ISSN 2616-5775 Vol. 8, Issue 11: 116-121, DOI: 10.25236/AJCIS.2025.081113

Application of Computer Vision Algorithms in Software Test Automation

Yuke Xing

Western Research Institute of Computing Technology, Chongqing, China, 400000

Abstract: *Against the backdrop of continuously accelerating software iteration speeds and increasing diversification of terminal devices, traditional automated testing tools that rely on control attributes struggle to meet requirements such as visual consistency and cross-device adaptation. Computer vision algorithms, with their capabilities in pixel-level analysis, accurate target localization, and dynamic scene recognition, have become a key technology to break through the "visual blind spots" in software testing. This paper sorts out the development trends of software test automation, analyzes the principles of core computer vision algorithms, expounds on application practices in scenarios such as UI (User Interface) verification and compatibility testing, examines current technical challenges, and proposes optimization paths. It aims to provide references for building a more comprehensive software test automation system.*

Keywords: *Computer Vision; Software Test Automation; UI Testing; Object Detection; Compatibility Verification*

1. Introduction

Driven by the popularity of Agile Development and DevOps (Development and Operations), the software delivery cycle has shortened from "month-level" to "day-level", which demands further improvements in testing efficiency and coverage. Traditional automated testing tools rely on underlying attributes such as ID and XPath (XML Path Language) for element localization and interaction. When the page is redesigned or the control type changes, the test scripts require extensive modifications, accounting for over 60% of maintenance costs. Additionally, some visual defects—such as shifted button positions, incorrect font colors, and stretched images—can only be identified by the human eye and cannot be verified automatically, becoming a "shortcoming" in the coverage of automated testing.

Computer vision algorithms simulate human visual recognition by detecting and identifying features in software interface screenshots and video streams, enabling comparison and verification without relying on control attributes. According to IDC (International Data Corporation) 2024 data, using computer vision for UI testing can reduce UI test maintenance costs to 45% while increasing the detection rate of visual defects to 93%. Furthermore, it fills the gap of lacking visual verification in the traditional software testing field, upgrading the logic-only verification mode to a "logic + visual" verification mode and enhancing the technical capabilities of software testing.

2. Development Trends of Software Test Automation

2.1 Testing Requirements: From "Functional Verification" to "Comprehensive Quality Assurance"

With the continuous expansion of software system scale and complexity, automated testing has gradually become a core direction of industry development. Early software test automation focused on functional logic verification, such as the correctness of interface calls and data processing accuracy. Nowadays, as users' requirements for software experience increase, the testing scope has expanded to non-functional requirements including visual presentation, interaction fluency, and cross-device adaptation [1]. For example, e-commerce APPs need to verify both "normal order submission function" and "visual consistency of product detail pages", while financial software must ensure both "correct transfer logic" and "compliant interface fonts". This requires automated testing tools to have more comprehensive verification capabilities.

2.2 Testing Scenarios: From "Single Terminal" to "Multi-Device Collaboration"

The development of the mobile Internet and the application of intelligent terminals require APPs to be compatible with devices such as mobile phones, tablets, smart TVs, and in-vehicle systems. These devices vary in screen size (e.g., 4.7-inch mobile phones, 10.9-inch tablets), resolution (720P/1080P/2K), and operating systems, leading to different interface display forms of APPs. Traditional compatibility testing methods require significant investment in real-device testing across different models and brands, yet still face issues such as missed tests for certain device models and difficulties in generating and executing cross-device test cases in real operating environments.

2.3 Testing Mode: From "Script-Driven" to "Intelligent Adaptive"

Traditional test automation relies on rigid script writing with extensive manual coding of fixed logic. Given the current high software iteration frequency and frequent changes in page elements, script maintenance has become increasingly challenging. The industry is gradually shifting to "intelligent adaptive testing", which uses algorithms to automatically detect interface elements and learn testing rules, reducing manual intervention [2]. For instance, computer vision algorithms can automatically locate the "login button" without manual definition of control attributes, significantly improving the stability and reusability of test scripts.

