applied sciences

MDPI

Article

Enhancement of GUI Display Error Detection Using Improved Faster R-CNN and Multi-Scale Attention Mechanism

Xi Pan [1], Zhan Huan [1,*], Yimang Li [2] and Yingying Cao [1]

[1] School of Microelectronics and Control Engineering, Changzhou University, Changzhou 213164, China; s21060858022@smail.cczu.edu.cn (X.P.); s22060858002@smail.cczu.edu.cn (Y.C.)
[2] School of Mechanical Engineering and Rail Transit, Changzhou University, Changzhou 213164, China; liyimang@cczu.edu.cn
* Correspondence: hzh@cczu.edu.cn

Abstract: Graphical user interfaces (GUIs) hold an irreplaceable position in modern software and applications. Users can interact through them. Due to different terminal devices, there are sometimes display errors, such as component occlusion, image loss, text overlap, and empty values during software rendering. To address the aforementioned common four GUI display errors, a target detection algorithm based on the improved Faster R-CNN is proposed. Specifically, ResNet-50 is used instead of the traditional VGG-16 as the feature extraction network. The feature pyramid network (FPN) and the enhanced multi-scale attention (EMA) algorithm are introduced to improve accuracy. ROI-Align is used instead of ROI-Pooling to enhance the generalization capability of the network. Since training models require a large number of labeled screenshots of errors, there is currently no publicly available dataset with GUI display problems. Therefore, a training data generation algorithm has been developed, which can automatically generate screenshots with GUI display problems based on the Rico dataset. Experimental results show that the improved Faster R-CNN achieves a detection accuracy of 87.3% in the generated GUI problem dataset, which is a 7% improvement compared to the previous version.

Keywords: automated testing; object detection; Faster R-CNN

Citation: Pan, X.; Huan, Z.; Li, Y.; Cao, Y. Enhancement of GUI Display Error Detection Using Improved Faster R-CNN and Multi-Scale Attention Mechanism. *Appl. Sci.* **2024**, *14*, 1144. https://doi.org/10.3390/app14031144

Academic Editor: José Salvador Sánchez Garreta

Received: 27 December 2023
Revised: 22 January 2024
Accepted: 25 January 2024
Published: 30 January 2024

1. Introduction

With the rapid development of the industrial internet, industrial software has become an integral component of industrial production. However, due to the complexity and diversity of industrial software, its testing process often necessitates significant manpower and resources. In order to enhance testing efficiency and accuracy, automated testing technology has emerged. Automated testing allows for continuous testing throughout the software development process, facilitating prompt identification and resolution of issues, thus mitigating software development costs and risks. Utilitarian tools such as Monkey [1,2] and Dynodroid [3] have gained widespread industrial application as practical automated testing instruments. Yet, these automated testing tools primarily concentrate on detecting crash issues rather than UI display problems. As an automated testing approach, GUI automation testing has been extensively implemented in software testing to boost efficiency, reduce costs, and improve quality. Central to GUI automation testing is the object recognition technology for target objects, which includes various controls within the GUI interface, such as buttons, text fields, and drop-down menus. Consequently, the accuracy and stability of object recognition technology bear the utmost significance for the effectiveness and reliability of automated testing.

In recent years, the rapid progress in artificial intelligence technology has given rise to innovative ideas and methodologies aimed at improving GUI testing. This involves leveraging deep learning for the automatic recognition of GUI elements in screen captures. The recognition of GUI elements within images, based on pixels, can be viewed as a

specialized object detection task within the domain of GUI testing. Object detection, a computer vision technology, facilitates the identification of specific semantic objects within digital images and videos.

This paper presents a target detection algorithm based on an improved Faster R-CNN. Initially, ResNet-50 is adopted as the feature extraction network, with the incorporation of enhanced multi-scale attention (EMA) within ResNet-50 to bolster attention to relevant channel information across both channel and spatial dimensions, thereby augmenting performance. Feature pyramid network (FPN) is introduced to bolster the algorithm's capability of handling small targets, resulting in improved robustness and efficiency during the learning process. ROI-Align supersedes ROI-Pooling for target feature mapping, effectively addressing false alarms and missed detections caused by limited model generalization and ultimately enhancing detection accuracy. Additionally, a training data generation algorithm has been devised, which automatically generates datasets with GUI display problems based on the Rico dataset.

The structure of this paper is as follows. Section 2 explores the existing research in the field of GUI-related studies. Section 3 provides a comprehensive exposition of the training data generation algorithm based on the Rico dataset, the improved Faster R-CNN, and their respective components. Section 4 describes the experimental setup and preparation, as well as the analysis of experimental results. Finally, in Section 5, this paper presents the conclusion and discusses future work.

2. Related Work

Graphical user interfaces (GUIs) serve as visual conduits between applications and users, warranting substantial attention to UI design quality. Developers craft UIs that not only interact effectively with users but also uphold visual aesthetics. Researchers strive to enhance UI quality by aiding software developers and UI designers in GUI search through image features. Reiss et al. introduced a method of generating complete and functional interactive user interfaces by employing code search to locate requisite interface sketches [4]. Behrang et al. proposed a code search technique leveraging the growing pool of open-source applications in public repositories to offer users foundational code for their desired applications [5]. Yang et al. devised a tool for automated detection of UI design issues, capable of statistically and concretely exemplifying consistency and violations for diverse design criteria [6]. Yeh et al. introduced a system utilizing GUI screenshots for visual search and automated graphical user interfaces [7]. Qian et al. harnessed computer vision techniques for robotically operating applications through a touch screen, automating GUI testing [8]. Chen et al. introduced a deep learning-based method encoding visual and text information to recover missing labels for the current UI, simplifying their retrieval via text queries [9]. They also examined the limitations and effective design of deep learning-based object detection methods for UI components, incorporating a new top-down strategy combined with a deep learning-based GUI text model [10]. Moran et al. scrutinized post-implementation GUI by comparing image similarity and employing computer vision techniques to ensure alignment with the original design [11]. In their subsequent work, they investigated and summarized GUI changes in constantly evolving applications [12].

Display error detection in Android apps, specifically component occlusion, is a challenging task. One approach is to use deep learning techniques to model the visual information of GUI screenshots and detect display issues [13]. Another approach is to utilize machine learning to automatically detect GUI errors, including widget errors, and analyze the positional relationship between widgets to detect more complex errors [14,15]. Regarding image loss issues, Li et al. conducted an empirical study on real-world Android apps and developed a static issue detection tool called TAPIR, which successfully detected previously unknown image-displaying issues [16]. Additionally, Guo et al. presented iFixDataloss, a tool that automatically detects and fixes data loss issues in Android apps, including those caused by screen rotation. iFixDataloss outperformed existing techniques in terms of the number of detected issues and the quality of generated patches [17]. Display

errors such as text overlap and empty values can negatively impact app usability and user experience. Nighthawk is a fully automated approach that uses deep learning to detect GUIs with display issues and locate the specific region of the issue in the GUI [18].

