Intelligent System For Visual Testing Of Software Products

Myroslav Komar[1], Viktor Fedorovych[1], Vladyslav Poidych[1] and Andrii Taborovskyi[1]

[1] *West Ukrainian National University, Lvivska str., 11, Ternopil, 46009, Ukraine*

Abstract

This article presents an innovative approach to software product testing through the development of an intelligent system for visual testing, leveraging artificial intelligence to automate and enhance the process. The paper outlines the creation of an integrated system designed to monitor and analyze visual changes in software graphical interfaces. By addressing the limitations of traditional visual snapshot testing, which struggles with the variability of web browsers' versions and types, this system uses a blend of Python and PHP programming languages, Selenium WebDriver testing framework, and ImageMagick library to offer a more robust solution for web interfaces testing.

The study emphasizes the growing importance of visual testing in software development, particularly for web and mobile applications, where user interface correctness and usability are critical for market success. It introduces an AI-based method for visual comparison, aiming to streamline the testing process, reduce error detection times, and improve overall product quality. The proposed solution is designed to overcome the challenges of traditional methods by enabling more accurate and efficient testing across different browsers and devices without the need for extensive manual coding or model training.

This contribution to the field of software testing and AI application in software development not only provides a practical solution to a common industry challenge but also opens avenues for further research and development in automated visual testing systems.

Keywords

Machine learning, automated testing, visual testing, artificial intelligence, visual element, pixel comparator, graphical interface

1. Introduction

Visual testing of software products is an important part of the software development process, as it allows you to identify errors related to the user interface and visual representation of data. This topic is especially important in the context of the growing popularity of web and mobile applications, where the importance of the correctness and usability of the interface is essential for the success of the product on the market.

Visual testing can help identify problems such as malfunctions in the display of graphic elements, incompatibility with different screen resolutions, incorrect display on different devices, etc. Given the rapid pace of technology development, it is important to constantly improve visual testing methods and tools to ensure software quality and user satisfaction.

Nowadays, visual testing is becoming very popular in the field of automated testing. Visual testing is a method used to check the graphical interfaces of programs or websites for their correctness and proper display [1].

Different programming languages have their own frameworks and libraries that allow you to make visual checks in the form of page snapshots, snapshots of dialog boxes and other components for automatic tests. This allows you to check how elements are displayed on the page, what color they are, and other data.

The First International Workshop of Young Scientists on Artificial Intelligence for Sustainable Development; May 10-11, 2024, Ternopil, Ukraine

mko@wunu.edu.ua (M. Komar); zizix781227@gmail.com (V. Fedorovych); poydich.vladislav@gmail.com (V. Poidych); taboroffsky6@gmail.com (A. Taborovskyi)

0000-0001-6541-0359 (M. Komar); 0009-0009-7806-7751 (V. Fedorovych); 0009-0007-5290-051X (V. Poidych); 0009-0001-9657-5364 (A. Taborovskyi)

