National College of Ireland

BSHC4

Software Development

2020/2021

Matthew Kearns

x17492632

x17492632@student.ncirl.ie

Auto-Trust

Technical Report

Contents

Executive Summary

This report outlines the development of my Final Year Project, with focus on project implementation through updated requirements scope to final deliverables.

The report highlights the importance of visual testing within the development lifecycle, providing a base framework of folder structure and test execution methods for automated testing with focus on automated visual testing. Offering a base for site comparisons of your development, staging, or production environments, encapsulated as continuous integration/continuous deployment delivery. Automated testing was done in a Behaviour Driven Development (BDD) approach by utilizing the natural language user behavioural scenarios of the Behave Python library, providing streamlined communication between developers, testing engineers, and business. The benefit of automating these scenarios is to inspect and assert site features before they reach production.

These visual comparisons are achieved with python toolkits to capture site element differences through exact pixel comparison, and site Image Quality Assessment (IQA) is evaluated with a custom-built machine learning model - a Convolutional Neural Network Image Classifier (CNN) that achieves 96.9% accuracy in assessing image quality under six categories: high resolution, blur, colour distortion, JPEG compression, noise, and spatial distortion. Assessing these qualities are essential in terms of customer usability and general development lifecycle endeavours, such as speeding up sprint achievements/deadlines, or avoiding painful deployment rollbacks.

CNN's can be built from scratch, or pre-built CNN's that have a very high success rate in classifying object features, edges etc, can be altered to suit your use case through methods of transfer learning, yielding high success rates (Brownlee, J., 2020). Through developing custom-built CNN's and implemented various methods of transfer learning in attempts to capture accurate results, this project reveals why incorporating IQA into your development lifecycle can speed up development processes by preventing image distortions that manual IQA may not be able to discover with the same accuracy, within the same time frame.

1.0 Introduction

1.1. Background

Following completion of my college internship program, upon reflection of a situation that I encountered led me to explore the possibilities of improving testing by adding the additional component of automated visual testing through image comparison and quality assessment to a testing environment to provide additional accuracy to areas that manual testing may bypass. There is a lot of room for human error within testing, and this can be drastically reduced with general automated testing. An issue with automated testing in many cases is the User Interface aspect, as this is omitted in many automation scripts and unit-tests and widely done manually. Even if the visual element's structure is tested, the background imagery or surrounding text distortion may not be captured by tests. Automated tests are usually done on a code-based assertion level, through Application Programming Interface (API) status assertion or element assertion. API status assertion compares the response of an API request to an expected response, and element assertion is achieved using the Document Object Model (DOM) to extract element values and compare them to expected values. An example of the problem at hand - if an API automated test script is built to test a company's login, it would pass if an API 200 status (success status) were returned, and the script could find the 'welcome' text on the homepage. However, if there is some text or imagery which is not rendering correctly this may be overseen by the automated test if not accounted for.

The issue occurred for me when these tests retrieved many elements, which were compared and successfully passed all assertion tests, yet there was a template rendering issue due text that was not de-coded correctly in response to a software update. This rendering issue was not caught and could have been extremely hazardous if it were released to production.

It is not practical to automate every aspect of a page, and tests cannot account for every possible scenario. However, the notion that this room for error can be drastically reduced inspired me to tackle visual impurities like the example just mentioned, by adding automated visual testing components to a developer's testing workflow.

1.2. Aims

This project introduces the structure of why this project was undertaken, and through what implementation of technologies the project aims have come to life.

In terms of an end goal of my personal development experience and learning outcomes was to collaborate my current knowledge of software development along with the challenge of new tools, technologies, and frameworks to bring light to the importance of visual testing when releasing code during the Software Development Lifecycle. The project aims to demonstrate this in a workflow that underlines the principles of automated browser testing.

Personal development aims were also tied to requirements gathering, where most requirements remain the same as proposed in my project proposal. The aim was to adhere to these projections under a strict agile development process tied to a project plan. This was a well thought out plan of future objectives based off researched technologies with the intention of avoiding future obstacles. The strictness added to this low-level granularity plan under current circumstances and unknown availability in the second semester due to unreleased timetables, was intentionally added to zone in on requirements gathering and sprint updates during my development process. This has truly stood to the final deliverables my personal development projected aims and reflective perspectives.