3. Principles and Features of Computer Vision Algorithms

The application of computer vision in software test automation relies on three core technical modules: image preprocessing, feature extraction and matching, and object detection. These modules collaborate to realize interface analysis and verification, with the following core principles and features:

3.1 Image Preprocessing: Eliminating Interference and Unifying Analysis Benchmarks

The images of software interfaces are easily affected by screen brightness, resolution, and compression formats. Before conducting subsequent analysis on software interface images, some preprocessing work is required. Commonly used methods are as follows:

Normalization processing: For screenshots with different resolutions, scale them to standard - sized images (e.g., 1920x1080 pixels). Use grayscale processing to compress and reduce the color channels of the images, eliminating image differences caused by hardware reasons and making it more convenient for subsequent feature extraction. The weighted average method is used for grayscale processing, and the formula is Gray = 0.299R + 0.587G + 0.114B, which conforms to the human eye's sensitivity to green.

Gaussian filtering is used to remove the noise introduced during the screenshot process, with the parameter set as cv2.GaussianBlur(src, (5,5), 0), where the size of the Gaussian kernel is 5x5 and the standard deviation is 0. Median filtering is used for edge noise. By sorting the grayscale values of neighboring pixels and taking the median, the noise can be removed. For example, when using a 3x3 window, sort the grayscale values of 9 pixels and take the middle value.

For images with unobvious detailed features, the Sobel operator is used to calculate the gradient of the image for edge enhancement. The formula is cv2.Sobel(src, ddepth, 1, 1, ksize = 3), where ddepth is the depth of the output image and ksize is the size of the convolution kernel. Then, the Laplace operator is used to increase the contrast of the image. A commonly used 3x3 convolution kernel is [0, -1, 0], [-1, 4, -1], [0, -1, 0], which improves the accuracy of subsequent element recognition.

3.2 Feature Extraction and Matching: Establishing "Expected-Actual" Correlation

Feature extraction and matching are core to determining whether interface elements are "correct", requiring the establishment of correlations between reference images (design drafts or production environment screenshots) and test images (to-be-verified interfaces). The mainstream technical approaches are divided into two categories [3]:

3.2.1 Traditional Feature Algorithms

Algorithms such as SIFT (Scale-Invariant Feature Transform) and SURF (Speed Up Robust

Features) extract feature points (e.g., icon corners, text edges) and construct 128-dimensional feature vectors. The FLANN (Fast Library for Approximate Nearest Neighbors) matcher is then used to calculate vector similarity. These algorithms are invariant to scaling and rotation, suitable for scenarios involving icon offset and control size changes, with a matching accuracy of over 85%.

3.2.2 Deep Learning-Based Feature Algorithms

Deep learning feature algorithm Based on the CNN model, multi - layer convolution is used to automatically learn high - order interface features, such as the red color, rounded corners and text combination of the payment button. In software testing, a three - layer architecture is commonly used, including the convolution layer, pooling layer and fully connected layer. The Dropout layer is combined to prevent overfitting. It represents features more comprehensively than traditional algorithms, and the accuracy can be improved by more than 30%. It is suitable for the recognition of elements in complex interfaces.

3.3 Object Detection: Localizing Elements for Interaction Verification

Object detection algorithms locate the position and category of specific elements in interfaces, serving as the key to "element interaction verification". Two technical solutions are commonly used in testing scenarios:

3.3.1 Two-Stage Detection Algorithms

Algorithms like Faster R-CNN (region-based CNN) follow a two-step process. It candidate region generation then feature classification. They first select regions likely containing targets, then classify objects within these regions and estimate bounding box coordinates. Positioning error can be controlled within 1-2 pixels, but the speed is slow (only 5 frames per second), making them suitable for precise matching of static elements in interfaces.

3.3.2 One-Stage Detection Algorithms

Algorithms such as YOLO (You Only Look Once) and SSD (Single Shot MultiBox Detector) complete object localization and classification in a single convolution operation, achieving a detection frame rate exceeding 30 fps, which meets the real-time requirements of dynamic interfaces. Therefore, most current interactive testing adopts this technology.