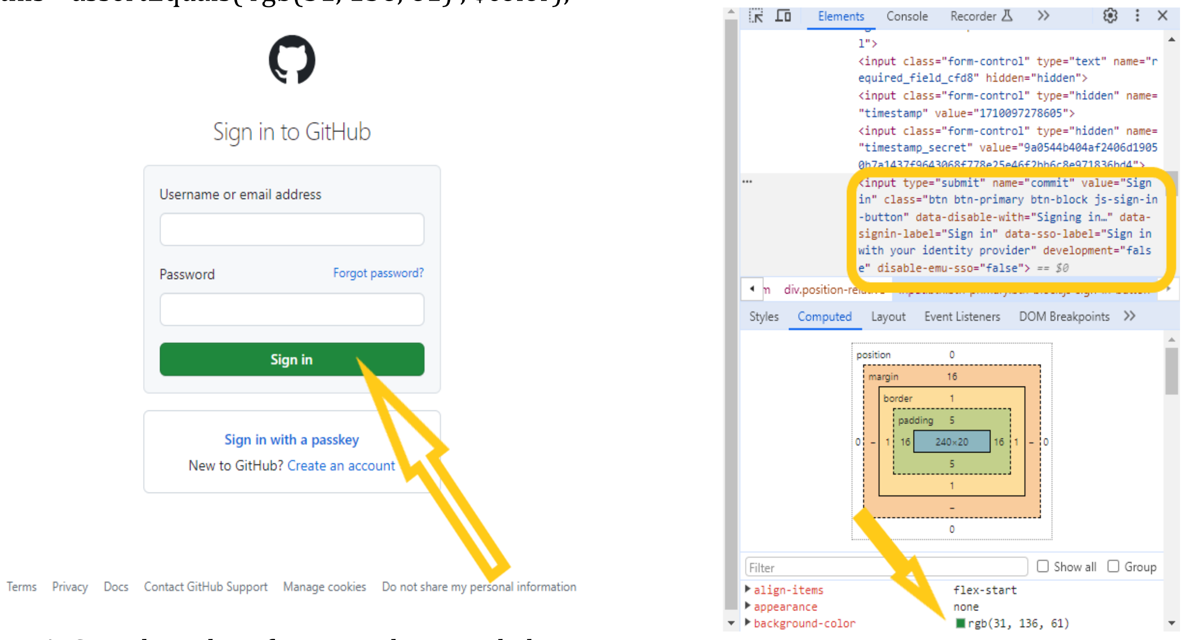
One of the standard methods of automated visual testing is to check the CSS styles that are responsible for the appearance of the corresponding element on the HTML page.

To check an element on a page, you can perform the following checks (Fig. 1):

1. The 'js-sign-in-button' element must contain text-align - center
2. The 'js-sign-in-button' element should contain the following color #1f883d

Next, let's consider an example of such checks using the PHP programming language and the Selenium Webdriver framework.

```
$element = $driver->findElement(WebDriverBy::className('js-sign-in-button'));
$color = $element->getCssValue('color');
$this->assertEquals('rgb(31, 136, 61)', $color);
```



```
1">
<input class="form-control" type="text" name="required_field_cfd8" hidden="hidden">
<input class="form-control" type="hidden" name="timestamp" value="1710097278605">
<input class="form-control" type="hidden" name="timestamp_secret" value="9a0544b404af2406d1905
0b7a1437f9643068f778e25e46f2bb6c8e971836bd4">
<input type="submit" name="commit" value="Sign in" class="btn btn-primary btn-block js-sign-in
-button" data-disable-with="Signing in…" data-signin-label="Sign in" data-sso-label="Sign in
with your identity provider" development="false" disable-emu-sso="false"> == $0
```

```
align-items        flex-start
appearance         none
background-color   rgb(31, 136, 61)
```

Figure 1: Sample styles of a particular visual element

When using conventional element validation using Selenium Webdriver or other frameworks, we need to know the following data about the element: background color, visibility, coordinates and size of the element, text. This means that for each such check we will have to write at least two lines of code, which is very costly and cumbersome.

It will be very difficult to find all visual errors, even if you use all the code above. For example, it is impossible to access a visual element if it is hidden behind another.

This method has a number of disadvantages: a large amount of additional code in the tests; difficulty in maintaining this code because style names can change; this method cannot detect defects when an element is overlapped by another element or may not be visible on the page.

Automated visual testing usually uses snapshot testing technology. During such testing, snapshots of the screen image are taken at different points during the test, and their pixels are compared to the base image [2-4]. Image testing algorithms are very simple: we compare the color codes of two pixels, and if they differ, a visual error report is generated. The difference from manual testing is that visual testing software allows you to quickly detect pixel differences, so that the computer can identify them easily and accurately (Fig. 2).

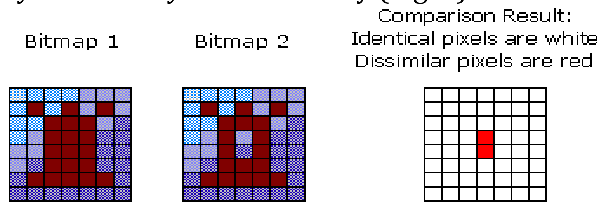


Figure 2: Pixel image comparison process

There's also the problem that these tests are often slow because they require a full browser compared to lighter unit tests.