Auto-Trust itself as a final deliverable aims to guide you in your project's development lifecycle by providing base automated visual testing framework that can be integrated to automate site comparisons of your development, staging, or production environments, encapsulated as a continuous integration/continuous deployment delivery. Testing is done in a BDD approach, assessing the behaviour of new developments against expected outcomes, and adjusting these tests when the behaviour changes.

To further elaborate what was mentioned in the executive summary, the framework offers the option the utilize Image Classification under two metrics:

1. Identical site element comparison to highlight any areas of a site that have changed during development, such buttons, Hyper Text Markup Language (HTML) tags that have not been closed/slipped through to the environment, or test distortions/formatting because of inappropriately unhandled encoding/decoding.
2. Image distortion classification to highlight any areas of a site image instances that may have lost quality through the deployment pipeline such as JPEG compression. It is difficult to pin-point what is causing such occurrences and having a second pair of eyes on your site with logic behind image quality assessment can point you in the right direction of a solution when previously the type of image distortion would be unknown and more difficult to troubleshoot the root cause and a solution.

There is nothing that stands out to me that a basic beginner, non-dev-op, or anyone without previous automation knowledge can use in their own backend workflow, already integrated as a base framework that demonstrates the main features of an automated workflow and visual testing. There are of course front-end tools that offer similar testing metrics, but these do not reveal what is happening in the background processes. This is great for most beginners as it de-couples the technical process from their experience. However, if someone wanted to integrate a similar testing architecture into their own workflow, there are many different interdependent

tools and technologies that can have tedious installations and configurations, and that is what I would like to change. To provide a framework that has the dependencies configured, automated workflow examples that can be easily adjusted to suite specific scenarios, and visual testing scripts that invoke image comparisons for visual testing, leaving only the complex operations of the CNN de-coupled from the framework, but the code to develop this CNN can still be found within the project structure. For these people or businesses, the information is there but the accessibility to bring this into your environment with basic knowledge at a very high granularity is not, and a problem that this project aims to address.

1.3. Technology

Most of the technologies that were originally proposed have been used, and additional technologies have been gathered through requirements and added to development. All technologies mentioned in this section are mentioned as a quick definition overview, and their full definitions and methodologies are described in detail in Section 2.3 - Implementation.

Test scenarios will be created in the Python Programming Language, extended by Behave and Selenium. Selenium is needed to access the web-browser itself, web-scraping a Chrome browser's DOM elements and navigate through the browser. Behave is required to overcome the obstacles of the workflow methodology mentioned in Section 2.3. Behave in this sense is needed for navigation throughout site locations by defining user behavioural scenarios to be executed. Python is the required language to run these Behave structured scenarios.

These three technologies and their tests scripts are stored on GitHub source control. An example of a user's workspace environment is also stored on GitHub. A Jenkins build and Circle-Ci build will be triggered when a developer pushes their new changes to a GitHub branch that has a pull request open to merge existing changes into a development, staging, or production environment GitHub branch. Jenkins and Circle-CI are continuous integration automation servers, which will allow me to set up a CI/CD workflow and elaborate examples of continuous integration server integrations. They both need to be integrated with GitHub and define the procedure of a build triggered, resulting in execution of automation scripts in the manner I would like them to be executed in, on what environment I would like them to be executed on.

Image distortion classification was developed using Keras and TensorFlow, which are open-source libraries for developing artificial neural networks (Gulli, A. & Pal, S., 2017) (TensorFlow. 2018). These will be used to create my new image data with image augmentation (Image augmentation applies various adjustment to an image such as zooming, rotation, etc to produce more data; a ground for better feature recognition through largely similar, yet different data). Keras and TensorFlow will also be used to develop my CNN.

1.4. Structure

All areas of the document are structured around Requirements, Design and Architecture, and Implementation/Graphical User Interface. The Requirements section is split up into Functional Requirements, Data Requirements, User Requirements, Environmental Requirements, Usability Requirements.

In order of Requirement structure, the Functional Requirements aim to detail an understanding of what the system must do, and how they must function to support an end user's workflow. By

designing a Use Case diagram, the aim was to visually represent the system boundary, the system requirements within this boundary which are defined by the relationships they hold with other system requirements. Each aspect adhering to the requirements engineering specification needed to execute a full-satisfactory User Workflow. The main functional requirements are outlined underneath by their order of integrating into an end user's framework when an end user decides to use Auto-Trust. requirements priority is defined in the section followed. An individual requirement's use case is broken down to imply the preconditions required in each case, the main flow, and alternative flows of the functional requirements.