4. Application Scenarios of Computer Vision Algorithms in Software Test Automation

4.1 UI Automated Testing: Breaking Dependence on Control Attributes

UI testing is a crucial component of software testing. Computer vision algorithms use visual features for localization, significantly improving script stability and maintainability [4]. For element localization and interaction verification, the YOLOv8 algorithm is used to identify elements such as "login input boxes" and "verification code icons" in interfaces, simulating human operation logic and verifying interaction results through template matching. For example, in bank APP testing, the algorithm can automatically locate the "transfer amount input box"; after entering test data, it verifies whether the "confirm button" is activated by matching its visual changes (e.g., changing from gray to blue).

Visual Consistency Verification compares test interface screenshots with standard baseline screenshots at the pixel level, using the SSIM (Structural Similarity Index) to measure similarity (ranging from 0 to 1). An alarm is triggered if the index is below a threshold (e.g., 0.95). In scenarios such as e-commerce "product detail pages", this method can easily detect issues like text color differences, image stretching (from 16:9 to 4:3), and button offset (shifted 10 pixels to the right).

Text Recognition and Verification combines OCR algorithms (e.g., PaddleOCR, Tesseract) to recognize, compare, and verify on-screen text. In government service software testing, OCR technology can automatically identify content such as "Processing Progress: Approved" and "Processing Result: Passed", ensuring the accuracy of information display.

4.2 Cross-Device Compatibility Testing: Reducing Hardware Dependence Costs

Traditional compatibility testing requires a large number of physical devices, resulting in high costs and low efficiency. Computer vision algorithms realize cross-device test automation through "image standardization + feature transfer".

Based on the reference interface, "expected templates" for different resolutions are generated via image scaling. Screenshots from actual devices (e.g., 720P mobile phones, 2K tablets) are then compared with features in the "expected templates" to verify adaptive effects—for example, whether the "chat input box" height is 1/5 of the screen height and whether the send button is positioned on the right, regardless of the device (720P mobile phone or 2K tablet).

Cross-System Visual Difference Filtering addresses visual differences (e.g., button rounded corners, window shadows) across operating systems (Windows, macOS, Linux). CNN models learn the visual features of different systems, enabling the differentiation between "system-native differences" and "software defects" during testing.

Multi-Scenario Adaptation Testing for Mobile Devices combines device sensor data (screen rotation, brightness adjustment) with computer vision to monitor interface changes in mobile APP testing. For example, when a mobile phone switches from portrait to landscape mode, the algorithm can automatically verify whether "the video playback window becomes full-screen" and "the list layout changes from single-column to double-column". When brightness is adjusted, it verifies whether text contrast meets accessibility standards.

4.3 Dynamic Interaction and Anomaly Monitoring: Covering Traditional Testing Blind Spots

Traditional automated testing struggles to cover dynamic interactions and abnormal scenarios during software operation. Computer vision algorithms enable monitoring through real-time video stream analysis:

1) Dynamic Interaction Process Verification: The video stream of software operation processes is split into frames, with each frame compared against expected templates to verify step correctness. For example, in food delivery APP testing, the algorithm can monitor the entire process of "selecting products → adding to cart → clicking checkout → filling in address → submitting order", ensuring each interface change meets expectations (e.g., "quantity +1" after adding to cart).

2) Abnormal Scenario Automatic Recognition: Temporal image analysis identifies abnormal features: a white screen is characterized by "uniform pixel values in 5 consecutive frames"; a crash is marked by "sudden jump to the system desktop"; and a freeze is indicated by "no change in the loading animation for 10 consecutive frames". When anomalies are detected, the algorithm automatically captures screenshots and records videos to preserve fault data.

3) Performance-Related Visual Monitoring: Image change frequency is used to indirectly monitor software performance. For example, if the frame rate (FPS) is below 24 during "list scrolling", the interface is considered laggy; if the "loading" animation persists for more than 3 seconds without image display, the loading performance is abnormal.

5. Challenges and Optimization Directions in Software Test Automation

5.1 Core Technical Challenges

5.1.1 High Dependence on Labeled Data

Deep learning-based object detection and feature extraction algorithms require large amounts of labeled data (e.g., bounding box annotations for "buttons" and "input boxes"). However, software interfaces iterate rapidly, leading to high costs for updating labeled data. Statistics show that a UI testing model for a medium-complexity APP requires at least 5,000 labeled screenshots to achieve a recognition accuracy of over 90%.

