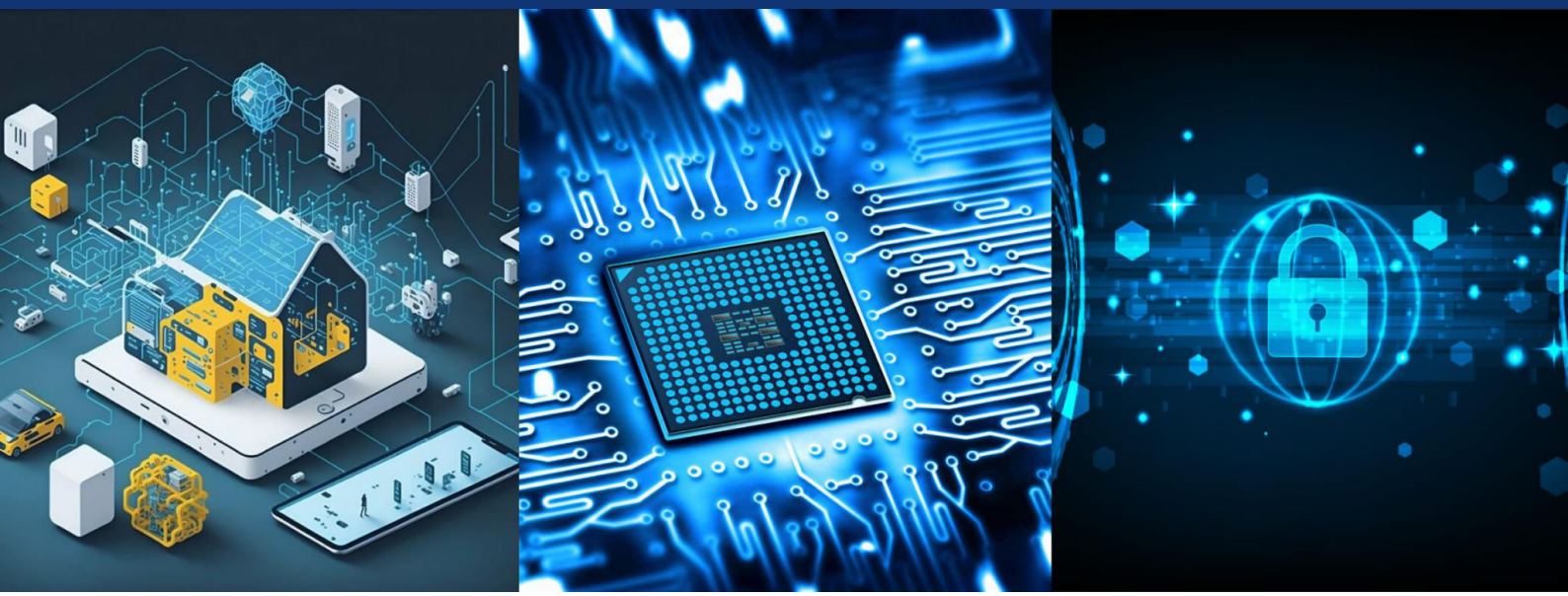
IJIRCCE

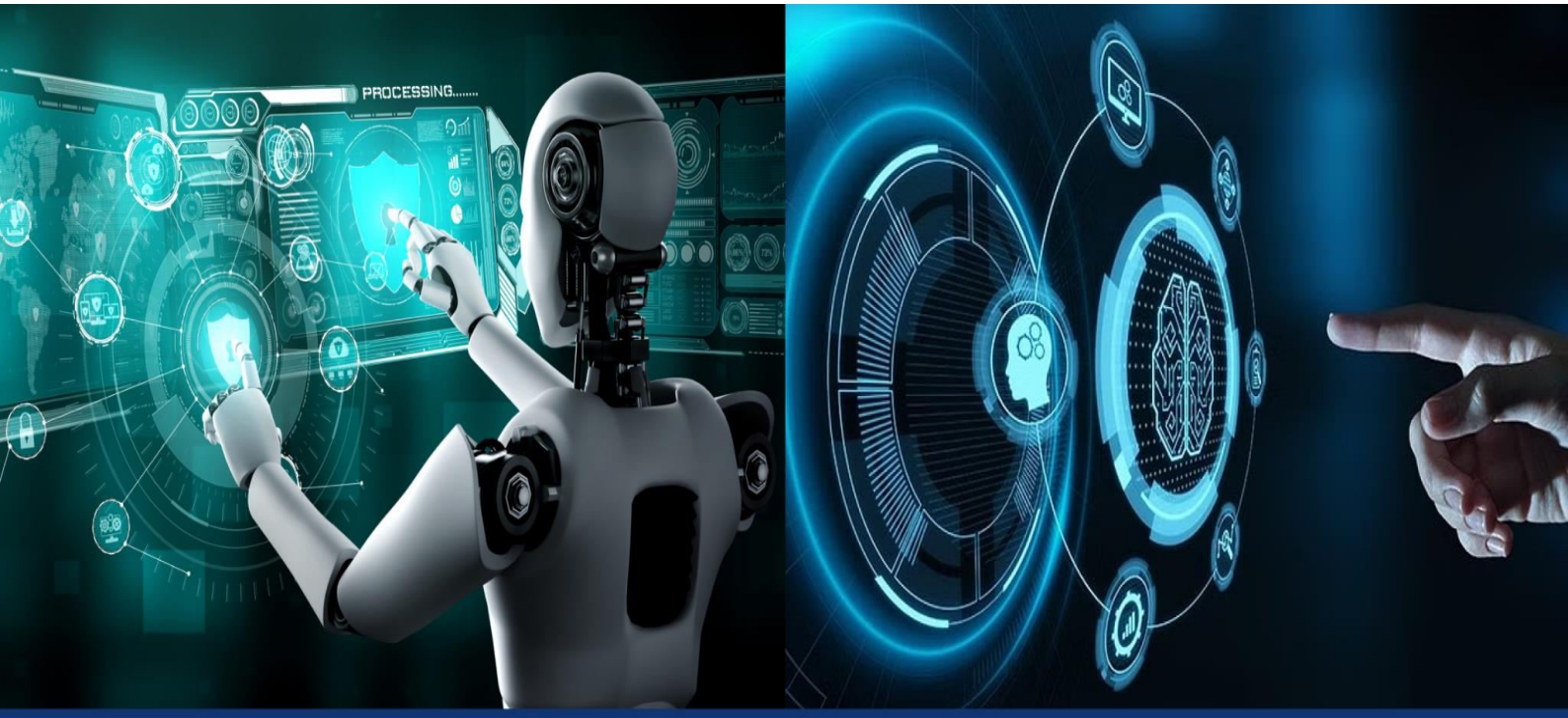
ISSN(O): 2320-9801

ISSN(P): 2320-9798



International Journal of Innovative Research in Computer and Communication Engineering

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 9.927

Volume 14, Issue 7, July 2026

© 2026 IJIRCCE | Volume 14, Issue 7, July 2026 DOI: 10.15680/IJIRCCE.2026.1407019

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

YOLO-Based GUI Element Detection for User Interaction Analysis

Ms. G. Sneha, Ms. P. Alekhya

Department of Computer Science and Engineering, St. Mary's Women's Engineering College, Budampadu, Guntur, Andhra Pradesh, India

Department of Computer Science and Engineering, St. Mary's Women's Engineering College, Budampadu, Guntur, Andhra Pradesh, India

ABSTRACT: Understanding how users interact with software interfaces increasingly depends on the ability to automatically locate and classify the on-screen elements - buttons, text inputs, checkboxes, sliders, dropdowns and menu items - that a user can act upon. Manual annotation of graphical user interface (GUI) elements does not scale to the pace at which modern applications change, motivating an automated, vision-based detection approach. This paper applies a You Only Look Once (YOLO) single-stage object detector to the task of GUI element detection, framing each screenshot as an object-detection problem in which every interactive widget is localized with a bounding box and assigned an element-type label. The detector is trained on a custom UI-element dataset spanning thirteen widget categories, and is evaluated using standard detection metrics - precision, recall, mean Average Precision at IoU 0.5 (mAP@0.5) and across IoU thresholds 0.5:0.95 (mAP@0.5:0.95) - together with per-image inference latency. Training logs show the box, classification and distribution-focal losses converging steadily, while validation results reveal a strong class-imbalance effect: frequent classes such as button and link are detected with useful precision and mAP, whereas rare classes with only a handful of annotated instances (checkbox, slider, radio, clickable) are not yet reliably detected. Despite the modest overall mAP obtained from this first-pass, imbalanced dataset, the model achieves a per-image inference time of close to twenty milliseconds, confirming that a YOLO-based pipeline is fast enough to support real-time downstream user-interaction analysis such as automated UI testing, accessibility tooling and usability analytics. The paper closes with a concrete plan - dataset rebalancing, stronger augmentation and transfer learning - for closing the accuracy gap on the minority element classes.

