Journal of Theoretical and Applied Information Technology
30th November 2022. Vol.100. No 22

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

AN AUTOMATED TESTING TOOL BASED ON GRAPHICAL USER INTERFACE WITH EXPLORATORY BEHAVIOURAL ANALYSIS

MS.K.JAGANESHWARI [1] , DR.S.DJODILATCHOUMY[2]

[1]Research Scholar, SCSVMV University, Kanchipuram.
Assistant Professor, CTTE College, Chennai, India.
[2]Head of the Department of Computer Science,
Chellammal Women's College, Chennai, India.
Email: [1]kjaganeshwari@gmail.com, [2]djodilatchoumy@hotmail.com

ABSTRACT

Web-based applications have complex mechanisms, and it is challenging to perform any test. Computerization examination applies automation tools to decrease individual interference and repeatable assignments. In this article, we have designed and implemented an automation testing framework for web applications. The Selenium WebDriver tool was used to execute this innovative automated testing model. With this structure, testers can match the widgets with pre-trained dataset by taking screenshots of the web pages and by clicking automatically using PyAutoGUI. The screenshot property of the framework is useful to creators for evaluating their design of the web page. We introduced a novel methodology for widget detection, widget classification and testing coverage using machine learning and image processing concept. In this approach, we can give the URL as input, so, the GUI widget images can be quickly captured from the server and do not require large storage repositories to be used as training samples. Widgets of GUI images are detected from the pre-trained datasets by applying BLOB text detection. After identifying the widgets, they are classified into domain-specific categories, for example, labels, buttons, input boxes, check boxes, and links). Finally, it is evaluated to achieve a mean GUI component categorization of superior precision. Previous works have mostly used Java GUIs, Python GUIs, and Visual Basic GUIs to distinguish the types of widgets, but in our work, we observed the types of widgets using Image Processing. From this automated testing implementation on web applications, we have achieved 98.4%. of test coverage accuracy.

Keywords: *Automated Testing, GUI, DNN, Computer Vision, Image processing*

1. INTRODUCTION

Graphical User Interfaces (GUIs) are significant component in most of the current software. In addition to extensive testing of the GUI, the fundamental code also requires undergoing massive testing regime to make certain that it performs accurately [1]. Also, In the graphics user interface (GUI) mapping, many functions in the image processing (IP) field are integrated into a call back so that it can perform actions related to IP functions, including image detection and object classification [2]. A secondary image can be created based on the provided functionalities after loading any of the image file types.

For an appropriate functioning of the web-based applications requires software systems. So, the web applications quality is a main factor during deployment of these applications. To enhance the quality of software, repeated testing plays a crucial position [3]. As a result of automation testing, software quality is improved, and human intervention is minimized. The testing tasks can be accomplished with a variety of open-source and commercial tools, such as Watir, JMeter, Selenium, and QTP. The Selenium automation tool is commonly used open-source kits for testing the web applications [4]. The main idea of this paper is to develop automated testing tool with computer vision techniques. Also, this research involves on image processing techniques such as feature extraction, widget detection, widget classification, testing coverage & comparison.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

2.RELATED WORK

In [1], discovered a new way to improve GUI testing by applying machine learning techniques to recognize and classify GUI widgets. Moreover, we found that using training sample, URL linkages, and screen links, we could recognize and categorize GUI items in screenshots and report depending on their positions (x, y coordinates) and types. For improving the GUI testing by utilizing machine learning techniques to automatically detect GUI widgets in screen photos [2]. We produce randomized GUI as trained data in order to extract widget information automatically. We contemporary a neural device decipherer that utilizes the modern developments in computer image and instrument interpretation to translate UI design images into GUI skeletons [3]. We employ mature approaches from the computer vision (CV) region to recognize GUI elements, including old-school method that depend on classic image processing characteristic and deep learning techniques trained to detect massive volumes of GUI data [4]. Nevertheless, these CV techniques were not designed with the unique characteristics of GUIs and GUI components in mind [2], or with the superior localization precision required for the GUI elements recognition assignment. Simple and efficient image annotation model use Convolutional Neural Network (DNN) extracted features from a pictures and word embed vectors to represent their relevant tags [12]. The Canonical Correlation Analysis (CCA) framework, which aids in modelling both text and visual components of the dataset, is the foundation for our first set of models. UIED (User Interface Element Detection), a toolkit developed to give users a basic and easy-to-use platform for detecting GUI elements accurately [6]. To grip various and intricate GUI pictures, UIED includes multiple detection algorithms, including traditional computer vision (CV) methodologies and deep learning models [2]. UIED has been found to be capable of accurate detection and beneficial in subsequent projects. The depth of the analysis is decided about how many of the program capabilities are covered [13]. Typically, these solutions include executing an app in a sandbox. For enormous number of applications and limits human resources, auditor find it difficult to manually execute the app. Mobile app inputs are often represented via interactions with the app's graphical user interface (GUI).

