

Using Machine Learning and Image Recognition in User Interface Test Automation

DISSERTATION

Submitted in partial fulfillment of the requirements of the
MTech Data Science and Engineering Degree programme

By

V Nithin Chary
2020FC04570

Under the supervision of

Ish Abbi
Associate Director, Sprinklr

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
Pilani (Rajasthan) INDIA
(March, 2023)

Acknowledgements

First and foremost, I want to express my gratitude to Mr. Ish Abbi, my M.Tech supervisor, for his helpful advice and unwavering support during my studies.

I'd also want to thank Mr. TNGK Ranganath for his technical insights and guidance, without which completing this work wouldn't have been possible.

Thank you everyone on my team Sprinklr India Pvt. Ltd. for their support.

Finally, I want to thank my parents and friends for their relentless emotional support during my course of study.

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

CERTIFICATE

This is to certify that the Dissertation entitled “Using Machine Learning and Image Recognition in UI Test Automation” and submitted by Mr. V Nithin Chary ID No. 2020FC04570 in partial fulfillment of the requirements of DSECLZG628T Dissertation, embodies the work done by him/her under my supervision.

Ish Abbi



Name: Ish Abbi

Designation: Associate Director

Place: Delhi

Date: 10 Mar 2023

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
FIRST SEMESTER 2022-23

DSECLZG628T DISSERTATION

Dissertation Title: Using Machine Learning and Image Recognition in UI Test Automation

Name of Supervisor: Ish Abbi

Name of Student: V Nithin Chary

ID No. of Student: 2020FC04570

Abstract

Automated testing of web user interfaces (UIs) involves simulating user interactions and verifying the resulting UIs. However, using locators to identify web elements can become obsolete due to changes in the application, causing test scripts to fail. Additionally, manual visual verification of web pages is time-consuming and error-prone. This paper proposes a solution to these challenges by using image recognition for web element identification and machine learning to perform visual testing of UIs. Screenshots of web pages are used as training data to build a machine-learning model that classifies inputs. During testing, the model categorizes the application's screen into a class and validates it against the class's baseline data, detecting anomalies at the pixel level. This approach offers a promising solution to automate UI tests, reduce errors, and improve software development efficiency.

Keywords:

Automated testing, locators, user interfaces, visual testing, anomalies

List of Symbols & Abbreviations used

| | |
|---------------|------------------------------------|
| UI | User Interfaces |
| AUT | Application Under Test |
| ML | Machine Learning |
| CNN | Convolutional Neural Network |
| Web UI | Website User Interface |
| Web | Website |
| App | Application |
| GUI | Graphical User Interface |
| IDE | Integrated Development Environment |
| API | Application Programming Interface |
| JSON | JavaScript Object Notation |
| REST | Representational State Transfer |
| DOM | Document Object Model |

List of Tables

| | |
|--|----|
| Table 3.4.6.1.t1: Model accuracy at various noise levels | 64 |
| Table 3.4.7.2.t1: Tuned LeNet-5 Accuracy at various noise levels | 68 |
| Table 3.4.7.2.t2: Model prediction speed at various noise levels | 69 |
| Table 3.4.7.2.t3: Models h5 file size | 70 |

List of Figures

| | |
|--|----|
| Figure: 2.3.1.f1 Sample Login Screen | 7 |
| Figure: 2.3.1.f2 Sample Login Screen with Password Field element capture option. | 7 |
| Figure: 2.3.1.f3 Captured Password Field Webelement png file | 8 |
| Figure: 3.3.1.f1 Classes directory containing categories(folders/labels) | 24 |
| Figure: 3.3.1.f2 Properties of a single category directory | 24 |
| Figure: 3.4.4.1.f1 Training and Validation Accuracy/Loss for Alexnet | 42 |
| Figure: 3.4.4.2.f1 Training and Validation Accuracy/Loss for LeNet-5_32x32 | 44 |
| Figure: 3.4.4.3.f1 Training and Validation Accuracy/Loss for CustomeModel | 46 |
| Figure: 3.4.4.4.f1 Training and Validation Accuracy/Loss for VGG16 | 48 |
| Figure: 3.4.4.5.f1 Training and Validation Accuracy/Loss for SqueezeNet | 50 |
| Figure: 3.4.5.1.f1 Sample Image of a base class in challenging_dom category | 51 |
| Figure: 3.4.5.1.f2 Sample screen of testclasses | 52 |
| Figure: 3.4.5.1.f3 Sample screen of highnoise level | 52 |
| Figure: 3.4.5.1.f4 Sample screen of veryhighnoise level | 53 |
| Figure: 3.4.5.1.f5 Sample screen of ultrahighnoise level | 53 |
| Figure: 3.4.5.1.f6 Sample screen of ultraprnoiselevel | 54 |
| Figure: 3.4.6.1.f1 Model Accuracy at various noise levels | 64 |
| Figure: 3.4.7.2.f1: Tuned LeNet-5+Models Accuracy at various noise levels | 68 |
| Figure: 3.4.7.2.f2: Model prediction speed at various noise levels | 69 |
| Figure: 3.4.7.2.f3: Models h5 file size | 70 |
| Figure: 3.5.f1:Architecture of service/category service | 71 |
| Figure: 3.5.f2:Architecture of service/screendiff service | 73 |
| Figure: 4.1.f1: Project Architecture | 77 |
| Figure: 4.1.f2: localhost:5000 web screen | 79 |
| Figure: 7.1.f1: Pipeline test script architecture | 92 |
| Figure: 7.1.f2: getScreenCategory function design | 92 |
| Figure: 7.1.f3: getScreenDiff function design | 93 |
| Figure: 7.1.f4: Sample input screen containing noise. | 93 |
| Figure: 7.1.f5: Sample output screen highlighting noise | 94 |

Contents

| | |
|---|-------------|
| Cover | i |
| Acknowledgements | ii |
| Certificate from the Supervisor | iii |
| Dissertation Abstract | iv |
| List of Symbols & Abbreviations used | v |
| List of Tables | vi |
| List of Figures | vii |
| Contents | viii |
| 1. Introduction | 1 |
| 1.1. Software Development Lifecycle: Testing | 1 |
| 1.2. User Interface Testing. | 1 |
| 1.3. User Interface Test Automation | 2 |
| 1.3.1. Current Automation Process | 2 |
| 1.4. Shortcomings in UI Testing | 3 |
| 1.4.1. Current Automation process | 3 |
| 1.4.2. User Interface Visual Regression Tests. | 3 |
| 1.5. Problem Statements the Dissertation addresses | 4 |
| 1.5.1. Locator Strategy in UI Test Automation | 4 |
| 1.5.2. Enhancing UI Tests to pixel perfection | 4 |
| 2. Problem 1: Locator Strategy | 5 |
| 2.1. Understanding the Problem: How locators work? | 5 |
| 2.2. Proposed Solution: Using Image Template Matching. | 5 |
| 2.3. Gathering the Data | 6 |
| 2.3.1. Locators-Web Element Image Templates. | 6 |
| 2.3.2. User Interface Screens | 8 |
| 2.4. Solution Implementation | 10 |
| 2.5. Scaling the solution implementation | 14 |
| 2.6. Challenges | 18 |
| 2.7. Conclusion | 18 |
| 3. Problem 2: Testing User Interfaces at pixel level | 19 |
| 3.1. Understanding the Problem: Visual Testing? What? Why?. | 19 |
| 3.2. Proposed Solution | 20 |
| 3.2.1. Screen Validation through Base Screens Dataset | 20 |
| 3.2.2. Necessity of an Image Classification Model | 20 |
| 3.2.3. Putting everything together | 21 |

| | | |
|----------|---|----|
| 3.3. | Gathering the Data | 21 |
| 3.3.1. | Base Screens | 21 |
| 3.4. | Image Classification Models | 25 |
| 3.4.1. | Requirements: Small size, High accuracy for web pages with noise, performance | 25 |
| 3.4.2. | Architectures | 25 |
| 3.4.2.1. | AlexNet | 25 |
| 3.4.2.2. | LeNet5. | 28 |
| 3.4.2.3. | Custom Model. | 30 |
| 3.4.2.4. | VGG16 | 32 |
| 3.4.2.5. | SqueezeNet | 35 |
| 3.4.3. | Preparing the Data for Model training | 40 |
| 3.4.4. | Training the models | 41 |
| 3.4.4.1. | AlexNet | 41 |
| 3.4.4.2. | LeNet5. | 43 |
| 3.4.4.3. | Custom Model. | 45 |
| 3.4.4.4. | VGG16 | 47 |
| 3.4.4.5. | SqueezeNet | 49 |
| 3.4.5. | Testing the Models | 51 |
| 3.4.5.1. | Test Strategy | 51 |
| 3.4.5.2. | Test Data Preparation | 54 |
| 3.4.6. | Models Comparison Results | 55 |
| 3.4.6.1. | Accuracy Table | 64 |
| 3.4.7. | Choosing a Model | 65 |
| 3.4.7.1. | Improvising Over LeNet5 | 65 |
| 3.4.7.2. | Comparison | 68 |
| 3.5. | Solution Implementation: API using Flask | 71 |
| 3.6. | Challenges | 76 |
| 4. | Deploying the Solution at Scale | 77 |
| 4.1. | Project Architecture | 77 |
| 4.2. | Tools and Technologies | 79 |
| 5. | Literature Survey | 82 |
| 6. | Future Work | 86 |
| 7. | Appendix | 87 |
| 7.1. | Code Snippets | 87 |
| | Bibliography | 97 |

Chapter 1

Introduction

1.1 Software Development Lifecycle: Testing

The Software Development Lifecycle (SDLC) is a process used by software development teams to design, build, test, and deploy high-quality software. It typically consists of several stages, including planning, requirements analysis, design, development, testing, deployment, and maintenance.

Testing is a critical component of the SDLC as it helps ensure that the software is functioning correctly and meets the specified requirements. Testing can help identify defects and issues early in the development process, which can save time and money in the long run by reducing the cost of fixing defects later in the development cycle.

In addition to catching defects early, testing also helps improve the overall quality of the software by ensuring that it meets the customer's expectations and performs as intended. By identifying and addressing issues before release, testing helps prevent costly rework and downtime, which can impact customer satisfaction and harm the reputation of the development team or organization.

In summary, testing is an essential part of the software development lifecycle, helping to ensure that software is of high quality, meets the customer's requirements, and is delivered on time and within budget.

1.2 User Interface Testing

User Interface (UI) testing is a type of software testing that verifies the functionality and performance of the graphical user interface (GUI) of a software application. UI testing involves simulating user interactions with the application's graphical elements and verifying that the application responds as expected. The purpose of UI testing is to ensure that the application's user interface meets the user's expectations, is easy to use, and provides a seamless user experience.

UI testing is important because the graphical user interface is the primary means through which users interact with the software application. A poorly designed user interface can lead to user frustration, reduced user adoption, and ultimately, a decrease in the value of the application. Therefore, UI testing plays a crucial role in ensuring that the user interface of the application is user-friendly, easy to navigate, and provides a positive user experience. UI testing can help identify issues such as layout problems, usability issues, and visual inconsistencies that could negatively impact the user experience. Ultimately, UI testing helps to increase user satisfaction and engagement, leading to a more successful software application.

1.3 User Interface Test Automation

User Interface (UI) test automation is the process of automating tests that validate the functionality and performance of a user interface. UI test automation involves simulating user interactions with a graphical user interface and verifying that the application behaves as expected. UI test automation can be done manually, but it is more efficient and effective when automated using specialized software tools.

UI test automation is essential in software development for several reasons. First, it helps ensure that the user interface works as expected, which is crucial to the success of any software application. It is particularly important in web applications, where the user interface is the primary means of interaction between the user and the application. UI test automation helps detect defects early in the development cycle, which can significantly reduce the cost and time of fixing them.

UI test automation can also save time and effort in testing, as it can perform repetitive tasks quickly and accurately. Automated tests can run faster and more consistently than manual tests, freeing up developers and testers to focus on more complex and creative aspects of testing.

Overall, UI test automation is an important part of software development, as it helps ensure the quality and functionality of the user interface while reducing the time and cost of testing.

1.3.1 Current Automation Process

In UI test automation using Selenium, the script acts on the web elements, and locators are used to define web elements. The locators contain the reference to the target web element within the Document Object Model(DOM). Selenium offers different types of locators, such as ID, Name, Class Name, Tag Name, Link Text, and XPath, which help in identifying web elements on a web page.

In order to automate UI tests using Selenium, testers write test scripts using a programming language such as Java, Python, or C#, which interact with the web page and validate the UI elements as per the expected result. Testers can use a variety of Selenium libraries and tools to create and execute automated UI tests, such as Selenium WebDriver, Selenium IDE, and Selenium Grid.

1.4 Shortcomings in UI Testing

1.4.1 Current Automation process

While Selenium provides a robust framework for UI test automation, it has limitations when it comes to identifying web elements. Changes in the application can cause the locators to become obsolete, leading to test failures. Additionally, manual verification of UI elements can be time-consuming and prone to errors.

There is a need for better approaches to UI test automation that help overcome the limitations of traditional UI test automation and increase the accuracy and efficiency of UI testing.

1.4.2 User Interface Visual Regression Tests

Few shortcomings include:

- **Dependency on manual testing:** The current visual testing methods heavily rely on manual testing, which is time-consuming, error-prone, and often results in inconsistent test results.
- **Inability to detect subtle changes:** Human visual perception can miss subtle changes in UI elements, such as font sizes, color contrast, or margins. These changes can impact the user experience, but they can be challenging to detect manually.
- **Limited scope:** Visual testing methods usually test a limited set of user scenarios and do not provide comprehensive coverage of the UI. This can result in overlooked issues that can impact the user experience.
- **Lack of scalability:** Manual testing can be challenging to scale, especially for large-scale applications. Automated visual testing can help overcome this challenge, but it requires substantial effort and resources to set up and maintain.

1.5 Problem Statements the Dissertation addresses

1.5.1 Locator Strategy in UI Test Automation

Locators in UI test automation can become unreliable because they rely on specific attributes of the UI elements, such as ID or name, to locate them in the Document Object Model (DOM). If the attributes of the UI elements change, such as due to a design update or other changes in the application code, the locators may no longer be valid and the automation scripts will fail. This can result in the need for frequent updates to the automation scripts and can reduce the efficiency and effectiveness of UI test automation.

This project aims to fix this problem through Image Recognition Techniques

1.5.2 Enhancing UI Tests to pixel perfection

Performing rigorous visual testing to achieve pixel-perfect screens presents several challenges, including:

- Time-consuming: Manual visual testing is a time-consuming process, especially for complex applications with multiple screens.
- Human error: Humans are prone to errors, and manual visual testing may lead to overlooking minor visual discrepancies or inconsistencies.
- Limited coverage: Manual visual testing can only cover a limited number of scenarios due to the extensive time required for manual testing.
- Inconsistencies: Manual testing can result in inconsistent test results as different testers may interpret the same visual elements differently.
- Maintenance: Maintaining a large number of visual tests can be challenging and time-consuming, and tests may need to be updated frequently due to changes in the application.

This project aims to propose a machine learning backed automated Visual Testing service that aids in achieving software development with pixel perfect screens.

Chapter 2

Problem 1: Locator Strategy

2.1 Understanding the Problem: How locators work?

In test automation, locators are used to identify and locate specific elements on a webpage. These elements could be buttons, text boxes, drop-down menus, or any other interactive component on the page.

Locators work by providing a reference to the element's location within the Document Object Model (DOM) of the webpage. The DOM is like a tree-like structure of all the elements and their relationships on the webpage.

Test automation tools like Selenium use locators to find and interact with these elements. The most common types of locators are:

- ID: A unique identifier for the element
- Name: A name attribute assigned to the element
- Class name: A class name assigned to the element
- Tag name: The type of element, such as "button" or "input"
- XPath: A path to the element based on its position within the DOM

Locators play a critical role in test automation because they allow the automation tool to interact with the elements on the page. However, locators can become unreliable if the page's HTML changes, such as when new features are added or existing ones are modified. In such cases, the locator may no longer reference the correct element, causing the automation script to fail.

2.2 Proposed Solution: Using Image Template Matching

Using a simple image template matching technique can help overcome challenges faced by locators in test automation by providing an alternative way to identify web elements on a webpage. Rather than relying solely on the HTML structure of the webpage, image recognition can use a screenshot of the webpage and compare it to a template image to identify specific elements. This can be particularly helpful in cases where the structure of the webpage changes frequently or where elements have no identifiable attributes or identifiers.

With image template matching, a template image is created for each unique element that needs to be identified. During the test, the script captures a screenshot of the webpage and compares it to the template images using a matching algorithm. If a match is found, the script can interact with the element in the same way it would with a traditional locator.

Advantages of using Image template matching technique over locators:

- **Robustness:** Image template matching is more robust than locators since it is not affected by changes to the web page structure or layout. This makes it more reliable in detecting web elements.
- **Ease of use:** Image template matching is easy to use and implement since it only requires a reference image of the element to be identified. In contrast, locators require knowledge of the web page structure and attributes of the element to be located.
- **Flexibility:** Image template matching is more flexible than locators since it can be used to identify any type of web element, including dynamic and non-standard ones.
- **Precision:** Image template matching is more precise than locators since it can identify elements at a pixel level, ensuring that the correct element is located.

2.3 Gathering the Data

2.3.1 Locators-Web Element Image Templates(Chrome)

A web element image can be captured through the following steps:

1. Open Google Chrome and navigate to the webpage containing the element to be captured.
2. Right-click on the element select "Inspect" from the context menu.
3. This will open the Chrome Developer Tools panel, with the element selected highlighted in the HTML code.
4. Right-click on the highlighted element and select "Capture node screenshot" from the context menu.
5. This will save the web element image as a PNG file to the default download location.

Note: The steps may vary slightly depending on the version of Chrome being used.

Screens going through the steps: Capture and download 'Enter Password' input webelement from a sample login screen