KEYWORDS: YOLO, Object Detection, GUI Element Detection, User Interaction Analysis, Human-Computer Interaction, Class Imbalance, Deep Learning, Real-Time Inference

I. INTRODUCTION

Graphical user interfaces mediate almost all interaction between people and software, and understanding how a user engages with an interface - which buttons are clicked, which fields are filled, which menus are opened - is central to automated software testing, accessibility support and usability analytics. Traditionally, this kind of interaction analysis has relied on instrumented logging inside the application itself, which requires access to source code or SDK integration and does not generalize across platforms. An alternative is to treat the interface visually: if the individual GUI elements present in a screenshot can be automatically localized and classified, user actions can be interpreted purely from what is on screen, without any modification to the underlying application.

Object detection provides a natural formulation for this problem. Given a screenshot, the goal is to draw a bounding box around every interactive widget - buttons, text inputs, checkboxes, radio buttons, sliders, toggles, dropdowns, menu items and labels - and assign it the correct category. You Only Look Once (YOLO) is a family of single-stage detectors that frames detection as a single regression problem over a dense grid of candidate boxes, predicting bounding-box coordinates and class probabilities in one forward pass. Because YOLO models avoid the two-stage region-proposal pipeline used by earlier detectors, they offer a favourable accuracy-speed trade-off that is well suited to interactive, near real-time analysis of GUI screenshots.

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

This paper investigates the use of a YOLO detector for GUI element detection as the foundation for downstream user-interaction analysis. A custom dataset of UI screenshots annotated across thirteen widget categories is used to fine-tune the detector, and its behaviour is analysed in detail - including the effect of severe class imbalance between common elements (buttons, inputs) and rare ones (sliders, toggles) - to establish a realistic baseline and a roadmap toward a production-ready GUI understanding pipeline.

II. CHALLENGES IN GUI ELEMENT DETECTION

GUI element detection differs from natural-image object detection in several important ways. First, class frequency is heavily skewed: buttons and text inputs dominate almost any UI corpus, while elements such as sliders, toggles, radio buttons and custom dropdowns appear rarely, so a detector trained with a standard loss will naturally learn the frequent classes far better than the rare ones. Second, many GUI elements are visually similar and small relative to the screen - a checkbox and a radio button may differ by only a few pixels of shape, and icons or labels can be only tens of pixels wide - which is a regime where single-stage detectors historically struggle compared to larger natural-image objects.

Third, interface style varies considerably across platforms, operating systems, themes (light/dark mode) and screen resolutions, so a detector must generalise across a wide visual vocabulary rather than a single consistent design language. Finally, because UI screenshots must typically be processed at interactive speed to support live testing or accessibility tooling, the detector cannot rely on heavy multi-stage pipelines; it must remain fast enough for near real-time inference. These four factors - class imbalance, small/similar objects, style variation and the need for speed - directly shape the dataset, augmentation and architecture choices described in the next section.

III. LITERATURE SURVEY

Object detection has progressed from two-stage, region-proposal architectures toward single-stage detectors that predict boxes and classes directly. Early two-stage detectors first generated candidate object regions and then classified each region independently, achieving strong accuracy at the cost of substantial inference latency, which makes them poorly suited to interactive applications. Deep convolutional backbones such as those introduced for large-scale image classification substantially improved the feature quality available to downstream detection heads, and benchmark corpora such as ImageNet and the PASCAL VOC challenge established the evaluation protocols - precision, recall and mean Average Precision - that remain standard today.