As the authors have established in modern software development [7], system testing is used to validate requirements conformance at all stages of system abstractions, including the system's graphical user interface. GUI-based automated tests, like all other automations, aim to cut testing costs and time in half when compared to human procedures [5]. In industrial practise, automated testing has proven to be efficient in lowering costs for a variety of tests (such as unit or integrated testing). For covering all conceivable GUI pathways with test scripts would be extreme time consuming and result in major maintenance concerns [8]. We propose using the open-source TESTAR tool to supplements scripted testing with script less test automation. Jonathan A. Saddler produced Event Flow Slicer. A GUI tester can use this tool to develop and create all of the actual test cases needed to accomplish a certain goal. The user registers pertinent actions [11] on the UI before setting limitations to narrow the scope of the activity. To acquire only the widgets that are relevant to that aim, an events flow graph is used. The author created a trained network that can predict widget location and size from display images, which has an impact on where and how GUI testers interact with the software [10]. Our SM gives an overview of existing GUI testing methodologies and assists in identifying areas in the industry that deserve greater research attention. Connecting theoretical model-based methodologies with commercially available technologies, for example, will take a lot of effort [9], to that objective, investigations comparing the hi-tech in GUI tested in academic methodologies and modern tools are required. Deep learning is a framework that uses cutting-edge deep icon-behaviour learning to train and detect intention-behaviour mismatches in a variety of popular apps [14]. Deep learning, in particularly, use programs analysis approaches to relate UI widget' programme behaviour to their intentions and inferred UI widget labels from the programmer behaviours, enabling for the production of a large-scale, high-quality training sample. First, there is a gap between the programmers' purpose and the outputs writing query, as well as a gap between the textual question and the visual GUI designs. Secondly, other developer may use the same GUI designs, resulting in high levels of similarities to other apps and lowering the application's uniqueness. Finally, several of the retrieved GUI may have outdated design style, making it difficult for programmers to keep up with current GUI design trends [15].

3. PROPOSED APPROACH

The main objective of this research is to present a data- driven approach for automatic prototype testing software for web applications and its implementation. And to explore and analyse the adoption of computer vision approaches in the field of software testing. To detect different GUI widgets, this design employs the testing concept assessment archives in the Python OpenCV library [1]. In addition, the novelty of this proposal is that we are providing the URL an input, so it immediately captured from the server and does not demand enormous sources of the GUI widget illustrations as training experiments. As an alternative, the GUIs are detected only on its leaping boxes and color deviation. The expert ideal dataset includes input data and reaction assessments. This development unsurprisingly generates precise tasks such as recognition, categorization, and assessment analysis. Profoundly, widgets of GUI images are associated from the unsubstantiated datasets using computer visualization systems. Later, individuals classified widgets are organized into domain-specific categories (eg. labels, buttons, input box, check box, and links, etc.) [1]. Ultimately, it is calculated to attain a mean GUI component categorization for superior precision. Recent literatures have used Java GUI, Python GUI, and VB GUI for detecting the type of widgets, but we have identified the type of widgets using Image processing methods [1]. Image processing based on machine learning that simulates human vision and automates image analysis [2].

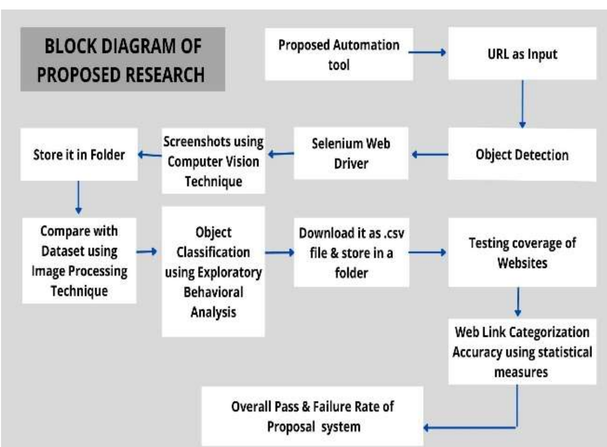


Figure 1: Block diagram of proposed system

3.1. Generation of Screenshot:

Selenium web-driver does not encourage the screenshot for pass and malfunction test cases [6]. We have applied novel work that will get the screenshot for pass and malfunction situation [7]. Utilizing this type of function tester can simply pick up the in accuracies transpired in web application. This method will aid the designer to investigate their testing. After completing the test suite, screenshots of success and failure widget links are stored in catalogue according to network smart database. Measures to create screenshot :1. Build catalogue where you store the screenshot of pass and failure widgets and links [8]. 2. Apprehend the result from selenium web driver. Validate the result is pass or fail 4. If the result is pass or fail, the screenshot for web page is obtained. 5. Place the network name as the filename for screenshot image. 6. Store up the image file in distinct index [8].

4.ENTITYSOURCE

Selenium web-driver endorses various forms of sensor to uncover the web page components [9]. To operate the selenium web driver, we have to schedule languages to imprint automation screenplays style [10]. Essentially, no matter what the language that we primarily chosen should have a selenium client library friendly [11]. All the selenium web driver highlights deliver gain access to the selenium web driver client collection for python. Broadly, we ought to recognize that the selenium impartial server for isolated and scattered testing for browser-based functions [12].

Web page features can be situated by labels, text boxes, combo boxes, buttons etc. Entity source store up all the widgets of web page sections. This will streamline the assignment of sustaining and refurbishing the practice circumstances. For example, previous version of web application contains ‘Login’ button. Subsequent type of web application ‘Login’ button changed to ‘Login Now’, so it is required to train the widgets [13]. To prevent any challenges, we have implemented object repository which contains the trained samples [14]. Maintenance cost will be reduced as well as the storage space. At any time, transformation arises in web application elements, tester no needs to change the object repository [14]. Unlike the existing architecture, it does not require huge repositories as training samples [1]. Here we have used the Frozen East Text detector dataset. OpenCV’s EAST text detector dataset is DNN prototype, built on a new structural design and guidance configuration. It is adept of operating at close instantaneous at 13 FPS on 720p images and it acquires hi-tech text recognition precision. And it uses OpenCV’s EAST sensor that’s spontaneously distinguish text in both images and video streams [15].

