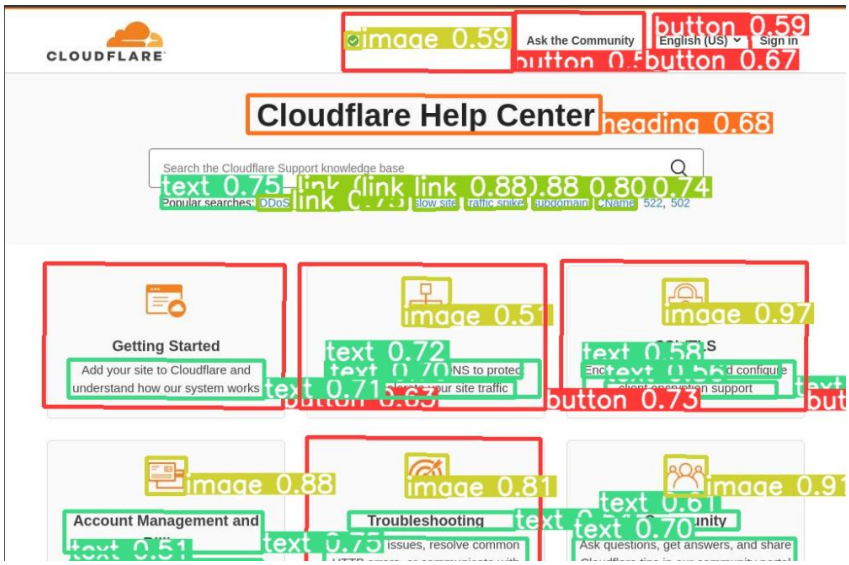

Figure 4. Web Element prediction.

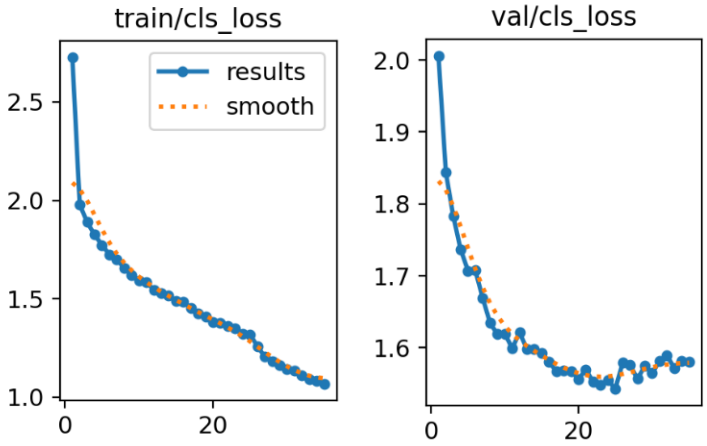

Figure 3. Classification during training and validation.

We validated our model on a separate validation set. In Table 2 below we have summarized an overview of our validation results on the dataset, considering all our classes present. From the table we can observe that the “iframe” and “label” classes had a worse precision rate compared to the other ones.

Table 2. Validation results.

Class	Images	Instances	Precision	Recall	mAP50	mAP50-95
Button	482	6188	0.557	0.599	0.551	0.302
Field	482	364	0.603	0.634	0.658	0.411

Heading	482	1508	0.383	0.532	0.407	0.228
Iframe	482	156	0.356	0.321	0.254	0.165
Image	482	3746	0.564	0.518	0.521	0.301
Label	482	34	0.242	0.176	0.19	0.141
Link	482	2650	0.452	0.347	0.351	0.202
Text	482	6030	0.437	0.344	0.168	0.168

As a result, we got better predictions for field web elements and images and poorer performance in the recognition of iframes and labels, which was expected to happen as iframes and labels can be very ambiguous when compared to other elements in the user interface. We expected the model to perform better in the detection of buttons given their more prominent structure compared to the other web elements.

5. Conclusions

Through the experiments we observed the performance of a small YoloV8 model in the task of GUI Web element recognition using webpages' screenshots. We got satisfying results in terms of precision improvement throughout the training and validation, starting at 0.368 and reaching up to 0.456 in the 35th epoch. We noticed a significant drop in precision in the 19th epoch, which could have come as a result of the small model's size compared to the other ones available for YoloV8 architecture as well as the dataset size and class variance imbalance. Additionally, recall increased from 0.296 to 0.425. The mAP50-95 metric showed an improvement from 0.101 to 0.232, while the mAP50 metric grew from 0.21 to 0.401. This provides us with content information to say that the model is able to perform well in different degrees of overlap threshold. However, these results still are not yet reliable enough to be used for fully automated GUI testing, therefore we would like to interpret them as a proof of concept baseline for future implementations and research, rather than a solution ready to be deployed.

The model performs well in interclass variation detection, by making the best predictions for buttons and input fields with the highest precision rate and mAP score. However, it performed much poorer when it came to predicting iframes or labels, which could directly be linked to the need for some more balanced class variance, but also indicating that further fine tuning or more data augmentation might be needed. The presence of low accuracy for these classes can also be related to their nature which tends to lean towards ambiguities as GUI components with iframes are sometimes tricky to detect even with the naked eye when implemented seamlessly and labels being often mistaken as text or paragraph content.