The YOLO family popularised single-stage detection by treating localisation and classification as one joint regression over a grid of anchors, trading a small amount of accuracy for a large gain in inference speed relative to two-stage detectors. Successive iterations - YOLOv3 through YOLOv5 and YOLOv8 - progressively improved backbone design, introduced anchor-free prediction heads, and adopted refined loss formulations such as Distribution Focal Loss (DFL) alongside box and classification losses, which together improve convergence stability, particularly on datasets with imbalanced or small objects.

Within the specific domain of GUI analysis, prior work has explored deep learning for recognising graphical interface components directly from screenshots, motivated by applications in automated UI testing and accessibility. Complementary datasets curated specifically for UI-element detection provide the annotated screenshots needed to fine-tune general-purpose detectors such as YOLO for this domain. Separately, data augmentation surveys and attention-based architectural improvements offer a toolbox of techniques - photometric jitter, geometric transforms, and learned feature re-weighting - that can be adapted to mitigate the class-imbalance and small-object challenges identified above. This paper draws on that combined body of work, applying a YOLO detector, standard detection-style augmentation, and rigorous per-class evaluation to the GUI element detection task.

IV. METHODOLOGY

A. Dataset and Class Distribution

The detector was fine-tuned on a custom UI-element detection dataset consisting of application screenshots annotated with bounding boxes across thirteen widget categories: button, link, input, text_block, checkbox, menu_item, radio, textarea, slider, toggle, dropdown, label and select_menu. As is typical of real-world interfaces, the class distribution is heavily skewed (Table I, Fig. 1): button and input dominate the training set, a second tier of classes (textarea, menu_item,

© 2026 IJIRCCE | Volume 14, Issue 7, July 2026 DOI: 10.15680/IJIRCCE.2026.1407019

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

radio, link) appears with moderate frequency, and several classes - checkbox, slider, toggle, dropdown, label and select_menu - are represented by only a small number of annotated instances. This imbalance is carried through to the validation results discussed in Section VII.

TABLE I. GUI ELEMENT CLASSES AND APPROXIMATE TRAINING-SET FREQUENCY

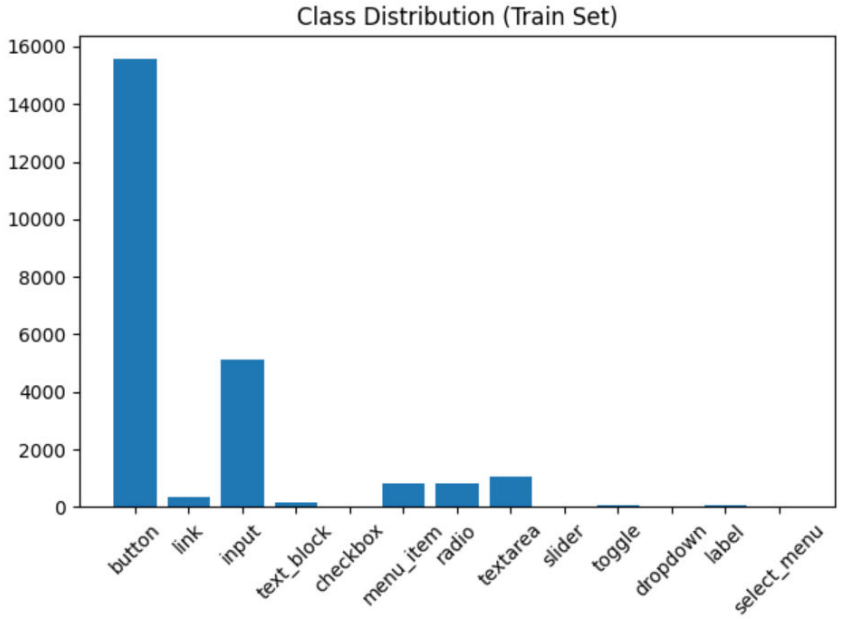


Figure 1. Class distribution of the GUI element detection dataset (training set), showing pronounced imbalance between common widgets (button, input) and rare ones (checkbox, slider, toggle, select_menu).

B. Annotation and Preprocessing

Each screenshot is annotated in YOLO format, storing for every element instance a class index together with its bounding-box centre coordinates, width and height, all normalised relative to the image dimensions. Prior to training, images are resized to a fixed input resolution (640 x 640) while preserving aspect ratio through padding, and pixel values are normalised to match the expected input distribution of the detector. The annotated corpus is split into training and validation subsets (71 validation images containing 2,478 element instances in total), with class labels retained in both subsets so that per-class metrics can be computed.