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

5. THE PROPOSED METHODOLOGIES-WIDGET DETECTION AND WIDGET CLASSIFICATION

This work tries to exploit the unique approach of investigative developmental detection. We have used the PyAutoGUI automation python library for controlling the mouse to click the numerous networks in the certain n URL [1]. By way of scrutinizing the subsequent screenshot using the OpenCV object recognition method for new discrepancies and GUI classification can be done by beyond engaging a DNN classifier. If the subsequent screenshot is solely new than the original GUI image, then the GUI can be recognized as a Hyperlink/ Button [1]. To implement this, we have defined the function called image decision function. The algorithm is given below:

```
Algorithm for Image Decision Function

Step 1: import  cv2, numpy, imutils
Step 2: Define Image_decision function with parameters name1, name2 and verbose
Step 3: Read source image and target image using cv2.imread
Step 4: Convert Color Image to Gray Image using cv2.cvtColor
Step 5: Resize the Second gray Image using cv2.resize
Step 6: Perform Gaussian Blur to the Source Gray Image and Target Gray Image using
        cv2.GaussianBlur
Step 7: Apply absolute difference for the first gray and second gray images using
cv2.absdiff
Step 8: Obtain threshold value to a variable thresh using cv2.threshold
Step 9: If len(rows)>1
        Step 9.1: obtain minimum value of rows and columns
        Step 9.2: Obtain maximum value of rows and column
        Step 9.3: Calculate the height = maxr-minr and width=maxc=minc
        Step 9.4: Display diff and threshold values
        Step 9.5: Assign orig_res=width*height and calc_org=width*height
        Step 9.6: res=orig_res-calc_res
        Step 9.7: diff_percent = ((calc/res/org_res)*100)
        Step 9.8: If(diff_percent)>80) then img_decision='Button/Anchor Link'
        Step 9.9: elseif(diff_percent)>70) then img_decision='Popup Target'
        Step 9.10: elseif(diff_percent)>5 then img_decision='Combo/List Box'
        Step 9.11: elseif(diff_percent)>2 then img_decision='Input Box'
        Step 9.12: else: img_decision="Label"
        Step 9.13: return the value of img_decision and diff_percent
        Step 9.14: if verbose!=0: then
                   Print Original Res, Calc Res, diff, diff percent and Decision
Step 10: Else return Image decision as Label
```

Figure 2: Image decision function algorithm

5.1. Widget Detection

This work used Open-Source Computer Vision Library (OpenCV). It is an icon managing tool with an open-source computer vision concept and object knowledge software. Astonishingly, it is the swiftest prototyping of computer image challenges. OpenCV python zips to the trepidation of videos, install images, and locate objects with both a normal webcam and a focused depth sensor. It carries refined boundaries for capturing, processing, and conferring picture data. OpenCV's Python supplements can assist us reconnoitrer vindications to these requirements in a high-level language and in a homogeneous data format that is interoperable with scientific libraries such as NumPy, beautifulsoup, imutils [16]. Although construing info from replicas vintages extremely precise leaping containers for GUI components, this data is not foreveraccessible due to restrictions in the photo-editing software used or alterations in design practises, such as delineating mock-ups digitally or physically using pen demonstrations, tablets, or paper. A mock-up relic could be nothing more than an image in some cases, necessitating the use of CV techniques to distinguish key GUI-component information. To accommodate these circumstances, our solution employs CV systems to recognise GUI component bounding boxes. To inferred joining frames across items comparable to GUI mechanisms in a picture, this procedure retains a variety of CV techniques. The edges of objects in an image are first detected using BLOB text detection algorithm. The edges are then dilated to blend edges that are close together. Finally, leaping frames over atomic GUI-modules are calculated using the contours of those edges. The algorithm is given below.

```
/*Algorithm for BLOB Text Detection*/

Step 1: import cv2, numpy, imutils, time,argparse,selenium webdriver, NMS
Step 2: Define handler function and Assign the Options for Chrome options using add
        arguments
Step 3: Assign the webdriver.chrome to the driver and place the Text for URL input.
Step 4: To find the element with longest height on page and save the driver screenshot using
        save_screenshot and quit driver
Step 5: Read the screenshot using cv2.imread, assign to img and show the image using cv2.
        imshow
Step 6: Assign the value of ArgumentParser to ap
Step 7: Determine the ArgumentParser ap with add_argument with type='str', 'i', 'image' and
        help ="path to input image"
Step 8: Determine the ArgumentParser ap with add_argument with type='str', '-east', '--east'
        and help ="path to input EAST text detector"
Step 9: Determine the ArgumentParser ap with add_argument with type='float', '-c',
        '--min-confidence', 'ast' and help ="minimum probability required to inspect a region"
Step 10:Determine the ArgumentParser ap with add_argument with type='int', '-w',  '--width',
        and help ="resized image width(should be multiple of 32)"
Step 11: Determine the ArgumentParser ap with add_argument with type='int', '-e',  '--height',
        and help ="resized image height(should be multiple of 32)"
Step 12: Assign ap.parse_args to args and load the input image and grab the image dimensions
        using cv2.imread
Step 13: Make a copy of image, assign to orig and get the shape of the image and assign to (H,W)
Step 14: Set the new width and height and then determine the ratio in change for both the width and
height using rW=W/float(newW) and rH=H/float(newH)
Step 15: Resize the image and assign it as image and get the layer options
Step 16: Assign the (deep neural network) cv2.dnn.readNet, "frozen_east_text_detection.pb" to net
Step 17: Construct a blob from the image and then perform a forward pass of blob using
cv2.dnn.blobFromImage
Step 18: setInput(blob) using net.forward(layerNames)
Step 19: Print the Object detection time taken in seconds using the standard format specified
Step 20: Using for with range from 0 to numRows
          Compute ScoresData, xData0, xData1, xData2, xData3 and anglesData
        Step 20.1: Using for with range from 0 to numCols
                Step 20.1.1: If our score does not have sufficient probability, ignore it
                        Step 20.1.1.1: Compute the offset factor as our resulting feature maps will be
                        4x smaller than the input image
                        Step 20.1.1.2: Extract the rotation angle for the prediction and then compute
                        the sine and cosine
                        Step 20.1.1.3: Compute the h=xData0[x]+xData2[x] and
                        w=xData1[x]+xData3[x]
                        Step 20.1.1.4: Compute the value of endX, endY, startX, startY and append it
                        with rects and confidences
Step 21: Compute the value of boxes using NMS and loop over the bounding boxes using for loop
        Step 21.1: Scale the bounding box coordinates based on the respective ratios
        Step 21.2 : Draw the bounding box on the image using cv2.rectangle
Step 22: Show the Output image using cv2. imshow and assign the gui with geometry and configure
          functions.
```