The data requirements refer to the projects use of data in obtainment of end deliverables, pre-processing, and maintenance of data throughout the project implementation.

User Requirements outline how to meet the physical and cognitive needs of the intended users. How they currently perform a task that Auto-Trust interface will support. The Environmental Requirements outline the visual, auditory, or tactile deficits. The Usability Requirements outlines the Efficiency of use, Intuitiveness, Ease of use.

Section 2.3 - Implementation, is key section that breaks down progression of systematic approaches through my chosen technology stack is, outlining what development approaches were taken with regards to projected goals, errors encountered, and solution countermeasures. Requirements gathered have been incrementally assessed and devoted to throughout Implementation, and the section dives into the framework which my project provides; an automated visual testing framework that offers clarity and ease to development lifecycles. This section also explains why BDD was chosen as this project's integrated testing architecture instead of a Test-Driven-Development (TDD) approach. BDD follows the structure of creating scenarios and test cases using structures, natural language that provides a testing approach of balanced communication between developers, testers, and business. The section describes how Behave was utilized, a structured implementation of natural language testing and how it was incorporated into my Implementation of visual testing image differences through more advanced methods such as A Convolutional Neural Network (CNN) Image Classifier, where data was acquired and manipulated to train this Neural Network. The challenges of this are described alongside constant reference to requirements engineering in response to challenges.

The focus is an automation workflow for visual testing, where the comparisons of an end user's User Interface (UI) are integrated into this project framework and invoked from within the test scenario architecture mention above. There are two scripts developed to achieve visual comparisons, and it is explained further how these were developed, what they product, and how they tie into the end goal. Graphical User Interface (GUI) in Section 2.4 is often linked in this document with emphasis on the differentiation from a typical front-end GUI that would be a final interactable product de-coupled from the back-end architecture as such, but rather focuses on the user's architectural setup that contains the projects main features as a final architectural deliverable.

Section 2.1.1.1 - Use Case Diagram compliments the whole report, offering a visual breakdown of the system. As an early section in this report, this diagram intends to visually demonstrate what structure has been put in place to adequately organize the technologies and functional requirements into a user's workflow in hopes of filtering out any previous confusion that may have existed while reading prior sections of the report, adding value to the system and subsequent sections. This has a relationship to Section 2.2 - Design & Architecture which

delegates the system architecture into key areas, in hopes of adding additional visualization structure to the understanding of the project.

2.0 System

2.1. Requirements

2.1.1. Functional Requirements

2.1.1.1. Use Case Diagram

The use case below describes the Requirements Engineering outlined in the project proposal and the adjustment since, additions and removals for a full system use case within an end user's workflow.