C. Data Augmentation

To improve robustness to the visual diversity of real interfaces, standard detection-oriented augmentations are applied during training: random horizontal flipping, HSV colour-space jitter to emulate light/dark theme and brand-colour variation, random scaling and translation to emulate different screen resolutions and crops, and mosaic augmentation, which combines four training images into one composite so that the detector is repeatedly exposed to elements at varied scales and surrounding contexts within a single training step. These augmentations particularly benefit the minority classes, since every occurrence of a rare widget is seen in several different augmented contexts across training rather than a single fixed presentation.

D. YOLO Detection Architecture

The detector follows the standard single-stage YOLO design: a convolutional backbone extracts multi-scale feature maps from the input screenshot, a path-aggregation neck fuses features across scales so that both large widgets (text areas, menus) and small ones (icons, checkboxes) are represented, and an anchor-free detection head predicts, at each spatial location, a bounding box, an objectness/box-quality score and a class-probability distribution over the thirteen GUI element categories. Training jointly optimises three loss terms that are tracked throughout this study: a box-regression loss (measuring bounding-box localisation quality), a classification loss (measuring category-prediction accuracy), and a Distribution Focal Loss (DFL) that refines box-edge localisation by modelling each edge as a discrete probability

© 2026 IJIRCCE | Volume 14, Issue 7, July 2026 DOI: 10.15680/IJIRCCE.2026.1407019

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

distribution rather than a single regression target. The combination of these losses is what is reported as box_loss, cls_loss and dfl_loss in the training logs discussed in Section VI.

E. From Detection to User-Interaction Analysis

Detecting and labelling GUI elements is a means to an end rather than the final goal: the motivating use case is to reconstruct what a user did, purely from what is visible on screen. Once every widget in a screenshot (or video frame) is localized and classified, a simple spatial-matching step can associate a recorded touch/click coordinate with the smallest enclosing detected box, yielding a symbolic action such as "tap on button #3" or "text entered into input #1" instead of a raw pixel coordinate. Chaining these symbolic actions across consecutive frames of a screen recording produces an interaction trace - an ordered sequence of (element type, action) pairs - that is directly usable for automated UI test-oracle generation (comparing an observed trace against an expected one), accessibility auditing (checking that every interactive element is reachable and appropriately sized), and usability analytics (mining common navigation paths or identifying elements that are frequently mis-tapped). This framing is also why detection speed matters as much as accuracy: an analysis pipeline that runs at screen-recording frame rates can operate live, rather than only as an offline batch process.

V. MODEL TRAINING AND EVALUATION

A. Training Setup

The detector was trained for 50 epochs at an input resolution of 640 x 640, using mini-batches drawn from the annotated training split, with validation performed after every epoch on the held-out 71-image validation set. Model weights corresponding to the best validation performance were checkpointed automatically. Training and inference were performed using the Ultralytics YOLO implementation, which provides built-in logging of per-epoch losses and validation detection metrics.

B. Evaluation Metrics

Detection performance is reported using precision (the fraction of predicted boxes that correctly match a ground-truth element of the same class), recall (the fraction of ground-truth elements successfully detected), and mean Average Precision at a single IoU threshold of 0.5 (mAP@0.5) as well as averaged across IoU thresholds from 0.5 to 0.95 (mAP@0.5:0.95), which jointly capture both classification correctness and bounding-box localisation quality. Per-image inference latency, broken down into preprocessing, network inference and post-processing (non-maximum suppression) stages, is additionally reported to assess suitability for real-time interaction analysis.

VI. TRAINING STRATEGY

Fig. 2 shows the training and validation loss curves - box, classification and DFL losses - over the course of training. All three losses decrease steadily through the early and middle epochs before flattening in the final stretch, indicating that the detector has largely converged given the available data and that further gains would depend more on data quantity/balance than on additional epochs alone. Table II lists the logged box, classification and DFL losses together with the corresponding validation precision, recall and mAP@0.5 for the final four epochs of training, illustrating the plateau: overall mAP@0.5 stabilises at approximately 0.04 with only small fluctuations from epoch to epoch.