Comparing images of the expected result and the actual result has a number of drawbacks:

- significant slowdown in the execution time of automated tests;
- inability to use the same reference images for different types and versions of browsers;
- a high probability of accidental triggering when the data on the page has not yet loaded.

The relevance of using artificial intelligence (AI) in the modern world is quite high, and this is reflected in the wide range of applied tasks where AI is used [5-9] . The relevance of using (AI) in visual testing of software products lies in its ability to automate and improve the testing process, reduce the time required to detect errors, and increase the accuracy of testing. AI can also detect problems such as misalignment of interface elements, color or font mismatches, and incompatibility with different devices and screen resolutions.

AI can help create test scripts that automatically check the visual appearance of a software product at different stages of development. This allows developers to detect and fix errors faster, which increases development speed and improves product quality.

In general, the use of AI and neural networks in visual testing allows companies to reduce testing costs, improve product quality, and increase user satisfaction, which makes this topic very relevant for modern software developers.

2. Related Work and Problem Statement

In the field of machine learning, a number of studies highlight key algorithms for comparing images. One study [10] shows the work of K-means and Mean Shift algorithms. The K-means algorithm for image comparison can be used to group images by their content or similarity. The basic idea is to divide a set of images into several clusters so that the images within each cluster are as similar as possible. Mean Shift is a non-directional clustering algorithm that can be used for image comparison. In the context of image comparison, the algorithm can be used to detect areas that have a similar color or texture. These algorithms are very useful, but they are difficult to use in cross-browser testing.

In another study [11], the author describes how cross-browser testing can be done. Test models are trained on different browsers so that snapshots can be compared. The comparison is done by saving the reference data for each version of the model. This process is very time-consuming and requires constant additional training of test models.

In a study for Philips Consumer Lifestyle BV [12] on visual testing, five different algorithms were used for comparison in visual testing of the company's logo - Hoeffding Tree (HT) [13], Hoeffding Adaptive Tree (HAT) [14], Stochastic Gradient Tree (SGT) [15], Streaming Logistic Regression (SLGR), and Streaming k-Nearest Neighbors (SKNN) [16]. According to each of the algorithms, models were trained and the results were checked accordingly. This study is interesting in terms of using different algorithms, but it is very costly and does not solve the problems described above.

In the study on visual recognition of GUI components using deep learning techniques [17], the author describes Deep Learning & CNN (convolutional neural networks). He explains how to define a dataset, prepare datasets with reference images for model training, and how to filter out unnecessary data to avoid accidental triggering during verification. This study will be useful in forming our dataset for training a neural network.

There are also several ready-made AI-based visual testing systems for software products on the market. Here are some of them:

1. Applitools - this platform uses AI to detect and track changes in the user interface, allowing you to automatically detect errors and shortcomings in the UI. It has integration with various test frameworks and programming languages [18]
2. Test.ai - this platform also uses AI to automate mobile app testing, including visual testing. It has the ability to analyze user interface elements to identify problems, and is supported for iOS and Android applications [19]

3. Mabl - uses reinforcement learning to automate testing, including visual testing of web applications. Automatically updates tests when changes are made to the web application [20]
4. Functionize - automates website testing using AI and analyzes dynamic web content [21]
5. Despite the development of AI, there are many shortcomings in visual testing systems currently on the market. Below are the main ones:

- high cost
- limited ability to work with certain technologies
- limited functionality
- high complexity of use
- the need for additional model training, which requires resources and money
- there is no possibility to compare images using additional correlation

There's a problem with how to do visual testing when you have tests running on different versions of browsers and different operating systems. Currently, there is no single standard for web content.

Also, web browsers usually implement their own extensions of web standards, which differs in their behaviour. This behaviour may also differ in the same version of the browser installed on different operating systems. This means that colours, fonts, and shadows will be displayed differently in a web application.

Another problem is the dynamic pulling of fonts to the web application. For example, Google fonts are very often used, which are dynamically pulled up from a specific resource. And accordingly, when Google changes the font style or adds a new shadow for the font, it affects the automatic tests.

The same problem will occur when a web application uses CSS styles, fonts, colours that are dynamically pulled from third-party resources.