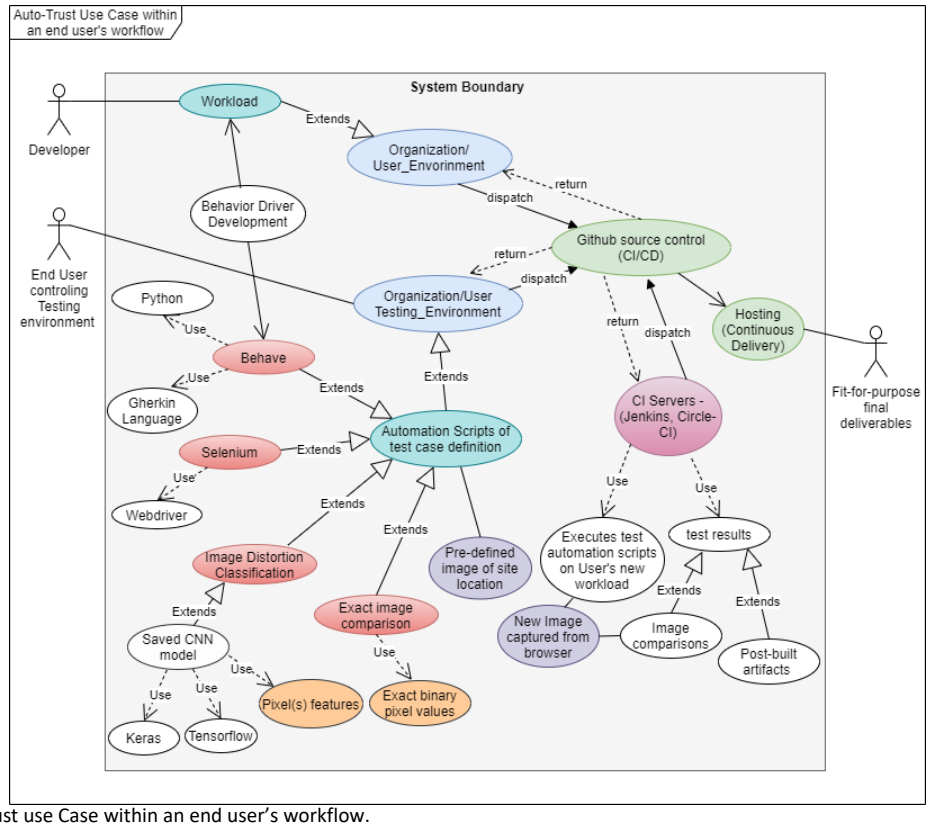


Fig. 1. AutoTrust use Case within an end user's workflow.

2.1.1.2.

Requirement 1: System must provide a suitable place for a user to obtain the framework, to successfully set up their environment.

Requirement 2: When the framework is acquired, a user must be able to successfully understand and execute the framework workflow provided for automated testing.

Requirement 3: A User must be able to successfully adjust the framework to their own scenarios to be integrated into their own workflow.

Requirement 4: User is notified on a failed test after visual testing is performed.

2.1.1.3. Description & Priority

Requirement 1: System must provide a suitable place for a user to obtain the framework, to successfully set up their environment.

This requirement holds the approach to installing and downloading the automated testing framework provided from a suitable location. Without this requirement all other requirements are not possible to implement, therefore, holds the highest priority. It outlines the user's ability to follow the description of an end site, what the framework offers, and a code infrastructure that will be provided to install. Through this function they will be able to set up the necessary architecture for performing automated testing with focus on visual testing.

Requirement 2: When the framework is acquired, a user must be able to fully understand and execute the framework workflow provided for automated testing.

This requirement outlines the required ground for a levelled understand for not only technical but non-technical users. This requirement is second priority. Code must be well defined, utilizing descriptive language for functions, methods, and variables throughout the framework. To allows the user to fully understand the execution workflow and execute themselves.

Requirement 4: User must be able to successfully adjust the framework to their own scenarios to be integrated into their own workflow.

This requirement is needed to scale the framework in adjustment to the users own site requirements. This requirement is third priority. The frameworks code must adhere to design principles. Instances of functions and methods must be closed, yet open to expansion. Common variables must be declared global and easily accessible. Overall separation of concerns must be adhered to through code implementation and folder structure. A user must not

Requirement 4: User is notified on a failed test after visual testing is performed.

This requirement is also essential to the architecture, but it comes after all previous requirements, therefore it is last priority. It is the post-condition to testing a user's environment and through this they are notified if a test has passed or failed. Without this the project deliverables are not working or fully delivered.

2.1.1.4. Use Case

Requirement 1

Scope

The scope of this use case is to highlight a user's interaction with Auto-Trust's site, installing the package in their testing environment.

Description

Describes the actions taken by a user to install Auto-Trust's framework, install the required dependencies, in a position to take the actions of requirement 2.

Use Case Diagram

Refer to Section 2.1.1.1 as the end user controlling the testing environment.

Flow Description

Precondition

Targeting primarily end users with a testing framework in place, but also with no testing framework in place. Either a high-level or low-level understanding of the BDD architecture of python, behave, PyCharm, Selenium, a web-driver.

Activation

This use case starts when the end user lands on Auto-Trust's site.

Main flow

1. A User without an environment consisting of the prerequisites outlined in the preconditions above, reads the site description and clicks on the GitHub link to download. A user with a testing framework sees step 2.
2. The User installs the dependencies.
3. The system now has the required dependencies.

Alternate flow

A1: User has an alternative testing environment setup.

1. The User reads the documentation and understands the features offered enough to incorporate the functions and features into their own testing framework.
2. This use case continues at position 2 of the main flow.

Exceptional flow

N/A

Termination

All imports throughout the framework are imported correctly now that dependencies are installed.

Post condition

The system goes into wait a state.

Requirement 2

Scope

The scope of this use case is to highlight a user's interaction with the framework navigation and code base implemented.

Description

Describes the actions taken by a user to navigate the site by following descriptive language in code comments, functions, and variables.

Use Case Diagram

Refer to Section 2.1.1.1 as the end user controlling the testing environment.

Flow Description

Precondition

An understanding of python, and familiar with the purpose of Selenium, or an expert at Selenium.