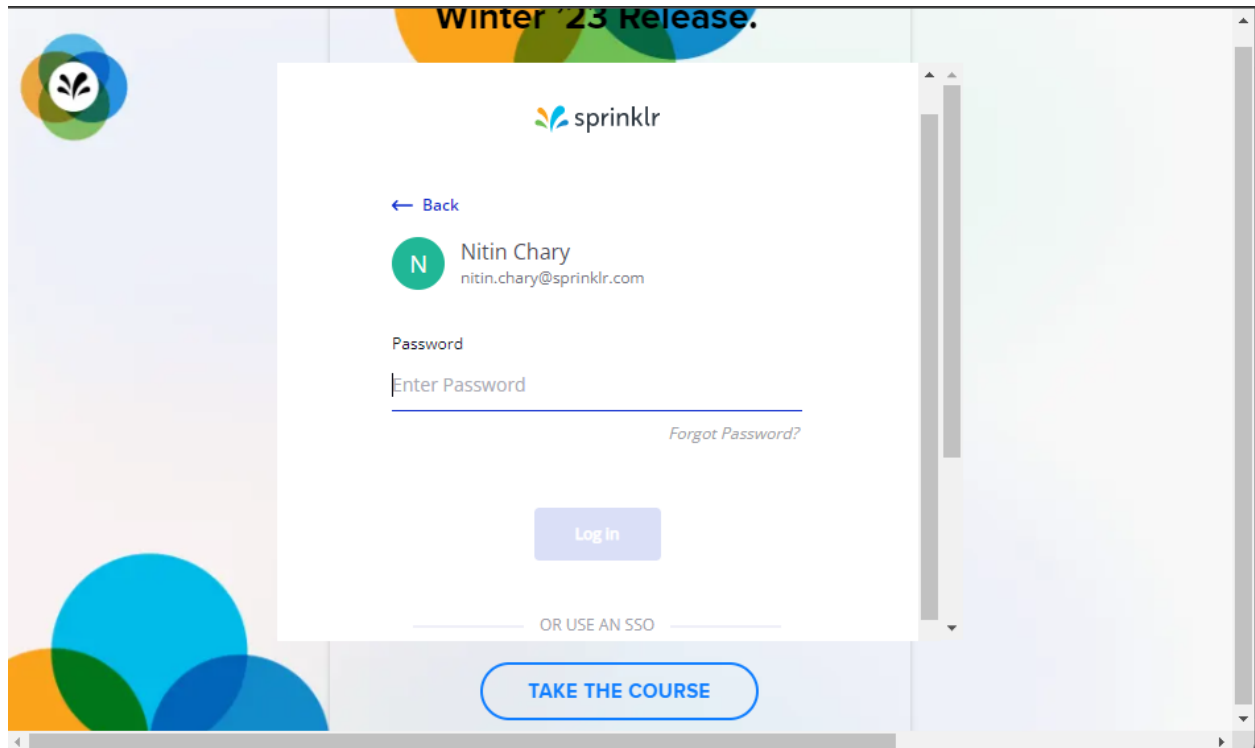


Figure: 2.3.1.f1 Sample Login Screen

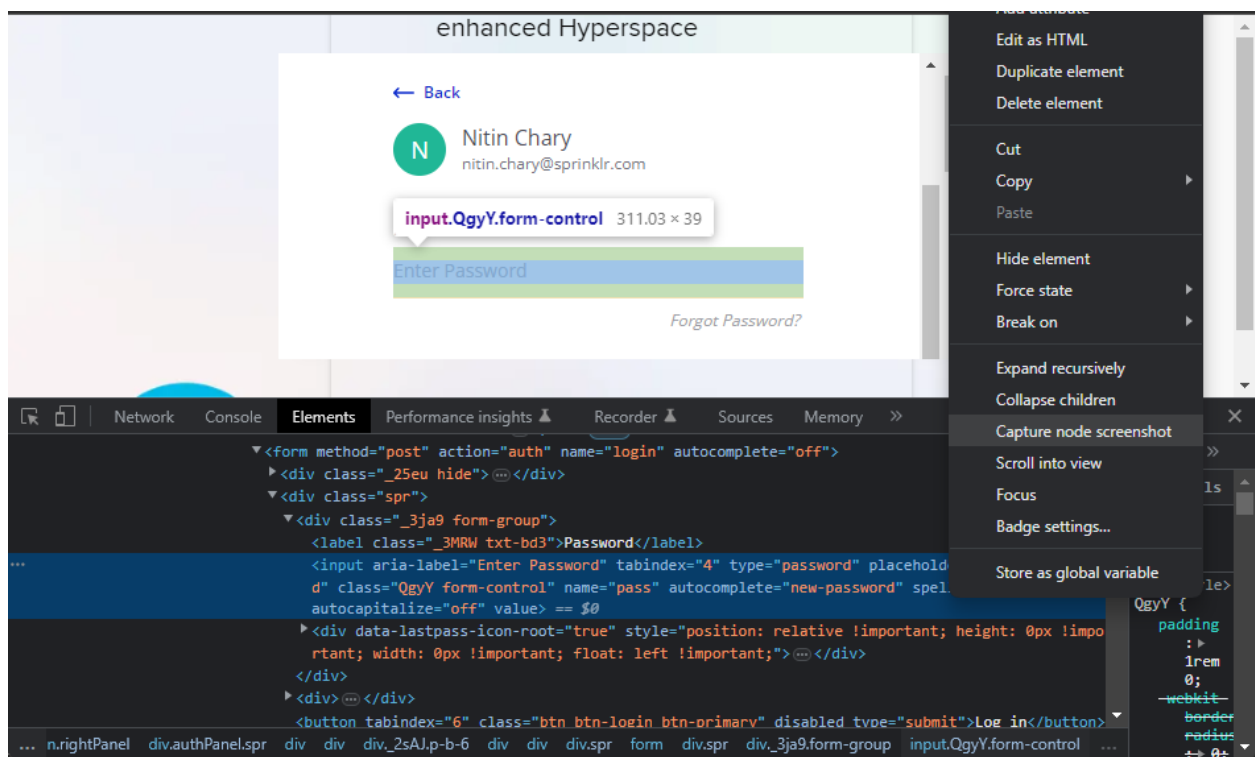


Figure: 2.3.1.f2 Sample Login Screen with Password Field element capture option

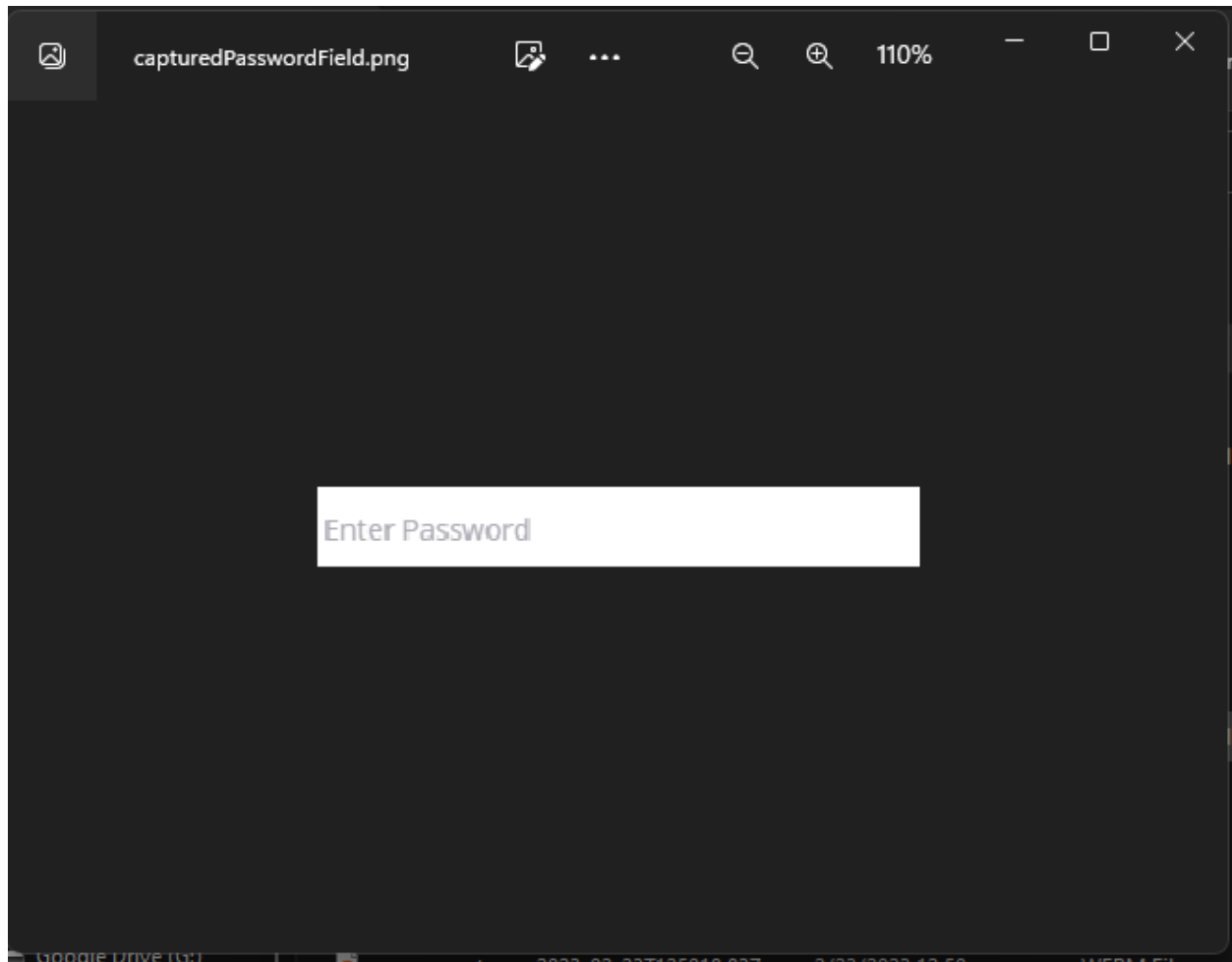


Figure: 2.3.1.f3 Captured Password Field WebElement png file

2.3.2 User Interface Screens

1. Steps to capture the Web page for template matching with the web element image:
Launch a web browser using Selenium WebDriver.
2. Navigate to the desired web page using the `get()` method of the WebDriver instance.
3. Use the `TakesScreenshot` interface of WebDriver to take a screenshot of the entire web page.
4. Save the screenshot as a file using the `FileUtils` class of the Apache Commons IO library.

Sample Java code for the above steps:

```
Java
import org.openqa.selenium.*;
```

```
import org.openqa.selenium.chrome.*;
import org.apache.commons.io.*;

public class WebPageScreenshot {
    public static void main(String[] args) throws Exception {
        // Set the path of the chromedriver.exe file
        System.setProperty("webdriver.chrome.driver",
            "path/to/chromedriver.exe");

        // Launch Chrome browser
        WebDriver driver = new ChromeDriver();

        // Navigate to the web page to be captured
        driver.get("https://www.example.com");

        // Typecast driver object to TakesScreenshot
        TakesScreenshot scrShot = ((TakesScreenshot)driver);

        // Call getScreenshotAs method to create an image file
        File srcFile=scrShot.getScreenshotAs(OutputType.FILE);

        // Save the screenshot as a file
        FileUtils.copyFile(srcFile, new File("path/to/screenshot.png"));

        // Close the browser
        driver.quit();
    }
}
```

Once the screenshot of the web page is captured, an image recognition or template matching technique is used to match the web element images with the screenshot image.

2.4 Solution Implementation

The proposed approach performs web element detection by using template matching, which is a technique used to find the position of a template (a small image) within a larger image (the source image).

The algorithm takes as input two images: the source image, which is the larger image that contains the object we are looking for, and the template image, which is the smaller image that we want to find within the source image. The algorithm then tries to find the template image within the source image by sliding the template image over the source image and computing the similarity between the two images at each position.

To compute the similarity between the two images, the algorithm uses a mathematical function called a correlation coefficient. The correlation coefficient is a measure of how similar two images are. The algorithm tries to find the position in the source image where the correlation coefficient between the template image and the source image is highest. This position is the location of the template image within the source image.

The algorithm uses the OpenCV library to perform template matching. It first preprocesses the template and source images by converting them to grayscale and applying the Canny edge detection algorithm to create an edges-only image. It then resizes the source image to a smaller size if necessary to speed up the template matching process. The algorithm slides the template image over the source image at different scales and computes the correlation coefficient between the two images at each position using the `matchTemplate` function of OpenCV. It then selects the position where the correlation coefficient is highest as the location of the template image within the source image.

Finally, the algorithm returns the coordinates of the center of the bounding box around the template image within the source image.

Pseudocode for the proposed approach:

Unset

1. Define a function named `find_coordinates` that takes four arguments - `keyImage`, `fullImage`, `width`, `height`.
2. Decode the base64 encoded `keyImage` and `fullImage` using the `base64` module.
3. Generate unique names for the decoded `keyImage` and `fullImage`.

4. Write the decoded keyImage and fullImage to disk using the generated names.
5. Convert the keyImage to grayscale and generate edges-only image using Canny-edge detection method.
6. Read the fullImage and resize it if its dimensions are greater than the specified width and height.
7. Create a copy of the fullImage.
8. Convert the copy to grayscale.
9. Set the found variable to None.
10. Loop over a range of scales for the fullImage and resize the image accordingly.
11. Compute the ratio of the resizing and resize the edge-only keyImage accordingly.
12. Compute the edges for the resized fullImage.
13. Compute the match between the keyImage and the edges of the fullImage using the specified method.
14. Compute the maximum match value and location.
15. If found is None or maxVal is greater than found[0], set found to (maxVal, maxLoc, ratio).
16. Compute the top-left and bottom-right corners of the found location.
17. Draw a rectangle around the found location in the copy of the fullImage.
18. Compute the center of the rectangle as the (x,y) coordinate of the found location.
19. Return the (x,y) coordinate.

Code:

Python

```
def find_coordinates(keyImage, fullImage, width, height):
    base64Template = base64.b64decode(payload["keyImage"])
    base64Source = base64.b64decode(payload["fullImage"])
    sourceName = str("source-" + str(timestamp) + str(random.randint(0,
999)) + str(random.randint(0, 999)) + ".png")
    templateName = str("template-" + str(timestamp) +
str(random.randint(0, 999)) + str(random.randint(0, 999)) + ".png")
    logger.info(sourceName)
    logger.info(templateName)
```

```

with open(sourceName, "wb") as file:
    file.write(base64Source)
with open(templateName, "wb") as file:
    file.write(base64Template)
logger.info(payload["dimensions"])
x, y = getElementCoordinates(templateName, sourceName, width,
height),cv2.TM_CCOEFF)
os.remove(sourceName)
os.remove(templateName)
logger.info("Captured location of element is : {0},
{1}".format(str(x), str(y)))
return (x,y)

def getElementCoordinates(template, source, width, height, method):
    timestamp = int(datetime.now().timestamp())
    templateImage = cv2.imread(template)
    templateImage = cv2.cvtColor(templateImage, cv2.COLOR_BGR2GRAY)
    templateImage = cv2.Canny(templateImage, 50, 200) # Create an
edges-only image using Canny-image detection method
    sourceImage = cv2.imread(source)
    logger.debug("Source Shape : " + str(sourceImage.shape))
    if sourceImage.shape[0] > height or sourceImage.shape[1] > width:
        logger.info("Resizing the source image to {0} x
{1}".format(width, height))
        sourceImage = cv2.resize(sourceImage, (width, height),
interpolation=cv2.INTER_LINEAR)
        templateImage = imutils.resize(templateImage,
width=int(templateImage.shape[1] * 0.5))
        img_to_draw = sourceImage.copy()
        grayScaleImage = cv2.cvtColor(img_to_draw, cv2.COLOR_BGR2GRAY)
        found = None
        h, w = templateImage.shape[:2]
        for scale in np.linspace(0.2, 1.0, 20)[::-1]:
            # resize the image according to the scale and keep track
            # of the reatio of the resizing
            resized = imutils.resize(grayScaleImage,
width=int(grayScaleImage.shape[1] * scale))
            ratio = grayScaleImage.shape[1] / float(resized.shape[1])
            logger.info("Ratio : " + str(ratio))

```

```

        # resize till the source image dimensions are greater than
        # the template image dimensions. Pre-requisite of template
matching
        if resized.shape[0] < h or resized.shape[1] < w:
            break
        sourceEdges = cv2.Canny(resized, 50, 200)
        # match the two images
        result = cv2.matchTemplate(sourceEdges, templateImage, method)
        _, maxVal, _, maxLoc = cv2.minMaxLoc(result)
        if found is None or maxVal > found[0]:
            found = (maxVal, maxLoc, ratio)
        (_, maxLoc, r) = found
        (startX, startY) = (int(maxLoc[0] * r), (int(maxLoc[1] * r))
        (endX, endY) = (int((maxLoc[0] + w) * r), (int((maxLoc[1] + h) *
r))
        logger.info("Top left : " + str((startX, startY)))
        logger.info("Bottom right : " + str((endX, endY)))
        locatedTemplate = str(str(source) + str(random.randint(0, 999)) +
".png")
        logger.info(str("identified") + locatedTemplate)
        cv2.rectangle(img_to_draw, (startX, startY), (endX, endY), (0, 0,
255), 5)
        logger.info("Highlighting source with identification")
        cv2.imwrite(str("identified") + str(locatedTemplate), img_to_draw)
        x = (startX + endX) / 2
        y = (startY + endY) / 2
        return x, y

```

Step-by-Step breakdown of Code:

1. The function `find_coordinates` takes in an image of a key element, an image of the full webpage, as well as the width and height of the full webpage.
2. The key element and full webpage images are in base64 format, so they are decoded using `base64.b64decode`.
3. Two new file names are created for the source and template images, with random numbers appended to ensure uniqueness.
4. The decoded images are written to their respective files using `with open(file_name, "wb")` as file.
5. The function `getElementCoordinates` is called with the template image, source image, width, height, and a method for matching the template.

6. In `getElementCoordinates`, the template image is read, converted to grayscale, and edges are detected using the Canny algorithm.
7. The source image is read and resized if it is larger than the specified width and height.
8. A loop is run over a range of scales, from 0.2 to 1.0 with 20 steps, in decreasing order.
9. At each scale, the source image is resized by the ratio of the current scale, and the Canny edge detection algorithm is applied.
10. The template image is matched against the edges of the source image using the specified method.
11. The maximum match value and location are recorded if they are better than the previous maximum.
12. Once the loop is finished, the best match location is used to draw a rectangle around the identified element in the source image.
13. The coordinates of the center of the rectangle are calculated and returned.
14. The source and template images are deleted from the file system.
15. The coordinates of the element are returned by `find_coordinates`.

2.5 Scaling the solution implementation

By implementing the web element detection functionality as an API accessible through the endpoint `service/coordinates`, the code can be used by multiple clients across different platforms and technologies without the need to install the entire codebase locally. The API service accepts inputs in the form of a template image, full image, and the dimensions of the full image. The API service then performs the necessary image processing operations to detect the template in the full image and returns the coordinates of the center of the detected element. These coordinates can then be used by the clients, such as Selenium library, to click or interact with the detected element. By deploying the API service on a server, the clients can access the service remotely, thus enabling the functionality to be scaled and made accessible across the internet.

Code for API service implementation using flask:

Python

```
@app.route("/service/coordinates", methods=["POST"])
def find_coordinates():
    timestamp = int(datetime.now().timestamp())
    logger.debug("Incoming request : " + str(request.json))
    payload = request.get_json()
    if payload["keyImage"] is None or payload["fullImage"] is None:
```

```

        return jsonify({"status": 500, "error": "Expected either a
template or source image"})
    if payload["dimensions"] is None:
        return jsonify({"status": 500, "error": "Dimension of the source
is mandatory"})
    base64Template = base64.b64decode(payload["keyImage"])
    base64Source = base64.b64decode(payload["fullImage"])
    sourceName = str("source-" + str(timestamp) + str(random.randint(0,
999)) + str(random.randint(0, 999)) + ".png")
    templateName = str("template-" + str(timestamp) +
str(random.randint(0, 999)) + str(random.randint(0, 999)) + ".png")
    logger.info(sourceName)
    logger.info(templateName)
    with open(sourceName, "wb") as file:
        file.write(base64Source)
    with open(templateName, "wb") as file:
        file.write(base64Template)
    logger.info(payload["dimensions"])
    x, y = getElementCoordinates(templateName, sourceName,
int(payload["dimensions"]["width"]),int(payload["dimensions"]["height"])
,cv2.TM_CCOEFF)
    os.remove(sourceName)
    os.remove(templateName)
    logger.info("Captured location of element is : {0},
{1}".format(str(x), str(y)))
    return jsonify({"x": x, "y": y})

def getElementCoordinates(template, source, width, height, method):
    timestamp = int(datetime.now().timestamp())
    templateImage = cv2.imread(template)
    templateImage = cv2.cvtColor(templateImage, cv2.COLOR_BGR2GRAY)
    templateImage = cv2.Canny(templateImage, 50, 200) # Create an
edges-only image using Canny-image detection method
    sourceImage = cv2.imread(source)
    logger.debug("Source Shape : " + str(sourceImage.shape))
    if sourceImage.shape[0] > height or sourceImage.shape[1] > width:
        logger.info("Resizing the source image to {0} x
{1}".format(width, height))
        sourceImage = cv2.resize(sourceImage, (width, height),
interpolation=cv2.INTER_LINEAR)

```



```

        templateImage = imutils.resize(templateImage,
width=int(templateImage.shape[1] * 0.5))
        img_to_draw = sourceImage.copy()
        grayScaleImage = cv2.cvtColor(img_to_draw, cv2.COLOR_BGR2GRAY)
        found = None
        h, w = templateImage.shape[:2]
        for scale in np.linspace(0.2, 1.0, 20)[::-1]:
            # resize the image according to the scale and keep track
            # of the reatio of the resizing
            resized = imutils.resize(grayScaleImage,
width=int(grayScaleImage.shape[1] * scale))
            ratio = grayScaleImage.shape[1] / float(resized.shape[1])
            logger.info("Ratio : " + str(ratio))
            # resize till the source image dimensions are greater than
            # the template image dimensions. Pre-requisite of template
matching
            if resized.shape[0] < h or resized.shape[1] < w:
                break
            sourceEdges = cv2.Canny(resized, 50, 200)
            # match the two images
            result = cv2.matchTemplate(sourceEdges, templateImage, method)
            _, maxVal, _, maxLoc = cv2.minMaxLoc(result)
            if found is None or maxVal > found[0]:
                found = (maxVal, maxLoc, ratio)
            (_, maxLoc, r) = found
            (startX, startY) = (int(maxLoc[0] * r)), (int(maxLoc[1] * r))
            (endX, endY) = (int((maxLoc[0] + w) * r)), (int((maxLoc[1] + h) *
r))
            logger.info("Top left : " + str((startX, startY)))
            logger.info("Bottom right : " + str((endX, endY)))
            locatedTemplate = str(str(source) + str(random.randint(0, 999)) +
".png")
            logger.info(str("identified") + locatedTemplate)
            cv2.rectangle(img_to_draw, (startX, startY), (endX, endY), (0, 0,
255), 5)
            logger.info("Highlighting source with identification")
            cv2.imwrite(str("identified") + str(locatedTemplate), img_to_draw)
            x = (startX + endX) / 2
            y = (startY + endY) / 2
            return x, y

```

Java Code to integrate with existing selenium framework:

```
Java
public WebElement getElementFromImage(File keyImage, int height, int
width) {
    String browserImage = ((TakesScreenshot)
webDriver).getScreenshotAs(OutputType.BASE64);
    Map<String, Float> coordinates = new HashMap<>();

    try {
        byte[] keyBytes = FileUtils.readFileToByteArray(keyImage);
        String keyBase64 =
Base64.getEncoder().encodeToString(keyBytes);

        //Send Data to Image Service
        coordinates = //API call to service/coordinates //returning
x and y;
    } catch (IOException e) {
        e.printStackTrace();
    }

    String script = String.format("return
document.elementFromPoint(%s, %s)", coordinates.get("x"),
coordinates.get("y"));
    Object obj = ((JavascriptExecutor)
webDriver).executeScript(script);
    return (WebElement) obj;
}
```

This function is used to get the web element located at a specific coordinate on the web page. It takes in an image file (keyImage), height, and width as input parameters. It first takes a screenshot of the web page using webDriver, and then sends both the keyImage and the screenshot to an image service to get the coordinates of the element. Once the coordinates are obtained, the function executes a Javascript script to get the web element at those coordinates and returns it.

2.6 Challenges

Some of the challenges in template matching approach:

- Sensitivity to changes in UI: Since image template matching relies on pixel-by-pixel comparisons, any changes to the UI (e.g., new buttons, color changes, etc.) can cause the template matching to fail.
- Performance: Image template matching can be computationally expensive, especially when dealing with large images or complex templates.
- Maintenance: Any changes to the UI (e.g., new buttons, color changes, etc.) may require the creation of new templates or the modification of existing ones, which can be time-consuming and error-prone.
- Accuracy: Image template matching can sometimes return false positives or false negatives, which can be problematic in automated testing scenarios.
- Limited applicability: Image template matching may not be suitable for all types of web elements, particularly those that cannot be represented by a static image (e.g., dynamic menus, sliders, etc.).

2.7 Conclusion

Locators are generally more suitable for identifying web elements that have a specific and consistent attribute or property. For example, if a button on a webpage always has the same ID, using the ID locator would be the most suitable and reliable option.

Image template matching is more suitable when dealing with dynamic or changing web elements that cannot be easily identified using traditional locators. For example, if a button on a webpage has a dynamic ID that changes every time the page is loaded, using image template matching to locate the button based on its visual appearance can be more reliable.

Both approaches can be integrated for better test automation. For example, if a web element can be identified using a locator, that should be the first choice. However, if the locator approach fails due to dynamic changes, image template matching can be used as a backup option. Additionally, using both approaches together can increase the reliability and accuracy of test automation, as the combination of both methods can provide more robust and comprehensive element identification.

Chapter 3

Problem 2: Testing User Interfaces at pixel level

3.1 Understanding the Problem: Visual Testing? What? Why?

Visual testing is the process of verifying that the user interface of an application appears correctly to end-users. This type of testing aims to detect any issues in the appearance, layout, and behavior of the application's UI, such as incorrect fonts, colors, alignment, and graphical artifacts.

Visual testing is important because the appearance of an application's UI can have a significant impact on its usability and user experience. A poorly designed or implemented UI can lead to user frustration, decreased productivity, and even a loss of business.

Visual testing can be hard to perform because of the large number of possible UI states that can exist, including different screens, window sizes, fonts, colors, and layouts. Additionally, visual testing is often performed manually, which raises the following issues:

- Human errors: Humans can easily miss visual defects or variations in UI elements, especially when the changes are subtle or occur in a large UI.
- Cognitive biases: Humans can also be influenced by cognitive biases, which can cause them to overlook defects that don't match their expectations or assumptions about how the UI should look.
- Fatigue and boredom: Visual testing requires sustained focus and attention, which can be difficult to maintain for long periods. This can lead to fatigue and boredom, which can impair a tester's ability to detect defects.
- Subjectivity: Different humans can have different opinions on what constitutes a defect or an acceptable UI. This can lead to inconsistencies in testing and make it hard to establish a consistent baseline for testing.
- Scalability: Visual testing can be time-consuming and challenging to scale for large or complex applications. This can make it impractical or impossible to manually test all UI elements.

Automating visual testing can help overcome many of these challenges and improve the efficiency and effectiveness of the testing process.

3.2 Proposed Solution

The proposed solution to automate visual testing involves using machine learning to predict the category of the screen and then comparing the current screen to a base screen dataset under the predicted category. The result would be a difference image that highlights any anomalies in the current screen.

This solution aims to overcome the challenges of visual testing by automating the process and reducing the time and effort required to perform visual testing manually. The use of machine learning enables the system to identify and categorize screens accurately and quickly, and the comparison with the base screen dataset allows for efficient detection of anomalies.

3.2.1 Screen Validation through Base Screens Dataset

The current screen of an application under test is compared to a set of expected screen iterations, called the base dataset. The base dataset is created by gathering design specifications from the design team and capturing screen snapshots from the production environment that are assumed to be valid.

To compare the current screen to the base dataset, a pixel-by-pixel comparison is performed. Any differences found between the two sets of screens are highlighted in the current screen. This highlighting shows regions where there are differences in the expected placement of elements and where new elements are present. This process is used to identify user interface issues such as alignment, layout, missing or new elements on the screen, incorrect colors or fonts, broken images, overlapping elements, incorrect text formatting, and spelling mistakes at the pixel level, bringing attention to subtle differences that might be difficult to spot otherwise.

3.2.2 Necessity of an Image Classification Model

Using an image classification model can help to reduce the amount of manual effort required to validate screens during the visual testing process. The model can classify the current screen into one of several predefined categories, such as login screen, home screen, settings screen, etc. Then, the appropriate base screen dataset can be selected and compared against the current screen.

Using machine learning models to classify screens has several advantages over manual classification. Firstly, it can handle a large number of screens more quickly and accurately than a human tester. Additionally, it can adapt to changes in the application over time, such as new screens or changes to existing screens. Finally, it can reduce the risk of human error, as manual classification can be subjective and inconsistent.

On the other hand, not using a machine learning model can lead to more manual effort required to validate screens, as each screen would need to be manually categorized and compared against the appropriate base screen dataset. This can be time-consuming and error-prone, particularly if the application has a large number of screens. It can also lead to inconsistencies in the visual testing process if different testers categorize screens differently.