Sometimes it happens that an application can use an iframe that opens a third-party application from another system or web resource. Or the application uses a micro frontend that works remotely and independently of our web application. Then we can't protect ourselves from the changes that occur in these third-party services and, accordingly, the content in the application may be different each time.

In all these cases, we have to take different versions of the images and compare them later. And this, in turn, is very time-consuming, difficult to maintain, and there is a high risk of accidental triggering of automatic tests.

Therefore, the development of a new AI-based visual testing system, taking into account the problems and shortcomings described above, is relevant. It is necessary to consider the basic methods and principles of visual testing that can be used to solve this problem. The Estimation Process of Functioning Efficiency Level of Information Web-Resources considered in work [30]. Approaches to using artificial intelligence to identify security malware are presented in paper [31].

3. Methods of Image Segmentation and Processing

Comparison of objects in visual testing can be done using various methods. Here are some methods and tools that can be used for visual testing [22]:

1. Manual inspection - manual inspection of the user interface to detect errors such as misplacement, color changes, etc
2. Specialized tools for visual comparison - for example, Selenium, Appium, Cypress, which provide automated visual testing for websites and mobile applications
3. Visual regression tests are tests that automate the inspection of visual elements of the interface to detect any deviations from predefined standards [23]
4. Cross browser and cross device testing is a visual check to make sure that the interface looks and works the same on different browsers and devices [24]
5. Responsive design testing is a test that includes testing at different screen resolutions

AI is used in various aspects of software testing to automate the process, analyze, and identify problems. Therefore, the following principles can be used to automatically compare screenshots in visual testing [11-13]:

1. Pixel-to-pixel comparison
2. Using hash functions
3. Use of structural or content hashes
4. Comparison with computer vision
5. Comparison with vector or vectorized images

Pixel-to-pixel comparison - this method involves comparing pixels in two images to identify any differences [25]. It includes the following steps: collecting images; converting to a format suitable for comparison; comparing images; detecting differences. For pixel comparison, various formulas are often used to calculate the difference between pixel colors. The main formulas include:

1. Mean Squared Error (MSE): This formula measures the root mean square difference between each pair of pixels in two images. It is calculated by the formula:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(I_1(i) - I_2(i))^2$$

where I_1 and I_2 - are vectors of pixel values for each image, and n - is the number of pixels in the image.

2. Peak Signal-to-Noise Ratio (PSNR): This formula measures the ratio of the maximum possible signal strength to the false signal that occurs when a signal is encoded or transmitted. It is calculated by the formula:

$$PSNR = 10 * log_{10}(\frac{MAX^2}{MSE})$$

where MAX - is the maximum pixel value (for example, 255 for color images).

3. Structural Similarity Index (SSI): This formula takes into account not only the difference in color intensity of pixels, but also the structural differences between them. It is a more complex formula that takes into account the size, contrast, and structure of the image.

$$SSI(x,y) = \frac{2 * \mu_x * \mu_y + c_1}{\mu_x^2 + \mu_y^2 + c_1} * \frac{2 * \sigma_{xy} + c_2}{\sigma_x^2 + \sigma_y^2 + c_2}$$

where:

- x and y - the images being compared
- - μ_x i μ_y are the average pixel values in the images x and y respectively
- - σ_x^2 i σ_y^2 pixel variances in the images x and y respectively
- - σ_{xy} - is the covariance between pixels in images x and y
- - c_1 i c_2 -states used to stabilize the formula

This formula gives an SSI value ranging from -1 (complete difference) to 1 (complete similarity). It is generally considered that an SSI value of more than 0.75 indicates a fairly high similarity between images. This approach is simple, but can be quite resource intensive for large images.

Using hash functions - hash functions can convert an image into a unique hash that can be compared to another hash to determine differences. During testing, the software can store the hash value of the image to which the current image is compared. This allows you to quickly identify any changes or differences between images. Another important aspect of using hash functions is reducing the amount of data to store. Because hash values are short, they are easy to store and compare, making them ideal for use in automated testing where large numbers of images or other big data need to be stored. The use of hash functions in automated visual testing allows for fast and efficient image comparisons, reduces the amount of data to store, and verifies data integrity during testing.