Figure3: BLOB Text detection algorithm

i) Before WidgetDetection- Web page of yahoo.com captured as screenshot before detecting the widgets using image processing techniques python openCV.

Journal of Theoretical and Applied Information Technology
30th November 2022. Vol.100. No 22

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

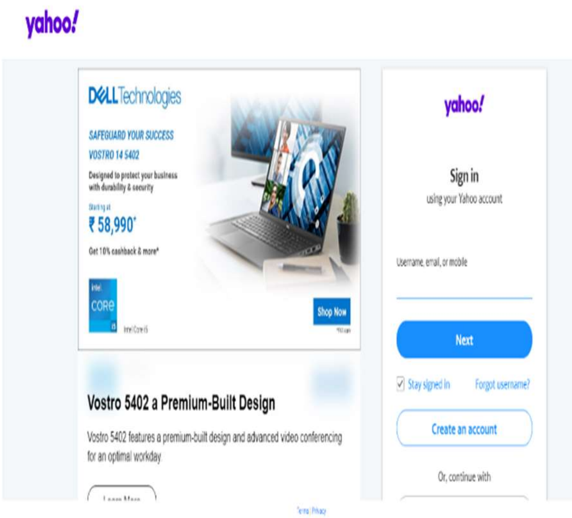


Figure4: screenshot of yahoo.com before widget detection

ii)After Widget Detection- Web page of yahoo.com captured as screenshot after detecting the widgets using image processing techniques using python openCV. And the widgets are highlighted in green rectangular box using bounding boxes techniques.

Figure5: Widget detection of Yahoo web page.

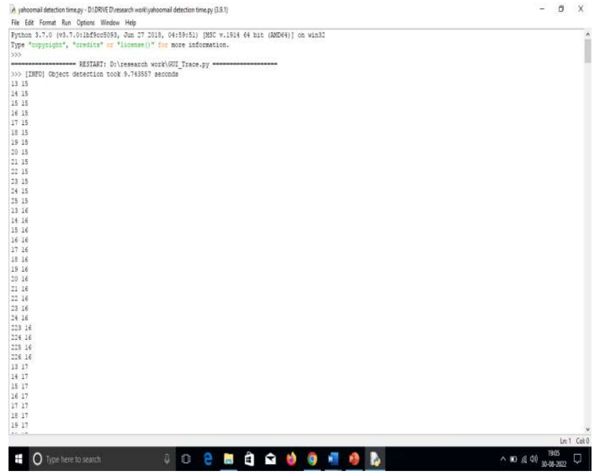


Figure6: Widget detection time of yahoo.com in the Python IDE

The widget detection time of various real time websites are tabulated and given below:

URL of Web Pages	Object detection time
https://www.gmail.com/	9.958827 seconds
https://www.google.com/	9.760212 seconds
https://services.gst.gov.in/services/login	10.047040 seconds
https://www.instagram.com/accounts/login/	9.672259 seconds
https://www.linkedin.com/login	9.711001 seconds
https://twitter.com/login?lang=en-gb	9.660448 seconds
https://profile.w3schools.com/log-in	9.824095 seconds
https://login.skype.com/	9.810490 seconds
https://www.snapdeal.com/login	10.351611 seconds
https://www.website.com/sign-in/	10.276383 seconds
https://login.yahoo.com/	9.743557 seconds
https://www.amazon.in/ap/signin	9.633661 seconds

Table 1: Object detection time of web pages

The widget detection time of various real time websites are represented in graphical format and given below:

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

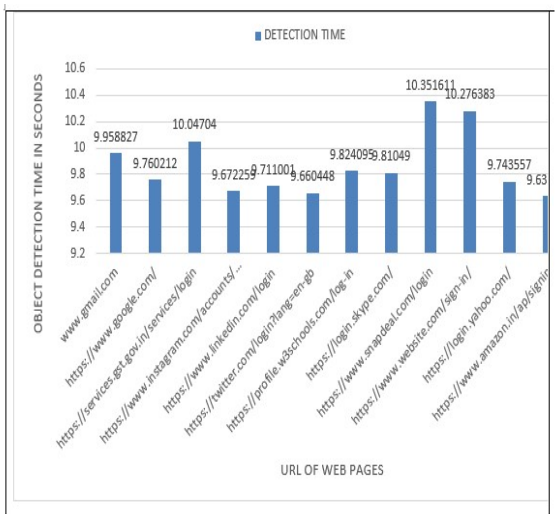


Figure7: Chart Representation of Widget Detection

5.2. Widget Classification

It recommends a unique approach of investigative developmental recognition. We have applied the PyAutoGUI automation python library for controlling the mouse to click the several connections in the particular URL [1]. Following the pick-up of the resultant screenshot employing the OpenCV object recognition procedure for new modifications and GUI classification, which can be done by additional commissioning a DNN classifier [18]. If the resultant screenshot is totally different than the prototype GUI imaged, then the GUI can be classified as a Hyperlink/ Button. If the Cursor image is activated, it can be discovered as an Input box or if within the same page a list of items is shown, it can be recognized as a Combo Box [19]. This assessment approach demands to be trialled with different real-time websites for additional authenticity and precision. Accordingly, without controlled training samples, the suggested approach can generate precision in GUI component recognition and categorization. Blob text detection algorithm use computer vision to detect regions of a digital image that differ from surrounding regions in terms of properties, such as brightness or color [20]. This algorithm is needed to classify the extracted text blobs into horizontal text regions. This process is also referred to as text segmentation [21]. This classification is based on the size and position of the blobs. Image classification algorithm with exploratory behavioural analysis algorithm is used to classify the widgets. The algorithm is given below:

Algorithm for Widget Classification with EBA

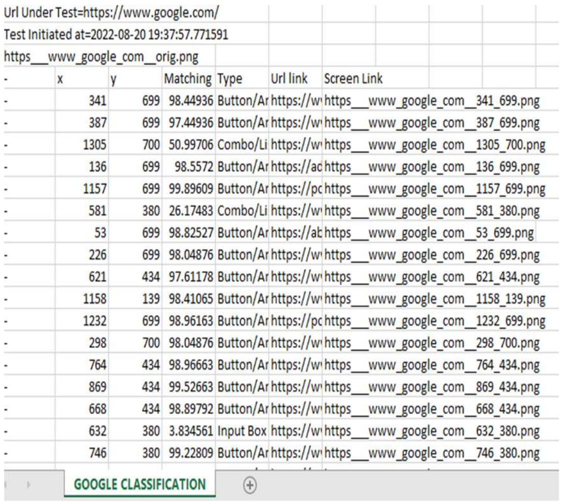
Step 1: import cv2, time, selenium webdriver, chrome, requests, pyautogui, numpy, argparse, beautifulsoup, imutils, image decision function.
Step 2: Assign minarea to 500, and Input the actual_url and name_url
Step 3: Assign the response with get request from actual_url, Assign the value of soup with BeautifulSoup library and Assign the value of link, page, c and unique
Step 4: Using for loop with range for i to link, assign the value of i with href
    Step 4.1.: If x not in unique then append to x Print Total Reference links, Obtain the driver with webdriver using chromedriver.exe
Step 5: Identify the longest height on page and set the driver window size to save_screenshot and Assign the value of ArgumentParser to ap
Step 6: Determine the ArgumentParser ap with add_argument with type='str', 'i', 'image' and help ="path to input image"
Step 7: Determine the ArgumentParser ap with add_argument with type='str', '-east', '--east' and help ="path to input EAST text detector"
Step 8: Determine the ArgumentParser ap with add_argument with type='float', '-c', '--min-confidence', 'ast' and help ="minimum probability required to inspect a region"
Step 9: Determine the ArgumentParser ap with add_argument with type='int', '-w', '--width', and help ="resized image width(should be multiple of 32)"
Step 10: Determine the ArgumentParser ap with add_argument with type='int', '-e', '--height', and help ="resized image height(should be multiple of 32)"
Step 11: Assign ap.parse_args to args, Load the input image and grab the image dimensions using cv2. Imread, Make a copy of image and assign to orig and Get the shape of the image and assign to (H,W)
Step 12: Set the new width and height and then determine the ratio in change for both the width and height
Step 13: Resize the image and grab the new image dimensions
Step 14: Define the two output layer names for the EAST detector model that we are interested-the first is the output probabilities and the second can be used to derive the bounding box coordinates of text layer names
Step 15: Load the pre-trained EAST text detector using cv2.dnn.readNet to net, Construct a blob from the image and then perform a forward pass of the model to obtain the two output layer sets
Step 16: Show timing information on text prediction
Step 17: Grab the number of rows and columns from the scores volume, then initialize our set of bounding box rectangles and corresponding confidence scores
Step 18: Loop over the number of rows, Extract the scores (probabilities) followed by the geometrical data used to derive potential bounding box coordinates that surround text
Step 19: Loop over the number of columns
    Step 19.1: If our score does not have sufficient probability, ignore it, Compute the offset factor as our resulting feature maps will be 4x smaller than the input image
    Step 19.2: Extract the rotation angle for the prediction and then compute the sine and cosine
    Step 19.3: Use the geometry volume to derive the width and height of the bounding box
    Step 19.4: Compute both the starting and ending (x,y)- coordinates for the text prediction bounding box, Add the bounding box coordinates and probability score to our respective lists
Step 20: Apply non-maxima suppression to suppress weak, overlapping bounding boxes using NMS
Step 21: Set test initiated date and time, input url and write it as .csv file with x,y coordinates of the widget position, matching percentage for the widget with the dataset, url link and screen link as .png
Step 22: Loop over the bounding boxes
    Step 22.1: Scale the bounding box coordinates based on the respective ratios, Click the bounding box on the image using pyautogui move, press and click, Print the length of windows
        Step 22.2.1: If label not in decision, then assign as combo box, else write fobj
        Step 22.2.2: If actual_url is not equal to current_url then make time to sleep, Execute driver script
        Step 22.2.3: else then make time to sleep, write fobj and execute script
Step 23: Close fobj

Figure8: Image classification with exploratory behavioural analysis algorithm

The report with x,y coordinates of widgets positions, matched percentage of widgets with trained samples, type of the widgets, URL Link, and the screen link (.png)of Google.com is downloaded as .csv file, which is given below:

Journal of Theoretical and Applied Information Technology
30th November 2022. Vol.100. No 22

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195



Url Under Test=https://www.google.com/
Test Initiated at=2022-08-20 19:37:57.771591
https__www_google_com__orig.png

-	x	y	Matching	Type	Url link	Screen Link
-	341	699	98.44936	Button/Ar	https://w	https__www_google_com__341_699.png
-	387	699	97.44936	Button/Ar	https://w	https__www_google_com__387_699.png
-	1305	700	50.99706	Combo/Li	https://w	https__www_google_com__1305_700.png
-	136	699	98.5572	Button/Ar	https://ac	https__www_google_com__136_699.png
-	1157	699	99.89609	Button/Ar	https://pc	https__www_google_com__1157_699.png
-	581	380	26.17483	Combo/Li	https://w	https__www_google_com__581_380.png
-	53	699	98.82527	Button/Ar	https://ab	https__www_google_com__53_699.png
-	226	699	98.04876	Button/Ar	https://w	https__www_google_com__226_699.png
-	621	434	97.61178	Button/Ar	https://w	https__www_google_com__621_434.png
-	1158	139	98.41065	Button/Ar	https://w	https__www_google_com__1158_139.png
-	1232	699	98.96163	Button/Ar	https://pc	https__www_google_com__1232_699.png
-	298	700	98.04876	Button/Ar	https://w	https__www_google_com__298_700.png
-	764	434	98.96663	Button/Ar	https://w	https__www_google_com__764_434.png
-	869	434	99.52663	Button/Ar	https://w	https__www_google_com__869_434.png
-	668	434	98.89792	Button/Ar	https://w	https__www_google_com__668_434.png
-	632	380	3.834561	Input Box	https://w	https__www_google_com__632_380.png
-	746	380	99.22809	Button/Ar	https://w	https__www_google_com__746_380.png

GOOGLE CLASSIFICATION

Figure9: Widget classification report of Google.com

The widget classification of Google.com website is represented as graphical format with the position of the x, y co-ordinates of widgets and given it is displayed below:

load googlet.txt

plot(googlet,'DisplayName','googlet');

xlabel("x values"); ylabel("y values");

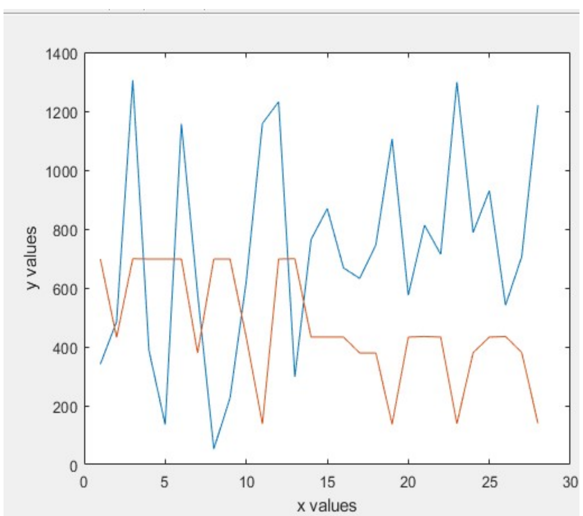


Figure 10: x,y co-ordinates of widget position

The widget classification of Google.com website is represented as graphical format with percentage of matched widgets with trained data sample and given it is displayed below:

xlabel("widgets"); ylabel("%matchedwidgets");

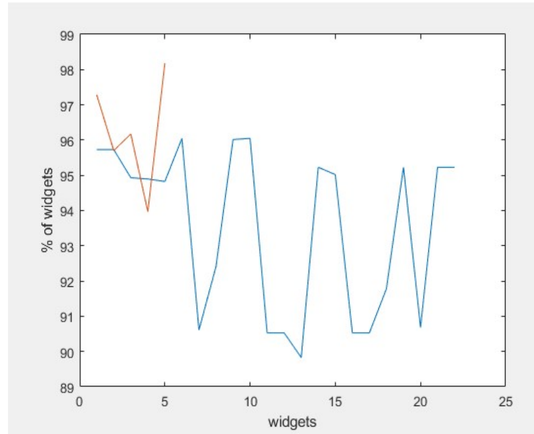


Figure11: Percentage of matched widgets with the trained samples

6. TEST COMPARISON

GUI widget detection, classification accuracy, GUI categorized link accuracy, testing coverage probability are analysed with many real-time scenarios and which is tabulated and displayed below:

Websites	Existing Categorized Links	Test coverage in %	Proposed Categorized links	Test coverage in %
Google.com	30	98	31	99
Twitter.com	24	97	25	98
Gmail.com	20	98	21	99
Yahoo.com	24	96	26	98
Skype.com	18	95	21	99
Snapdeal.com	32	94	34	98
Linkedin.com	18	95	20	97
Instagram.com	26	97	28	99
Amazon.in	23	96	25	98
W3school.com	35	97	36	99

Table 2: Test coverage of Existing and Proposed system

7. RESULT AND DISCUSSION

In the existing work of literature, a deep learning method of prototyping a GUI with supervised training is done [1]. A labelled image repository is used as training samples and DNN architecture is used to classify the GUI components [1]. Finally, it is used to derive a GUI code based on a mock input image. This idea spikes many curious possibilities in software GUI testing [1]. Due to this problem, novel methodologies are developed to find out the accuracy in automated testing. The Proposed method

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

is compared with existing system for test coverage accuracy. And the proposed method captured links are compared and categorised with existing system categorized links. The accuracy of the proposed system is compared with existing system using Root mean square error (RMSE), Mean square root (MSE).