3.2.3 Putting everything together

Here are the steps to put the solution at work:

1. Capture the current screen under view for the application being tested.
2. Feed the captured screen image into an image classification model to predict the category of the screen.
3. Use the predicted category to select the corresponding base screens dataset.
4. Compare the current screen image with the base screens dataset at the pixel level.
5. Highlight any differences between the current screen and the expected screens in the base dataset.
6. Identify any UI issues, such as alignment, layout, and spelling errors.
7. Generate a difference image that highlights the areas where the UI issues are present.
8. Output the difference image along with a pass/fail result for the screen validation test.

3.3 Gathering the Data

3.3.1 Base Screens

Screen data can be gathered using a python script.

Example URLs for sample classes:

```
JavaScript
// class names mapped to respective page URLs
references = {

  "challenging_dom": "https://the-internet.herokuapp.com/challenging_dom",
```

```

"disappearing_elements": "https://the-internet.herokuapp.com/disappearing\_elements",

"dynamic_content": "https://the-internet.herokuapp.com/dynamic\_content",

"floating_menu": "https://the-internet.herokuapp.com/floating\_menu",

"infinite_scroll": "https://the-internet.herokuapp.com/infinite\_scroll",

"shifting_content": "https://the-internet.herokuapp.com/shifting\_content/menu?mode=random&pixel\_shift=100",

"typos": "https://the-internet.herokuapp.com/typos"

}

```

Script to capture User Interfaces

```

Python
# class names mapped to respective page URLs
references = {
    "challenging_dom":
"https://the-internet.herokuapp.com/challenging\_dom",
    "disappearing_elements":
"https://the-internet.herokuapp.com/disappearing\_elements",
    "dynamic_content":
"https://the-internet.herokuapp.com/dynamic\_content",
    "floating_menu": "https://the-internet.herokuapp.com/floating\_menu",
    "infinite_scroll":
"https://the-internet.herokuapp.com/infinite\_scroll",
    "shifting_content":
"https://the-internet.herokuapp.com/shifting\_content/menu?mode=random&pixel\_shift=100",
    "typos": "https://the-internet.herokuapp.com/typos"
}

```

```

def openBrowser(driverInner, url):
    driverInner.maximize_window()
    # logger.debug("Opening Browser url " + str(url))
    driverInner.get(url)

def createClasses(reference):
    for key in reference:
        # Directory
        directory = str(key)
        # Parent Directory path
        parent_dir = "G:\My
Drive\Projects\Auto\VisionFramework_UI\malgo\herokuappdemo\classes"
        # Path
        path = os.path.join(parent_dir, directory)
        # Create the directory
        try:
            os.makedirs(path, exist_ok=True)
            print("Directory '%s' created successfully" % directory)
        except OSError as error:
            print("Directory '%s' can not be created" % directory)

for i in range(1, 800):
    driver = webdriver.Chrome()
    for classname in references:
        openBrowser(driver, references[classname])
        time.sleep(1)
        driver.save_screenshot(
            "path/to/project/directory/herokuappdemo/classes/" +
            str(classname)+ "/" + str(classname) + "_" + str(i) + ".png")
    driver.quit()

```

The script opens a web browser using the Selenium webdriver and navigates to different URLs corresponding to different page classes. It then takes a screenshot of each page and saves it to a specific directory for that page class. This process is repeated 800 times. The script also includes a function to create a directory for each page class if it does not already exist. Overall, the script is used to generate a dataset of screenshots for different page classes, which can be used for visual testing and machine learning models

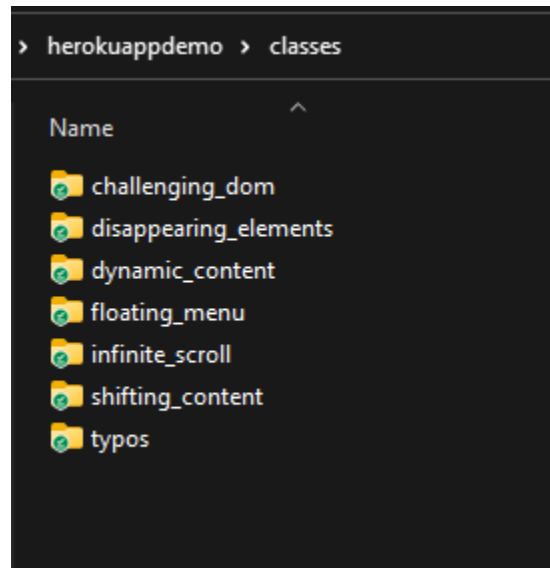
Output Directories:

Figure: 3.3.1.f1 Classes directory containing categories(folders/labels)

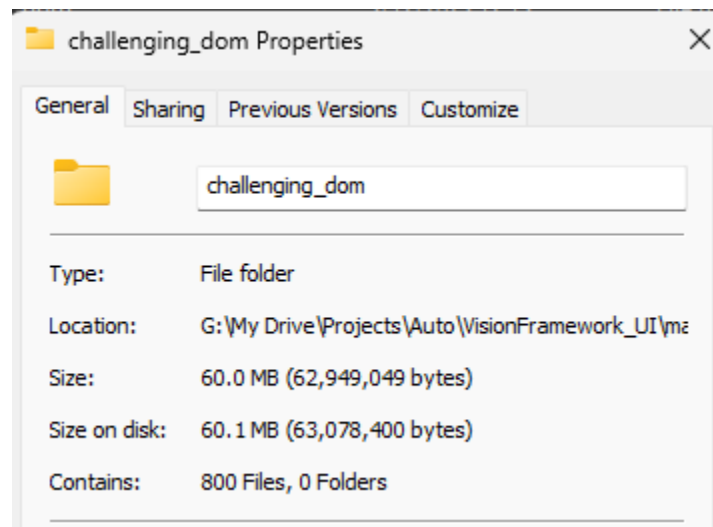


Figure: 3.3.1.f2 Properties of a single category directory

3.4 Image Classification Models

3.4.1 Requirements: Small size, High accuracy, Performance

An Image Classification model should be chosen that satisfies the following requirements:

- The model should be of small size: The model size is an important consideration, especially if we plan to deploy it as an API. A smaller model will take up less space and be faster to load, which will result in faster prediction times. Additionally, a smaller model will require less memory and computational power to run, which can reduce costs and improve scalability.
- The model prediction speed should be fast: Since the model will be deployed as an API, it is important that the prediction speed is fast. This is especially true if the API is being used in a production environment where fast response times are critical. A fast model will enable the API to handle more requests per second, improving its overall performance.
- The model accuracy should be high: The accuracy of the model is critical to ensure that the API provides reliable predictions. A high accuracy model will minimize the number of false positives and false negatives, reducing the chances of errors and improving the user experience. Additionally, high accuracy models are more likely to be adopted by users and stakeholders, which can increase the success and adoption of the API.

3.4.2 Image Classification Model Architectures

3.4.2.1 AlexNet

AlexNet is a deep neural network architecture that was developed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton.

The architecture consists of 8 layers in total, with 5 convolutional layers and 3 fully connected layers. The input to the network is a 227x227 RGB image. The first layer consists of 96 kernels of size 11x11x3, with a stride of 4 and a rectified linear activation function. This is followed by a max-pooling layer of size 3x3 with a stride of 2. The second layer is similar to the first, with 256 kernels of size 5x5x48, and is again followed by a max-pooling layer. The third layer consists of 384 kernels of size 3x3x256, and the fourth and fifth layers consist of 384 and 256 kernels of size 3x3x192 and 3x3x192, respectively.

The output of the fifth convolutional layer is then flattened and passed through two fully connected layers, each with 4096 neurons, followed by a final fully connected layer with 7 neurons (corresponding to the 7 classes in Base Screens class references). The activation function used in the fully connected layers is again rectified linear.

Model Setup in python:

Python

model setup

```
model = keras.models.Sequential([
    keras.layers.Conv2D(filters=96, kernel_size=(11, 11), strides=(4,
4), activation='relu', input_shape=(227, 227, 3)),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3, 3), strides=(2, 2)),
    keras.layers.Conv2D(filters=256, kernel_size=(5, 5), strides=(1, 1),
activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3, 3), strides=(2, 2)),
    keras.layers.Conv2D(filters=384, kernel_size=(3, 3), strides=(1, 1),
activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=384, kernel_size=(3, 3), strides=(1, 1),
activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=256, kernel_size=(3, 3), strides=(1, 1),
activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3, 3), strides=(2, 2)),
    keras.layers.Flatten(),
    keras.layers.Dense(4096, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(4096, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(7, activation='softmax')
])

model.compile(optimizer=tf.optimizers.SGD(learning_rate=0.001),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])

print(model.summary())
```

Model Summary:

Python

Model: "sequential"

| Layer (type) | Output Shape | Param # |
|---|---------------------|---------|
| conv2d (Conv2D) | (None, 55, 55, 96) | 34944 |
| batch_normalization (Batch Normalization) | (None, 55, 55, 96) | 384 |
| max_pooling2d (MaxPooling2D) | (None, 27, 27, 96) | 0 |
| conv2d_1 (Conv2D) | (None, 27, 27, 256) | 614656 |
| batch_normalization_1 (Batch Normalization) | (None, 27, 27, 256) | 1024 |
| max_pooling2d_1 (MaxPooling2D) | (None, 13, 13, 256) | 0 |
| conv2d_2 (Conv2D) | (None, 13, 13, 384) | 885120 |
| batch_normalization_2 (Batch Normalization) | (None, 13, 13, 384) | 1536 |
| conv2d_3 (Conv2D) | (None, 13, 13, 384) | 1327488 |
| batch_normalization_3 (Batch Normalization) | (None, 13, 13, 384) | 1536 |
| conv2d_4 (Conv2D) | (None, 13, 13, 256) | 884992 |
| batch_normalization_4 (Batch Normalization) | (None, 13, 13, 256) | 1024 |
| max_pooling2d_2 (MaxPooling2D) | (None, 6, 6, 256) | 0 |

| | | |
|---------------------|--------------|----------|
| flatten (Flatten) | (None, 9216) | 0 |
| dense (Dense) | (None, 4096) | 37752832 |
| dropout (Dropout) | (None, 4096) | 0 |
| dense_1 (Dense) | (None, 4096) | 16781312 |
| dropout_1 (Dropout) | (None, 4096) | 0 |
| dense_2 (Dense) | (None, 7) | 28679 |

=====

Total params: 58,315,527
 Trainable params: 58,312,775
 Non-trainable params: 2,752

3.4.2.2 LeNet5

LeNet-5 is a convolutional neural network architecture designed for handwritten digit recognition. It consists of seven layers, including three convolutional layers and two subsampling layers, followed by two fully connected layers. The input to the network is a 32x32 grayscale image, which is convolved with a set of filters in the first layer. This is followed by a subsampling layer that reduces the dimensionality of the feature maps. The process of convolution and subsampling is repeated in the second and third layers, and the resulting feature maps are flattened and fed into two fully connected layers for classification. The final layer uses a softmax activation function to produce a probability distribution over the possible classes. LeNet-5 was one of the first successful applications of convolutional neural networks for image classification and is still used as a benchmark for evaluating new architectures.

Model Setup in Python:

```
Python
model = keras.Sequential()
```

```

model.add(layers.Conv2D(filters=6, kernel_size=(3, 3),
activation='relu', input_shape=(img_height, img_width, 1)))
model.add(layers.AveragePooling2D())
model.add(layers.Conv2D(filters=16, kernel_size=(3, 3),
activation='relu'))
model.add(layers.AveragePooling2D())
model.add(layers.Flatten())
model.add(layers.Dense(units=120, activation='relu'))
model.add(layers.Dense(units=84, activation='relu'))
model.add(layers.Dense(units=7, activation='softmax'))

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])

print(model.summary())

```

Model Summary:

Python

Model: "sequential"

| Layer (type) | Output Shape | Param # |
|--|--------------------|---------|
| conv2d (Conv2D) | (None, 30, 30, 6) | 60 |
| average_pooling2d (AveragePooling2D) | (None, 15, 15, 6) | 0 |
| conv2d_1 (Conv2D) | (None, 13, 13, 16) | 880 |
| average_pooling2d_1 (AveragePooling2D) | (None, 6, 6, 16) | 0 |
| flatten (Flatten) | (None, 576) | 0 |
| dense (Dense) | (None, 120) | 69240 |

```

dense_1 (Dense)          (None, 84)          10164

dense_2 (Dense)          (None, 7)           595

=====
Total params: 80,939
Trainable params: 80,939
Non-trainable params: 0
-----

```

3.4.2.3 Custom Model

The model consists of three Convolutional layers, each followed by a MaxPooling layer to downsample the feature maps and reduce the number of parameters. Each Convolutional layer has a different number of filters (16, 32, and 64) and a kernel size of 3x3.

After the last MaxPooling layer, the feature maps are flattened and passed through two Dense layers. The first Dense layer has 128 neurons with ReLU activation and the second Dense layer has a number of neurons equal to the number of classes to be predicted.

The output of the last Dense layer is the final prediction probabilities for each class. Additionally, this particular model architecture is relatively simple and can be trained quickly, making it useful for smaller image classification tasks.

Model Setup in Python

```

Python
# model setup
class_names = ["challenging_dom", "disappearing_elements",
               "dynamic_content", "floating_menu", "infinite_scroll",
               "shifting_content", "typos"]

num_classes = len(class_names)
model = Sequential([
    tf.keras.layers.Conv2D(16, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(32, 3, padding='same', activation='relu'),

```

```

tf.keras.layers.MaxPooling2D(),
tf.keras.layers.Conv2D(64, 3, padding='same', activation='relu'),
tf.keras.layers.MaxPooling2D(),
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(128, activation='relu'),
tf.keras.layers.Dense(num_classes)
])

model.compile(optimizer='adam', loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])

print(model.summary())

```

Model Summary:

Python

Model: "sequential"

| Layer (type) | Output Shape | Param # |
|--------------------------------|----------------------|---------|
| conv2d (Conv2D) | (None, 200, 200, 16) | 448 |
| max_pooling2d (MaxPooling2D) | (None, 100, 100, 16) | 0 |
| conv2d_1 (Conv2D) | (None, 100, 100, 32) | 4640 |
| max_pooling2d_1 (MaxPooling2D) | (None, 50, 50, 32) | 0 |
| conv2d_2 (Conv2D) | (None, 50, 50, 64) | 18496 |
| max_pooling2d_2 (MaxPooling2D) | (None, 25, 25, 64) | 0 |
| flatten (Flatten) | (None, 40000) | 0 |
| dense (Dense) | (None, 128) | 5120128 |


```

dense_1 (Dense)          (None, 7)          903

=====
Total params: 5,144,615
Trainable params: 5,144,615
Non-trainable params: 0
-----

```

3.4.2.4 VGG16

VGG16 is a deep convolutional neural network architecture that was developed for image classification tasks. The name "VGG" stands for the Visual Geometry Group, which is a research group at the University of Oxford that developed the architecture.

VGG16 consists of 16 layers, including 13 convolutional layers and 3 fully connected layers. The convolutional layers have small 3x3 filters and use a stride of 1 pixel, while the pooling layers use 2x2 filters with a stride of 2 pixels. The network takes an input image of size 224x224 and produces an output vector of length 1000, where each element corresponds to a different object category

Model Setup in python:

```

Python
# model setup
model = Sequential()

# Block 1
model.add(Conv2D(64, (3, 3), activation='relu', padding='same',
input_shape=(224, 224, 3)))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2), strides=(2, 2)))

# Block 2
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2), strides=(2, 2)))

```

Block 3

```
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2), strides=(2, 2)))
```

Block 4

```
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2), strides=(2, 2)))
```

Block 5

```
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2), strides=(2, 2)))
```

Flatten the output of the last convolutional layer to feed into a dense layer

```
model.add(Flatten())
```

Add fully connected layers

```
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(7, activation='softmax'))
```

Compile the model with a suitable optimizer and loss function

```
# sgd = SGD(lr=0.001, decay=1e-6, momentum=0.9, nesterov=True)
```

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
```

```
print(model.summary())
```

Model Summary:

Python

Model: "sequential"

| Layer (type) | Output Shape | Param # |
|--------------------------------|-----------------------|---------|
| conv2d (Conv2D) | (None, 224, 224, 64) | 1792 |
| conv2d_1 (Conv2D) | (None, 224, 224, 64) | 36928 |
| max_pooling2d (MaxPooling2D) | (None, 112, 112, 64) | 0 |
| conv2d_2 (Conv2D) | (None, 112, 112, 128) | 73856 |
| conv2d_3 (Conv2D) | (None, 112, 112, 128) | 147584 |
| max_pooling2d_1 (MaxPooling2D) | (None, 56, 56, 128) | 0 |
| conv2d_4 (Conv2D) | (None, 56, 56, 256) | 295168 |
| conv2d_5 (Conv2D) | (None, 56, 56, 256) | 590080 |
| conv2d_6 (Conv2D) | (None, 56, 56, 256) | 590080 |
| max_pooling2d_2 (MaxPooling2D) | (None, 28, 28, 256) | 0 |
| conv2d_7 (Conv2D) | (None, 28, 28, 512) | 1180160 |
| conv2d_8 (Conv2D) | (None, 28, 28, 512) | 2359808 |
| conv2d_9 (Conv2D) | (None, 28, 28, 512) | 2359808 |
| max_pooling2d_3 (MaxPooling2D) | (None, 14, 14, 512) | 0 |
| conv2d_10 (Conv2D) | (None, 14, 14, 512) | 2359808 |

| | | |
|---------------------------------|---------------------|-----------|
| conv2d_11 (Conv2D) | (None, 14, 14, 512) | 2359808 |
| conv2d_12 (Conv2D) | (None, 14, 14, 512) | 2359808 |
| max_pooling2d_4 (MaxPooling 2D) | (None, 7, 7, 512) | 0 |
| flatten (Flatten) | (None, 25088) | 0 |
| dense (Dense) | (None, 4096) | 102764544 |
| dropout (Dropout) | (None, 4096) | 0 |
| dense_1 (Dense) | (None, 4096) | 16781312 |
| dropout_1 (Dropout) | (None, 4096) | 0 |
| dense_2 (Dense) | (None, 7) | 28679 |


```

=====
Total params: 134,289,223
Trainable params: 134,289,223
Non-trainable params: 0
-----

```

3.4.2.5 SqueezeNet

SqueezeNet is a deep neural network architecture for image classification that was designed to have a small memory footprint and to be efficient for mobile and embedded devices. It was introduced in a research paper in 2016 by Iandola et al.

The architecture of SqueezeNet is based on the idea of reducing the number of parameters in the network without sacrificing accuracy. It achieves this by replacing the standard 3x3 filters in convolutional layers with 1x1 filters, which dramatically reduces the number of parameters while maintaining the same receptive field.

SqueezeNet also includes a number of fire modules, which consist of a squeeze layer followed by an expand layer. The squeeze layer performs 1x1 convolutions on the input, while the expand

layer performs a combination of 1x1 and 3x3 convolutions to increase the depth of the output. This design allows for a high degree of feature reuse, which further reduces the number of parameters.

Model Setup in python:

Python

```
def fire_module(x, squeeze, expand):
    y = Conv2D(filters=squeeze, kernel_size=1, activation='relu',
padding='same')(x)
    y = Conv2D(filters=expand, kernel_size=1, activation='relu',
padding='same')(y)
    y = Conv2D(filters=expand, kernel_size=3, activation='relu',
padding='same')(y)
    output = Concatenate()([x, y])
    return output

def SqueezeNet():
    input_shape = (img_height, img_width, 3)
    input_tensor = Input(shape=input_shape)

    x = Conv2D(64, (3, 3), strides=(2, 2), padding='valid',
activation='relu')(input_tensor)
    x = MaxPooling2D(pool_size=(3, 3), strides=(2, 2))(x)

    x = fire_module(x, squeeze=16, expand=64)
    x = fire_module(x, squeeze=16, expand=64)
    x = MaxPooling2D(pool_size=(3, 3), strides=(2, 2))(x)

    x = fire_module(x, squeeze=32, expand=128)
    x = fire_module(x, squeeze=32, expand=128)
    x = MaxPooling2D(pool_size=(3, 3), strides=(2, 2))(x)

    x = fire_module(x, squeeze=48, expand=192)
    x = fire_module(x, squeeze=48, expand=192)
    x = fire_module(x, squeeze=64, expand=256)
    x = fire_module(x, squeeze=64, expand=256)

    x = Dropout(0.5)(x)
```

```

x = Conv2D(num_classes, (1, 1), padding='valid')(x)
x = Activation('softmax')(x)
x = GlobalAveragePooling2D()(x)
x = Dense(7, activation='softmax')(x)

model = Model(inputs=input_tensor, outputs=x)
return model

# Create and compile model
model = SqueezeNet()
model.compile(optimizer=SGD(lr=0.01),
loss=tf.keras.losses.SparseCategoricalCrossentropy(),
metrics=['accuracy'])

print(model.summary())

```

Model Summary:

Python

Model: "model"

```

-----
Layer (type)                 Output Shape              Param #
Connected to
-----
input_1 (InputLayer)         [(None, 224, 224, 3)      0
                                     )]
conv2d (Conv2D)              [(None, 111, 111, 64)     1792
['input_1[0][0]']
                                     )
max_pooling2d (MaxPooling2D) [(None, 55, 55, 64)       0
['conv2d[0][0]']
conv2d_1 (Conv2D)            [(None, 55, 55, 16)       1040
['max_pooling2d[0][0]']

```

```

conv2d_2 (Conv2D)                (None, 55, 55, 64)    1088
['conv2d_1[0][0]']
conv2d_3 (Conv2D)                (None, 55, 55, 64)    36928
['conv2d_2[0][0]']
concatenate (Concatenate)        (None, 55, 55, 128)   0
['max_pooling2d[0][0]',
                                     'conv2d_3[0][0]']
conv2d_4 (Conv2D)                (None, 55, 55, 16)    2064
['concatenate[0][0]']
conv2d_5 (Conv2D)                (None, 55, 55, 64)    1088
['conv2d_4[0][0]']
conv2d_6 (Conv2D)                (None, 55, 55, 64)    36928
['conv2d_5[0][0]']
concatenate_1 (Concatenate)      (None, 55, 55, 192)   0
['concatenate[0][0]',
'conv2d_6[0][0]']
max_pooling2d_1 (MaxPooling2D)   (None, 27, 27, 192)   0
['concatenate_1[0][0]']
conv2d_7 (Conv2D)                (None, 27, 27, 32)    6176
['max_pooling2d_1[0][0]']
conv2d_8 (Conv2D)                (None, 27, 27, 128)   4224
['conv2d_7[0][0]']
conv2d_9 (Conv2D)                (None, 27, 27, 128)   147584
['conv2d_8[0][0]']
concatenate_2 (Concatenate)      (None, 27, 27, 320)   0
['max_pooling2d_1[0][0]',
'conv2d_9[0][0]']
conv2d_10 (Conv2D)              (None, 27, 27, 32)    10272
['concatenate_2[0][0]']
conv2d_11 (Conv2D)              (None, 27, 27, 128)   4224
['conv2d_10[0][0]']
conv2d_12 (Conv2D)              (None, 27, 27, 128)   147584
['conv2d_11[0][0]']
concatenate_3 (Concatenate)      (None, 27, 27, 448)   0
['concatenate_2[0][0]',
'conv2d_12[0][0]']

```