For effective GUI display, automation testing dynamically detects GUI elements. Mirzaei et al. proposed a fully automatic method utilizing a novel composition technique to generate GUI system tests for Android applications [19]. Baek et al. introduced an automated testing framework with a multi-level GUI comparison criterion (multi-level GUICC) for Android applications [20]. Su et al. presented a novel automated model-based testing approach to enhance GUI testing [21]. Gao et al. and Li et al. scrutinized potential GUI rendering issues and devised automatic detection methods. Gao et al. introduced a mobile application for GUI rendering analysis tools [22]. In recent years, deep learning techniques have gained prominence in GUI automation testing. White et al. proposed a GUI testing improvement by automatically identifying GUI widgets in screen screenshots using machine learning techniques [23]. Degott et al. introduced a reinforcement learning-based approach that autonomously learns interactions suitable for elements, guiding test generation [24]. Diverging from traditional GUI testing methods, deep learning techniques employ computer vision to detect GUI components on the screen and determine subsequent steps by identifying anomalies. Thomas et al. suggested employing machine learning to automatically identify GUI widgets in screen screenshots and using this information to guide testing [23].

There are several approaches for data generation in machine learning. One approach is synthetic data generation, which involves generating artificial data that is similar to real-world data. This can be done using techniques such as combinatorial testing and variational autoencoder [25]. Another approach is constraint-based generation, where a dataset is generated to satisfy given support constraints on itemsets [26]. Probabilistic generative modeling is another approach that models data generation as a sampling process from a parametric distribution, often encoded as a neural network [27].

3. Preliminaries

3.1. Faster R-CNN

The Faster R-CNN represents a two-stage object detection algorithm [28,29], an enhancement derived from the R-CNN and Fast RCNN frameworks. In contrast to single-stage object detection algorithms like YOLO [29,30] and SSD [31], Faster R-CNN divides the object detection process into two stages: candidate box generation and subsequent object recognition within those candidate boxes. Consequently, while this two-stage approach yields higher accuracy, it exhibits a relatively slower detection speed compared to single-stage counterparts. Following feature extraction, the data undergo initial processing through the region proposal network (RPN) to generate candidate regions. The content of these regions is then categorized as foreground or background. If considered foreground, it undergoes further processing via ROI pooling for subsequent object classification and position regression; otherwise, the candidate box is discarded. Concurrently, if determined as foreground, the alternate branch of the RPN network adjusts the size and geometric coordinates of the candidate box. The Faster R-CNN employs ResNet-50 as the backbone feature extraction network [32] and integrates the feature pyramid network (FPN) structure [33]. Utilizing three feature maps at distinct scales for multi-scale feature fusion, the model outputs prediction results aimed at enhancing accuracy.

3.2. ResNet-50

The residual network (ResNet) primarily consists of residual blocks. Prior to the advent of ResNet, deep learning training encountered limitations in achieving considerable depth, as the number of layers led to network degradation. ResNet emerged to address this degradation phenomenon. Various ResNet model variants exist, including ResNet34, ResNet-50, and ResNet101, among others. The model structure of ResNet-50 is illustrated in Figure 1. ResNet-50 is segmented into 5 stages, where Stage 0 involves input preprocessing,

and the subsequent 4 stages comprise residual blocks with a similar structure. Comprising a total of 50 layers, ResNet-50 includes 48 convolutional layers, 1 max pooling layer, and 1 average pooling layer. The input UI image undergoes initial feature extraction via the ResNet-50 backbone feature extraction network. Subsequently, the outputs from stage 1 to stage 4 within the backbone network serve as features at distinct scales for feature fusion in the feature pyramid network, as depicted in Figure 1.

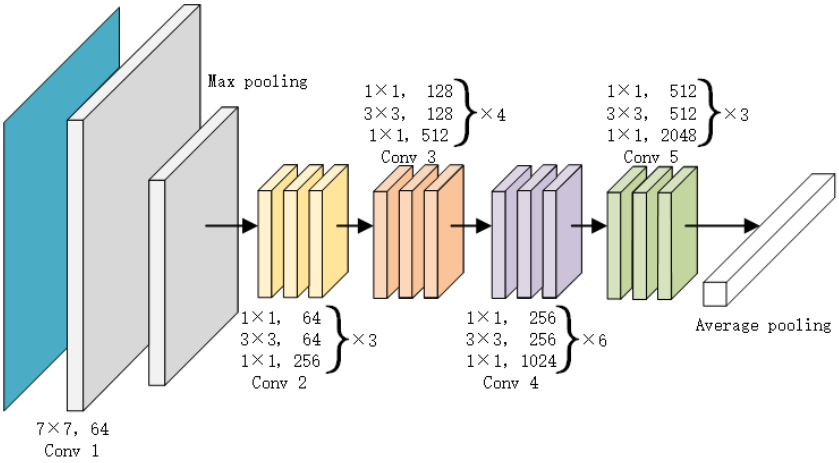

Figure 1. Network structure of ResNet-50.

3.3. Feature Pyramid Networks

The feature pyramid network (FPN) extracts feature maps at diverse scales and supplies them to subsequent networks for prediction tasks. As depicted in Figure 2, FPN is primarily composed of a bottom-up path and a top-down path. The bottom-up path encompasses the forward feature extraction process within the deep convolutional network, while the top-down path involves the upsampling of feature maps from the final convolutional layer. Lateral connections play a crucial role in amalgamating features from deep and shallow convolutional layers. This integration accounts for its commendable detection performance, particularly for small objects. In the FPN, an ROI with width (w) and height (h) is assigned to level the Pk of the feature pyramid in Equation (1). Here, value 224 represents the standard ImageNet pre-training size, where k_0 is the target width and height ($w \times h = 224^2$) to which the ROI should be mapped horizontally.

$$k = \left\lfloor k_0 + \log_2(\sqrt{wh}/224) \right\rfloor \tag{1}$$

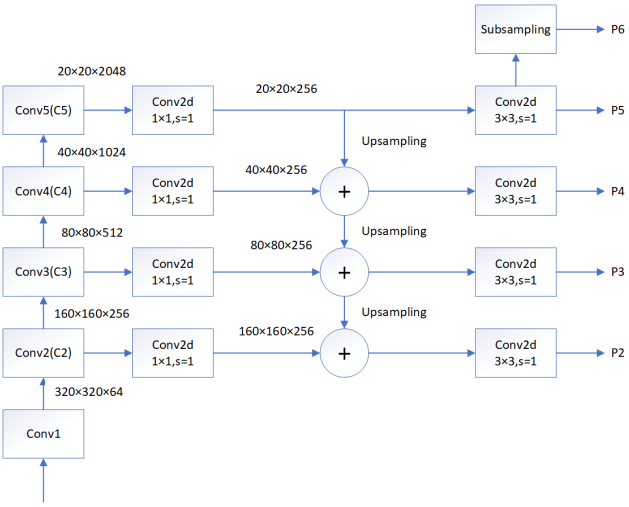

Figure 2. Network structure of ResNet-50 and feature pyramid networks.

4. Research Method

4.1. Motivation

To ensure the accurate rendering of graphical user interfaces (GUIs), software companies must utilize a considerable workforce of testers who are dedicated to the comprehensive evaluation of their applications. While these testers play a crucial role in detecting display anomalies within GUIs, various limitations persist. Firstly, the verification of correct GUI presentation necessitates a substantial number of testers who must manually navigate through numerous pages using diverse interaction methods. Additionally, they are required to scrutinize the UI display across devices with distinct operating system versions, resolutions, and screen dimensions. Secondly, testers often inadvertently neglect minor errors in GUI presentation, such as text overlapping and component occlusion. Addressing these challenges calls for the imperative development of automated GUI testing methodologies.