Activation

This use case starts when the end user begins to look at the feature code of the framework.

Main flow

1. Th system provided a BDD architecture of Behave written in Gherkin natural language.
2. The User understands the natural language and follows direction of natural language transitioning into programming logic.
3. The User then understands the behavioural scenario described, retraces to the feature file, and executes the test.

Alternate flow

A1: User has an alternative testing environment setup.

1. The User understands the features offered for visual testing and directly translates or extracts image comparison scripts into their own test framework.
2. This use case continues at position 2 of the main flow.

Exceptional flow

N/A

Termination

Basic knowledge of the Behave and text execution methods used are understood.

Post condition

The system goes into wait a state.

Requirement 3

Scope

The scope of this use case is to highlight a user's interaction with Auto-Trust's site, installing the package in their testing environment.

Description

Describes the actions taken by a user when expanding and utilizing the pre-defined functions of the architecture to suit their own use case of their own architecture/architectural requirements.

Use Case Diagram

Refer to Section 2.1.1.1 as the end user controlling the testing environment.

Flow Description

Precondition

Has explored a large area of the workflow in requirement 3, and well understood. Knows Object Oriented Programming (OOP) principles.

Activation

This use case starts when a user is confidence in integrating the framework to suit their own testing desires.

Main flow

4. The user begins to use their OOP knowledge to expand test functions.
5. The user can access all areas of the framework for expansion except the trained weight of the CNN.
6. The system is adjusted to the user's desires.

Alternate flow

A1: User has an alternative testing environment setup.

3. User
4. This use case continues at position 2 of the main flow.

Exceptional flow

N/A

Termination

All imports throughout the framework are imported correctly now that dependencies are installed.

Post condition

The system goes into wait a state.

Requirement 4

Scope

The scope of this use case is to alert an end user of a test result.

Description

Describes the user's basic integration and development steps to execute a test when developing.

Use Case Diagram

Refer to Section 2.1.1.1 as the Developer.

Flow Description

Precondition

The previous requirement has been met, under focus is the preconditions and post conditions.

Activation

This use case starts when the end user wants to test their environment.

Main flow

1. The User pushes their changes to GitHub.
2. The projects domain provider Netlify creates a new Staging environment build, updating the live web application.
3. The GitHub recognises this push through its Actions defined in settings.
4. GitHub Actions executes a Jenkins build using this new environment hosted URL as the domain subject in testing.
5. Jenkins executes shell commands to install dependencies and run Behave.
6. Jenkins returns the test results to the user.
7. Jenkins returns the test results.

Alternate flow

A1: User executes Jenkins build manually

1. This flow starts at position (5.) of the main flow.
2. The user wants to clarity the result and navigates to Jenkins.
3. The user executes a Jenkins build manually.
4. Jenkins returns the test results to the user.

A2: User has Circle-CI as continuous integration server

5. Circle-CI executes shell commands to install dependencies and run Behave.
6. Circle-CI returns the test results to the user.
7. Circle-CI returns the test results.

A3: User executes Circle-CI build manually

1. This flow starts at position (5.) of the Alternate Flow 3 (A2).
2. The user wants to clarity the result and navigates to Jenkins.
3. The user executes a Jenkins build manually.
4. Jenkins returns the test results to the user.

Exceptional flow

N/A

Termination

Continuous integration server provides a passed or failed result to the User's environments GitHub, or the manual execution post built actions on continuous integration project dashboard view.

Post condition

The system goes into a wait state, waiting on a user to push more environment changes, or execute build jobs manually.

2.1.2. Data Requirements

The Data Requirements for this project will be focusing on my implementation of the project objectives, and the end user's implementation once the project is complete, from identifying the data to achieving the end framework.

On my side of development, a suitable dataset must be acquired to train my model, consisting of large enough data specific to each distortion class in subject, providing the means for more accurate training and feature extraction from the image pixel values

a key step to the end framework is the web application subjected in testing. This application needs to have a hosted domain to access it from the end user's testing scripts. It is required to choose a secure hosting available, and Netlify was chosen as the hosting service provider. To build this domain It is essential to be careful with all data, especially image data which must be copyright free. Any images shown from image datasets must not be shown if the dataset does not offer the availability for academic research. I must target data carefully.

The data requirements for the user will corresponds to the in the project documentation user requirements and outlines the user's interactions with their own environmental dart such as their web application.