```

max_pooling2d_2 (MaxPooling2D) (None, 13, 13, 448) 0
['concatenate_3[0][0]']
conv2d_13 (Conv2D) (None, 13, 13, 48) 21552
['max_pooling2d_2[0][0]']
conv2d_14 (Conv2D) (None, 13, 13, 192) 9408
['conv2d_13[0][0]']
conv2d_15 (Conv2D) (None, 13, 13, 192) 331968
['conv2d_14[0][0]']
concatenate_4 (Concatenate) (None, 13, 13, 640) 0
['max_pooling2d_2[0][0]',

'conv2d_15[0][0]']
conv2d_16 (Conv2D) (None, 13, 13, 48) 30768
['concatenate_4[0][0]']
conv2d_17 (Conv2D) (None, 13, 13, 192) 9408
['conv2d_16[0][0]']
conv2d_18 (Conv2D) (None, 13, 13, 192) 331968
['conv2d_17[0][0]']
concatenate_5 (Concatenate) (None, 13, 13, 832) 0
['concatenate_4[0][0]',

'conv2d_18[0][0]']
conv2d_19 (Conv2D) (None, 13, 13, 64) 53312
['concatenate_5[0][0]']
conv2d_20 (Conv2D) (None, 13, 13, 256) 16640
['conv2d_19[0][0]']
conv2d_21 (Conv2D) (None, 13, 13, 256) 590080
['conv2d_20[0][0]']
concatenate_6 (Concatenate) (None, 13, 13, 1088) 0
['concatenate_5[0][0]',

)
'conv2d_21[0][0]']
conv2d_22 (Conv2D) (None, 13, 13, 64) 69696
['concatenate_6[0][0]']
conv2d_23 (Conv2D) (None, 13, 13, 256) 16640
['conv2d_22[0][0]']
conv2d_24 (Conv2D) (None, 13, 13, 256) 590080
['conv2d_23[0][0]']
concatenate_7 (Concatenate) (None, 13, 13, 1344) 0
['concatenate_6[0][0]',

```



```

)
'conv2d_24[0][0]'
dropout (Dropout) (None, 13, 13, 1344 0
['concatenate_7[0][0]']
)
conv2d_25 (Conv2D) (None, 13, 13, 7) 9415
['dropout[0][0]']
activation (Activation) (None, 13, 13, 7) 0
['conv2d_25[0][0]']
global_average_pooling2d (Glob (None, 7) 0
['activation[0][0]']
alAveragePooling2D)
dense (Dense) (None, 7) 56
['global_average_pooling2d[0][0]']

```

```

=====

Total params: 2,481,983
Trainable params: 2,481,983

Non-trainable params: 0
-----

```

3.4.3 Preparing the Data for Model training

1. The data for training the models is present in the 'classes' folder.
2. There are seven classes in total: "challenging_dom", "disappearing_elements", "dynamic_content", "floating_menu", "infinite_scroll", "shifting_content", and "typos".
3. Each class has around 800 images.
4. The images are resized as per the architecture of the model.
5. For the LeNet-5 model, the resized input images are converted to grayscale.
6. Each class folder in the 'classes' folder represents a label.

3.4.4 Training the models

3.4.4.1 AlexNet

Code to train AlexNet Model

```
Python
batch_size = 32
img_height = 227
img_width = 227

train_ds = tf.keras.utils.image_dataset_from_directory(
    "classes",
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)

val_ds = tf.keras.utils.image_dataset_from_directory(
    "classes",
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)

epochs = 50
history = model.fit(
    train_ds,
    validation_data=val_ds,
    validation_freq=1,
    epochs=epochs
)

model.save("alexnet_255_800_50epoch.h5")
#
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
```

```

val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()

```

Training Output:

- Alexnet_255_800_50epoch.h5 model file that can be imported into scripts for prediction
- Graphs

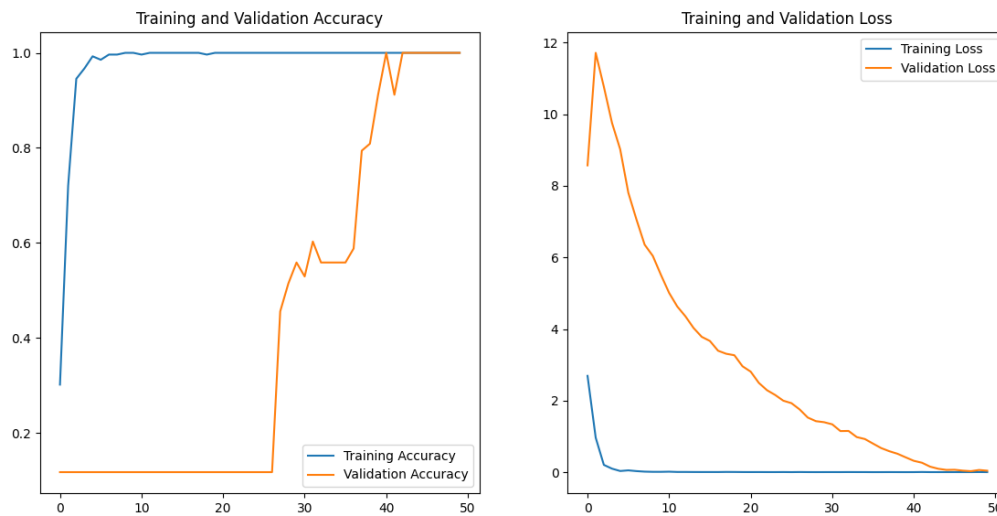


Figure: 3.4.4.1.fl Training and Validation Accuracy/Loss for Alexnet

3.4.4.2 LeNet5

Code to train LeNet5 Model

Python

```
batch_size = 32
img_height = 32
img_width = 32

train_ds = tf.keras.utils.image_dataset_from_directory(
    "grayclasses",
    color_mode="grayscale",
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)

val_ds = tf.keras.utils.image_dataset_from_directory(
    "grayclasses",
    color_mode="grayscale",
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)

epochs = 10
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs
)

model.save("leNet-5_32x32_255_800.h5")
#
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
```

```

val_loss = history.history['val_loss']

epochs_range = range(epochs)
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()

```

Training Output:

- leNet-5_32x32_255_800.h5 model file that can be imported into scripts for prediction
- Graphs

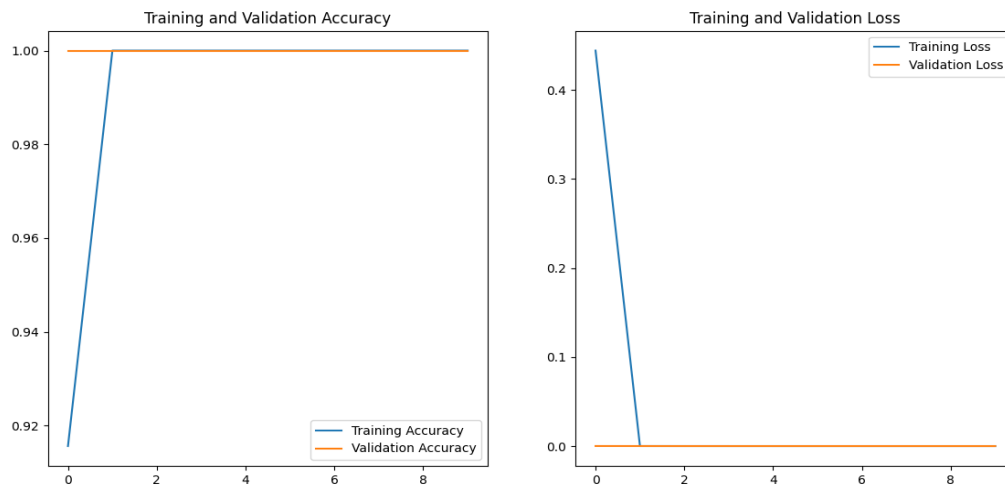


Figure: 3.4.4.2.f1 Training and Validation Accuracy/Loss for LeNet-5_32x32

3.4.4.3 Custom Model

Code to train Custom Model

```
Python
batch_size = 32
img_height = 200
img_width = 200

train_ds = tf.keras.utils.image_dataset_from_directory(
    "classes",
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)

val_ds = tf.keras.utils.image_dataset_from_directory(
    "classes",
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)

epochs = 10
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs
)

model.save("custom_200x200_255_800.h5")

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']
```

```

epochs_range = range(epochs)
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()

```

Training Output:

- custom_200x200_255_800.h5 model file that can be imported into scripts for prediction
- Graphs

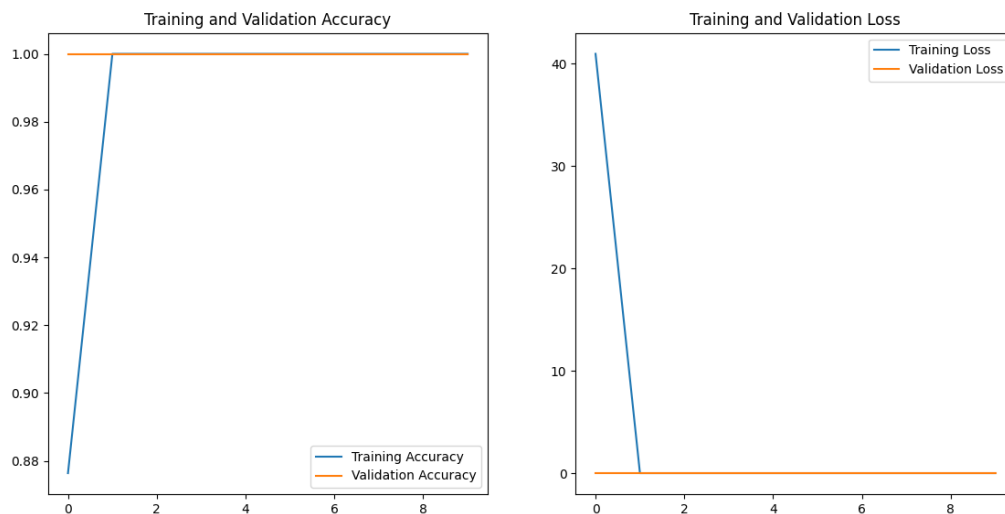


Figure: 3.4.4.3.fl Training and Validation Accuracy/Loss for CustomeModel

3.4.4.4 VGG16

Code to train VGG16 Model

Python

```
batch_size = 32
img_height = 224
img_width = 224

train_ds = tf.keras.utils.image_dataset_from_directory(
    "classes",
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)

val_ds = tf.keras.utils.image_dataset_from_directory(
    "classes",
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)

epochs = 15
history = model.fit(
    train_ds,
    validation_data=val_ds,
    validation_freq=1,
    steps_per_epoch=len(train_ds),
    epochs=epochs,
    validation_steps=len(val_ds)
)

model.save("vgg16_model_255_800samples.h5")

epochs_range = range(epochs)
plt.figure(figsize=(8, 8))
```



```

plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()

```

Training Output:

- vgg16_model_255_800samples.h5 model file that can be imported into scripts for prediction
- Graphs

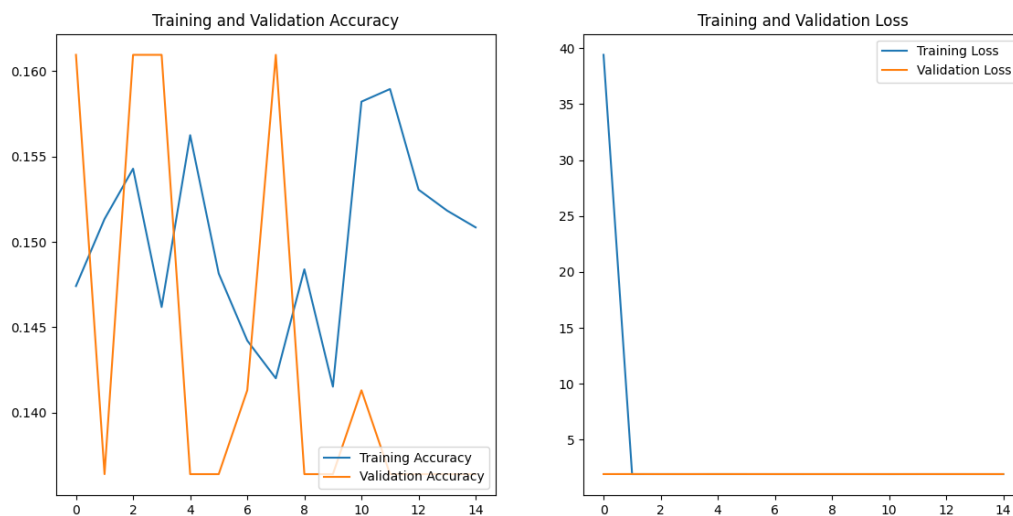


Figure: 3.4.4.4.f1 Training and Validation Accuracy/Loss for VGG16

3.4.4.5 SqueezeNet

Code to train SqueezeNet Model

```
Python
img_height, img_width = 224, 224
num_classes = 7
batch_size = 64
epochs = 20

train_ds = tf.keras.utils.image_dataset_from_directory(
    "classes",
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)

val_ds = tf.keras.utils.image_dataset_from_directory(
    "classes",
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)

history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs
)

model.save("squeezeNet_255_800.h5")
#
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)
```

```
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

Training Output:

- SqueezeNet_255_800.h5 model file that can be imported into scripts for prediction
- Graphs

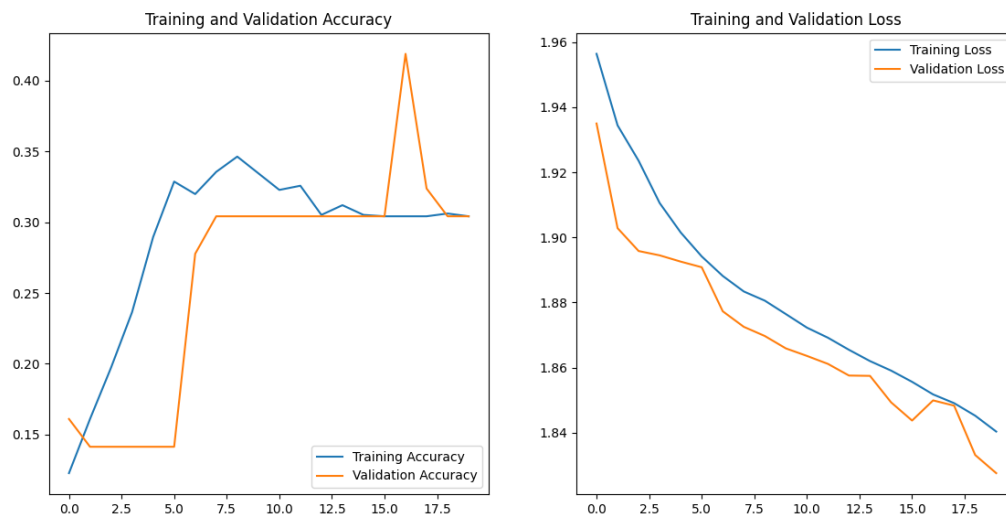


Figure: 3.4.4.5.f1 Training and Validation Accuracy/Loss for SqueezeNet

3.4.5 Testing the Models

3.4.5.1 Test Strategy

1. A set of test images are created with added noise. This is done by applying shapes, lines, characters to a subset of the original images from the 'classes' folder.
2. Test the models on the set of noisy images and record their accuracy scores.
3. Analyze the results and compare the models' accuracy scores. Consider which model performs the best with the added noise.
4. Once a final model has been selected, test it on the separate testing set of noisy images and measure the accuracy score.
5. Repeat steps 2-3 with different levels of noise to ensure the robustness of the model.

A test data set has been created with various noise levels.

Noise levels:

1. testclasses
2. testclasses-highnoise
3. testclasses-veryhighnoise
4. testclasses-ultrahighnoise
5. testclasses-ultrapronoise

Base class Image:

Challenging DOM

The hardest part in automated web testing is finding the best locators (e.g., ones that well named, unique, and unlikely to change). It's more often than not that the application you're testing was not built with this concept in mind. This example demonstrates that with unique IDs, a table with no helpful locators, and a canvas element.

| qux | | | | | | | |
|----------|-----------|-----------|-------------|---------------|-----------|---|--|
| bar | | | | | | | |
| foo | | | | | | | |
| Lorem | Ipsum | Dolor | Sit | Amet | Diceret | Action | |
| luvaret0 | Apeirian0 | Adipisci0 | Definiebas0 | Consequuntur0 | Phaedrum0 | edit delete | |
| luvaret1 | Apeirian1 | Adipisci1 | Definiebas1 | Consequuntur1 | Phaedrum1 | edit delete | |
| luvaret2 | Apeirian2 | Adipisci2 | Definiebas2 | Consequuntur2 | Phaedrum2 | edit delete | |
| luvaret3 | Apeirian3 | Adipisci3 | Definiebas3 | Consequuntur3 | Phaedrum3 | edit delete | |
| luvaret4 | Apeirian4 | Adipisci4 | Definiebas4 | Consequuntur4 | Phaedrum4 | edit delete | |
| luvaret5 | Apeirian5 | Adipisci5 | Definiebas5 | Consequuntur5 | Phaedrum5 | edit delete | |
| luvaret6 | Apeirian6 | Adipisci6 | Definiebas6 | Consequuntur6 | Phaedrum6 | edit delete | |
| luvaret7 | Apeirian7 | Adipisci7 | Definiebas7 | Consequuntur7 | Phaedrum7 | edit delete | |
| luvaret8 | Apeirian8 | Adipisci8 | Definiebas8 | Consequuntur8 | Phaedrum8 | edit delete | |
| luvaret9 | Apeirian9 | Adipisci9 | Definiebas9 | Consequuntur9 | Phaedrum9 | edit delete | |

Figure: 3.4.5.1.f1 Sample Image of a base class in challenging_dom category

Test class Image: level-testclasses

311

Challenging DOM

The hardest part in automated web testing is finding the best locators (e.g., ones that well named, unique, and unlikely to change). It's more often than not that the application you're testing was not built with this concept in mind. This example demonstrates that with unique IDs, a table with no helpful locators, and a canvas element.



| Lorem | Ipsum | Dolor | Sit | Amet | Diceret | Action |
|----------|-----------|-----------|-------------|---------------|-----------|---|
| Iuvaret0 | Apeirian0 | Adipisci0 | Definiebas0 | Consequuntur0 | Phaedrum0 | edit delete |
| Iuvaret1 | Apeirian1 | Adipisci1 | Definiebas1 | Consequuntur1 | Phaedrum1 | edit delete |
| Iuvaret2 | Apeirian2 | Adipisci2 | Definiebas2 | Consequuntur2 | Phaedrum2 | edit delete |
| Iuvaret3 | Apeirian3 | Adipisci3 | Definiebas3 | Consequuntur3 | Phaedrum3 | edit delete |
| Iuvaret4 | Apeirian4 | Adipisci4 | Definiebas4 | Consequuntur4 | Phaedrum4 | edit delete |
| Iuvaret5 | Apeirian5 | Adipisci5 | Definiebas5 | Consequuntur5 | Phaedrum5 | edit delete |
| Iuvaret6 | Apeirian6 | Adipisci6 | Definiebas6 | Consequuntur6 | Phaedrum6 | edit delete |
| Iuvaret7 | Apeirian7 | Adipisci7 | Definiebas7 | Consequuntur7 | Phaedrum7 | edit delete |
| Iuvaret8 | Apeirian8 | Adipisci8 | Definiebas8 | Consequuntur8 | Phaedrum8 | edit delete |
| Iuvaret9 | Apeirian9 | Adipisci9 | Definiebas9 | Consequuntur9 | Phaedrum9 | edit delete |

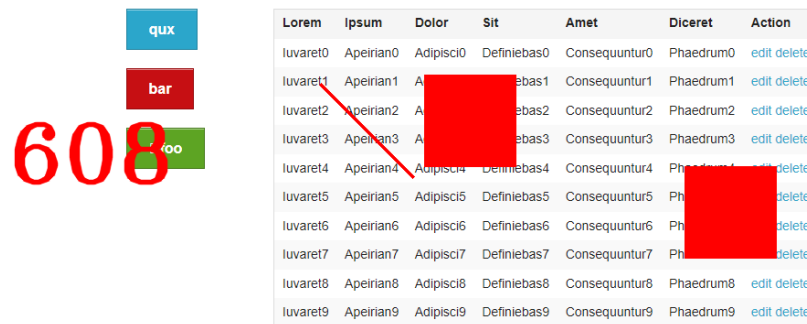
Figure: 3.4.5.1.f2 Sample screen of testclasses

Test class high noise Image: level-testclasses-highnoise

311

Challenging DOM

The hardest part in automated web testing is finding the best locators (e.g., ones that well named, unique, and unlikely to change). It's more often than not that the application you're testing was not built with this concept in mind. This example demonstrates that with unique IDs, a table with no helpful locators, and a canvas element.



| Lorem | Ipsum | Dolor | Sit | Amet | Diceret | Action |
|----------|-----------|-----------|-------------|---------------|-----------|---|
| Iuvaret0 | Apeirian0 | Adipisci0 | Definiebas0 | Consequuntur0 | Phaedrum0 | edit delete |
| Iuvaret1 | Apeirian1 | Adipisci1 | Definiebas1 | Consequuntur1 | Phaedrum1 | edit delete |
| Iuvaret2 | Apeirian2 | Adipisci2 | Definiebas2 | Consequuntur2 | Phaedrum2 | edit delete |
| Iuvaret3 | Apeirian3 | Adipisci3 | Definiebas3 | Consequuntur3 | Phaedrum3 | edit delete |
| Iuvaret4 | Apeirian4 | Adipisci4 | Definiebas4 | Consequuntur4 | Phaedrum4 | edit delete |
| Iuvaret5 | Apeirian5 | Adipisci5 | Definiebas5 | Consequuntur5 | Phaedrum5 | edit delete |
| Iuvaret6 | Apeirian6 | Adipisci6 | Definiebas6 | Consequuntur6 | Phaedrum6 | edit delete |
| Iuvaret7 | Apeirian7 | Adipisci7 | Definiebas7 | Consequuntur7 | Phaedrum7 | edit delete |
| Iuvaret8 | Apeirian8 | Adipisci8 | Definiebas8 | Consequuntur8 | Phaedrum8 | edit delete |
| Iuvaret9 | Apeirian9 | Adipisci9 | Definiebas9 | Consequuntur9 | Phaedrum9 | edit delete |