4.2. Automatic Training Data Generation

Object detection algorithms necessitate an extensive dataset for training models, making the collection of numerous screen captures depicting GUI display issues labor-intensive. Currently, there is a dearth of publicly available datasets explicitly tailored for addressing GUI display problems. This paper introduces a methodology for the automatic generation of training data. Our approach relies on the Rico dataset [34], comprising design data from 93,000 Android applications and offering over 66,000 UI screen captures, each paired with its corresponding JSON file.

Algorithm 1 presents a heuristic-based approach for the automatic generation of training data. Initially, it accepts input parameters, including error-free screen captures, their associated JSON files, the specified category of UI display problems to be generated, and pre-prepared image icons for the "image missing" class. The algorithm systematically traverses the JSON file, extracting relevant component information, such as TextView and ImageView. Utilizing this component information, Algorithm 1 duplicates and adjusts positions and sizes according to predefined rules, thereby generating UI screen captures with the associated issues, as illustrated in Figures 3–6.

In the event of a component occlusion error, the text or image becomes obscured by other components. To induce such errors, Algorithm 1 initiates the process by generating a color block matching the background of the current component, possessing the same width but differing in height. Subsequently, this generated color block is repositioned to partially cover the component.

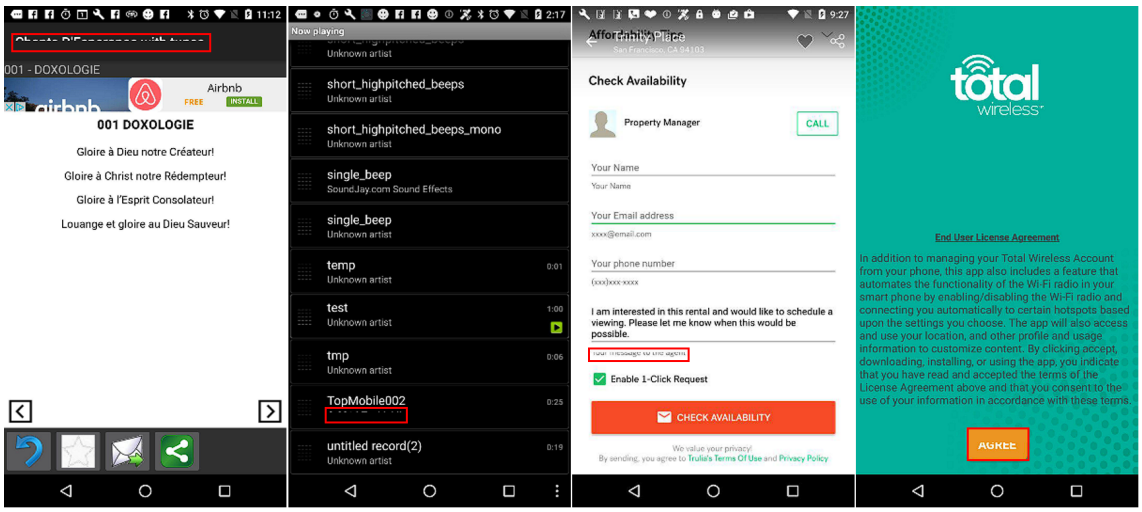

Figure 3. Example of an algorithm generating a component occlusion problem.

Algorithm 1: Heuristic-based training data auto-generation.

```
Input  : category
Output : image
if category is 'component occlusion' then
    rand ← randomUniform(0, 1);
    if rand > 0.5 then
        range ← randomUniform(0.5, 0.7);
    end
    else
        range ← randomUniform(−0.7, −0.5);
    end
    Generate block image with [width, height × range] and background;
    if range ≥ 0 then
        pasteBlockOn(image, x1, y1);
    end
    else
        pasteBlockOn(image, x1, y2 + height × range);
    end
end
if category is 'missing image' then
    generateAndPasteBlock(image, x1, y1, genericWidth, genericHeight);
    resizeIconTo(image, x1, y1, genericWidth, genericHeight);
end
if category is 'null value' then
    generateAndPasteBlock(image, x1, y1, genericWidth, genericHeight);
    drawNullStringOn(image, x1, y1, font, background);
end
if category is 'text overlap' then
    range ← randomUniform(0.8 × genericWidth, 0.85 × genericWidth);
    rand ← randomUniform(0, 1);
    cropped → cropImageBy(image, x1, y1, x2, y2);
    if rand > 0.5 then
        pasteCroppedOn(image, cropped, x2 − range, y1);
    end
    else
        pasteCroppedOn(image, cropped, x1 − genericWidth + range, y1);
    end
end
return image;
```

In the case of an image missing error, where the original image fails to display correctly and is represented by an image icon, Algorithm 1 employs four common image icons to emulate situations where the image is not properly rendered. Initially, a color block, identical in size to the current image and with a background color corresponding to the predetermined color of the image, is created. The display area of the original image is then covered.

In instances of a null value error, manifested as the display of "null" in the text area, Algorithm 1 first determines the height of text in a TextView and converts it into a font of the same height. Subsequently, a "null" text with an identical background color is generated and superimposed onto the original TextView.

When confronted with a text overlap error, characterized by the overlapping of text content, thereby impairing readability, Algorithm 1 initially acquires the position and size

of the TextView. Subsequently, it duplicates an identical TextView and, through image cropping to render it transparent, achieves the effect of text overlap by repositioning it.

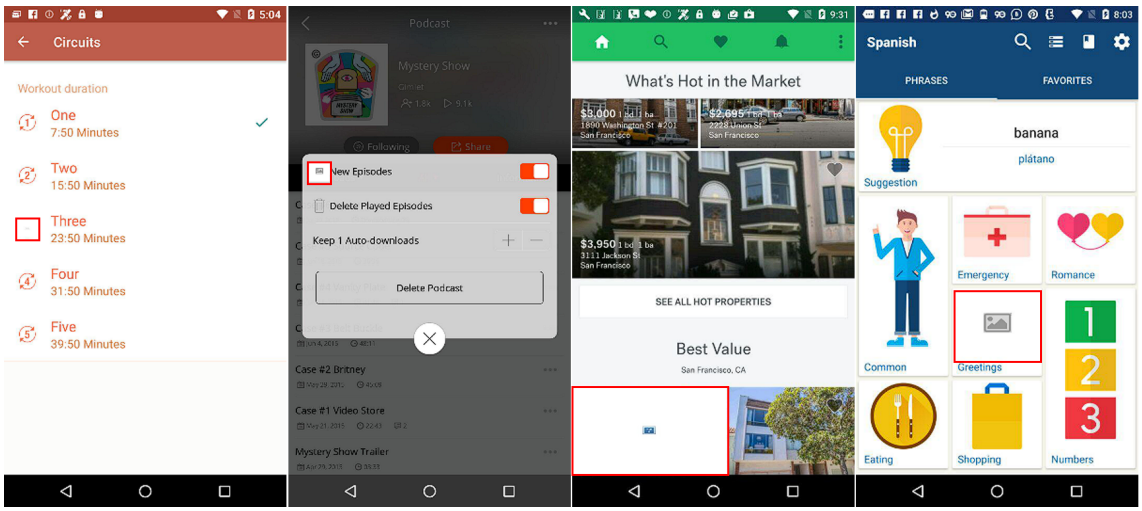

Figure 4. Example of the algorithm generation of the missing image problem.