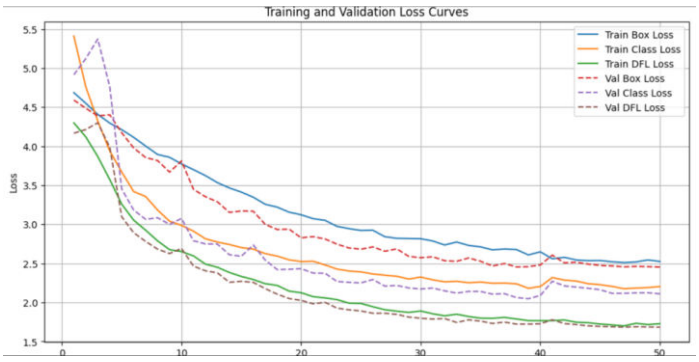


Figure 2. Training and validation loss curves (box, classification and distribution-focal losses) over the training run, showing convergence and a late plateau.

© 2026 IJIRCCE | Volume 14, Issue 7, July 2026 DOI: 10.15680/IJIRCCE.2026.1407019

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

TABLE II. TRAINING LOG SNIPPET (FINAL FOUR EPOCHS)

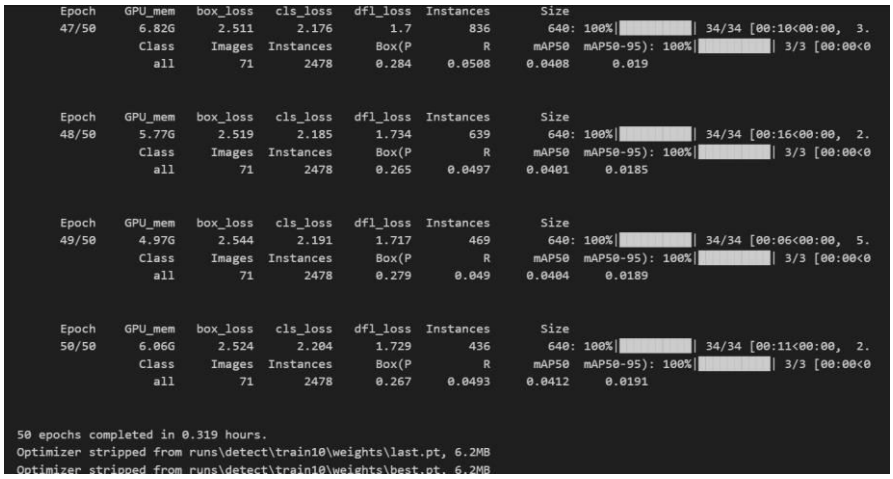


Figure 3. Raw Ultralytics YOLO training-log excerpt for epochs 47-50, reporting per-epoch box/class/DFL losses and validation precision, recall and mAP.

VII. RESULTS AND DISCUSSION

Table III reports the final per-class validation results. The two most frequent classes, button and link, are the only ones with clearly non-trivial detection performance: button reaches a precision of 0.330 and mAP@0.5 of 0.207, while link achieves the highest recall in the table (0.287) and an mAP@0.5 of 0.161. The dropdown class, despite only 79 annotated instances, achieves a moderate precision of 0.383, suggesting its visual signature (a bordered box with a chevron icon) is comparatively easy to localise even with limited data. In contrast, classes with fewer than ten validation instances - textarea, radio, slider, clickable and icon - obtain zero measured precision, recall and mAP, since the detector effectively never learns a reliable decision boundary for categories it observes only a handful of times. The input and menu_item rows show an important caveat of small-sample evaluation: a precision of 1.000 alongside a recall near zero indicates that the few predictions the model does make for these classes happen to be correct, but it fails to propose boxes for most true instances - a classic symptom of insufficient positive training examples rather than a genuinely reliable detector.

TABLE III. PER-CLASS VALIDATION RESULTS (FINAL EPOCH)

Overall, the model attains a validation precision of 0.268, recall of 0.050, mAP@0.5 of 0.041 and mAP@0.5:0.95 of 0.019 across all thirteen classes and 2,478 validation instances - a modest result driven almost entirely by the severe class imbalance in the underlying dataset rather than by a fundamental weakness of the YOLO architecture itself, as shown by the comparatively strong numbers on the well-represented button and link classes.