What we would suggest for future studies to focus on would be the improvement of the model in recognition of ambiguous web elements like iframes or labels, which were also the ones that our model did not perform well in classifying. The performance of the model may be enhanced by thorough hyperparameter optimization and sophisticated data augmentation methods as well as further data augmentation like random cropping, rotation or color jittering to be able to fix the class variance a bit. Finally, adding extra assessment measures like the F1-score and thorough error analysis can offer a more thorough evaluation of the model's performance. These issues can be resolved to greatly improve the

YOLOv8 model's robustness and accuracy in web screenshot object recognition, increasing its suitability for real-world use.

References

[1] P. J. Van de Broek, C. Onime, J. O. Uhomobhi, M. Santachiara, "Evolution of User Interface and User Experience in Mobile Augmented and Virtual Reality Applications" in P. Bourdot & D. J. Kasik (Eds.), Haptic Technology - Intelligent Approach to Future Man-Machine Interaction, 2022, Chapter 3.

[2] T. Zhang, Tao et al. "Deep Learning-Based Mobile Application Isomorphic GUI Identification for Automated Robotic Testing" *IEEE Software*, 37, 67-74, 2020. https://doi.org/10.1177/15501329221115

[3] T. Yeh, T.H. Chang, and R.C. Miller, "Sikuli: Using GUI screenshots for search and automation". In Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology (UIST '09), pp. 183-192. 2009, Association for Computing Machinery. https://doi.org/10.1145/1622176.1622213

[4] X. Sun, T. Li and J. Xu, "UI Components Recognition System Based On Image Understanding" 2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C), Macau, China, pp. 65-71, 2020. https://doi.org/10.1109/QRS-C51114.2020.00022

[5] M.F. Yilmaz and B. Karlik, "Comparison of deep learning algorithms with different activation functions for brightness image enhancement". *International Journal of Artificial Intelligence and Expert Systems (IJAE)*, 13(2), pp.12-24, 2024.

[6] B. Selcuk, T. Serif. "A Comparison of YOLOv5 and YOLOv8 in the Context of Mobile UI Detection." *Journal of Computer Vision and Pattern Recognition*, Array 2024, vol. 22, 100351, 2024. https://doi.org/10.1007/978-3-031-39764-6_11

[7] K. Moran, C. Bernal-Cárdenas, M. Curcio, "Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps." In Proceedings of the 26th ACM/IEEE International Conference on Mobile Software Engineering and Systems (ESEC/FSE'18), vol. 46, pp. 196–221, 2018. https://doi.org/10.1109/TSE.2018.2844788

[8] S. N. Cavsak, A. Deliahmetoglu, B. T. Ay and S. Tanberk, "GUI Component Detection Using YOLO and Faster-RCNN," 2023 14th International Conference on Electrical and Electronics Engineering (ELECO), Bursa, Turkiye, pp. 1-5, 2023. https://doi.org/10.1109/ELECO60389.2023.10415929

[9] M. D. Altinbas, and T. Serif, T. (2022). GUI Element Detection from Mobile UI Images Using YOLOv5. In Computer Vision and Pattern Recognition (pp. 45-62). Springer.25- H. Agrawal and K. Desai, "Canny edge detection: A comprehensive review," *Int. J. Tech. Res. & Sci.*, vol. 5, no. 1, pp. 126, 2024. https://doi.org/10.30780/specialissue-ISET-2024/023

[10] M. Xie, S. Feng, Z. Xing, J. Chen, and C. Chen, "UIED: A Hybrid Tool for GUI Element Detection." In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (Virtual Event, USA) (ESE/FSE, 2020), Association for Computing Machinery, New York, NY, USA, pp. 1655-1659, 2020. https://doi.org/10.1145/3368089.3417940

[11] J. Cheng, D. Tan, T. Zhang, A. Wei, and J. Chen, "YOLOv5-MGC: GUI Element Identification for Mobile Applications Based on Improved YOLOv5", *Mobile Information Systems*, 9800734, 9 pages, 2022. https://doi.org/10.1155/2022/8900734

[12] C. Jemmali, C. Harteveld, Y. Fu, and M. Seif El-Nasr, "VINS: Visual Search for Mobile User Interface Design. CHI '21: ACM CHI Conference on Human Factors in Computing Systems, (CHI '21)." Association for Computing Machinery, New York, NY, USA, Article 423, pp. 1–14, 2021.

[13] J. Vyskočil and L. Picek, "Improving Web User Interface Element Detection Using Faster R-CNN," in the Conference and Labs of the Evaluation Forum (CLEF), 2021, Bucharest, Romania. CEUR Workshop Proceedings, vol. 2936, pp. 1375-1386, 2021.

[14] R. Girshick J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Columbus, OH, USA, pp. 580-587, 2014. https://doi.org/10.1109/CVPR.2014.81

[15] A.A.M. Al-Saffar, T. Hai, M. A. Talab, "Review of Deep Convolution Neural Network in Image Classification." In ACM Computing Surveys, 2017 International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications, pp. 26-31, 2017. https://doi.org/10.1109/ICRAMET.2017.8253139

[16] https://universe.roboflow.com/yolo-z8ekd/ui-component/dataset/2https://docs.ultralytics.com/

[17] H. Prifti. "Recognition of GUI Web Elements Using the Deep Learning Approach and Yolo Architecture." Master Thesis, Epoka University, Tirana, Albania, 2025.