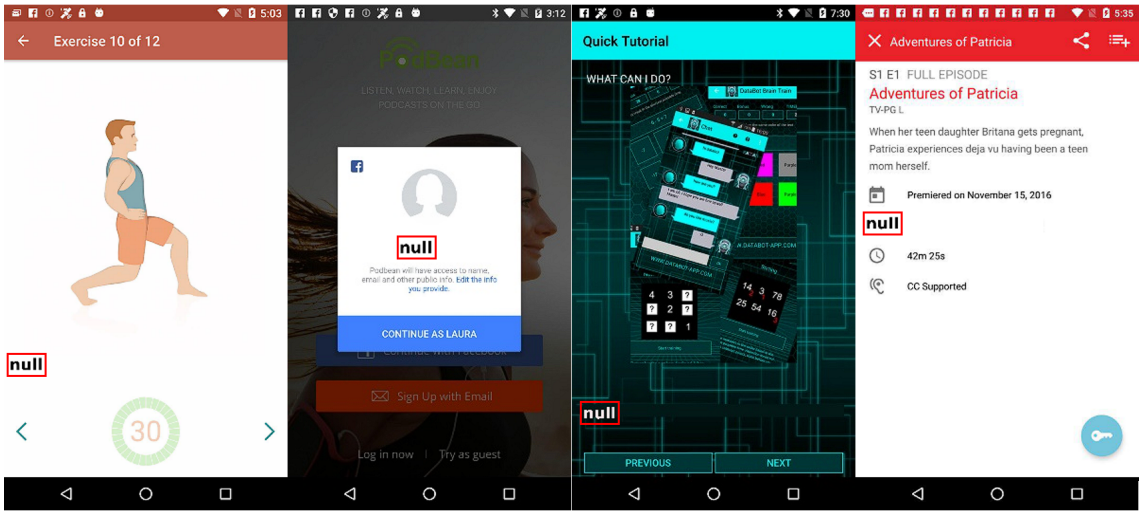

Figure 5. Example of the algorithmic generation of the null problem.

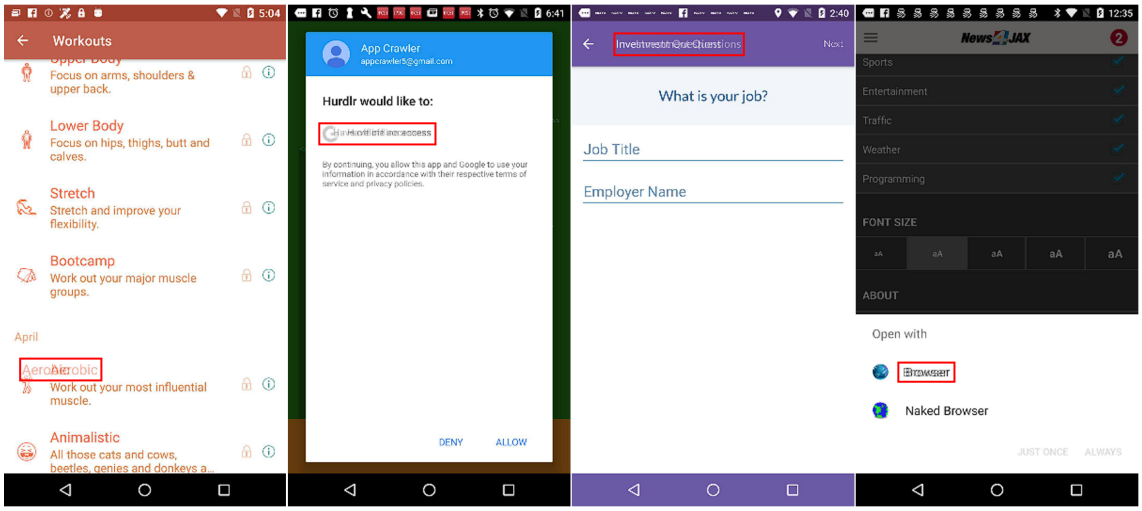

Figure 6. Examples of the generation of the text overlap problem example.

4.3. Improved Faster R-CNN with Multi-Scale Attention Mechanism

4.3.1. Overall Framework

This paper presents an enhanced Faster R-CNN algorithm shown in Figure 7, featuring targeted improvements. Notably, the conventional VGG-16 feature extraction network in the traditional Faster R-CNN is substituted with ResNet-50. Furthermore, ResNet-50 incorporates enhanced multi-scale attention (EMA) and a feature pyramid network (FPN). The enhanced multi-scale attention (EMA) mechanism is integrated after the bn3 output of each residual network in ResNet-50, positioned before the ReLU function. The introduced feature pyramid network (FPN) facilitates the retrieval of more pertinent information pertaining to smaller objects. Ultimately, the conventional ROI-Pooling in Faster R-CNN undergoes replacement with a more precise ROI-Align, augmenting the network's generalization capabilities.

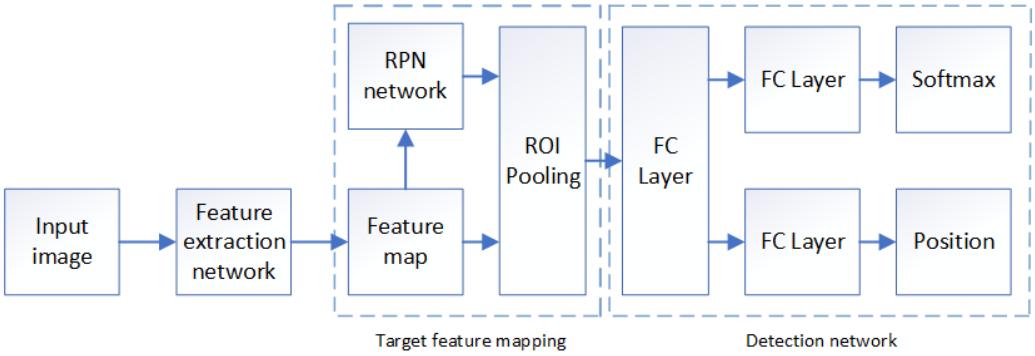

Figure 7. Network structure of the improved Faster R-CNN.

4.3.2. Multi-Scale Attention Mechanism

Enhanced multi-scale attention (EMA) represents an innovative and efficient non-dimensional reduction attention model based on the grouping structure [35]. This model departs from the sequential processing method of coordinated attention (coordinate attention, CA) [36].

As illustrated in Figure 8, coordinate attention (CA) employs an approach akin to the SE attention module, utilizing a global average pooling operation to model inter-channel information. However, in contrast to SE, CA integrates spatial positional information into the channel attention map to enhance feature aggregation. The 1D global average pooling representation, capturing global information along the horizontal dimension in the parallel pathway, is expressed as Equation (2).

$$z_c^H(H) = \frac{1}{W} \sum_{0 \le i \le W} x_c(H, i) \tag{2}$$

Here, C denotes the number of input channels, and H and W represent the spatial dimensions of the input features. Another pathway emerges directly from the 1D global average pooling along the horizontal direction, resembling a collection of positional information along the vertical direction. Therefore, the formula for the pooling output is articulated in Equation (3).

$$z_c^W(W) = \frac{1}{H} \sum_{0 \le j \le H} x_c(j, W) \tag{3}$$

In summary, CA transforms partial channels into batch dimensions and organizes channel dimensions into multiple sub-features, ensuring a robust spatial semantic distribution within each feature group. This strategy circumvents dimension reduction through conventional convolution, facilitating both strong generalization and computational capabilities. Moreover, it offers the advantages of flexibility and lightweight design, rendering

it suitable for diverse computer vision tasks aimed at enhancing overall performance. The configuration of the EMA module is elucidated in Figure 8.