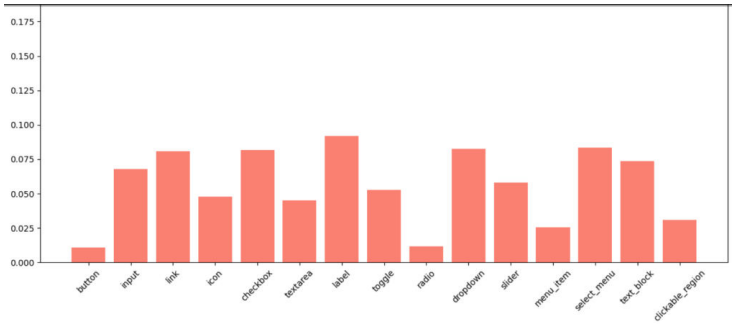


Figure 4. Per-class mAP@0.5:0.95 across the thirteen GUI element categories, highlighting the gap between well-represented classes and data-scarce ones.

© 2026 IJIRCCE | Volume 14, Issue 7, July 2026 DOI: 10.15680/IJIRCCE.2026.1407019

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

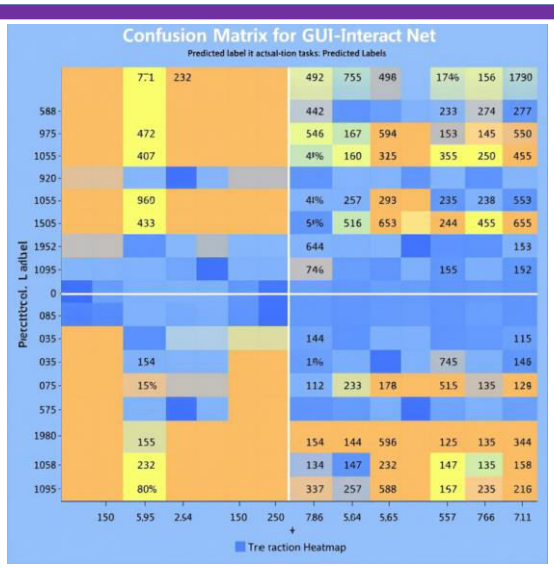


Figure 5. Confusion matrix of predicted versus actual GUI element labels, showing that most confusion occurs among the rare, visually similar classes.

Confusion-matrix analysis (Fig. 5) confirms that most misclassifications involve the rare classes being confused with background or with the dominant button/input/link categories, rather than systematic confusion between visually similar pairs such as checkbox and radio - a pattern consistent with those classes simply lacking enough examples for the detector to form a class-specific representation at all.

Per-image inference timing shows a preprocessing cost of 1.7 ms, network inference of 10.9 ms and post-processing (non-maximum suppression) of 8.1 ms, for a total of roughly 20.7 ms per screenshot - approximately 48 frames per second. This confirms that, independent of the accuracy limitations imposed by the current dataset, the YOLO-based pipeline is comfortably fast enough to support real-time downstream user-interaction analysis, such as live UI testing or accessibility overlays, once trained on a better-balanced dataset.

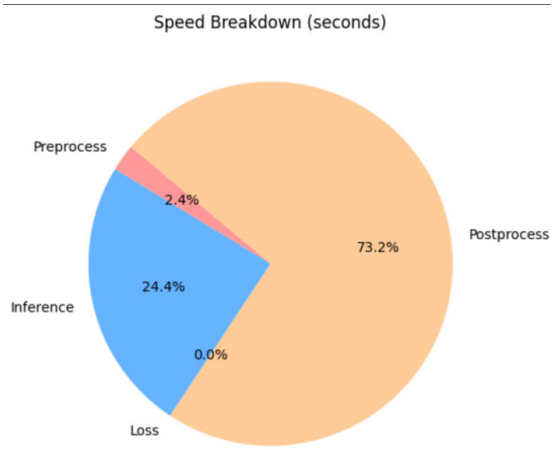


Figure 6. Per-image inference time breakdown across preprocessing, network inference and post-processing (NMS) stages; inference dominates the ~20.7 ms budget, confirming near real-time throughput.