Using structural or content hashes - they allow you to compare the structure or content of images, ignoring some small differences such as colors or sizes. A structural hash is a unique value that represents the structure of elements on a web page or other graphical interface. This hash can be created based on a variety of properties, such as dimensions, element location, textual content, etc. In automated visual testing, structural hashes are used to compare visual elements between test and control screens. If the structural hashes do not match, it may indicate that there are changes to the page layout that require attention. Content hashes are also used to verify the content of web pages or other graphical software interfaces, but they are based on the content of specific elements, such as images or textual content. When using content hashes, the system can compare images or text across test screens to detect minor changes, such as display errors or incorrect content.

Comparison using computer vision - computer vision methods can detect objects in an image and compare them with each other according to various parameters such as size, shape, and position [26].

Comparison using vector or vectorized images - this method uses the vector representation of images to compare them in a relatively scalable space, which allows you to identify differences even in the case of a slight change in the size or shape of objects. In vector images, information about the shape and size of objects is represented in the form of mathematical vectors, which allows you to compare them effectively. The following steps can be used to compare vectorized images [27-29]:

1. Converting a vector image to a mathematical vector: Each object in the image (e.g., line, curve, shape) can be described by a mathematical vector using parameters such as point coordinates, radii, and slope angles.
2. Comparing mathematical vectors: A single or standardized comparison method can be used to determine the similarities or differences between vectors. For example, you can use the distance between vectors (e.g., Euclidean distance) or the angle between vectors.
3. Setting a threshold: By defining a threshold for distance or angle, you can decide whether two images are considered the same or different.

It's also important to keep in mind that comparing vector images can be more demanding in terms of computational complexity than comparing raster images.

These methods and tools can be used separately or in combination to ensure high quality visual software testing.

Our approach will consist of a two-step process. The first step is to scan a web page and create an actual snapshot. The second step consists of visual comparison of the images based on a neural network. It is proposed to conduct the study using two principles of image comparison - pixel-to-pixel comparison and computer vision comparison. These processes are discussed in more detail in the next section.

4. Implementation

To solve this problem, it is proposed to develop a service based on a neural network that will be able to compare images on different versions of browsers and say whether these images are the same. That is, we have a reference image of a page. We feed it to the trained system, and the system, regardless of the browser version and operating system, will return the result whether the images are correct or not.

This system will be easily integrated into any project where there are written automated tests and visual verification. It will be independent of the programming language and frameworks in which the automatic tests are written.

It is proposed to divide this system into two parts. The first part will be a service that will be able to make changes to the software product. As input, it will receive a specific component of a web application - a whole page, a dialog, a page component, or even just the element itself. As an output, it will give a ready-made snapshot. Also, this service will be able to sketch certain elements of a web page if necessary and thus ignore them for verification. This is necessary when the page has different dynamic data and, accordingly, after each test run, the snapshot will contain different information.

The second part will be based on a neural network. It will receive two snapshots as input - the first is the one we have a reference with which we will make a comparison, the second is the current snapshot that has just been taken by the system. You will also need to set additional comparison correlation parameters such as:

- threshold for ignoring shadows
- font correction threshold
- pixel-to-pixel correlation comparison

Figure 4 shows how the system will work based on the principle of pixel-to-pixel comparison. The system accepts as input the component whose image is to be taken, and also accepts as input a reference image that was taken earlier and with which the comparison will be made.

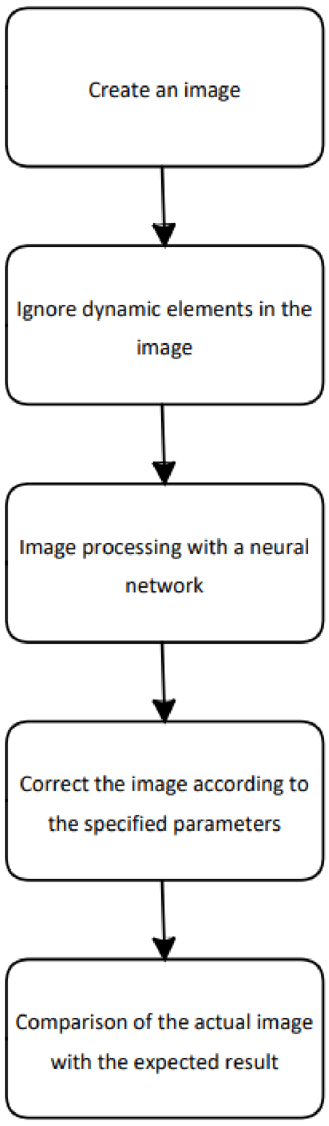

Create an image
Ignore dynamic elements in the image
Image processing with a neural network
Correct the image according to the specified parameters
Comparison of the actual image with the expected result