608

Figure: 3.4.5.1.f3 Sample screen of highnoise level

Test class very high noise: level-testclasses-veryhighnoise

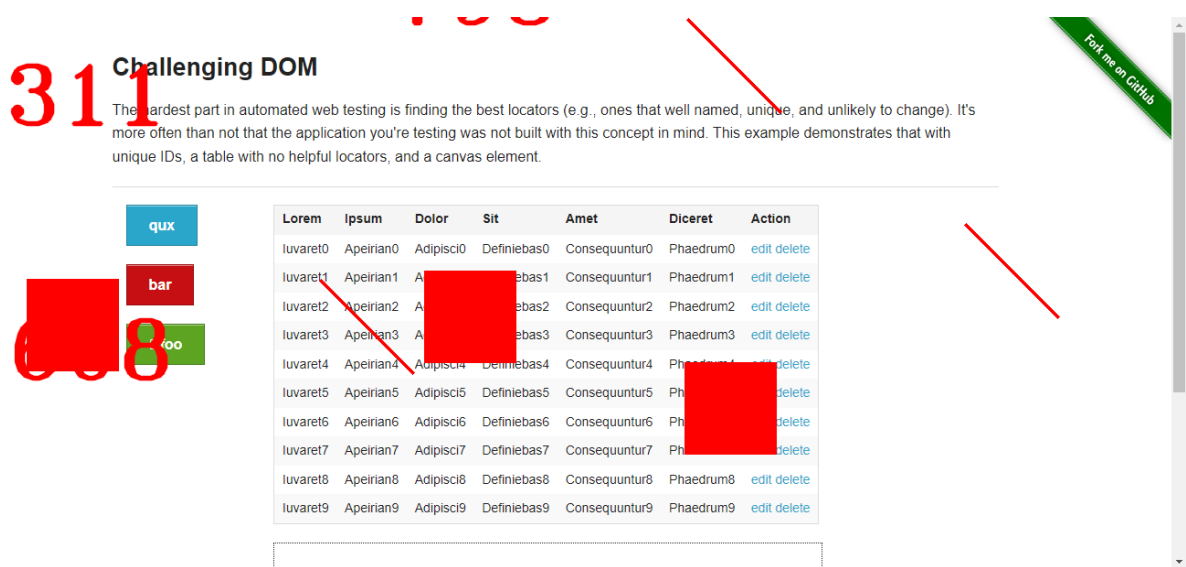


Figure: 3.4.5.1.f4 Sample screen of veryhighnoise level

Test class ultra high noise: level-testclasses-ultrahighnoise

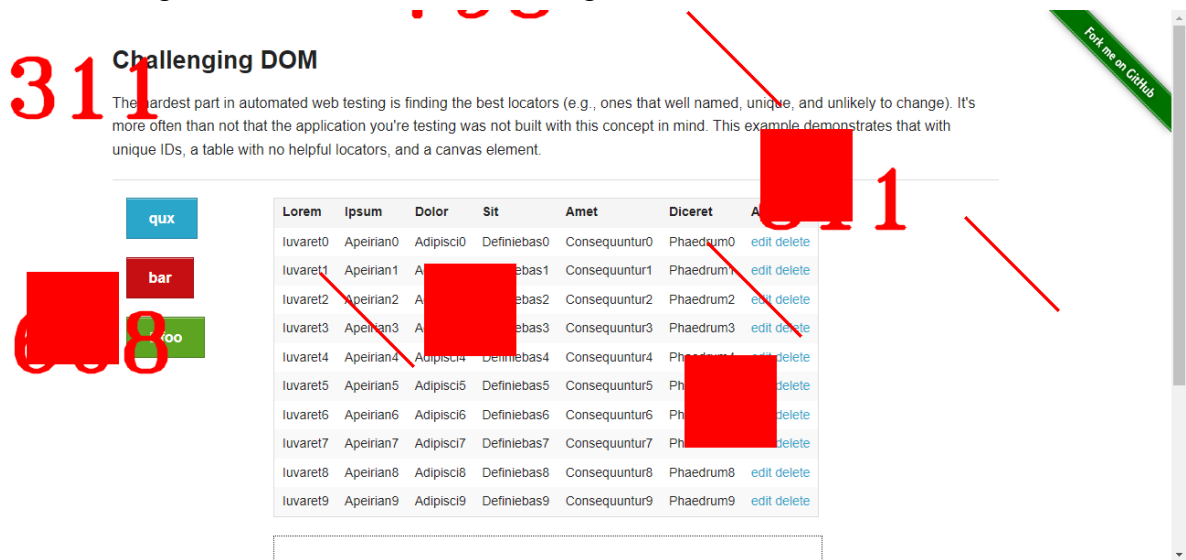


Figure: 3.4.5.1.f5 Sample screen of ultrahighnoise level

Test class ultrapro noise: level-testclasses-ultrapronoise

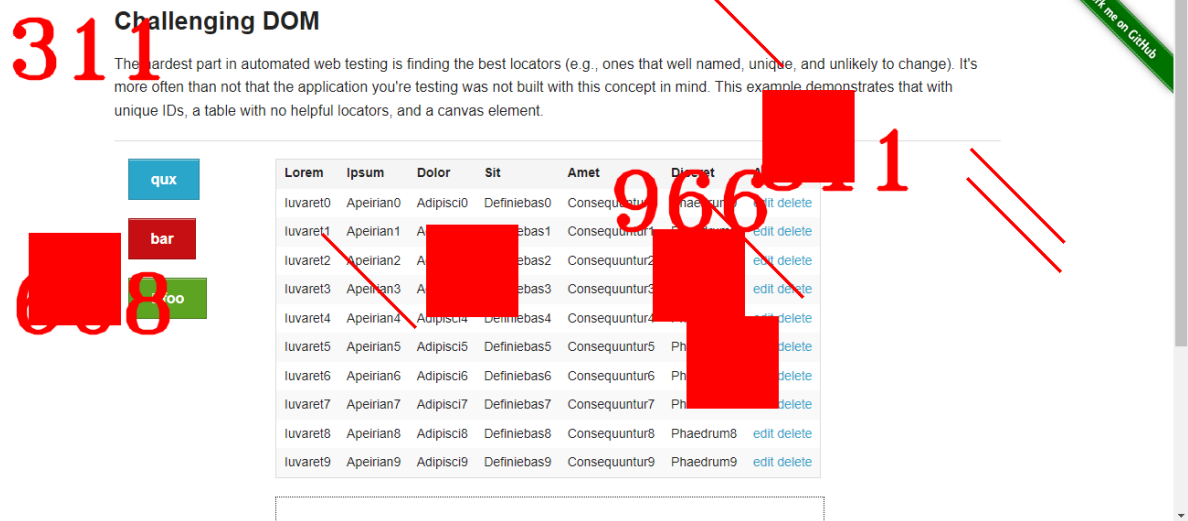


Figure: 3.4.5.1.f6 Sample screen of ultrapronoiselevel

3.4.5.2 Test Data Preparation

Python script to add noise to images:

Python

```
class_names = ["challenging_dom", "disappearing_elements",
               "dynamic_content", "floating_menu", "infinite_scroll",
               "shifting_content", "typos"]

for classer in class_names:
    input_dir = str("classes/"+str(classer))
    output_dir = str("testclasses-noise/"+str(classer))
    # Loop over the images in the input directory
    for filename in os.listdir(input_dir):
        if filename.endswith(".jpg") or filename.endswith(".png"):
            # Read the image
            image = cv2.imread(os.path.join(input_dir, filename))
            # Add rectangular block
            x = random.randint(0,1200)
            y = random.randint(0,500)
```

```

        xend = x+100
        yend = y+100
        cv2.rectangle(image, (x, y), (xend, yend), (0, 0, 255),
thickness=cv2.FILLED) # thickness= -1

        # add line
        x = random.randint(0, 1200)
        y = random.randint(0, 500)
        xend = x + 100
        yend = y + 100
        cv2.line(image, (x, y), (xend, yend), (0, 0, 255),
thickness=3)

        # add random text
        x = random.randint(0, 1000)
        y = random.randint(0, 400)
        cv2.putText(image, str(random.randint(0, 1200)), (x, y),
cv2.FONT_HERSHEY_TRIPLEX, 3.0, (0,0,255), 3, cv2.LINE_AA)
        noisy_image = image
        # Save the image
        cv2.imwrite(os.path.join(output_dir, filename), noisy_image)
        print(os.path.join(output_dir, filename))
    else:
        continue

```

Running the above script on base classes will result in testclasses. Using the testclasses as input will result in testclasses-highnoise. A resultant noisy image data set is used to create a higher level of noisy dataset

3.4.6 Models Comparison Results

Script to validate model performance with test classes:

```

Python
def evaluateModel_leNet5(directory):
    test_ds = tf.keras.utils.image_dataset_from_directory(
        directory,

```



```

        image_size=(32, 32),
        color_mode="grayscale",
        batch_size=32)
model = tf.keras.models.load_model('leNet-5_255_800.h5')
print(model.evaluate(test_ds))

def evaluateModel_alexNet(directory):
    test_ds = tf.keras.utils.image_dataset_from_directory(
        directory,
        image_size=(227, 227),
        batch_size=32)
    model = tf.keras.models.load_model('Alexnet_255_800_50epoch.h5')
    print(model.evaluate(test_ds))

def evaluateModel_squeezeNet(directory):
    test_ds = tf.keras.utils.image_dataset_from_directory(
        directory,
        image_size=(224, 224),
        batch_size=32)
    model = tf.keras.models.load_model('squeezeNet_255_800.h5')
    print(model.evaluate(test_ds))

def evaluateModel_customModel(directory):
    test_ds = tf.keras.utils.image_dataset_from_directory(
        directory,
        image_size=(200, 200),
        batch_size=32)
    model = tf.keras.models.load_model('custom_255_800.h5')
    print(model.evaluate(test_ds))

```

AlexNet Evaluation:

Python

TEST LEVEL: testclasses

```
AlexNet Model
Found 700 files belonging to 7 classes.
22/22 [=====] - 8s 335ms/step - loss: 45.2726 -
accuracy: 0.1800
Accuracy: 18.000000715255737
Loss: 45.27260971069336
```

```
*****
TEST LEVEL: testclasses-highnoise
```

```
AlexNet Model
Found 700 files belonging to 7 classes.
22/22 [=====] - 9s 341ms/step - loss: 70.5321 -
accuracy: 0.1886
Accuracy: 18.857142329216003
Loss: 70.5320816040039
```

```
*****
TEST LEVEL: testclasses-veryhighnoise
```

```
AlexNet Model
Found 700 files belonging to 7 classes.
22/22 [=====] - 9s 347ms/step - loss: 95.9190 -
accuracy: 0.1757
Accuracy: 17.571428418159485
Loss: 95.91903686523438
```

```
*****
TEST LEVEL: testclasses-ultrahighnoise
```

```
AlexNet Model
Found 700 files belonging to 7 classes.
22/22 [=====] - 9s 359ms/step - loss: 118.7585
- accuracy: 0.1643
Accuracy: 16.428571939468384
Loss: 118.75851440429688
```

```
*****
TEST LEVEL: testclasses-ultrapronoise
```

```

AlexNet Model
Found 700 files belonging to 7 classes.
22/22 [=====] - 9s 370ms/step - loss: 137.9092
- accuracy: 0.1671
Accuracy: 16.7142853140831
Loss: 137.90919494628906

```

LeNet-5 Evaluation:

```

Python
*****
TEST LEVEL: testclasses

leNet-5 32x32 Model
Found 700 files belonging to 7 classes.
22/22 [=====] - 2s 43ms/step - loss: 0.5829 -
accuracy: 0.8971
Accuracy: 89.71428275108337
Loss: 0.5829128623008728

*****
TEST LEVEL: testclasses-highnoise

leNet-5 32x32 Model
Found 700 files belonging to 7 classes.
22/22 [=====] - 2s 47ms/step - loss: 1.5962 -
accuracy: 0.7814
Accuracy: 78.1428575515747
Loss: 1.5962468385696411

*****
TEST LEVEL: testclasses-veryhighnoise

leNet-5 32x32 Model
Found 700 files belonging to 7 classes.

```

```
22/22 [=====] - 2s 44ms/step - loss: 2.8082 -
accuracy: 0.6743
Accuracy: 67.42857098579407
Loss: 2.808243751525879
```

```
*****
TEST LEVEL: testclasses-ultrahighnoise
```

```
leNet-5 32x32 Model
Found 700 files belonging to 7 classes.
22/22 [=====] - 2s 44ms/step - loss: 3.9500 -
accuracy: 0.5900
Accuracy: 58.99999737739563
Loss: 3.950012445449829
```

```
*****
TEST LEVEL: testclasses-ultrapronoise
```

```
leNet-5 32x32 Model
Found 700 files belonging to 7 classes.
22/22 [=====] - 2s 46ms/step - loss: 5.0065 -
accuracy: 0.5414
Accuracy: 54.14285659790039
Loss: 5.006483554840088
```

```
Process finished with exit code 0
```

CustomModel Evaluation:

```
Python
*****
TEST LEVEL: testclasses

CustomModel
Found 700 files belonging to 7 classes.
22/22 [=====] - 4s 143ms/step - loss: 1.3182 -
accuracy: 0.8400
```

Accuracy: 83.99999737739563

Loss: 1.3181616067886353

TEST LEVEL: testclasses-highnoise

CustomModel

Found 700 files belonging to 7 classes.

22/22 [=====] - 4s 146ms/step - loss: 4.9889 -

accuracy: 0.5871

Accuracy: 58.71428847312927

Loss: 4.988931179046631

TEST LEVEL: testclasses-veryhighnoise

CustomModel

Found 700 files belonging to 7 classes.

22/22 [=====] - 4s 151ms/step - loss: 9.7491 -

accuracy: 0.4314

Accuracy: 43.142858147621155

Loss: 9.749069213867188

TEST LEVEL: testclasses-ultrahighnoise

CustomModel

Found 700 files belonging to 7 classes.

22/22 [=====] - 4s 149ms/step - loss: 14.8249 -

accuracy: 0.3557

Accuracy: 35.57142913341522

Loss: 14.824920654296875

TEST LEVEL: testclasses-ultrapronoise

```
CustomModel
Found 700 files belonging to 7 classes.
22/22 [=====] - 4s 163ms/step - loss: 19.5064 -
accuracy: 0.3029
Accuracy: 30.28571307659149
Loss: 19.506423950195312

Process finished with exit code 0
```

VGG16 Evaluation:

```
Python
*****
TEST LEVEL: testclasses

VGG16 Model
Found 700 files belonging to 7 classes.

22/22 [=====] - 87s 4s/step - loss: 1.9749 -
accuracy: 0.1429
Accuracy: 14.28571492433548
Loss: 1.9748685359954834

*****
TEST LEVEL: testclasses-highnoise

VGG16 Model
Found 700 files belonging to 7 classes.
22/22 [=====] - 84s 4s/step - loss: 1.9749 -
accuracy: 0.1429
Accuracy: 14.28571492433548
Loss: 1.9748682975769043

*****
TEST LEVEL: testclasses-veryhighnoise

VGG16 Model
Found 700 files belonging to 7 classes.
```

```
22/22 [=====] - 91s 4s/step - loss: 1.9749 -
accuracy: 0.1429
```

```
Accuracy: 14.28571492433548
```

```
Loss: 1.9748685359954834
```

```
*****
```

```
TEST LEVEL: testclasses-ultrahighnoise
```

```
VGG16 Model
```

```
Found 700 files belonging to 7 classes.
```

```
22/22 [=====] - 99s 4s/step - loss: 1.9749 -
accuracy: 0.1429
```

```
Accuracy: 14.28571492433548
```

```
Loss: 1.9748685359954834
```

```
*****
```

```
TEST LEVEL: testclasses-ultrapronoise
```

```
VGG16 Model
```

```
Found 700 files belonging to 7 classes.
```

```
22/22 [=====] - 119s 5s/step - loss: 1.9749 -
accuracy: 0.1429
```

```
Accuracy: 14.28571492433548
```

```
Loss: 1.9748685359954834
```

```
Process finished with exit code 0
```

SqueezeNet Evaluation:

```
Python
```

```
*****
```

```
TEST LEVEL: testclasses
```

```
SqueezeNet Model
```

```
Found 700 files belonging to 7 classes.
```

```
22/22 [=====] - 9s 377ms/step - loss: 1.8855 -
accuracy: 0.3057
```

```
Accuracy: 30.571427941322327
```

```
Loss: 1.8855056762695312
```

TEST LEVEL: testclasses-highnoise

SqueezeNet Model

Found 700 files belonging to 7 classes.

22/22 [=====] - 9s 382ms/step - loss: 1.9041 -
accuracy: 0.2400

Accuracy: 23.999999463558197

Loss: 1.9041235446929932

TEST LEVEL: testclasses-veryhighnoise

SqueezeNet Model

Found 700 files belonging to 7 classes.

22/22 [=====] - 10s 387ms/step - loss: 1.9206 -
accuracy: 0.1529

Accuracy: 15.285713970661163

Loss: 1.9206159114837646

TEST LEVEL: testclasses-ultrahighnoise

SqueezeNet Model

Found 700 files belonging to 7 classes.

22/22 [=====] - 10s 393ms/step - loss: 1.9350 -
accuracy: 0.1429

Accuracy: 14.28571492433548

Loss: 1.935038685798645

TEST LEVEL: testclasses-ultrapronoise

SqueezeNet Model

Found 700 files belonging to 7 classes.

22/22 [=====] - 11s 440ms/step - loss: 1.9478 -
accuracy: 0.1429

Accuracy: 14.28571492433548

Loss: 1.9478093385696411

Process finished with exit code 0

3.4.6.1 Accuracy Table

| | <i>testclasses</i> | <i>highnoise</i> | <i>veryhighnoise</i> | <i>ultrahighnoise</i> | <i>ultrapronoise</i> |
|---------------------|--------------------|------------------|----------------------|-----------------------|----------------------|
| AlexNet | 18 | 18.8 | 17.5 | 16.4 | 16.7 |
| LeNet5_32x32 | 89.7 | 78.1 | 67.4 | 58.9 | 54.1 |
| CustomModel | 83.9 | 58.7 | 43.1 | 35.5 | 30.2 |
| VGG16 | 14.2 | 14.2 | 14.2 | 14.2 | 14.2 |
| SqueezeNet | 30.5 | 23.9 | 15.2 | 14.2 | 14.2 |

Table 3.4.6.1.t1: Model accuracy at various noise levels

Graph representing Table information:

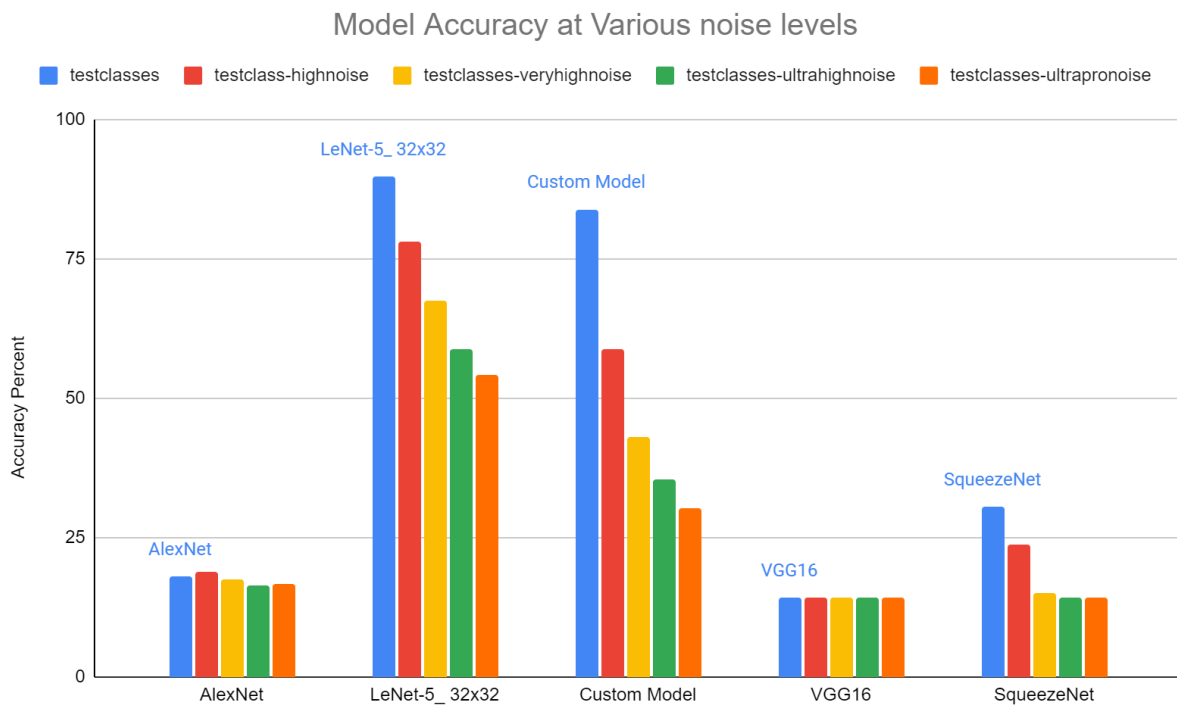


Figure: 3.4.6.1.f1 Model Accuracy at various noise levels

The data shows the performance of five different models, namely AlexNet, LeNet5_32x32, CustomModel, VGG16, and SqueezeNet, on four different levels of noise added to the test images. The noise levels are categorized as highnoise, veryhighnoise, ultrahighnoise, and ultraprnoise. The performance is measured in terms of accuracy, with higher accuracy indicating better performance.

The LeNet5_32x32 model performs the best across all noise levels, followed by CustomModel. VGG16, AlexNet, and SqueezeNet have comparable performance, with SqueezeNet performing slightly better. As expected, the accuracy of all models decreases as the noise level increases, with the LeNet5_32x32 model showing the most resilience to noise

3.4.7 Choosing a Model

3.4.7.1 Improvising Over LeNet5

The LeNet5_32x32 model has shown the highest accuracy among the tested models. In order to further enhance its performance, the hyperparameters of the LeNet5 architecture have been adjusted and trained on larger image sizes of 100x100 and 180x180.

Following are the outputs for improvised model Tests against the noisy datasets.

LeNet-5_100x100:

```
Python
*****
TEST LEVEL: testclasses
leNet5_100x100
Found 700 files belonging to 7 classes.
22/22 [=====] - 2s 59ms/step - loss: 0.0014 -
accuracy: 1.0000
Accuracy: 100.0
Loss: 0.0013804332120344043

*****
TEST LEVEL: testclasses-highnoise
leNet5_100x100
Found 700 files belonging to 7 classes.
22/22 [=====] - 2s 57ms/step - loss: 0.0871 -
accuracy: 0.9757
```

```

Accuracy: 97.57142663002014
Loss: 0.08714433014392853

*****
TEST LEVEL: testclasses-veryhighnoise
leNet5_100x100
Found 700 files belonging to 7 classes.
22/22 [=====] - 2s 63ms/step - loss: 0.5495 -
accuracy: 0.8786
Accuracy: 87.85714507102966
Loss: 0.5494567155838013

*****
TEST LEVEL: testclasses-ultrahighnoise
leNet5_100x100
Found 700 files belonging to 7 classes.
22/22 [=====] - 2s 66ms/step - loss: 1.4754 -
accuracy: 0.7671
Accuracy: 76.71428322792053
Loss: 1.4754234552383423

*****
TEST LEVEL: testclasses-ultrapronoise
leNet5_100x100
Found 700 files belonging to 7 classes.
22/22 [=====] - 2s 62ms/step - loss: 2.6584 -
accuracy: 0.6786
Accuracy: 67.85714030265808
Loss: 2.658419609069824

Process finished with exit code 0

```

LeNet-5_180x180:

```

Python
*****
TEST LEVEL: testclasses
leNet5_180x180
Found 700 files belonging to 7 classes.