VIII. LIMITATIONS AND FUTURE WORK

The principal limitation of the present study is the small, heavily imbalanced training corpus: several of the thirteen target classes are represented by fewer than ten annotated instances in validation, which is insufficient for the detector to learn

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

a reliable decision boundary regardless of architecture. This is reflected directly in the near-zero precision/recall/mAP obtained for checkbox, slider, radio, toggle and similar minority classes, and is the dominant explanation for the modest overall mAP@0.5:0.95 of 0.019. The immediate next step is therefore dataset rebalancing: collecting or synthetically generating additional annotated examples of the minority classes, combined with oversampling and class-weighted loss terms during training. Transfer learning from detectors pre-trained on large natural-image corpora, followed by fine-tuning on the UI-specific data, is also expected to improve convergence given the current dataset's limited size. On the architectural side, integrating attention-based feature re-weighting into the detection neck may help the model better exploit the limited signal available for visually similar, small GUI elements. Finally, to fully realise the 'user interaction analysis' goal implied by the system's purpose, future work will connect detected element boxes to logged user actions (taps, clicks, keystrokes), enabling automatic reconstruction of interaction sequences - e.g., "user tapped button X, then filled input Y" - directly from screen recordings, supporting automated UI testing, accessibility auditing and usability analytics without any application instrumentation. A further practical consideration for deployment is evaluation on interfaces the detector has not seen during training: because UI style varies across applications, operating systems and themes, a rigorous next step is to test the fine-tuned model on screenshots drawn from applications outside the training distribution, reporting accuracy separately for in-distribution and out-of-distribution interfaces. Establishing this generalisation gap will indicate whether additional fine-tuning is needed per application family or whether a single detector can serve a broad range of interfaces, which is an important product decision before the pipeline is used for live accessibility or testing tools.

IX. CONCLUSION

This paper applied a YOLO single-stage object detector to the task of GUI element detection as a foundation for vision-based user-interaction analysis, fine-tuning the detector on a custom, thirteen-class UI-element dataset. Training converged steadily, as shown by decreasing box, classification and distribution-focal losses, and per-image inference latency of roughly 20.7 ms confirms the approach is fast enough for real-time use. However, per-class evaluation revealed a stark class-imbalance effect: frequent classes such as button and link were detected with useful precision and mAP, while classes represented by only a handful of training instances were not reliably detected at all. These findings validate YOLO as an architecturally sound and sufficiently fast choice for this problem while clearly identifying dataset rebalancing - rather than model capacity - as the key lever for improving GUI element detection accuracy, and set a concrete, data-centric roadmap for the next iteration of this work.

REFERENCES

[1] A. A. Rahmadi and A. Sudaryanto, "Visual Recognition of Graphical User Interface Components Using Deep Learning Technique," J. Ilmu Komputer dan Informasi, vol. 13, no. 1, pp. 35-42, 2020.
[2] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," Proc. of NeurIPS, 2015.
[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," Proc. of CVPR, pp. 770-778, 2016.
[4] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," Proc. of CVPR, 2016.
[5] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," arXiv:1804.02767, 2018.
[6] G. Jocher et al., "YOLOv5," https://github.com/ultralytics/yolov5, 2020.
[7] C. Wang et al., "YOLOv8: A Real-Time Object Detection Framework with Distribution Focal Loss," arXiv:2304.08344, 2023.
[8] J. Deng et al., "ImageNet: A Large-Scale Hierarchical Image Database," Proc. of CVPR, pp. 248-255, 2009.
[9] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes (VOC) Challenge," Int. Journal of Computer Vision, vol. 88, no. 2, pp. 303-338, 2010.
[10] UI-Elements-Detection-Dataset. Available: https://github.com/YashJain0412/UI-Elements-Detection-Dataset
[11] A. Shorten and T. M. Khoshgoftaar, "A Survey on Image Data Augmentation for Deep Learning," Journal of Big Data, vol. 6, no. 60, 2019.
[12] A. Vaswani et al., "Attention Is All You Need," Proc. of NeurIPS, 2017.

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi crossref

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

9940 572 462 6381 907 438 ijircce@gmail.com

www.ijircce.com

Scan to save the contact details