5.1.2 Multiple Environmental Interference Factors

Dynamic elements and hardware differences in the testing environment can alter image features and interfere with algorithm judgments. For example, a pop-up mobile notification bar blocking the software interface may cause the algorithm to mistakenly judge "missing interface elements".

5.1.3 Insufficient Generalization Ability

Visual differences of the same type of element (e.g., varying colors and sizes of "confirm buttons" across pages) reduce recognition accuracy. For instance, in an e-commerce APP, the "Add to Cart" button on the product page is red, while the "Submit Order" button on the checkout page is orange—traditional models may fail to recognize them as the same type of "operation button".

5.1.4 Difficulty in Balancing Real-Time Performance and Accuracy

One-Stage algorithms (e.g., YOLO) offer high real-time performance but low accuracy, while Two-Stage algorithms (e.g., Faster R-CNN) provide high accuracy but poor real-time performance. In dynamic interface testing (e.g., game software), it is difficult to meet both "real-time monitoring" and "accurate recognition" requirements.

5.2 Key Optimization Directions

5.2.1 Few-Shot Learning and Transfer Learning for Cost Reduction

Few-Shot Learning algorithms use a "meta-learning" framework to train models with only dozens of labeled screenshots. Combined with transfer learning, pre-trained model parameters from general image datasets (e.g., ImageNet) are transferred to software interface recognition tasks, reducing labeling costs [5]. For example, based on the Mobile Net pre-trained model, 88% button recognition accuracy can be achieved with only 300 labeled APP screenshots.

5.2.2 Intelligent Filtering of Dynamic Interference

Semantic segmentation algorithms (e.g., Mask R-CNN) distinguish between "software-native elements" and "external interference elements" (e.g., system notifications, advertisements), automatically masking interference areas before testing. Adaptive threshold technology dynamically adjusts image comparison parameters based on screen brightness and color gamut, eliminating the impact of hardware differences.

5.2.3 Multi-Modal Feature Fusion for Efficiency Improvement

Combining visual features (color, shape) and semantic features (text content, function description) of interface elements improves generalization ability. For example, OCR (Optical Character Recognition) extracts the text "Submit Order", which is combined with the button's red color and rounded corners, ensuring accurate recognition even if the position and size change.

5.2.4 Lightweight Models and Hardware Acceleration

Model compression technologies (pruning, quantization) improve real-time performance while ensuring accuracy. For example, quantizing the YOLOv8 model to INT8 precision reduces its size by 75% and doubles detection speed. Additionally, GPU and NPU hardware acceleration further enhances algorithm efficiency, meeting the "60fps real-time monitoring" requirement for game software.

6. Conclusion

By simulating human visual perception logic, computer vision algorithms break the limitation of traditional automated testing's dependence on control attributes. They demonstrate efficient and stable advantages in scenarios such as UI verification, cross-device compatibility testing, and dynamic anomaly monitoring, serving as an important supplement to the software test automation system. Currently, with the development of deep learning technology, the application of computer vision in software testing is evolving from "single-element recognition" to "full-process intelligent analysis".

In the future, its development will focus on three directions: first, multi-modal fusion—combining visual, text, and log data to realize end-to-end automation of "defect localization - root cause analysis"; second, adaptive learning—enabling test models to automatically optimize rules during iteration, reducing manual intervention; third, cross-domain transfer—applying mature algorithms from industrial vision and autonomous driving to AR/VR software testing, further expanding the testing boundary.

References

[1] Cui, Jing. (2025). Study on the Automated Testing for Computer Application Software based on Deep Learning. China-Arab States Science and Technology Forum (Chinese & English), (06), 97-100.

[2] Wang, Peng. (2025). Research on the Application of Artificial Intelligence in Computer Application Software Development. Mobile Information, 47(07), 335-337.
[3] Yin, Jie. (2024). Analysis of Automated Development Technology for Computer Application Software. Information Recording Materials, 25(03), 165-167.
[4] Li, Liang. (2025). An Automated Assembly Line Quality Inspection System Based on Computer Vision. Modern Manufacturing Technology and Equipment 61(06), 211-213, 218.
[5] Wang, Zheng. (2025). Research on Intelligent Testing and Optimization Technology of Artificial Intelligence in Computer Software Development. Information Recording Materials, 26(9), 30-32.