```

```

22/22 [=====] - 3s 81ms/step - loss: 1.9369e-05
- accuracy: 1.0000
Accuracy: 100.0
Loss: 1.9369366782484576e-05
*****
TEST LEVEL: testclasses-highnoise
leNet5_180x180
Found 700 files belonging to 7 classes.
22/22 [=====] - 3s 82ms/step - loss: 5.4889e-04
- accuracy: 1.0000
Accuracy: 100.0
Loss: 0.0005488891038112342

*****
TEST LEVEL: testclasses-veryhighnoise
leNet5_180x180
Found 700 files belonging to 7 classes.
22/22 [=====] - 3s 81ms/step - loss: 0.0110 -
accuracy: 0.9971
Accuracy: 99.71428513526917
Loss: 0.010999184101819992

*****
TEST LEVEL: testclasses-ultrahighnoise
leNet5_180x180
Found 700 files belonging to 7 classes.
22/22 [=====] - 3s 85ms/step - loss: 0.0568 -
accuracy: 0.9800
Accuracy: 98.00000190734863
Loss: 0.05680794268846512

*****
TEST LEVEL: testclasses-ultrapronoise
leNet5_180x180
Found 700 files belonging to 7 classes.
22/22 [=====] - 3s 86ms/step - loss: 0.1866 -
accuracy: 0.9571
Accuracy: 95.71428298950195
Loss: 0.1866360455751419
Process finished with exit code 0

```

3.4.7.2 Comparison

Accuracy Comparison

| | <i>testclasses</i> | <i>highnoise</i> | <i>veryhighnoise</i> | <i>ultrahighnoise</i> | <i>ultrapronoise</i> |
|------------------------|--------------------|------------------|----------------------|-----------------------|----------------------|
| LeNet-5_32x32 | 89.7 | 78.1 | 67.4 | 58.9 | 54.1 |
| LeNet-5_100x100 | 100 | 97.5 | 87.8 | 76.7 | 67.8 |
| LeNet-5_180x180 | 100 | 100 | 99.7 | 98 | 95.7 |

Table 3.4.7.2.t1: Tuned LeNet-5 Accuracy at various noise levels

Graph:

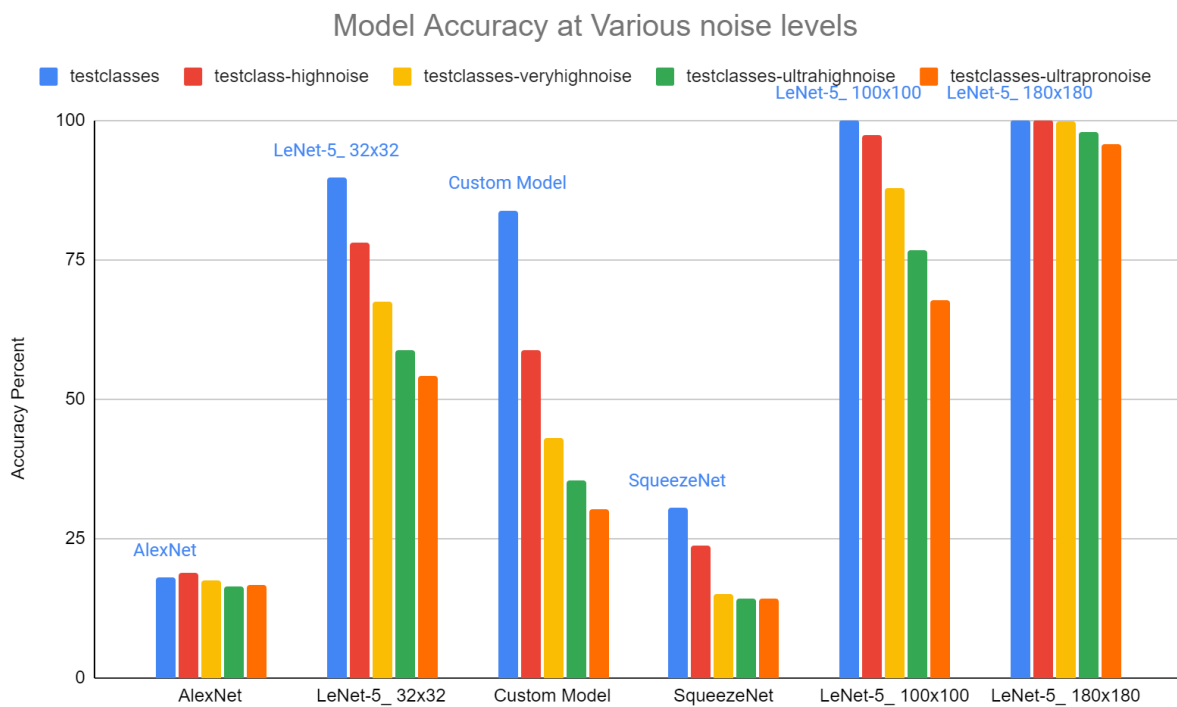


Figure: 3.4.7.2.f1: Tuned LeNet-5+Models Accuracy at various noise levels

Performance Comparison

| milliseconds per step | testclasses | highnoise | veryhighnoise | ultrahighnoise | ultrapronoise |
|-------------------------|-------------|-----------|---------------|----------------|---------------|
| AlexNet | 335 | 341 | 347 | 359 | 370 |
| LeNet-5_ 32x32 | 43 | 47 | 44 | 44 | 46 |
| Custom Model | 143 | 146 | 151 | 149 | 163 |
| SqueezeNet | 377 | 382 | 387 | 393 | 440 |
| LeNet-5_ 100x100 | 59 | 57 | 63 | 66 | 62 |
| LeNet-5_ 180x180 | 81 | 82 | 81 | 85 | 86 |

Table 3.4.7.2.t2: Model prediction speed at various noise levels

Graph:

testclasses, highnoise, veryhighnoise, ultrahighnoise and ultrapronoise

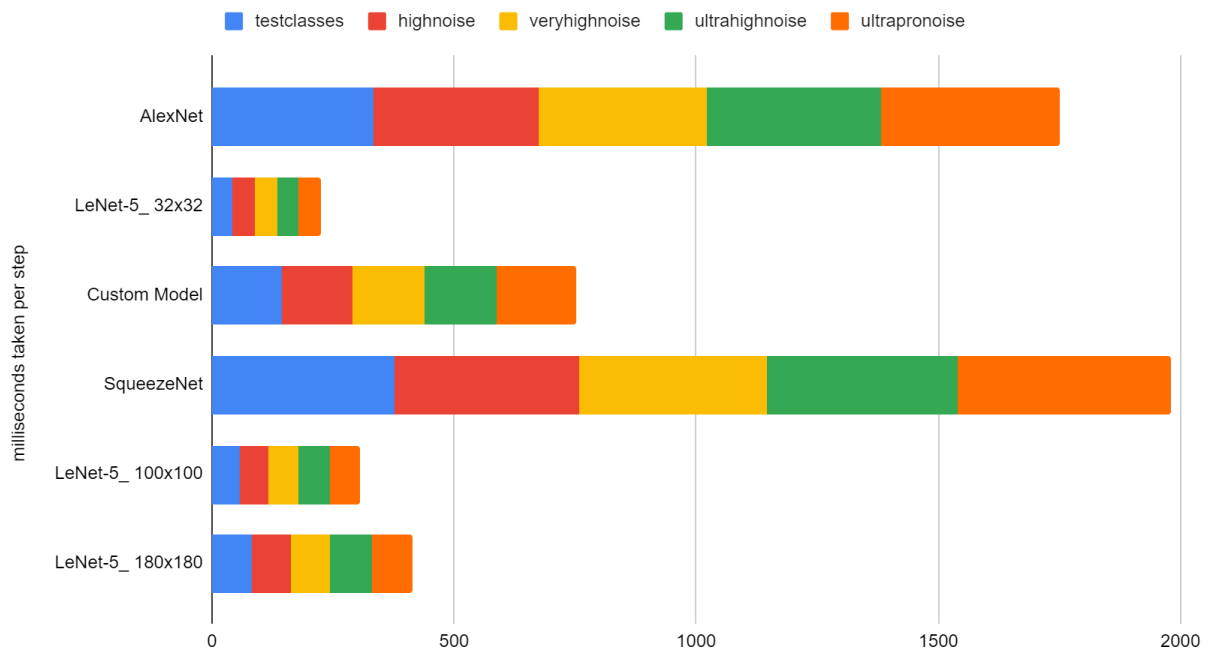


Figure: 3.4.7.2.f2: Model prediction speed at various noise levels

Size Comparison h5 file

| Size in MB | h5 file size |
|-------------------------|--------------|
| AlexNet | 455 |
| LeNet-5_ 32x32 | 1 |
| Custom Model | 60 |
| VGG16 | 1500 |
| SqueezeNet | 9 |
| LeNet-5_ 100x100 | 12 |
| LeNet-5_ 180x180 | 41 |

Table 3.4.7.2.t3: Models h5 file size

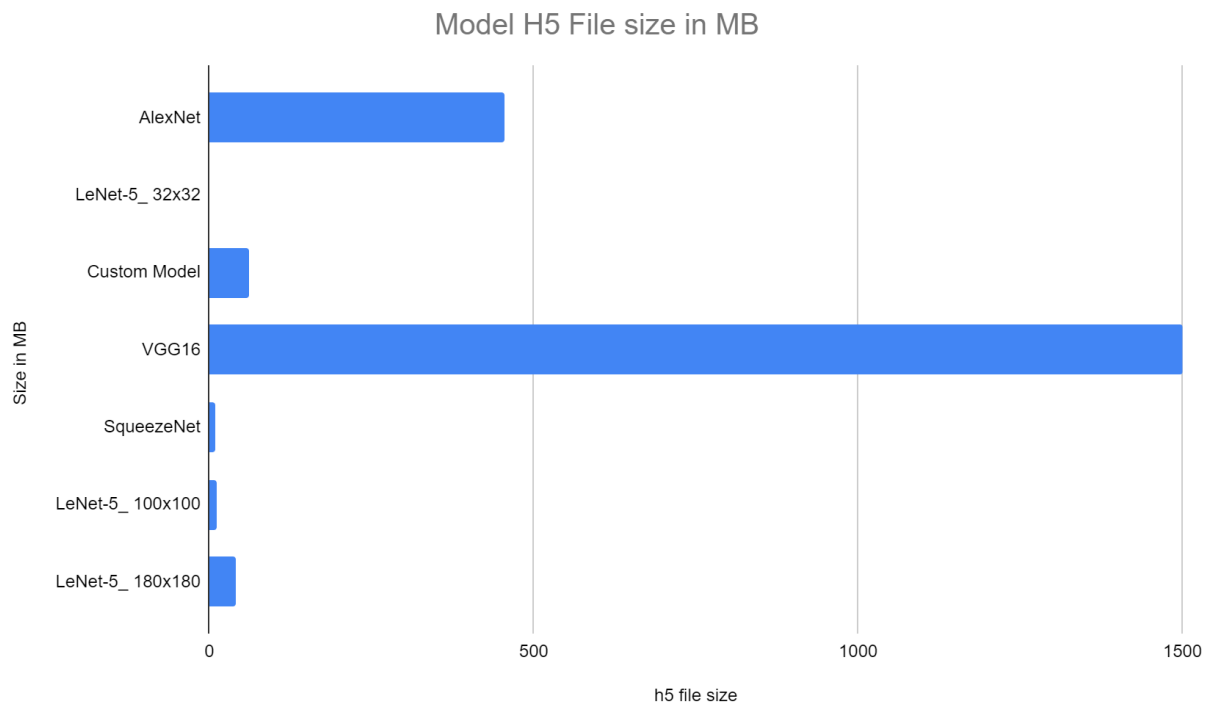
Graph:

Figure: 3.4.7.2.f3: Models h5 file size

In conclusion, LeNet-5_180x180 architecture is chosen to be used in web screens categorization due to high accuracy levels, lower h5 file size and fast performance.

3.5 Solution Implementation: API Using Flask

service/category:

This web service receives an image in base64 format, saves it as a PNG file, predicts the category of the image using a pre-trained deep learning model, and returns the category as a JSON response. The function `category()` is an API endpoint that listens for incoming POST requests with a JSON payload that contains a base64-encoded image. The function decodes the image and saves it as a PNG file with a unique name using the current timestamp. Then it calls the function `predictscreencategory()` to predict the category of the image. Finally, it returns the predicted category as a JSON response. The function `predictscreencategory()` loads a pre-trained deep learning model stored in an H5 file and predicts the category of the given image. It first loads the image from the given file path using `load_img()` function from Keras, resizes it to the required dimensions (180 x 180), and converts it to a grayscale image. Then, it converts the image to a numpy array and expands its dimensions to create a batch of size 1. The pre-trained model is used to predict the probabilities of each class for the input image, and the class with the highest probability is selected as the predicted category. The function returns the predicted category as a string.

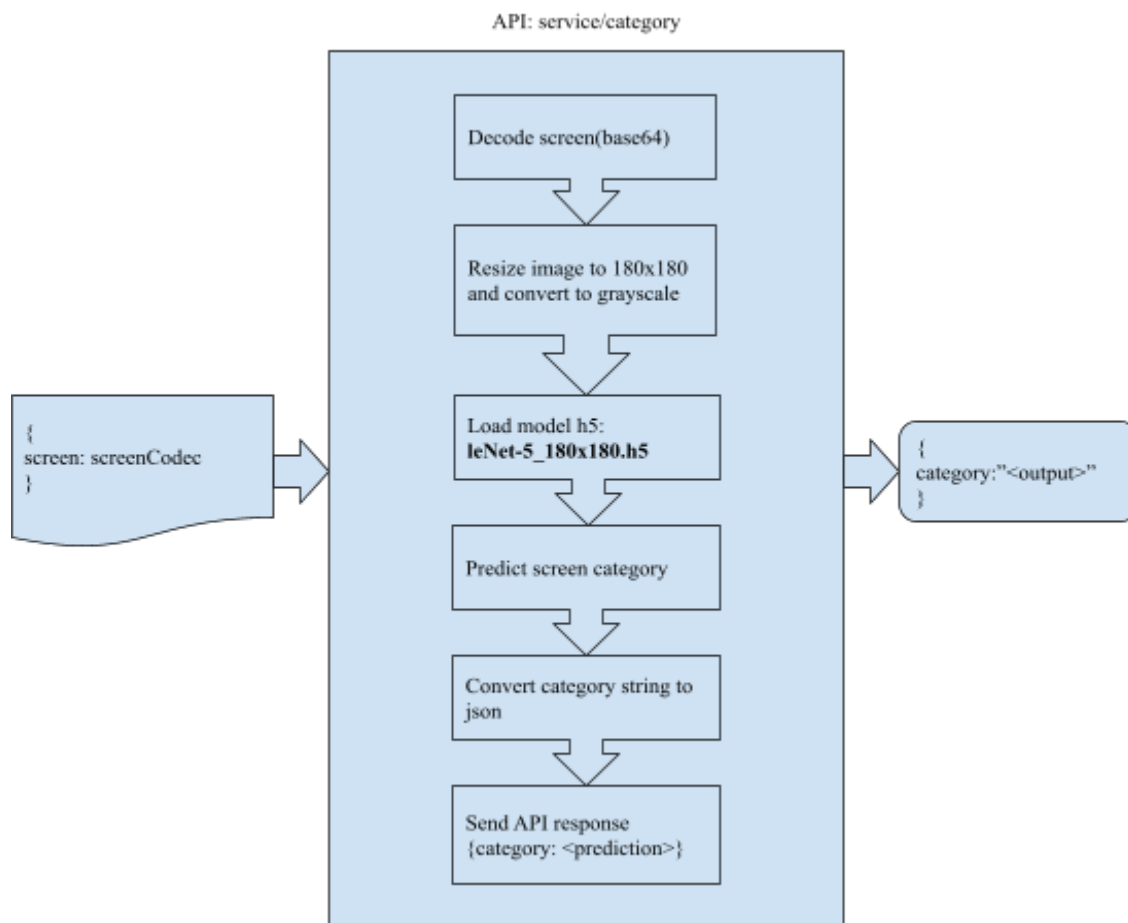


Figure: 3.5.fl: Architecture of service/category service

Code:

Python

```

def predictscreencategory(testImagePath):
    img_height, img_width = 180, 180
    model =
    tf.keras.models.load_model('model/leNet-5_180x180_255_800.h5')

    class_names = ["challenging_dom", "disappearing_elements",
                    "dynamic_content", "floating_menu", "infinite_scroll",
                    "shifting_content", "typos"]

    img = tf.keras.utils.load_img(
        testImagePath, target_size=(img_height, img_width),
        grayscale=True
    )
    img_array = tf.keras.utils.img_to_array(img)
    img_array = tf.expand_dims(img_array, 0) # Create a batch

    predictions = model.predict(img_array)
    score = tf.nn.softmax(predictions[0])
    category = class_names[np.argmax(score)]
    return category

@app.route("/service/category", methods=["POST"])
def category():
    timestamp = int(datetime.now().timestamp())
    if request.get_json()["screen"] is None:
        return jsonify({"status": 500, "error": "Expected screen"})
    logger.info("Incoming request : for category")

    base64Screen = base64.b64decode(request.get_json()['screen'])
    screenName = str("screenCategory" + str(timestamp) + ".png")
    logger.info("screenName : " + screenName)
    with open(screenName, "wb") as file:
        file.write(base64Screen)
    logger.info("screen written successfully")
    category = predictscreencategory(screenName)
    logger.info("Prediction Done: Image belongs to "+category)

    return jsonify({"category": category})

```

service/screendiff:

This web service receives an HTTP POST request with a base64 encoded image and a category name. The service uses the received image and category name to compare the image with a set of base images of the same category to determine if there are any differences. The processdiff function takes the path to the received image and a list of paths to the base images of the same category. For each base image, the function compares it with the received image using the structural similarity index and computes the difference between them. If the difference is above a certain threshold, the function detects and highlights the differences between the two images. The function returns the difference image. The run_image_diff_processor function takes the path to the received image and the category name. It constructs the paths to the base images of the same category and passes them to the processdiff function to compute the difference image. The function returns the difference image. The /service/screendiff endpoint is the HTTP POST endpoint of the web service. It receives the base64 encoded image and the category name as JSON data in the request body. The endpoint decodes the image and writes it to a file. It then calls the run_image_diff_processor function to compute the difference image and saves it to a file, encodes it to base64 before returning it as a JSON response.

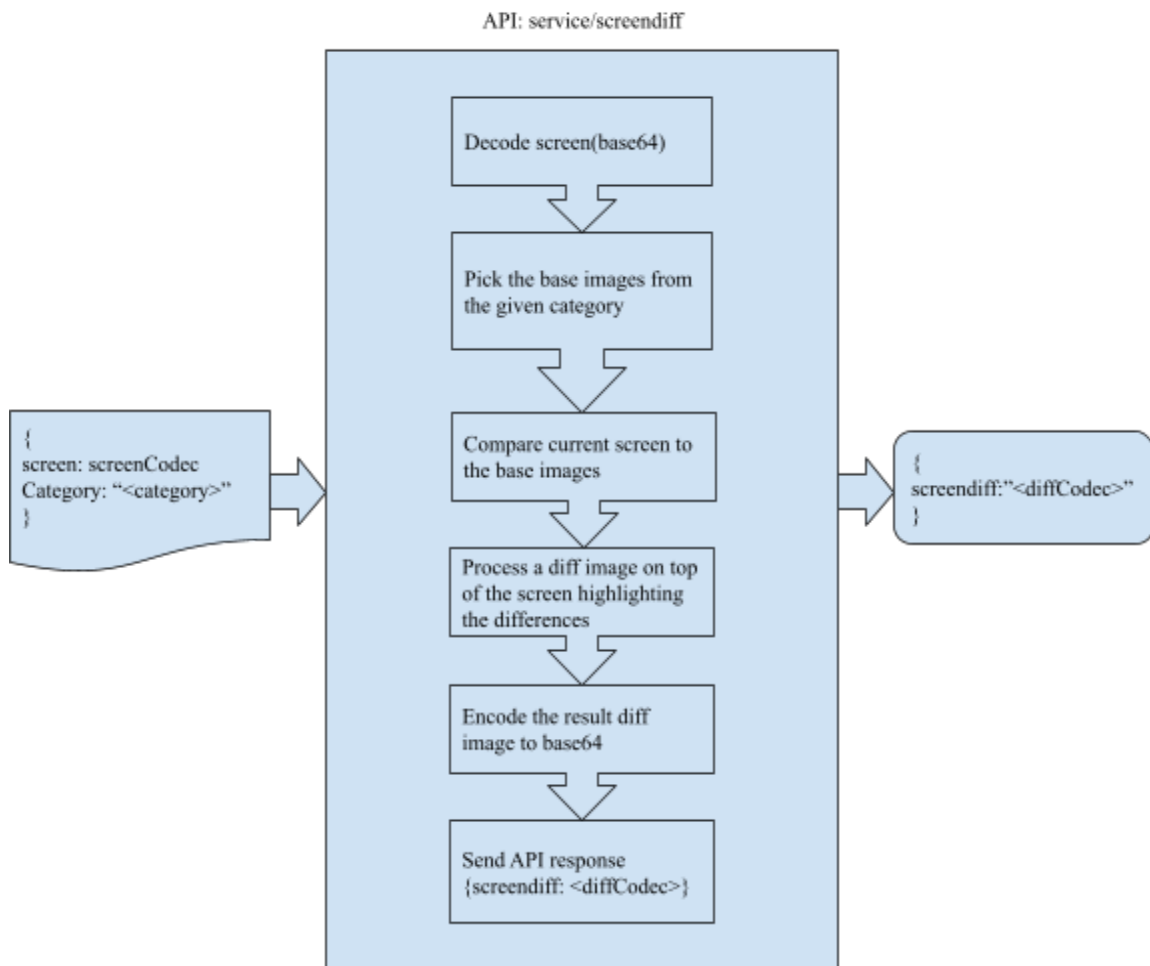


Figure: 3.5.f2: Architecture of service/screendiff service

Code:

Python

```

def processdiff(left, baseImages):
    diffImage = cv2.imread(str(left)).copy()
    for right in baseImages:
        # Load images
        before = cv2.imread(str(left))
        after = cv2.imread(str(right))

        before_gray = cv2.cvtColor(before, cv2.COLOR_BGR2GRAY)
        after_gray = cv2.cvtColor(after, cv2.COLOR_BGR2GRAY)

        (score, diff) = structural_similarity(before_gray, after_gray,
        full=True, win_size=3)
        print("Image Similarity: {:.4f}%".format(score * 100))

        diff = (diff * 255).astype("uint8")

        thresh = cv2.threshold(diff, 0, 255, cv2.THRESH_BINARY_INV |
        cv2.THRESH_OTSU)[1]
        contours = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE)
        contours = contours[0] if len(contours) == 2 else contours[1]

        for c in contours:
            area = cv2.contourArea(c)
            if area > 0.1:
                cv2.drawContours(diffImage, [c], 0, (0, 255, 0), -1)
    return diffImage

def run_image_diff_processor(pathToTestImage, category):
    left = pathToTestImage # "classes/" + str(category) + "/" +
    str(category) + "_1.png"
    baseImages = []
    for i in range(1, 49):
        right = "baseclasses/" + str(category) + "/" + str(category) +
        "_" + str(i) + ".png"
        print(right)
        baseImages.append(right)

```

```

diffImage = processdiff(left, baseImages)
# cv2.imwrite(pathToTestImage + 'difflogger.png', diffImage)
return diffImage

@app.route("/service/screendiff", methods=["POST"])
def screendiff():
    timestamp = int(datetime.now().timestamp())
    if request.get_json()["screen"] is None or
request.get_json()["category"] is None:
        return jsonify({"status": 500, "error": "Expected screen and
category"})
    logger.info("Incoming request : for diff")
    category = request.get_json()["category"]
    logger.info("category : "+category)

    base64Screen = base64.b64decode(request.get_json()['screen'])
    screenName = str("screenNow" + str(timestamp) + ".png")
    logger.info("screenName : "+screenName)
    logger.info(os.getcwd())
    with open(screenName, "wb") as file:
        file.write(base64Screen)
    logger.info("screen written successfully")

    resultDiff = run_image_diff_processor(screenName, category)

    cv2.imwrite("resultDiff" + str(screenName), resultDiff)
    with open("resultDiff" + str(screenName), "rb") as img:
        base64DiffImage = base64.b64encode(img.read()).decode('utf-8')

    return jsonify({"screendiff": base64DiffImage})

```

service/category will be called first to get the category of the screen followed by the service/screendiff API that responds with the diff image.

3.6 Challenges

Some challenges while using the proposed solution:

- Changes in the screen: Screens are subject to change frequently, such as with updates or new releases of applications. This can result in the ML model and screen difference script becoming outdated and requiring updates.
- Scalability: The ML model and screen difference script may need to be updated or adjusted to handle changes in the number of screen categories or variations in the screens within a category. This can make the system less scalable and require more maintenance.