The RMSE can plot a difference between the predicted value and observed value of a parameter of the model and we can find the efficiency of the testing model. So, the RMSE lower value shows the better results. And our results found that the proposed testing model has lower RMSE value compared to existing testing model.

Root Mean square (RMSE) = √[Σ(Pi – Oi)2 / n]

Root Mean Square Erro (RMSE)					
Existing			Proposed		
Predicted Value	Observed Value	RMSE	Predicted Value	Observed Value	RMSE
100	98	0.63246	100	99	0.3162278
100	97	1	100	98	0.6666667
100	98	0.70711	100	99	0.3535534
100	96	1.51186	100	98	0.7559289
100	95	2.04124	100	99	0.4082483
100	94	2.68328	100	98	0.8944272
100	95	2.5	100	97	1.5
100	97	1.73205	100	99	0.5773503
100	96	2.82843	100	98	1.4142136
100	97	3	100	99	1

Table 3: RMSE of Existing and Proposed system

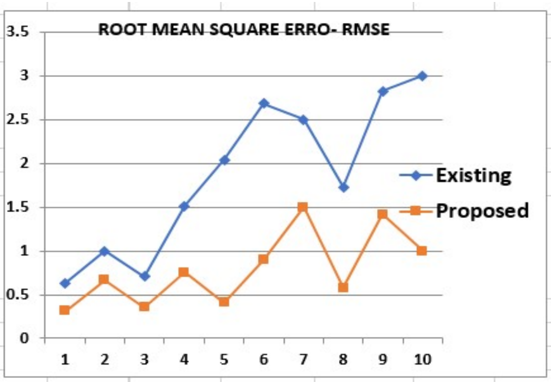


Figure 12: Graph representation of RMSE

The Mean squared error (MSE) measures the amount of error in statistical model. It finds the average squared difference between the observed and predicted values. When a testing model has no error, then the MSE is equal to zero. Our proposed testing model values are close to zero.

Mean Squared Error (MSE): = √[Σ(Pi – Oi)2 / n]

Mean Squared Error(MSE)					
Existing			Proposed		
Predicted Value	Observed Value	MSE	Predicted Value	Observed Value	MSE
100	98	0.4	100	99	0.1
100	97	1	100	98	0.44444444
100	98	0.5	100	99	0.125
100	96	2.285714	100	98	0.57142857
100	95	4.166667	100	99	0.16666667
100	94	7.2	100	98	0.8
100	95	6.25	100	97	2.25
100	97	3	100	99	0.33333333
100	96	8	100	98	2
100	97	9	100	99	1

Table 4: MSE of Existing and Proposed system

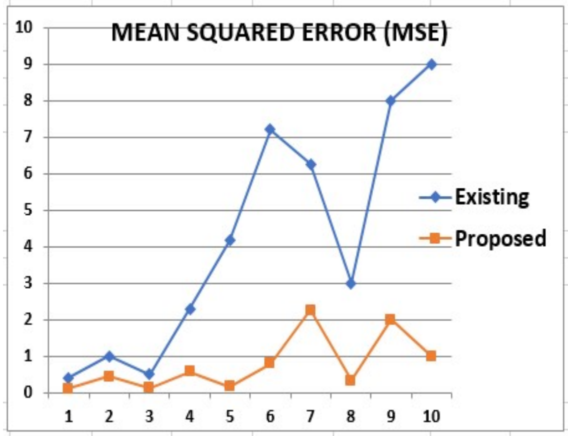


Figure 13: Graph representation of MSE

Finally testing coverage and categorization accuracy of existing and proposed system is tabulated and presented below:

Existing system Categorization Accuracy in %	Proposed system Categorization Accuracy in %
96.3	98.4

Table 5: Categorization Accuracy of Existing and Proposed system

The below table shows the parameters like overall pass rate, overall failure rate, execution time and maintenance cost of proposed system and existing system [23]:

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Parameters	Overall pass rate	Overall failure rate	Testing time	Maintenance cost
Proposed system	98.4	1.6	Low	Low
Existing System	96.3	3.7	High	High

Table 6: Overall results of proposed system and existing system

8. CONCLUSION

In this article, we have suggested innovative computerization examination structure to assess the web-based testing function centred on selenium web-driver. With the intention of assessing the web application anticipated automation structure certainly diminishes the moment necessary to compose the test cases and expand the pass percentage of test coverage. It also diminishes frenetic capability of a tester. By applying this structure, one can create the tailored test statements and also investigate the malfunctions using screenshots of failed test cases. Tester can maintain all data from central place. This framework is very effective for enthusiastically altering web applications. Like this, automation framework helps businesses to test web applications economically. Also, this research is concentrated on automation testing of real-time web applications of the User Interface and User Experience. This instrumental functions economically in a reduced amount of time with fewer maintenance cost. This framework produces the tailored test report for testers. It is convenient, streamlined, and superior compared to other works available on the current literature. Thus, without supervised training samples the proposed method can yield accuracy in GUI component detection and Classification [1]. The main contribution is to reduce the investment of human resources, training and testing time. Also, this work focused on automation testing of real time websites of User Interface and User Experience. The limitations of the work are, we have used single dataset and we didn't find the 100% accuracy in this work. So, the future enhancement of this work is a testing model with more datasets, accuracy and involves efficient statistical measures with large number of web links will be achieved.

REFERENCES

[1]Jaganeshwari,K. and S.Djodilatchoumy. "A Novel approach of GUI Mapping with image based widget detection and classification. " In 2021 2nd International ConferenceonIntelligent Engineering and Management (ICIEM), pp. 342-346. IEEE, 2021.

[2] White, Thomas D., Gordon Fraser, and Guy J. Brown. "Improving random GUI testing withimage-based widget detection." In Proceedings of the 28th ACM SIGSOFT InternationalSymposium on Software Testing and Analysis, pp. 307-317. 2019.