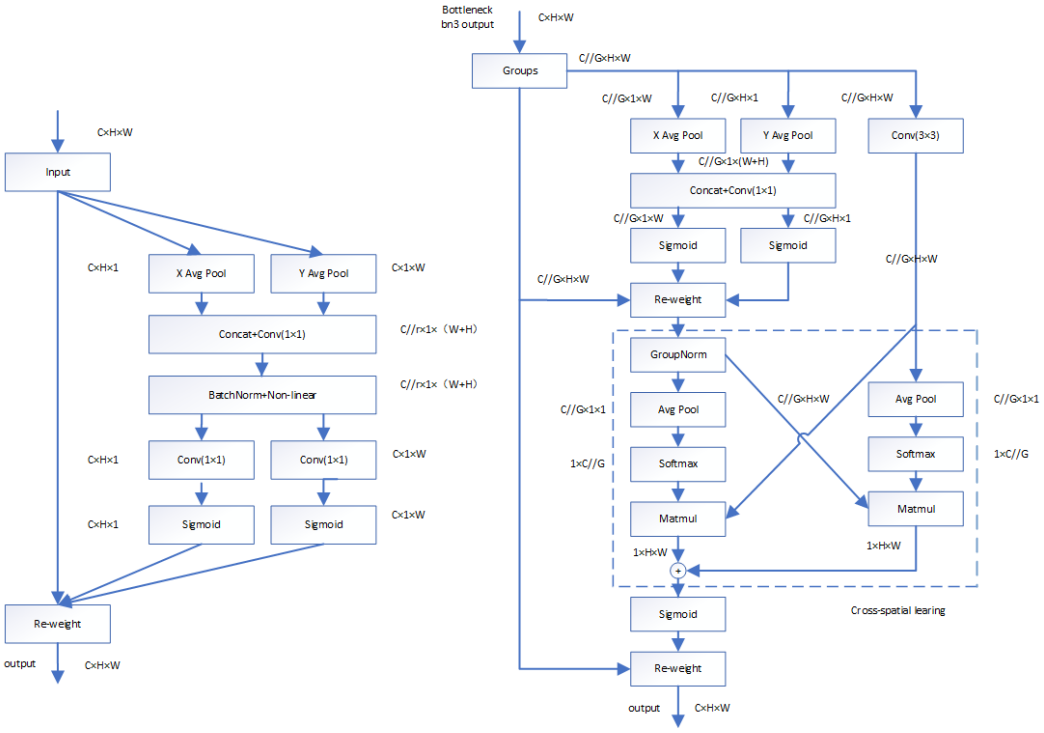

Figure 8. Network structure of coordinate attention (**left**) and multi-scale attention (**right**).

Specifically, EMA proposes a cross-space information aggregation method with different spatial dimension directions. First, EMA uses three parallel pathways to extract attention-weight descriptors of grouped feature maps, with two parallel pathways in the 1×1 branch and the third pathway in the 3×3 branch. In the 1×1 branch, there are two average pooling layers to encode channels along the X and Y spatial directions. After factorized convolutions in the 1×1 branch, the output is decomposed into two tensors, which are fitted with two Sigmoid functions to model the 2D binomial distribution of the linear convolutions. In the 3×3 branch, only a 3×3 convolution is used to capture multi-scale features. Then, the two tensors outputted by the Sigmoid functions are merged into one tensor through Re-weighting. The 2D global average pooling is applied to encode global spatial information in the output of the branches, and the output of the smallest branch is directly transformed into the corresponding dimension shape before the joint activation mechanism of channel features. The formula for the 2D global pooling operation is shown in Equation (4).

$$z_c = \frac{1}{H \times W} \sum_{j}^{H} \sum_{i}^{W} x_c(i, j) \tag{4}$$

In the mathematical expressions, c denotes the number of input channels, H and W signify the spatial dimensions of the input features, and xc represents the input feature at the c-th channel. Following the application of the softmax fitting transformation, the output feature maps within each group are computed by aggregating two spatial attention weight values. Subsequently, these maps are output as tensors of equivalent size to the input through the Sigmoid function.

4.3.3. Improved ROI-Alignment

The ROI-Pooling layer was originally introduced in Fast R-CNN and has since found widespread application in Faster R-CNN. In the Faster R-CNN framework, the RPN network generates candidate boxes of diverse sizes. Subsequently, in the ROI-Pooling

layer, feature pooling is executed to map candidate boxes of various sizes onto fixed-size feature maps. ROI-Pooling incorporates two rounding quantizations. The first quantization converts the floating-point coordinates of candidate boxes into integers, while the second quantization further divides the quantized candidate regions into N × N equally-sized sub-regions, rounding the edges of each sub-region, as illustrated in Figure 9. However, ROI-Pooling suffers from a drawback where the two rounding operations lead to an approximation of the region's value and varying sizes of each region. This issue can result in position errors when measuring boxes, consequently diminishing detection accuracy, especially for smaller targets. Given the focus of this paper on detecting errors in GUI interface display, some of which entail small geometric areas and occupy fewer image pixels, we address this limitation by employing ROI-Align as an enhancement. This improves the network's generalization capability and detection accuracy, as depicted in Figure 9.

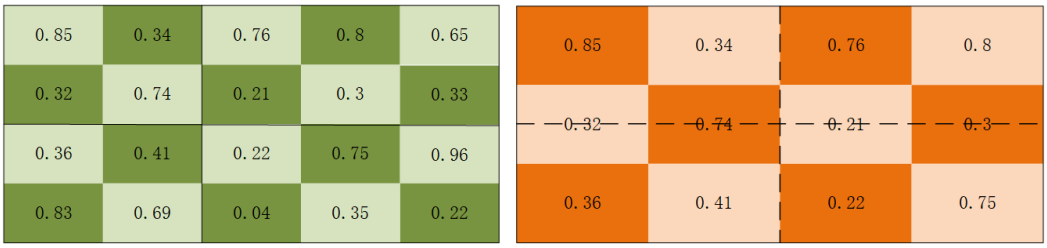

Figure 9. Example of ROI-Pooling (**left**) and ROI-Align (**right**).

ROI-Align, introduced in Mask RCNN [37], addresses the quantization errors inherent in ROI-Pooling. In contrast to ROI-Pooling, ROI-Align retains the original floating-point values, eliminating the need for quantization. Bilinear interpolation is employed to calculate pixel values based on floating-point image coordinates, ensuring precise alignment between the pixels of the original image and the feature map (Figure 10).