Figure 3: Scheme of the system operation

The general scheme of the improved system is shown in (Fig. 3). We will also develop a mechanism for visual highlighting of elements when the images differ. At the output, this system will tell whether the images are the same or not. If the images are the same, then the visual check in the tests will be successful. If the system says that the images are different, then data will be returned on the current, expected images and data on the image in which the pixel-by-pixel difference will be shown.

Then the system takes a new picture of the component that was passed to the input. After that, the images are compared pixel by pixel. Since the images are not identical, the system took an additional image and highlighted the places on the image that differed in red.

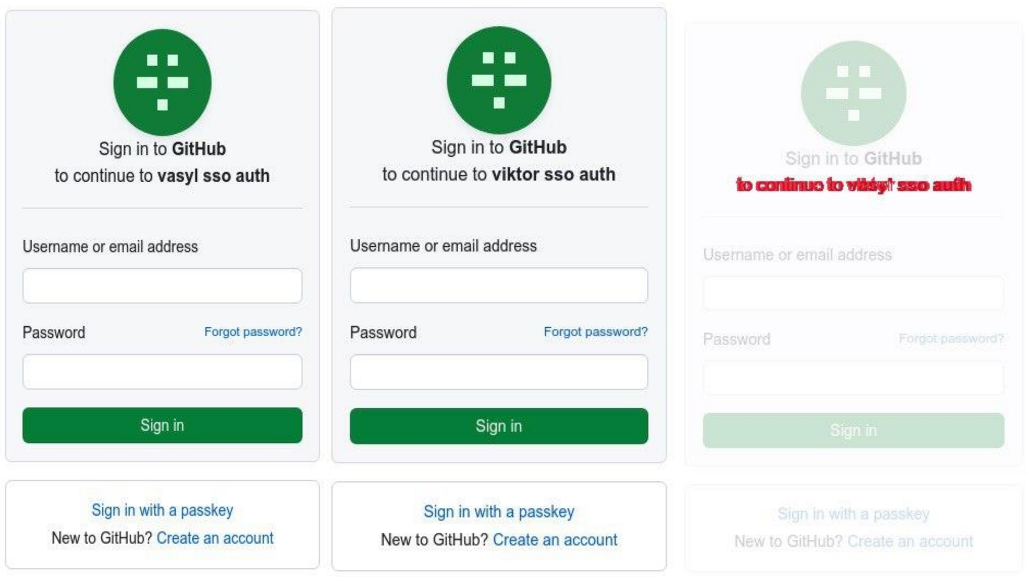

Sign in to GitHub
to continue to vasyl sso auth
Username or email address
Password
Forgot password?
Sign in
Sign in with a passkey
New to GitHub? Create an account
Sign in to GitHub
to continue to viktor sso auth
Username or email address
Password
Forgot password?
Sign in
Sign in with a passkey
New to GitHub? Create an account
Sign in to GitHub
Username or email address
Password
Forgot password?
Sign in
Sign in with a passkey
New to GitHub? Create an account

Figure 4: Pixel-to-pixel comparison of images

As a result, the system returned three images - the one we sent as a reference, the one that was taken and the image with elements that differ from the reference image. Figure 5 shows how the system works when you want to ignore dynamic elements in the image.