[3] Chen, Chunyang, Ting Su, Guozhu Meng, Zhenchang Xing, and Yang Liu. "Fromui designimage to gui skeleton: a neural machine translator to bootstrap mobile gui implementation."In Proceedings of the 40th International Conference on Software Engineering, pp. 665-676.2018.

[4] Chen, Jieshan, MulongXie, Zhenchang Xing, Chunyang Chen, Xiwei Xu, Liming Zhu, andGuoqiang Li. "Object detection for graphical user interface: old fashioned or deep learning ora combination?." In Proceedings of the 28th ACM Joint Meeting on European SoftwareEngineering Conference and Symposium on the Foundations of Software Engineering, pp.1202-1214. 2020.

[5]Moniruzzaman, Md, Syed Mohammed Shamsul Islam, Paul Lavery, and MohammedBennamoun. "Faster r-cnn based deep learning for seagrass detection from underwater digitalimages." In 2019 Digital Image Computing: Techniques and Applications (DICTA), pp. 1-7.IEEE, 2019.

[6]Xie, Mulong, Sidong Feng, Zhenchang Xing, Jieshan Chen, and Chunyang Chen. "UIED: ahybrid tool for GUI element detection." In Proceedings of the 28th ACM Joint Meeting Software Engineering Conference and Symposium on the Foundations of SoftwareEngineering, pp. 1655-1659. 2020.

[7] Nass, Michel, Emil Alégroth, and Robert Feldt. "Why many challenges with GUI testautomation (will) remain." Information and Software Technology 138 (2021): 106625.

[8] Vos, Tanja EJ, PekkaAho, Fernando Pastor Ricos, Olivia Rodriguez-Valdes, and Ad Mulders."testar–scriptless testing through graphical user interface." Software

Testing, Verification andReliability 31, no. 3 (2021): e1771.

[9]Kanglinli&MemggiWu, “Effective GUI Test Automation developing an automated GUItesting tool” ISBN 0-7821-4321-2.

[10] Jonathan A. Saddler, Myra B. Cohen, “EventFlowSlicer: A Tool for Generating RealisticGoal-Driven GUI Tests”, 2017 IEEE.

[11] Murthy, Venkatesh N., Subhransu Maji, and R. Manmatha. "Automatic image annotationusing deep learning representations." In Proceedings of the 5th ACM on InternationalConference on Multimedia Retrieval, pp. 603-606. 2015.

[12] Li, Yuanchun, Ziyue Yang, Yao Guo, and Xiangqun Chen. "Humanoid: A deep learning-based approach to automated black-box android app testing." In 2019 34th IEEE/ACMInternational Conference on Automated Software Engineering (ASE), pp. 1070-1073. IEEE,2019.

[13] Xi, Shengqu, et al. "DeepIntent: Deep icon-behavior learning for detecting intention-behaviordiscrepancy in mobile apps." Proceedings of the 2019 ACM SIGSAC Conference onComputer and Communications Security. 2019.

[14] Zhao, Tianming, Chunyang Chen, Yuanning Liu, and Xiaodong Zhu. "GUIGAN: Learning toGenerate GUI Designs Using Generative Adversarial Networks." In 2021 IEEE/ACM 43rdInternational Conference on Software Engineering (ICSE), pp. 748-760. IEEE, 2021.

[15] Nguyen, Tam, Phong Vu, Hung Pham, and Tung Nguyen. "Deep learning UI design patternsof mobile apps." In 2018 IEEE/ACM 40th International Conference on Software Engineering:New Ideas and Emerging Technologies Results (ICSE-NIER), pp. 65-68. IEEE, 2018.

[16] Chen, Jieshan, Chunyang Chen, Zhenchang Xing, Xiwei Xu, Liming Zhut, Guoqiang Li, andJinshui Wang. "Unblind your apps: Predicting natural-language labels for mobile guicomponents by deep learning." In 2020 IEEE/ACM 42nd International Conference onSoftware Engineering (ICSE), pp. 322-334. IEEE, 2020.

[17] Zhao, Dehai, Zhenchang Xing, Chunyang Chen, Xiwei Xu, Liming Zhu, Guoqiang Li, andJinshui Wang. "Seenomaly: vision-based linting of GUI animation effects against design-don't guidelines." In 2020 IEEE/ACM 42nd International Conference on (ICSE), pp. 1286-1297. IEEE, 2020.

[18] De Souza Baulé, Daniel, Christiane Gresse von Wangenheim, Aldo von Wangenheim, andJean CR Hauck. "Recent Progress in Automated Code Generation from GUI Images UsingMachine Learning Techniques." J. Univers. Comput. Sci. 26, no. 9 (2020): 1095-1127.

[19]Xiao, Xusheng, Xiaoyin Wang, Zhihao Cao, Hanlin Wang, and Peng Gao. "Iconintent:automatic identification of sensitive UI widgets based on icon classification for android apps."In 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), pp. 257-268. IEEE, 2019.

[20]Satish Gojarea,Rahul Joshib,DhanashreeGaigawarec Analysis and Design of Selenium WebDriver Automation Testing Framework,ScienceDirect, Procedia Computer Science (2015).

[21]Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps, Kevin Moran, Member, IEEE, Carlos Bernal-C´ardenas, Student Member, IEEE,IEEE Transactions on Software Engineering, VOL. #, NO. #, 2018.

[22] Gojare, Satish, Rahul Joshi, and Dhanashree Gaigaware. "Analysis and Design of Selenium WebDriver Automation Testing Framework" Procedia Computer Science, 2015.