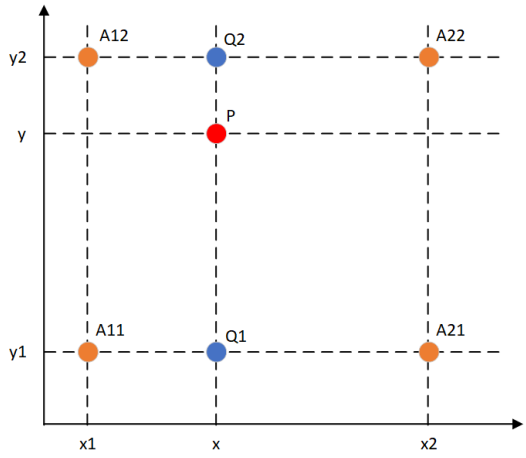

Figure 10. Example of bilinear interpolation.

The bilinear interpolation algorithm, as shown in Figure 7, first performs linear interpolation between A11 and A21 to obtain Q1. Similarly, Q2 can be obtained by performing linear interpolation between A12 and A22. Q1 and Q2 can be calculated using Equation (5).

$$\begin{aligned} f(Q_1) &\approx \frac{x_2 - x}{x_2 - x_1} f(A_{11}) + \frac{x - x_1}{x_2 - x_1} f(A_{21}) \text{ Where } Q_1 = (x, y_1) \\ f(Q_2) &\approx \frac{x_2 - x}{x_2 - x_1} f(A_{12}) + \frac{x - x_1}{x_2 - x_1} f(A_{22}) \text{ Where } Q_1 = (x, y_2) \end{aligned} \tag{5}$$

Then, linear interpolation is performed between Q1 and Q2 in the y direction with Equation (6).

$$f(P) \approx \frac{y_2 - y}{y_2 - y_1} f(Q_1) + \frac{y - y_1}{y_2 - y_1} f(Q_2) \tag{6}$$

Finally, the final position is obtained through Equation (7).

$$\begin{aligned} f(x,y) \approx & \frac{(x_2 - x)(y_2 - y)}{(x_2 - x_1)(y_2 - y_1)} f(A_{11}) + \frac{(x_2 - x)(y - y_1)}{(x_2 - x_1)(y_2 - y_1)} f(A_{12}) \\ & + \frac{(x - x_1)(y_2 - y)}{(x_2 - x_1)(y_2 - y_1)} f(A_{21}) + \frac{(x - x_1)(y - y_1)}{(x_2 - x_1)(y_2 - y_1)} f(A_{22}) \end{aligned} \tag{7}$$

As illustrated in Figure 9, the dashed box represents the feature map, while the solid box signifies an ROI feature (with a size of 2×2) featuring four sampling points per sub-region. ROI-Align utilizes bilinear interpolation to estimate pixel values corresponding to these grid points and subsequently employs max pooling or average pooling for each sub-region to yield a 2×2 output result.

4.3.4. Evaluation Metrics

All quantitative analysis experiments in this paper are evaluated using the AP50 metric, a standard evaluation metric in the COCO dataset, to measure the detection performance of the network on test images. To explain the meaning of the aforementioned metric, the concepts of precision and recall are introduced. The formulas for precision and recall are shown in Equation (8).

$$\begin{aligned} \text{Precision} &= \frac{TP}{TP + FP} \\ \text{Recall} &= \frac{TP}{TP + FN} \end{aligned} \tag{8}$$

Here, TP represents true positives, FP represents false positives, FN represents false negatives, and TN represents true negatives. Precision indicates the proportion of correctly detected objects (when the intersection over union, IOU, between the predicted bounding box and the ground truth box exceeds a certain threshold) among all detected objects. Recall represents the proportion of detected objects by the model among the total number of true objects. Normally, precision and recall are mutually exclusive, meaning that high recall tends to have lower precision and vice versa. Therefore, using these two metrics alone cannot intuitively compare the performance of different networks. Hence, this paper uses the AP metric to quantitatively analyze the algorithm's performance. The formula is seen in Equation (9).

$$\begin{aligned} AP &= \int_0^1 P_{\text{smooth}}(r) dr \\ mAP &= \sum_{i=1}^{N} AP(i)/N \end{aligned} \tag{9}$$

In the equation, $P_{\text{smooth}}(r)$ represents the PR curve after applying smoothing, where the curve is plotted with recall on the x-axis and precision on the y-axis within the range of 0 to 1. N represents the number of validation sets.

5. Experiments

5.1. Experiment Settings

The experimental hardware configuration for this study comprises an Intel Xeon Cascade Lake 8255C (2.5 GHz) processor, 16 GB + 16 GB dual-channel memory, and an NVIDIA Tesla V100 graphics card. Software conditions involve a deep learning environment running on Ubuntu 20.04 with the PyTorch 1.31 framework. In this experiment, we employ the transfer learning approach within the realm of deep learning. Initially, pre-training

is executed on the COCO2017 dataset, followed by training the enhanced Faster R-CNN algorithm using the pre-trained parameters. The objective is to identify four types of GUI display issues: image missing, component occlusion, text overlap, and empty values.

This experiment is dedicated to investigating four prevalent GUI display issues: component occlusion, image missing, text overlap, and empty values. Leveraging the Rico dataset, an algorithm is employed to autonomously produce images representative of these aforementioned GUI issues. As shown in Table 1, the resulting training dataset comprises a total of 40,000 images, evenly distributed with 10,000 images for each issue type, accompanied by corresponding annotation files. To facilitate model training and evaluation, the dataset is partitioned into training, validation, and test sets with an 8:1:1 ratio.

Table 1. GUI displays a problematic dataset.

	Text Overlap	Image Missing	Component Occlusion	Empty Values
Training Set	8000	8000	8000	8000
Validation Set	1000	1000	1000	1000
Test Set	1000	1000	1000	1000
Sum	10,000	10,000	10,000	10,000

5.2. Experiment Result Analysis

To ensure a fair experimental comparison, we adopt the method of controlling variables. All aspects, aside from the variation in networks, encompassing the dataset, training steps, and parameter configurations, remain consistent. Throughout the experimental process, the datasets containing GUI display issues are partitioned into proportions of 8:1:1, as previously described. Models undergo training utilizing pre-trained models from the COCO2017 dataset, with 20 training iterations, an initial learning rate of 0.005, and a batch size of 8; the optimal weight parameters are selected based on the training results.

In Experiment 1, improvements are sequentially implemented in the original Faster R-CNN network. Specifically, VGG16 is replaced with ResNet-50 as the feature extraction network. A feature pyramid network (FPN) and multi-scale attention (EMA mechanism are incorporated, and the more accurate ROI-Align replaces ROI-Pooling. Figure 11 illustrates a bar chart depicting the iterative enhancements in Faster R-CNN. Experiment names are abbreviated using capital letters, where R represents ResNet-50, F represents the feature pyramid network (FPN), E represents the enhanced multi-scale attention (EMA) mechanism, and A represents ROI-Align. Notably, each improvement experiment demonstrates advancements. Compared to the original Faster R-CNN algorithm, the final improved algorithm achieves notable enhancements in text overlap (0.094), image missing (0.028), component occlusion (0.126), and empty value (0.031). The final detection effectiveness reaches 0.784 (text overlap), 0.964 (image missing), 0.812 (component occlusion), and 0.930 (empty value).

As observed in Table 2, the Faster R-CNN algorithm, incorporating all the enhancements, demonstrates a 7% increase in mAP compared to the original Faster R-CNN algorithm. Moreover, there is a notable surge in the AP value for the detection of each GUI display problem.