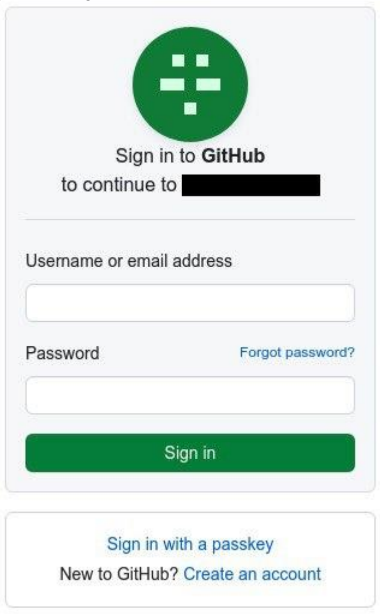

Sign in to GitHub
to continue to [redacted]
Username or email address
Password
Forgot password?
Sign in
Sign in with a passkey
New to GitHub? Create an account

Figure 5: Ignoring an element in an image

The system works according to the same approach as described above, only here an additional parameter is passed to the element that needs to be ignored in the image. Also here, the principle of computer vision is used to better determine the placement of elements that are necessary to ignore dynamic data in the image and avoid accidental negative triggers in tests. As you can see, this element will be painted black.

Python, PHP, the ImageMagick library, and the Selenium Webdriver test framework were used to design the automated system. The result of the work is an automated visual testing system for web interfaces using artificial intelligence methods for more accurate comparison of images.

5. Conclusions

Modern software products are becoming increasingly complex, and testing requires significant effort to detect errors and defects. Using neural networks and AI in visual testing of software products can significantly improve the quality of testing, reduce the time required to detect bugs, can be extremely accurate in detecting even the smallest anomalies or defects in the program code or user interface, and provide a more reliable and high-quality product.

This article discusses research into artificial intelligence tools and technologies for analyzing, processing, and comparing GUI images obtained during automated regression functional tests. It also describes the creation of a system that allows users to detect errors in the web application interface and generate reports. Snapshot comparison is performed using a developed testing service that can take a snapshot of a page or corresponding elements on a web page, perform additional correlation of the snapshot, and has the ability to ignore certain elements in the snapshot.

The preliminary results of the system using pixel-to-pixel comparison and comparison at using computer vision showed a good result in comparing images and allowed us to partially solve the problems described above.

Our current approach can be improved by using the principle of comparing structural or content hashes, which will allow for better training of the neural model, as well as allow for more accurate transfer of additional parameters for correlation of image comparison. This will directly reduce the false positive rate at the screen level and increase the accuracy of our approach.

Another useful direction is to study the integration of our system with existing AI services to share neural models. This will allow access to other models that could be used for similar research. Combined with our service, this could allow us to create a powerful tool for visual testing of software products.

References

[1] Graphic User Interface Modelling and Testing Automation [Online]. Available: https://vuir.vu.edu.au/16066/1/Xuebing_Yang_PhD_thesis%2C_May_2011.pdf.

[2] Robert, M., Linda, H., Shapiro, G.: Image segmentation techniques. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0734189X85901537.

[3] Hlaing, S., Khaing, A.: Weed and crop segmentation and classification using area thresholding. International Journal of Research in Engineering and Technology, 2321-7308 (2014).

[4] Lottes, P., Stachniss, C.: Semi-Supervised Online Visual Crop and Weed Classification in Precision Farming Exploiting Plant Arrangement. [Online]. Available: http://flourishproject.eu/fileadmin/user_upload/publications/lottes17iros.pdf.

[5] Turchenko, I., Osolinsky, O., Kochan, V., Sachenko, A., Tkachenko, R., Svyatnyy, V., Komar, M. Approach to neural-based identification of multisensor conversion characteristic. Proceedings of the 5th IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS'2009, 2009, pp. 27-31.

[6] Komar, M., Kochan, V., Dubchak, L., Sachenko, A., Golovko, V., Bezobrazov, S., Romanets, I. High performance adaptive system for cyber attacks detection. Proceedings of the 2017 IEEE 9th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2017, 2017, 2, pp. 853-858, 8095208

[7] Komar, M., Dorosh, V., Hladiy, G., Sachenko, A. Deep neural network for detection of cyber attacks. 2018 IEEE 1st International Conference on System Analysis and Intelligent Computing, SAIC 2018 - Proceedings, 2018, 8516753.