Chapter 4

Deploying the Solution at Scale

4.1 Project Architecture

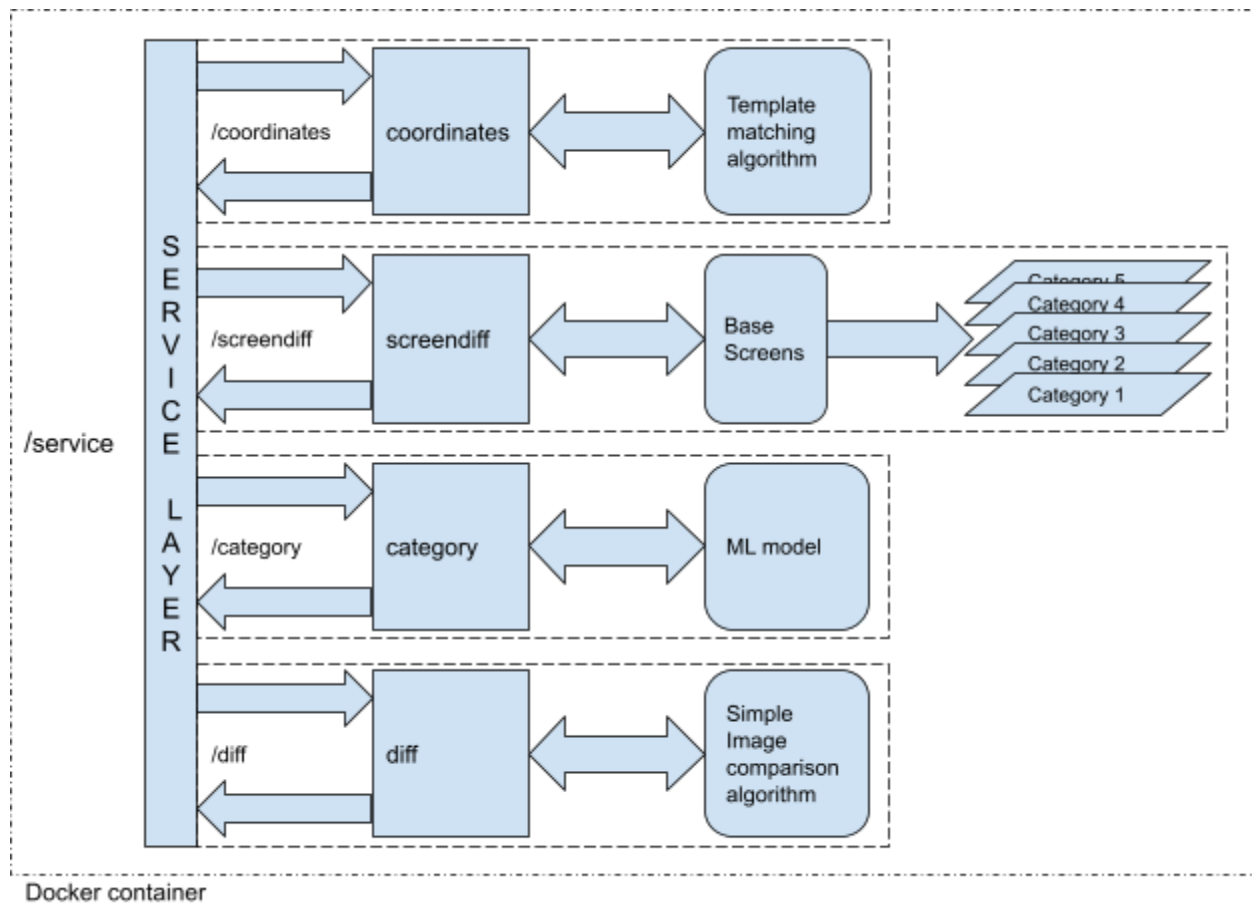


Figure: 4.1.f1: Project Architecture

The project's APIs are designed as RESTful endpoints that accept JSON requests and return JSON responses. The entire project is containerized as a Docker build image, allowing it to be run on any platform. Since the Docker images are platform agnostic, they can be deployed on various systems without modification.

Steps used to deploy the flask service on Docker:

- Navigate to project directory in the terminal
- Create a file called Dockerfile in this directory: This Dockerfile sets up a base image, copies the code and dependencies into the image, exposes port 5000 (the default Flask port), and sets the command to run your Flask application.

```
Unset
FROM python:3.9-slim
# Install system packages
RUN apt-get update \
    && apt-get install -y --no-install-recommends \
    ffmpeg libsm6 libxext6 \
    libatlas-base-dev curl

# Setting up working directory
RUN mkdir /python-server
WORKDIR /python-server
COPY . .
RUN pip3 install --upgrade pip
RUN pip3 install --no-cache-dir -r requirements.txt
RUN chmod +x ./entrypoint.sh
CMD ["sh", "entrypoint.sh"]
```

- A file called requirements.txt is created in the same directory as your Dockerfile, add any Python dependencies that the Flask application needs to run.
- Build the Docker image by running the following command in terminal: This command will build a Docker image with the tag `mytestapplicationservice`.

```
Unset
docker build -t mytestapplicationservice .
```

- Once the image is built, the container can be run from it with the following command:

```
Unset
docker run -p 5000:5000 mytestapplicationservice
```

This command will start a container from the `mytestapplicationservice` image, and map port 5000 inside the container to port 5000 on your host machine. The service will be up and running at `http://localhost:5000` in the web browser.

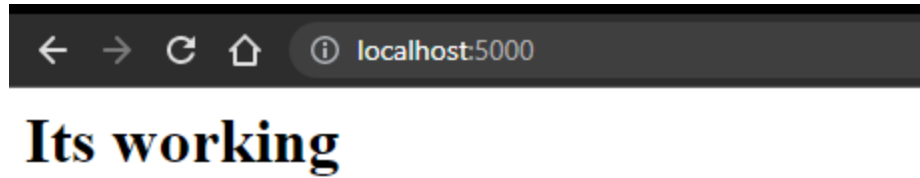


Figure: 4.1.f2: localhost:5000 web screen

The docker image `mytestapplicationservice` can be copied onto any machine that supports docker, and can be triggered using the run command to expose the service APIs. To reuse the image in another project, `dockerfile` and `requirements.txt` files can be copied into the new project directory, and the image can be rebuilt using the docker build command.

4.2 Tools and Technologies

Here are the libraries used for project implementation:

- `absl-py`: A library of Python utils used by Google AI.
- `astunparse`: A Python module used to unparse AST back to source code.
- `async-generator`: A library for making generators asynchronous.
- `attrs`: A Python library for creating classes with named attributes.
- `cachetools`: A set of caching utilities for Python.
- `certifi`: A Python library providing a set of trusted SSL certificates.
- `cffi`: A library for calling C code from Python.
- `charset-normalizer`: A Python library for character encoding detection and normalization.
- `click`: A Python package for creating command-line interfaces.
- `colorama`: A library for adding color and styling to command-line output.
- `contourpy`: A Python library for plotting contour and contour-like plots.
- `cycler`: A Python library for cycling through colors and styles for plots.
- `exceptiongroup`: A Python library for grouping exceptions into groups.
- `Flask`: A micro web framework for Python.
- `flatbuffers`: A library for efficient serialization of structured data.
- `fonttools`: A Python library for manipulating fonts.
- `gast`: A Python library for abstract syntax trees used by TensorFlow.
- `google-auth`: A Python library for Google authentication.
- `google-auth-oauthlib`: A Python library for Google authentication with OAuth.
- `google-pasta`: A Python library for manipulating tensors used by TensorFlow.
- `grpcio`: A Python library for communicating between distributed systems using gRPC.

- `h11`: A Python library for HTTP/1.1.
- `h5py`: A Python library for storing and manipulating large datasets.
- `idna`: A Python library for Internationalized Domain Names in Applications.
- `imageio`: A library for reading and writing image data in Python.
- `importlib-metadata`: A library for working with Python package metadata.
- `imutils`: A library for image processing tasks in Python.
- `input`: A Python library for reading user input.
- `itsdangerous`: A Python library for secure token generation.
- `Jinja2`: A template engine for Python.
- `keras`: A high-level neural networks API written in Python.
- `kiwisolver`: A Python library for solving equations.
- `libclang`: A Python library for interacting with the Clang C++ compiler.
- `Markdown`: A Python library for generating HTML from Markdown text.
- `MarkupSafe`: A Python library for escaping HTML, XML, and other markup languages.
- `matplotlib`: A Python library for creating static, animated, and interactive visualizations.
- `networkx`: A Python library for creating and manipulating complex networks.
- `numpy`: A Python library for scientific computing with Python.
- `oauthlib`: A Python library for implementing OAuth1 and OAuth2.
- `opencv-python`: A Python library for computer vision and machine learning.
- `opt-einsum`: A Python library for optimizing Einstein summation notation.
- `outcome`: A Python library for tracking outcomes of operations.
- `packaging`: A Python library for working with Python packages.
- `Pillow`: A Python library for image processing.
- `protobuf`: A Python implementation of Google's Protocol Buffers.
- `pyasn1`: A Python library for ASN.1 data structures.
- `pyasn1-modules`: A Python library for ASN.1 modules.
- `pycparser`: A C parser and AST generator for Python.
- `pyparsing`: A Python library for parsing text data.
- `PySocks`: A Python library for working with SOCKS proxy servers.
- `python-dateutil`: A Python library for working with dates and times.
- `python-version`: A Python library for getting information about the Python version being used.
- `PyWavelets`: A Python library for wavelet transforms.
- `requests`: A Python library for sending HTTP requests.
- `requests-oauthlib`: A Python library for OAuth1 and OAuth2 authentication with Requests.
- `rsa`: A Python library for RSA encryption and decryption.
- `scikit-image`: A Python library for image processing and computer vision.
- `scipy`: A Python library for scientific computing.
- `selenium`: A Python library for automating web browsers

- six: Python 2 and 3 compatibility utilities
- sniffio: Asyncio-compatible library for working with sockets, serial ports, and asyncio streams
- sortedcontainers: Python sorted collections library
- tensorboard: TensorFlow's visualization toolkit for machine learning experimentation
- tensorboard-data-server: Server for serving data to TensorBoard
- tensorboard-plugin-wit: TensorFlow plugin for displaying data with What-If Tool (WIT)
- tensorflow: Open-source machine learning library for numerical computation and large-scale machine learning
- tensorflow-estimator: High-level TensorFlow library for building and training machine learning models
- tensorflow-intel: Optimized TensorFlow library for Intel CPUs
- tensorflow-io-gcs-filesystem: TensorFlow plugin for reading and writing data from Google Cloud Storage (GCS)
- termcolor: Python library for ANSI color formatting in terminal output
- tiffio: Python library for reading and writing image files in Tagged Image File Format (TIFF)
- trio: Python library for async/await-native I/O, concurrency, and parallelism
- trio-websocket: Python library for websockets based on Trio and Python's standard library
- typing_extensions: Backported and experimental typing features for Python
- urllib3: HTTP client library for Python
- Werkzeug: WSGI utility library for Python
- wrapt: Python library for decorators, wrappers and monkey patching
- wsproto: WebSocket protocol implementation in Python
- zipp: Backport of pathlib-compatible object wrapper for zip files in Python 3.8+

Chapter 5

Literature Survey

Locators: Image Template Matching

In web testing, there are a few alternatives to using locators:

- Text-based search: Rather than using locators like ID or class name, you can search for web elements based on the text they contain. This can be useful when the text is unique and not likely to change frequently.
- CSS selectors: CSS selectors can be used to locate elements based on their style attributes. This can be helpful when the element does not have a unique ID or class name.
- XPath: XPath is a powerful tool for locating elements in an HTML document. It allows you to search for elements based on their tag names, attributes, and text content.
- Image recognition: With the help of image recognition tools, one can locate web elements based on their visual appearance. This can be particularly useful for identifying elements that don't have unique IDs or class names.
- JavaScript injection: In some cases, you can use JavaScript injection to locate web elements. This involves injecting JavaScript code into the browser to manipulate the DOM and locate the desired element. However, this approach requires a deep understanding of JavaScript and can be complex to implement.

Out of the mentioned alternatives, Image recognition is advantageous compared to other methods because it can process large amounts of visual data quickly and accurately. Here are some advantages of image recognition:

- Speed and accuracy: Image recognition algorithms can process large amounts of visual data in real-time, making it a very fast and efficient way to analyze images. They can also achieve very high accuracy rates, especially when compared to manual methods.
- Scalability: Image recognition can be easily scaled to process large volumes of images, making it ideal for applications such as image search engines, video surveillance systems, and social media platforms.
- Objectivity: Image recognition algorithms are not influenced by personal biases or subjective opinions, making it a reliable and objective method for analyzing visual data.
- Non-invasive: Image recognition can analyze images without the need for physical contact, making it a non-invasive method for studying living organisms and other sensitive subjects.

- Automation: Image recognition can be automated, which reduces the need for manual labor and makes it more efficient and cost-effective.

Few existing Solutions that can aid in Image template matching:

- OpenCV template matching: OpenCV provides a built-in template matching function that can be used for basic image matching tasks.
- Template matching with normalized cross-correlation: Normalized cross-correlation is a technique that can help improve template matching accuracy. OpenCV provides a function called `matchTemplate()` that implements normalized cross-correlation.
- Scale-invariant feature transform (SIFT): SIFT is a popular method for feature extraction and matching that is commonly used in computer vision applications. It is robust to scaling, rotation, and translation and can be used to match images containing different objects or scenes.
- Speeded up robust features (SURF): SURF is a feature extraction and matching technique that is similar to SIFT but is more efficient and faster.
- Fast R-CNN: Fast R-CNN is a deep learning-based object detection framework that can be used for template matching and object recognition tasks.
- You Only Look Once (YOLO): YOLO is another deep learning-based object detection framework that can be used for template matching and object recognition tasks. It is known for its speed and accuracy.
- Haar Cascades: Haar Cascades are a machine learning-based approach for object detection that can be used for template matching tasks. It is commonly used for face detection and other applications.

Simple template matching is a basic approach for image template matching that works well in situations where the object to be detected has a distinctive and easily identifiable shape or pattern, and the background is relatively uniform. It involves comparing a small template image with the larger search image, pixel by pixel, to find the best match.

One advantage of simple template matching is its simplicity and ease of implementation. It requires no specialized equipment or algorithms and can be implemented using basic image processing libraries such as OpenCV. Additionally, simple template matching is computationally efficient, making it ideal for real-time applications such as object detection in video streams.

However, simple template matching has some limitations. It may not work well in situations where the object of interest has complex features or is partially occluded by the background or other objects. In such cases, more advanced techniques such as feature-based methods or deep learning-based approaches may be more effective.

Testing User Interfaces at pixel level

Some Tools and libraries available for visual testing of web pages:

Percy: Percy is a visual testing tool that helps automate visual testing for web applications. It integrates with popular testing frameworks like Jest, Cypress, and WebDriver.

Shortcomings:

- **Cost:** Percy is a paid service and can be expensive for small teams or projects.
- **Limited language support:** Percy supports only a few programming languages and frameworks, which can limit its use in certain projects.
- **Setup time:** Setting up Percy can take some time, especially for complex applications or when integrating with existing testing frameworks.
- **Limited customization:** While Percy offers some customization options, it may not be sufficient for some teams or projects that require more flexibility.

Applitools: Applitools is a visual testing tool that uses AI and machine learning algorithms to analyze and compare visual differences between web pages.

Shortcomings:

- **Cost:** Applitools can be quite expensive, especially for larger teams or organizations. Pricing is based on usage, and the more tests and executions you run, the more expensive it becomes.
- **Learning curve:** Applitools has a steep learning curve, and it may take some time for teams to get up to speed with the platform and its features.
- **Limited support for certain platforms:** While Applitools supports a wide range of web and mobile platforms, there are still some platforms that are not fully supported or may have limitations.
- **Dependence on third-party tools:** Applitools relies on integrations with other tools and frameworks, such as Selenium and Appium, which can create dependencies and potential compatibility issues.
- **Lack of control over visual baseline:** Applitools automatically sets a visual baseline for each test, which can be convenient but may also result in false positives or other issues if the baseline is not set correctly or if changes are made to the visual elements of the application.

Galen Framework: Galen Framework is a layout and functional testing framework that allows developers to test their web applications across multiple browsers and devices.

Shortcomings:

- **Steep Learning Curve:** The Galen Framework has a relatively steep learning curve, especially for testers who are not familiar with the command line interface, CSS and HTML.
- **Limited Cross-Browser Testing:** Galen has limited cross-browser testing capabilities. Although Galen can run tests on multiple browsers, it can be challenging to write tests that work across all of them.
- **Technical Expertise Required:** The Galen Framework requires some technical expertise to set up and use effectively, and may not be suitable for less technical testers.

Wraith: Wraith is a screenshot comparison tool that allows developers to compare screenshots of their web pages across multiple devices and browsers.

Shortcomings:

- **Limited browser support:** Wraith only supports Firefox and Chrome browsers, which may not be suitable for all users or projects.
- **Limited integration options:** Wraith does not have many integration options with other testing tools, which can limit its usefulness in larger testing workflows.
- **Difficulty in setting up:** Setting up Wraith can be challenging for users who are not familiar with the command line interface and configuration files.
- **Limited visual testing capabilities:** Wraith primarily focuses on visual regression testing, and does not have advanced image recognition or AI-based testing capabilities.
- **Maintenance overhead:** Wraith requires regular maintenance and updates, which can be time-consuming and require a dedicated team to manage.

While tools and frameworks like Percy, Applitools, and Galen are useful for automated visual testing of web pages, they are primarily focused on detecting visual differences between two versions of a page. These tools typically do not have the ability to understand the content of the page and identify specific UI elements or components.

On the other hand, building a visual testing framework with an image classification model allows for a more granular approach to visual testing. The model can be trained to recognize specific UI elements or components, such as buttons or forms, and detect changes or anomalies in their appearance. This can help catch visual issues that may not be detected by the above tools.

Additionally, an image classification model can also be used to perform visual testing on non-web applications or interfaces, such as desktop or mobile applications. This can provide a more comprehensive visual testing solution across a variety of platforms and interfaces.

Chapter 6

Future Work

There are several potential future directions for research and development on image template-based web element detection and visual testing using an image classification model, including:

- Integration with machine learning algorithms: Image classification models can be further improved by incorporating other machine learning algorithms, such as deep learning or reinforcement learning, to enhance their accuracy and efficiency.
- Dynamic web element detection: Current methods for web element detection typically rely on static image templates, but dynamic web pages with changing content can present a challenge. Future work could explore methods for dynamically detecting and verifying web elements.
- Cross-browser and cross-platform testing: The majority of visual testing tools currently available are limited to testing web applications on a single browser or platform. Future work could explore ways to develop more comprehensive cross-browser and cross-platform visual testing frameworks.
- Improved reporting and analysis: While many visual testing tools provide comprehensive reporting and analysis capabilities, there is still room for improvement. Future work could focus on developing more sophisticated reporting and analysis features, such as the ability to identify trends and patterns in testing results.
- Integration with CI/CD pipelines: As continuous integration and continuous deployment (CI/CD) become more widely adopted, there is a growing need for visual testing frameworks that can be seamlessly integrated into these pipelines. Future work could focus on developing tools that can be easily integrated into CI/CD workflows, and that can provide fast and reliable feedback on the visual changes introduced by code changes.

Chapter 7

Appendix

7.1 Code Snippets

Filename: requirements.txt

Unset

```
absl-py==1.4.0
astunparse==1.6.3
async-generator==1.10
attrs==22.2.0
cachetools==5.3.0
certifi==2022.12.7
cffi==1.15.1
charset-normalizer==2.1.1
click==8.1.3
colorama==0.4.6
contourpy==1.0.7
cyclер==0.11.0
exceptiongroup==1.1.0
Flask==2.2.2
flatbuffers==23.1.21
fonttools==4.38.0
gast==0.4.0
google-auth==2.16.0
google-auth-oauthlib==0.4.6
google-pasta==0.2.0
grpcio==1.51.1
h11==0.14.0
h5py==3.8.0
idna==3.4
imageio==2.24.0
importlib-metadata==6.0.0
imutils==0.5.4
input==0.0.0
itsdangerous==2.1.2
```



```
Jinja2==3.1.2
keras==2.11.0
kiwisolver==1.4.4
libclang==15.0.6.1
Markdown==3.4.1
MarkupSafe==2.1.1
matplotlib==3.6.3
networkx==3.0
numpy==1.24.1
oauthlib==3.2.2
opencv-python==4.7.0.68
opt-einsum==3.3.0
outcome==1.2.0
packaging==23.0
Pillow==9.4.0
protobuf==3.19.6
pyasn1==0.4.8
pyasn1-modules==0.2.8
pyparser==2.21
pyparsing==3.0.9
PySocks==1.7.1
python-dateutil==2.8.2
python-version==0.0.2
PyWavelets==1.4.1
requests==2.28.1
requests-oauthlib==1.3.1
rsa==4.9
scikit-image==0.19.3
scipy==1.10.0
selenium==4.7.2
six==1.16.0
sniffio==1.3.0
sortedcontainers==2.4.0
tensorboard==2.11.2
tensorboard-data-server==0.6.1
tensorboard-plugin-wit==1.8.1
tensorflow==2.11.0
tensorflow-estimator==2.11.0
tensorflow-intel==2.11.0
tensorflow-io-gcs-filesystem==0.30.0
```

```
termcolor==2.2.0
tiff==2022.10.10
trio==0.22.0
trio-websocket==0.9.2
typing_extensions==4.4.0
urllib3==1.26.13
Werkzeug==2.2.2
wrapt==1.14.1
wsproto==1.2.0
zipp==3.11.0
```

Filename: dockerfile

```
Unset
FROM python:3.9-slim
# Install system packages
RUN apt-get update \
    && apt-get install -y --no-install-recommends \
    ffmpeg libsm6 libxext6 \
    libatlas-base-dev curl

# Setting up working directory
RUN mkdir /python-server
WORKDIR /python-server
COPY . .
RUN pip3 install --upgrade pip
RUN pip3 install --no-cache-dir -r requirements.txt
RUN chmod +x ./entrypoint.sh
CMD ["sh", "entrypoint.sh"]
```

Filename: entrypoint.sh

```
Unset
#!/bin/bash
exec gunicorn --config gunicorn_config.py run:app
```

Filename: gunicorn_config.py

```
Python
bind = "0.0.0.0:7600"
workers = 4
threads = 4
timeout = 120
loglevel = 'debug'
```

Filename: run.py

```
Python
from app import app

if __name__ == '__main__':
    app.run(host="0.0.0.0")
```

Filename: pipeline.py to test the Visual Testing framework

```
Unset
import base64
import os
import sys

import requests
from datetime import datetime

def getScreenDiff(screen, category):
    with open(screen, "rb") as img:
        screenCodec = base64.b64encode(img.read()).decode('utf-8')
        # print("encoding done")
        content_type = 'application/json'
        headers = {'content-type': content_type}
        response = requests.post('http://localhost:5000/service/screendiff',
                                json={"screen": screenCodec, 'category':
category}, headers=headers)
        # print("request sent")
        if response.ok:
            print("")
        else:
            # print("Response is NOT OK")
```

```

        sys.exit("Response not OK @getScreenDiff")
    return dict(response.json())

def getScreenCategory(screen):
    with open(screen, "rb") as img:
        screenCodec = base64.b64encode(img.read()).decode('utf-8')
    content_type = 'application/json'
    headers = {'content-type': content_type}
    response = requests.post('http://localhost:5000/service/category',
                             json={"screen": screenCodec},
    headers=headers)
    if response.ok:
        print("")
    else:
        # print("Response is NOT OK")
        sys.exit("Response not OK @getScreenCategory")
    return dict(response.json())

directory = "testsrealtime"
testscreens = []
for root, dirs, files in os.walk(directory):
    for filename in files:
        testscreens.append(str(filename))

for capturedScreen in testscreens:
    imagePath = directory + "/" + capturedScreen

print("\n\n*****")
print("Begin tests for "+capturedScreen)
print("\nPredicting screen category...")
categoryServiceResponse = getScreenCategory(imagePath)
screenCategory = categoryServiceResponse.get('category')
print(str(capturedScreen) + " belongs to " + screenCategory)
print("Proceeding with screen validation")
print("Screen validation is in progress...")
screenDiffResponse = getScreenDiff(imagePath, screenCategory)
print("Screen validation Complete for "+capturedScreen)
diffImageCodec = screenDiffResponse.get('screendiff')
resultDiffImage = base64.b64decode(diffImageCodec)
resultImageName = "results/" + capturedScreen + "response-diff-" +
str(int(datetime.utcnow().timestamp())) + ".png"

```

```

with open(resultImageName,"wb") as file:
    file.write(resultDiffImage)
    print("Please find the validation result for "+capturedScreen+"
here: " + resultImageName)
print("Folder validation tests complete.")