In Experiment 2, the enhanced Faster R-CNN algorithm undergoes a comparative analysis with alternative CNN-based object detection algorithms. These algorithms are trained and evaluated using the identical dataset, and the ensuing comparison results are depicted in Table 3. Notably, the improved Faster R-CNN algorithm attains superior detection accuracy when juxtaposed with other CNN-based object detection algorithms.

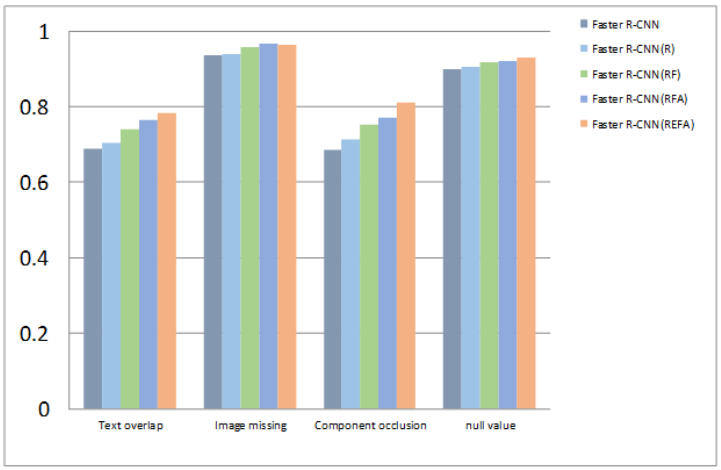

Figure 11. Comparison of different algorithms.

Table 2. Comparison of accuracy between original and improved algorithms.

Algorithm	Average Accuracy AP/%				mAP/%
	Text Overlap	Image Missing	Component Occlusion	Empty Values	
Faster R-CNN	0.69	0.936	0.686	0.899	0.803
Faster R-CNN(R)	0.703	0.94	0.714	0.907	0.816
Faster R-CNN(RF)	0.741	0.959	0.753	0.917	0.843
Faster R-CNN(RFA)	0.766	**0.966**	0.771	0.922	0.856
Faster R-CNN(REFA)	**0.784**	0.964	**0.812**	**0.93**	**0.873**

Table 3. Comparison of accuracy between YOLOv3 and Faster R-CNN.

Algorithm	mAP/%
Faster R-CNN	0.803
YOLOv3	0.792
Improved Faster R-CNN	0.873

Discussions

The experimental approach in this research leveraged the enhanced multi-scale attention (EMA) mechanism to enhance the Faster R-CNN object detection algorithm. Specifically, the EMA was incorporated into each convolutional layer of the residual networks within the feature extraction network, ResNet-50. This addition occurred after the output of the bn3 layer and before the application of the ReLU function. This strategic placement ensured that the structure of each residual block in ResNet-50 remained unchanged, enabling the use of pre-trained models.

The motivation behind employing EMA lies in its strong generalization and computational capabilities, coupled with its flexibility and lightweight nature. The experimental results presented in Table 2 clearly demonstrate a significant improvement in mean average precision (mAP) for the Faster R-CNN algorithm with the inclusion of EMA. Specifically, there is a 7% enhancement compared to the original Faster R-CNN algorithm. When compared to a variant of Faster R-CNN that only replaces ROI-Align, the version incorporating EMA exhibits a 1.7% mAP increase, showcasing a distinct advantage in accuracy.

Furthermore, the improved Faster R-CNN algorithm exhibits enhanced detection performance across four GUI error types. Notably, the detection performance for image absence and null values achieved high precision scores of 0.964 and 0.930, respectively. The detection of text overlap and component occlusion also demonstrated improvements, reaching precision scores of 0.784 and 0.812, respectively.

However, some areas warrant further improvement. The generation of a realistic dataset, crucial for training the algorithm, is based on automatically generating screenshots

with real GUI display issues from the Rico dataset. The quality of this generated dataset significantly influences the experimental results. Therefore, there is a need for optimization in aspects related to the algorithm for training data generation.

In the experiment, screenshots are employed to detect GUI display issues. Due to the minimal differences observed in screenshots from various platforms, the methodology can be transferred and extended to detect GUI display problems on other platforms. For instance, we identified similar display issues on the Windows platform, affirming the universality of these problems. Further, more in-depth experiments will be conducted in the future.

Regarding the computation complexity, the enhanced Faster R-CNN algorithm is expected to have a higher computational complexity compared to the traditional Faster R-CNN due to the use of ResNet-50, EMA, FPN, and ROI-Align. While these enhancements contribute to improved performance, they come at the cost of increased computational requirements.

6. Conclusions and Future Work

This paper introduces a framework designed to automatically identify GUI display issues in software screenshots through an enhanced Faster R-CNN object detection algorithm. In this modification, the conventional VGG-16 network is replaced with ResNet-50 as the feature extraction network, while the inclusion of a feature pyramid network (FPN) and enhanced multi-scale attention (EMA) enhances detection accuracy. Additionally, to improve the network's generalization ability, ROI-Align is employed in place of ROI-Pooling. Experimental results demonstrate that the improved Faster R-CNN algorithm achieves an average accuracy of 87.3%, marking a notable 7% enhancement over the original algorithm. This signifies the effectiveness of the improved algorithm in enhancing the accuracy of GUI display issues. Given the absence of a publicly available dataset for GUI display issues, a training data generation algorithm, based on the Rico dataset, is developed to automatically generate realistic screenshots featuring GUI display issues, mitigating the need for extensive manual collection.

In summary, this research approach offers valuable insights for software automation testing with broad applications, not only in the industrial sector but also for various terminal devices employing GUI displays. However, it is essential to note that this research is at an early stage. Future work includes expanding the focus from four to a more extensive range of GUI display issue categories, refining the current algorithm for generating training data, and optimizing it further. Additionally, while the Faster R-CNN algorithm has been improved in this study, there is a need for further research to enhance other types of object detection algorithms and identify the most optimal solution.