[8] Deep multilayer neural network for predicting the winner of football matches. Anfilets, S., Bezobrazov, S., Golovko, V., Sachenko, A., Komar, M., Dolny, R., Kasyanik, V., Bykovyy, P., Mikhno, E., Osolinskyi, O. International Journal of Computing, 2020, 19(1), pp. 70-77.

[9] Lipyanina, H., Sachenko, S., Lendyuk, T., Brych, V., Yatskiv, V., & Osolinskiy, O. (2021, January). Method of detecting a fictitious company on the machine learning base. In International

Conference on Computer Science, Engineering and Education Applications (pp. 138-146). Cham: Springer International Publishing.
[10] Artificial Intelligence in Automated System for WebInterfaces Visual Testing. [Online]. Available: https://ceur-ws.org/Vol-2604/paper68.pdf.
[11] Automated Cross-Browser Compatibility Testing - https://www.researchgate.net/publication/221554140_Automated_cross-browser_compatibility_testing
[12] Streaming Machine Learning and Online Active Learning for Automated Visual Inspection [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896322002075?ref=pdf_download&fr=RR-2&rr=862e3bc3fe2cfbce.
[13] Hulten, G., Spencer, L., and Domingos, P. (2001). Mining time-changing data streams. In Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, 97-106.
[14] Bifet, A. and Gavalda, R. (2009). Adaptive learning from evolving data streams. In International Symposium on Intelligent Data Analysis, 249-260. Springer
[15] Gouk, H., Pfahringer, B., and Frank, E. (2019). Stochastic gradient trees. In Asian Conference on Machine Learning, 1094-1109. PMLR.
[16] Lughofer, E. (2017). On-line active learning: A new paradigm to improve practical usability of data stream modeling methods. Information Sciences, 415, 356-376.
[17] Visual Recognition Of Graphical User Interface Components Using Deep Learning Technique [Online]. Available: https://jiki.cs.ui.ac.id/index.php/jiki/article/view/845.
[18] Applitools. [Online]. Available: https://applitools.com.
[19] Test.ai. [Online]. Available: https://test.ai/all-products.
[20] Mabl. [Online]. Available: https://mabl.com.
[21] Function. [Online]. Available: https://www.functionize.com.
[22] Tian, J., (2005). Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement. Wiley.
[23] Visual testing of Graphical User Interfaces: An exploratory study towards systematic definitions and approaches. [Online]. Available:: https://www.researchgate.net/publication/261486256_Visual_testing_of_Graphical_User_Interfaces_An_exploratory_study_towards_systematic_definitions_and_approaches.
[24] Mohanty, H., Mohanty, J., Balakrishnan, A.: Trends in Software Testing. Springer, Singapore (2017).
[25] B. Kim, S. Park, T. Won, J. Heo, and B. Kim, "Deep-learning based web UI automatic programming," in Proceedings of the 2018 Research in Adaptive and Convergent Systems, RACS 2018, 2018, pp. 64-65, doi: 10.1145/3264746.3264807.
[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks."
[27] M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," May 2019.
[28] Brooks, P., Robinson, B., and Memon, A., (2009). An Initial Characterization of Industrial Graphical User Interface Systems in Proceedings of International Conference on Software Testing Verification and Validation Denver, CO, IEEE, pp.11-20.
[29] Napoletano, P., Piccoli, F., and Schettini, R. (2021). Semi-supervised anomaly detection for visual quality inspection. Expert Systems with Applications, 115275.
[30] M. Dyvak, A. Melnyk, A. Kovbasistyi, R. Shevchuk, O. Huhul and V. Tymchyshyn, "Mathematical Modeling of the Estimation Process of Functioning Efficiency Level of Information Web-Resources," 2020 10th International Conference on Advanced Computer Information Technologies (ACIT), Deggendorf, Germany, 2020, pp. 492-496, doi: 10.1109/ACIT49673.2020.9208846
[31] S. Bezobrazov, A. Sachenko, M. Komar, and V. Rubanau, "THE METHODS OF ARTIFICIAL INTELLIGENCE FOR MALICIOUS APPLICATIONS DETECTION IN ANDROID OS", IJC, vol. 15, no. 3, pp. 184-190, Sep. 2016.