```

pipeline.py Design:

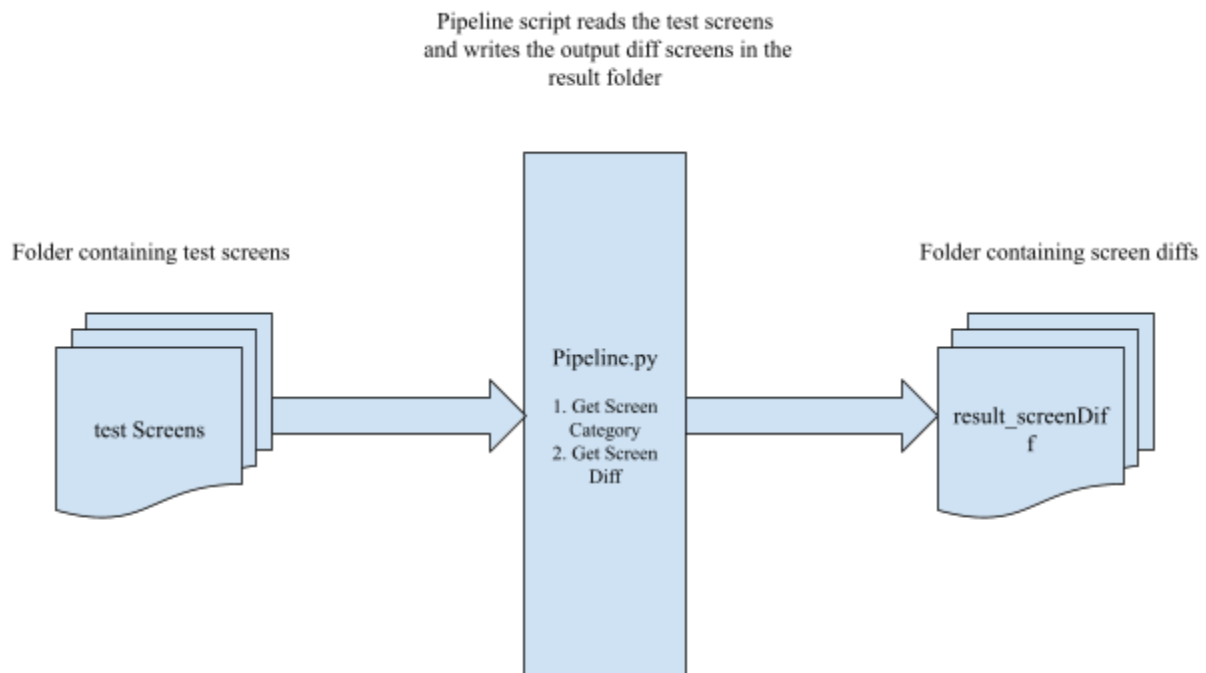


Figure: 7.1.f1: Pipeline test script architecture

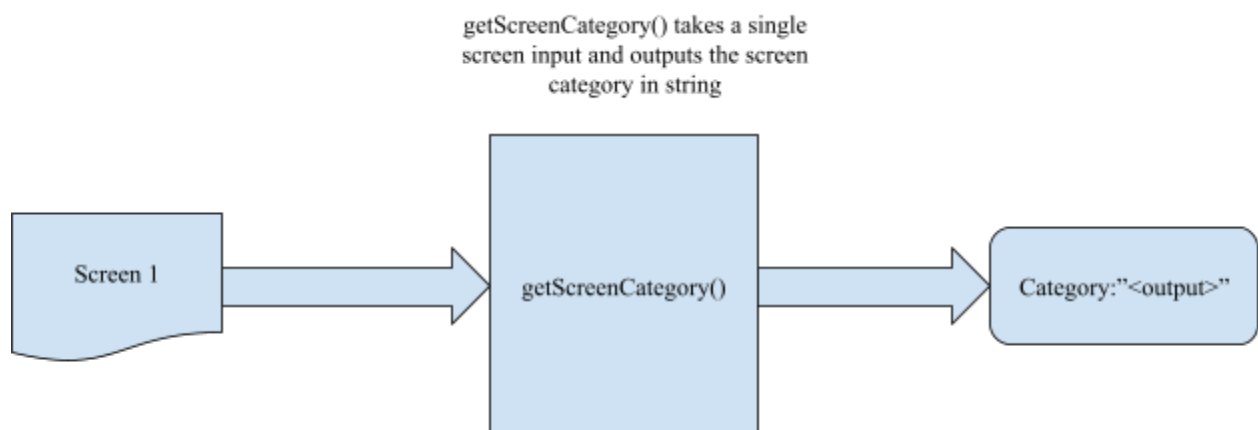


Figure: 7.1.f2: getScreenCategory function design

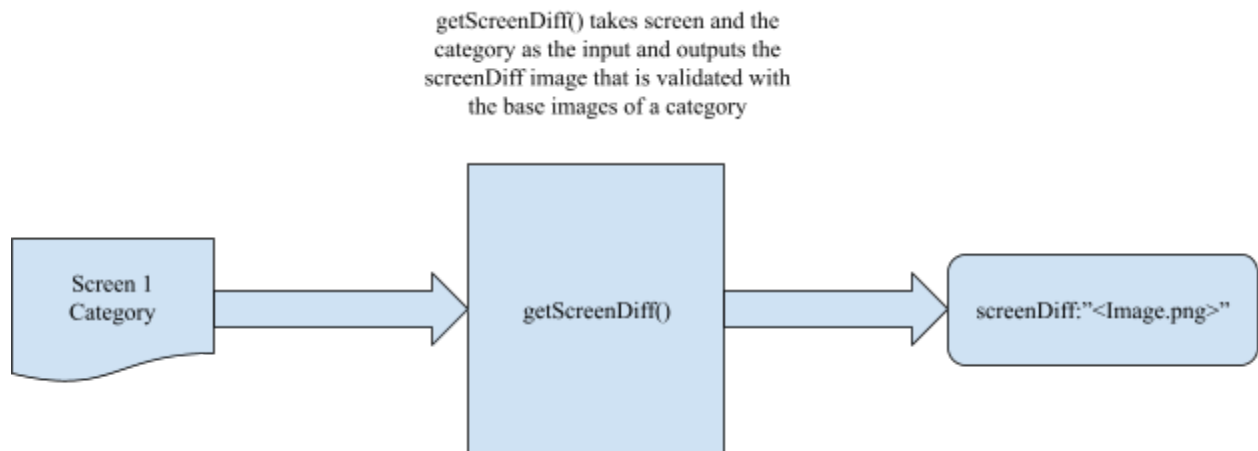


Figure: 7.1.f3: getScreenDiff function design

Sample Input for pipeline.py: red colored regions are added noise

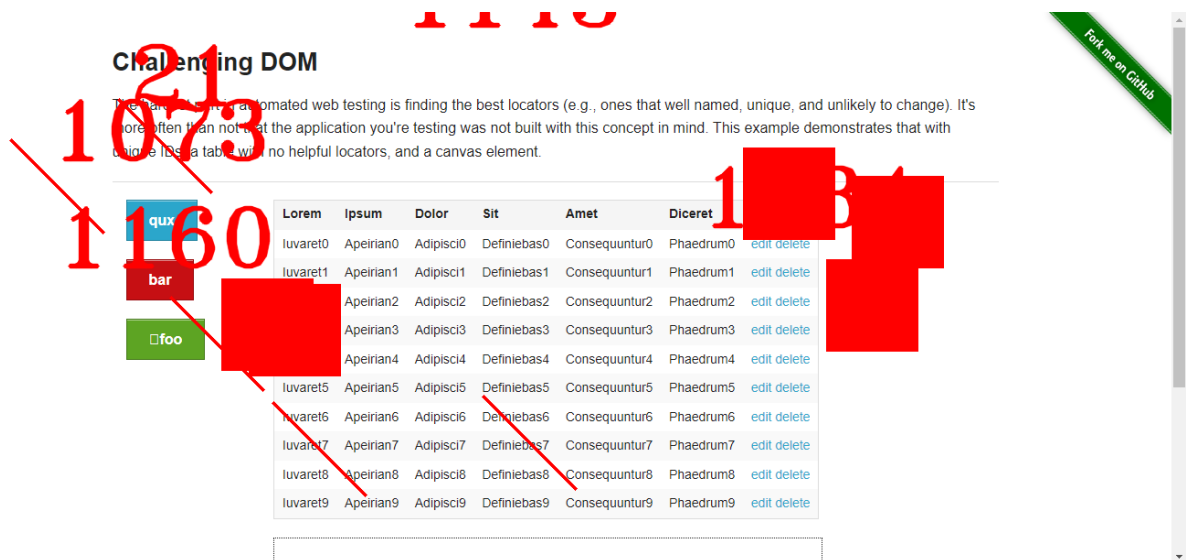


Figure: 7.1.f4: Sample input screen containing noise

Unset

```
public void poc_visual_identification() {
    WebDriver driver = BrowserInit.getDriver();
    driver.get("https://www.sprinklr.com");
    driver.manage().window().maximize();
    Waits waits = new Waits();
    waits.forSomeTime(5000);

    //      EVENTS
    //      Login Page Actions
        WebElement username = getElementsByCoordinates(new
File("files/loginPage/emailField.png"));
        highlightElement(driver, username);
        username.sendKeys("samplemail@sprinklr.com");
        waits.forSomeTime(1000);
        WebElement password = getElementsByCoordinates(new
File("files/loginPage/passwordField.png"));
        highlightElement(driver, password);
        password.sendKeys("password123");
        waits.forSomeTime(1000);
        WebElement loginButton = getElementsByCoordinates(new
File("files/loginPage/loginButton.png"));
        highlightElement(driver, loginButton);
        loginButton.click();
    //PageLoad
    waits.forPresenceOfElementLocatedBy(By.cssSelector("button[data-id=\"Mes
sages Posted\"]"));
        waits.forSomeTime(2000);
    //      ASSERTIONS
    //      ***** Icons Assertions *****
        WebElement universalSearch = getElementsByCoordinates(new
File("files/homePage/universalSearch.png"));
        highlightElement(driver, universalSearch);
        waits.forSomeTime(1000);
        WebElement homePageLoader = getElementsByCoordinates(new
File("files/homePage/homePageLoader.png"));
        waits.forSomeTime(1000);
        WebElement openLaunchpad = getElementsByCoordinates(new
File("files/homePage/openLaunchpad.png"));
        highlightElement(driver, openLaunchpad);
        waits.forSomeTime(1000);
}
```



```

        WebElement outboundCaller = getElementsByCoordinates(new
File("files/homePage/outboundCaller.png"));
        highlightElement(driver, outboundCaller);
        waits.forSomeTime(1000);
//        ***** Hover Assertions *****
        Actions actions = new Actions(driver);
//        searchIcon Hover message
        actions.moveToElement(universalSearch).perform();
        waits.forSomeTime(5000);

                                getElementsByCoordinates(new
File("files/assertors/universalSearchHoverMessage.png"));

//        openLaunchpadIcon Hover message
        actions.moveToElement(openLaunchpad).perform();
        waits.forSomeTime(5000);

                                getElementsByCoordinates(new
File("files/assertors/openLaunchpadHoverMessage.png"));

    }

    public void highlightElement(WebDriver driver, WebElement vElement)
    {
        try
        {
            JavascriptExecutor js = (JavascriptExecutor)driver;
            for (int i=0; i<3; i++)
            {
                js.executeScript("arguments[0].style.border='2px groove
green'", vElement);
                Thread.sleep(500); //blink
                js.executeScript("arguments[0].style.border='',
vElement);
            }
        }
        catch (Exception E)
        {
            System.out.println("Error in Highlight element");
        }
    }
}

```

Bibliography

1. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097-1105).
2. Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition* (pp. 248-255). IEEE.
3. Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European conference on computer vision* (pp. 818-833). Springer, Cham.
4. Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1), 1929-1958.
5. Krizhevsky, A. (2014). One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*.
6. Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.
7. LeCun, Yann, Leon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86, no. 11 (1998): 2278-2324.
8. LeCun, Yann, Léon Bottou, Bengio Yoshua, and Patrick Haffner. "Object recognition with gradient-based learning." In *Shape, contour and grouping in computer vision*, pp. 319-345. Springer, London, 1999.
9. Simard, Patrice Y., David Steinkraus, and John C. Platt. "Best practices for convolutional neural networks applied to visual document analysis." In *International Conference on Document Analysis and Recognition*, vol. 3, pp. 958-962. IEEE, 2003.
10. Boureau, Y-Lan, Jean Ponce, and Yann LeCun. "A theoretical analysis of feature pooling in visual recognition." In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 111-118. 2010.
11. Jarrett, Kevin, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. "What is the best multi-stage architecture for object recognition?." In *Proceedings of the 12th International Conference on Computer Vision (ICCV'09)*, pp. 2146-2153. IEEE, 2009.
12. Zhang, Xiangyu, Xinyu Zhou, Mengxiao Lin, and Jian Sun. "Shufflenet: An extremely efficient convolutional neural network for mobile devices." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6848-6856. 2018.
13. Jang, Dong-Won, Sung-Tae Kim, Kyung-Yeon Min, and Sungzoon Cho. "Performance analysis of convolutional neural networks for image recognition." *Journal of Applied Mathematics* 2017 (2017).

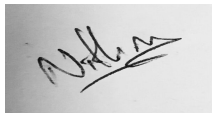
14. Zhao, Hao, and Xi Li. "A comprehensive survey of deep learning." In *Advances in Neural Networks*, pp. 5-17. Springer, Cham, 2017.
15. Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." In *European conference on computer vision*, pp. 818-833. Springer, Cham, 2014.
16. Weng, Lilian, Richard D. Chen, Yuanhua Huang, and Longhao Yuan. "Deep learning for image classification: A comprehensive review." *Neural Computation* 33, no. 4 (2021): 803-889.
17. Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations (ICLR)*.
18. Karen Simonyan and Andrew Zisserman. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556*.
19. Prakash, A., & Bosco, C. (2021). A Comprehensive Study of VGG16 and VGG19 Network Architectures for Image Recognition. *International Journal of Engineering Research & Technology (IJERT)*, 10(02), 54-61.
20. Tursunova, A., Kim, K., & Kim, S. (2020). Improved Image Classification using VGG16 and ResNet50. *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 1-6.
21. Naik, R., & Kumar, M. (2021). Comparative Study of VGG16 and ResNet50 Models for Image Classification using Transfer Learning. *2021 IEEE International Conference on Computer Communication and Informatics (ICCCI)*, 1-6
22. Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., & Keutzer, K. (2016). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size. *arXiv preprint arXiv:1602.07360*.
23. Sze, V., Chen, Y. H., Yang, T. J., & Emer, J. S. (2017). Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12), 2295-2329.
24. Rueda, L., Sanchez, V., Orozco, A. A., & Romero, E. (2018). Comparative analysis of deep learning algorithms for image classification. *Applied Sciences*, 8(6), 953.
25. Castro-Castro, A., & Carrasco-Ochoa, J. A. (2019). SqueezeNet and DenseNet for image classification tasks. In *Mexican Conference on Pattern Recognition* (pp. 363-373). Springer, Cham.
26. Aggarwal, H., Sharma, N., & Khatter, N. (2020). An Efficient CNN for Image Classification using SqueezeNet. *International Journal of Advanced Computer Science and Applications*, 11(4), 407-412.
27. Gani, A., Ali, M., & Han, J. H. (2021). SqueezeNet based deep learning framework for breast cancer classification. *Computer Methods and Programs in Biomedicine*, 200, 105908.
28. P. Surya, S. Jaiswal, and K. Nagwanshi, "Analysis of web element locators in selenium for web application testing," *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, Palladam, 2017, pp. 1-5.

29. M. Madheswari and R. Saravanaguru, "A survey on automated web application testing using locators," 2017 International Conference on Communication and Signal Processing (ICCSP), Chennai, 2017, pp. 1194-1199.
30. K. H. Chawla, A. K. Jain, and R. Sharma, "A comparative study of web element locators in selenium," 2017 International Conference on Computing, Communication and Automation (ICCCA), Greater Noida, 2017, pp. 269-273.
31. A. W. Al-Omari, "Efficient Web Element Locating Strategies for Test Automation," International Journal of Computer Applications, vol. 181, no. 31, pp. 23-30, 2018.
32. M. Singh, S. Sachan, and S. Sharma, "An empirical study on web element locators for automated testing," 2020 International Conference on Innovative Computing and Communication (ICICC), Ghaziabad, India, 2020, pp. 1-6.
33. Bradski, G., & Kaehler, A. (2008). Learning OpenCV: Computer vision with the OpenCV library. O'Reilly Media, Inc.
34. Bradski, G. (2000). The OpenCV Library. Dr. Dobb's Journal of Software Tools.
35. Guo, Q., Dai, Q., & Liu, B. (2017). A template matching algorithm based on OpenCV for positioning of LAMOST fiber positioning. Journal of Physics: Conference Series, 898(9), 092027.
36. Jo, Y. H., & Kim, S. H. (2016). A Real-time License Plate Recognition System using Raspberry Pi and OpenCV. Journal of the Korea Institute of Information and Communication Engineering, 20(12), 2767-2774.
37. Liu, T. J., Yen, Y. C., & Hsiao, S. W. (2013). License Plate Recognition Algorithm Based on Template Matching and Morphological Processing. International Journal of Advancements in Computing Technology, 5(7), 142-153.
38. Nguyen, T. H., Nguyen, V. Q., Nguyen, N. T., & Tran, Q. T. (2019, October). A template matching method for object detection on Raspberry Pi using OpenCV. In 2019 12th International Conference on Advanced Computing and Applications (ACOMP) (pp. 103-106). IEEE.
39. Prasad, D., & Prakash, A. (2017). Automatic license plate recognition using template matching algorithm. International Journal of Engineering and Technology, 9(3), 1895-1900.
40. Wang, Y., Qian, Y., & Zhou, M. (2018). A Rapid Image Stitching Algorithm Based on Template Matching. Journal of Computational and Theoretical Nanoscience, 15(9), 4139-4145.
41. Wu, X., Hu, J., & Wang, X. (2017). An automatic detection method for crack location based on template matching. In 2017 IEEE International Conference on Information and Automation (ICIA) (pp. 2507-2511). IEEE.
42. Zhang, Z. (2004). Iterative point matching for registration of free-form curves and surfaces. International Journal of Computer Vision, 60(2), 91-110.
43. Grano, M., Marchetto, A., & Tonella, P. (2018). Automated visual testing of web applications. IEEE Transactions on Software Engineering, 44(3), 193-212.

44. Palomba, F., Ferrucci, F., & De Lucia, A. (2016). User interface testing: A systematic mapping study. *Information and Software Technology*, 78, 13-32.
45. Memon, A. M., Banerjee, I., & Nagarajan, A. (2016). GUI testing: 20 years later. *Journal of Software: Evolution and Process*, 28(5), 413-439.
46. Nguyen, T. T., Nguyen, T. T., Nguyen, T. A., Nguyen, H. T., & Alshammari, R. (2019). Towards automated GUI testing using visual GUI element recognition. *Information and Software Technology*, 107, 163-180.
47. Zaidman, A., Romano, D., & Panichella, A. (2018). Automated testing of web applications: A survey. *Software Testing, Verification and Reliability*, 28(1), e1659.
48. Grinberg, M. (2018). *Flask web development: Developing web applications with Python*. O'Reilly Media, Inc.
49. Shalabh, K. (2019). *Flask Framework Cookbook: Over 80 proven recipes and techniques for Python web development with Flask, 2nd Edition*. Packt Publishing Ltd.
50. Huang, J., & Packt, T. (2019). *Flask Web Development with Python Tutorial: An easy to follow guide on Flask for beginners to get started with building web applications in Python*. Packt Publishing Ltd.
51. Poulton, A. (2018). *Docker Deep Dive*. John Wiley & Sons.
52. Duffield, D., & Cain, I. (2017). *Docker in Practice, Second Edition: Includes 17 Hands-On Labs*. Manning Publications.
53. Matsumoto, K. (2020). *Docker and Kubernetes Cookbook: Building scalable, reliable, and efficient applications, 2nd Edition*. Packt Publishing Ltd.
54. Gelperin, D., & Huggins, C. (2006). Selenium: a portable software testing framework. In *Proceedings of the 2006 OOPSLA workshop on Eclipse technology eXchange-Volume 1* (pp. 1-6).
55. Mishra, A., & Singh, D. (2014). Automation of web applications using Selenium. *International Journal of Computer Applications*, 99(19), 7-12.
56. Nguyen, T. T., Nguyen, T. H., Nguyen, V. T., & Nguyen, Q. V. (2019). A systematic literature review of web application testing. *Journal of King Saud University-Computer and Information Sciences*, 31(3), 297-316.
57. Paladugu, A. K., Adepalli, A., & Mandala, B. K. (2020). Design of an effective test automation framework for web applications. *International Journal of Advanced Science and Technology*, 29(06), 11184-11193.
58. Shi, Z., & Jiang, X. (2020). A lightweight and high-performance web testing automation framework based on Selenium. *IEEE Access*, 8, 153492-153503.
59. Singh, A., Jain, A., & Jain, V. (2018). Test automation framework for web applications. *International Journal of Engineering and Technology(UAE)*, 7(2.24), 105-110.
60. TensorFlow Official Documentation https://www.tensorflow.org/api_docs.
61. Python Official Documentation <https://docs.python.org/3/>.
62. Docker Official Documentation <https://docs.docker.com/>.
63. Flask Official Documentation <https://flask.palletsprojects.com/en/2.2.x/>.

Checklist of items for the Final report

1. Is the Cover page in proper format? **.Y / N**
2. Is the Title page in proper format? **.Y / N**
3. Is the Certificate from the Supervisor in proper format? Has it been signed? **.Y / N**
4. Is Abstract included in the Report? Is it properly written? **.Y / N**
5. Does the Table of Contents page include chapter page numbers? . . . **.Y / N**
6. Does the Report contain a summary of the literature survey? . . . **.Y / N**
7. Are the Pages numbered properly? **.Y / N**
8. Are the Figures numbered properly? **.Y / N**
9. Are the Tables numbered properly? **.Y / N**
10. Are the Captions for the Figures and Tables proper? **.Y / N**
11. Are the Appendices numbered? **.Y / N**
12. Does the Report have Conclusion / Recommendations of the work? . **.Y / N**
13. Are References/Bibliography given in the Report? **.Y / N**
14. Have the References been cited in the Report? **.Y / N**
15. Is the citation of References / Bibliography in proper format? . . . **.Y / N**



Nithin Chary
Student



Ish Abbi
Supervisor