Author Contributions: Conceptualization, X.P. and Y.L.; methodology, X.P. and Y.L.; software, Z.H. and Y.C.; validation, Z.H.; investigation, X.P. and Y.C.; writing—original draft preparation, X.P.; writing—review and editing, Z.H. and Y.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Arnatovich, Y.L.; Ngo, M.N.; Kuan, T.H.B.; Soh, C. Achieving high code coverage in android UI testing via automated widget exercising. In Proceedings of the 2016 23rd Asia-Pacific Software Engineering Conference (APSEC), Hamilton, New Zealand, 6–9 December 2016; pp. 193–200.
2. Wetzlmaier, T.; Ramler, R. Hybrid monkey testing: Enhancing automated GUI tests with random test generation. In Proceedings of the 8th ACM SIGSOFT International Workshop on Automated Software Testing, Paderborn, Germany 4–5 September 2017; pp. 5–10.
3. Machiry, A.; Tahiliani, R.; Naik, M. Dynodroid: An input generation system for android apps. In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, Saint Petersburg, Russia, 18–26 August 2013; pp. 224–234.
4. Reiss, S.P. Seeking the user interface. In Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering, Vasteras, Sweden, 15–19 September 2014; pp. 103–114.
5. Behrang, F.; Reiss, S.P.; Orso, A. GUIfetch: Supporting app design and development through GUI search. In Proceedings of the 5th International Conference on Mobile Software Engineering and Systems, Gothenburg, Sweden, 27–28 May 2018; pp. 236–246.
6. Yang, B.; Xing, Z.; Xia, X.; Chen, C.; Ye, D.; Li, S. UIS-hunter: Detecting UI design smells in Android apps. In Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), Madrid, Spain, 25–28 May 2021; pp. 89–92.
7. Yeh, T.; Chang, T.H.; Miller, R.C. Sikuli: Using GUI screenshots for search and automation. In Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology, Victoria, BC, Canada, 4–7 October 2009; pp. 183–192.
8. Qian, J.; Shang, Z.; Yan, S.; Wang, Y.; Chen, L. Roscript: A visual script driven truly non-intrusive robotic testing system for touch screen applications. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, Seoul, Republic of Korea, 27 June 2020–19 July 2020; pp. 297–308.
9. Chen, C.; Feng, S.; Liu, Z.; Xing, Z.; Zhao, S. From lost to found: Discover missing ui design semantics through recovering missing tags. *Proc. ACM Hum.-Comput. Interact.* **2020**, *4*, 1–22. [CrossRef]
10. Chen, J.; Xie, M.; Xing, Z.; Chen, C.; Xu, X.; Zhu, L.; Li, G. Object detection for graphical user interface: Old fashioned or deep learning or a combination? In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual, 8–13 November 2020; pp. 1202–1214.
11. Moran, K.; Li, B.; Bernal-Cárdenas, C.; Jelf, D.; Poshyvanyk, D. Automated reporting of GUI design violations for mobile apps. In Proceedings of the 40th International Conference on Software Engineering, Gothenburg, Sweden, 27 May 2018–3 June 2018; pp. 165–175.
12. Moran, K.; Watson, C.; Hoskins, J.; Purnell, G.; Poshyvanyk, D. Detecting and summarizing GUI changes in evolving mobile apps. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, Montpellier, France, 3–7 September 2018; pp. 543–553.
13. Li, B.; Hu, W.; Wu, T.; Zhu, S.C. Modeling occlusion by discriminative and-or structures. In Proceedings of the IEEE International Conference on Computer Vision, Sydney, Australia, 1–8 December 2013; pp. 2560–2567.
14. Vogel, D.; Balakrishnan, R. Occlusion-aware interfaces. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Atlanta, GA, USA, 10–15 April 2010; pp. 263–272.
15. Zhu, C.; Zhu, Z.; Xie, Y.; Jiang, W.; Zhang, G. Evaluation of machine learning approaches for android energy bugs detection with revision commits. *IEEE Access* **2019**, *7*, 85241–85252. [CrossRef]
16. Li, W.; Jiang, Y.; Xu, C.; Liu, Y.; Ma, X.; Lü, J. Characterizing and detecting inefficient image displaying issues in Android apps. In Proceedings of the 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), Hangzhou, China, 24–27 February 2019; pp. 355–365.
17. Kim, J.H.; Kong, K.; Kang, S.J. Image demoireing via U-Net for detection of display defects. *IEEE Access* **2022**, *10*, 68645–68654. [CrossRef]
18. Liu, Z.; Chen, C.; Wang, J.; Huang, Y.; Hu, J.; Wang, Q. Nighthawk: Fully automated localizing UI display issues via visual understanding. *IEEE Trans. Softw. Eng.* **2022**, *49*, 403–418. [CrossRef]
19. Mirzaei, N.; Garcia, J.; Bagheri, H.; Sadeghi, A.; Malek, S. Reducing combinatorics in GUI testing of android applications. In Proceedings of the 38th International Conference on Software Engineering, Austin, TX, USA, 14–22 May 2016; pp. 559–570.
20. Baek, Y.M.; Bae, D.H. Automated model-based android gui testing using multi-level gui comparison criteria. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, Singapore, 3–7 September 2016; pp. 238–249.
21. Su, T.; Meng, G.; Chen, Y.; Wu, K.; Yang, W.; Yao, Y.; Pu, G.; Liu, Y.; Su, Z. Guided, stochastic model-based GUI testing of Android apps. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, Paderborn, Germany, 4–8 September 2017; pp. 245–256.
22. Gao, Y.; Luo, Y.; Chen, D.; Huang, H.; Dong, W.; Xia, M.; Liu, X.; Bu, J. Every pixel counts: Fine-grained UI rendering analysis for mobile applications. In Proceedings of the IEEE INFOCOM 2017-IEEE Conference on Computer Communications, Atlanta, GA, USA, 1–4 May 2017; pp. 1–9.
23. White, T.D.; Fraser, G.; Brown, G.J. Improving random GUI testing with image-based widget detection. In Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, Beijing China, 15–19 July 2019; pp. 307–317.
24. Degott, C.; Borges Jr, N.P.; Zeller, A. Learning user interface element interactions. In Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, Beijing China, 15–19 July 2019; pp. 296–306.

25. Khadka, K.; Chandrasekaran, J.; Lei, Y.; Kacker, R.N.; Kuhn, D.R. Synthetic Data Generation Using Combinatorial Testing and Variational Autoencoder. In Proceedings of the 2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), Dublin, Ireland, 16–20 April 2023; pp. 228–236.
26. Manco, G.; Ritacco, E.; Rullo, A.; Saccà, D.; Serra, E. Machine learning methods for generating high dimensional discrete datasets. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* **2022**, *12*, e1450. [CrossRef]
27. Allen, C.; Bartók, A.P. Optimal data generation for machine learned interatomic potentials. *Mach. Learn. Sci. Technol.* **2022**, *3*, 045031. [CrossRef]
28. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster R-CNN: Towards real-time object detection with region proposal networks. *Adv. Neural Inf. Process. Syst.* **2015**, *28*. [CrossRef] [PubMed]
29. Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 28 June 2014; pp. 580–587.
30. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition, Las Vegas, NV, USA, 26 June –1 July 2016; pp. 779–788.
31. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.Y.; Berg, A.C. Ssd: Single shot multibox detector. In Proceedings of the Computer Vision—ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, 11–14 October 2016; Proceedings, Part I 14; Springer: Berlin/Heidelberg, Germany, 2016; pp. 21–37.
32. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 26 June 2016–1 July 2016; pp. 770–778.
33. Lin, T.Y.; Dollár, P.; Girshick, R.; He, K.; Hariharan, B.; Belongie, S. Feature pyramid networks for object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21 July 2017–26 July 2017; pp. 2117–2125.
34. Deka, B.; Huang, Z.; Franzen, C.; Hibschman, J.; Afergan, D.; Li, Y.; Nichols, J.; Kumar, R. Rico: A mobile app dataset for building data-driven design applications. In Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology, Québec City, QC, Canada, 22–25 October 2017; pp. 845–854.
35. Ouyang, D.; He, S.; Zhang, G.; Luo, M.; Guo, H.; Zhan, J.; Huang, Z. Efficient Multi-Scale Attention Module with Cross-Spatial Learning. In Proceedings of the ICASSP 2023–2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Rhodes Island, Greece, 4–10 June 2023; pp. 1–5.
36. Hou, Q.; Zhou, D.; Feng, J. Coordinate attention for efficient mobile network design. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, TN, USA, 20–25 June 2021; pp. 13713–13722.
37. He, K.; Gkioxari, G.; Dollár, P.; Girshick, R. Mask r-cnn. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 2961–2969.