The user's accessibility to their domain through authentication when the user is testing their web application may vary, as tests will need to be adjusted to firstly provide a means of de-coupling login credentials and keys from the testing framework, storing them in a safe location or through injected environment variables within the continuous integration servers, such as main domain login credentials or Docker credentials if a Docker image is used within the CI continuous integration server build script.

2.1.3. User Requirements

The focus from the start of this project was to make software engineers from developers to testers, and managers lives easier by offering a clear and concise solution to their already existing testing framework, as mentioned in the use case preconditions "Targeting primarily end users with a testing framework in place". While this still holds as the key concept of end user implementation, as stated in the aims, section 1.2, a shift of focus from the above precondition is to also add more attention to end users that have no knowledge of a testing framework. This slight requirement shift occurred while analysing previous projections of user requirements, there are a lot of tools and technologies that end users may or may not

incorporate already in their existing environment. This way, by offering an easy to understand, well commented framework, and overview of how this project's end visual testing features are integrated into the project architecture was used to build and test the visual testing features, it not only provides the end users with pre-existing knowledge the ability to integrate the visual testing features into their existing testing-framework, but it offers knew knowledge to complete beginners to testing or these specific testing tech stack. It shall give users the ability to start integrating it as their own similar architecture. The ideal architecture of an end user that has pre-conceived knowledge of testing and the technology stack, consists of a behaviour-driven-development testing workflow that uses python, behave, PyCharm, Selenium, and a web-driver, along with their project on GitHub and Jenkins configured to their GitHub repositories.

Another aspect to this requirement shift is that visual testing may be enough to satisfy a user's testing needs, without wanting to dive deep into the automation offered by this framework through Behave and Selenium, but instead configure a quick implementation which will utilize solely site navigation and visual testing as strong addition to how they currently perform developing/testing.

2.1.4. Environmental Requirements

The purpose of Environmental Requirements is to outline the visual, auditory, or tactile deficits. – N/A.

2.1.5. Usability Requirements

Efficiency of use: Once our User Requirements are met, as mentioned the primary focus is on end users that have a testing framework in place, the usability of our system must be user-efficient in its verification of functionality promised to the user and in a timely manner. This verification through visual testing results must satisfy the user and add value to their workflow.

Intuitiveness: The CNN must be consistent in results when comparing two screenshots. This simplicity of learning must be present throughout to offer the user the ability to continuously develop and continuously integrate without delays.

Low perceived workload: The system must always hold its core objective through the user's perception, that being to enhance testing results accuracy, and speed up development cycles with trusted automated measures, and at no stage should this appear to be intimidating, or a liability to the user's workflow.

2.2. Design & Architecture

Auto-Trust's design principles revolve around the BDD systematic approach to developing and testing user or organization site components. Auto-Trust is designed to put forth a solution to

visual testing and capture UI defects with regards to an addition to one's web application that needs to be released. Below is the design and architecture of this release process as a System based architecture with regards to the Use Case in section 2.1.1.4.

- Phase 1: The architecture takes in the user's workload when the user is satisfied with their changes and would like to release these changes to their site environment. The system is designed using Netlify hosting, using Netlify's ability to trigger new hosting of an environment once code has been pushed to that environment's GitHub source control.

- Phase 2: The execution of this environment. This execution exercises Jenkins and Circle-CI as examples of continuous integration servers, executing the automated test workflow. The Jenkins execution has a series of components for this build such as setting the location of the environment's GitHub repository and SHH credentials to retrieve the code. The second components structure the Build Actions, defining what steps are to be taken in this build, such as initializing a virtual environment, installing dependencies (caches or un-cached), executing Behave feature files to kick off the automated test scripts, and storing results. Similarly, Circle-CI is connected to the environment's GitHub repository with SSH-key authorization. The build defined has a similar process to Jenkins, initializing a virtual environment, installing dependencies, executing scripts, and storing results. However, Circle-CI differs in how this execution is invoked, as the build steps are defined in a YAML file that is stored within your repository. A Jenkins build steps are defined within a BASH script or a SHELL script. Both executions within this systems phase 2, execute within a command line interface.

- Phase 3: The results phase. Within the system, the automated scripts capture results and passes them to our next build procedure: Post-Build-Actions. Both Jenkins and Circle-CI can be given a specification to dump post-build artifacts after each build, in this system case they specify the folder location containing failed image comparisons. Email notifications are also set to be sent on builds in the Post-Build-Actions.

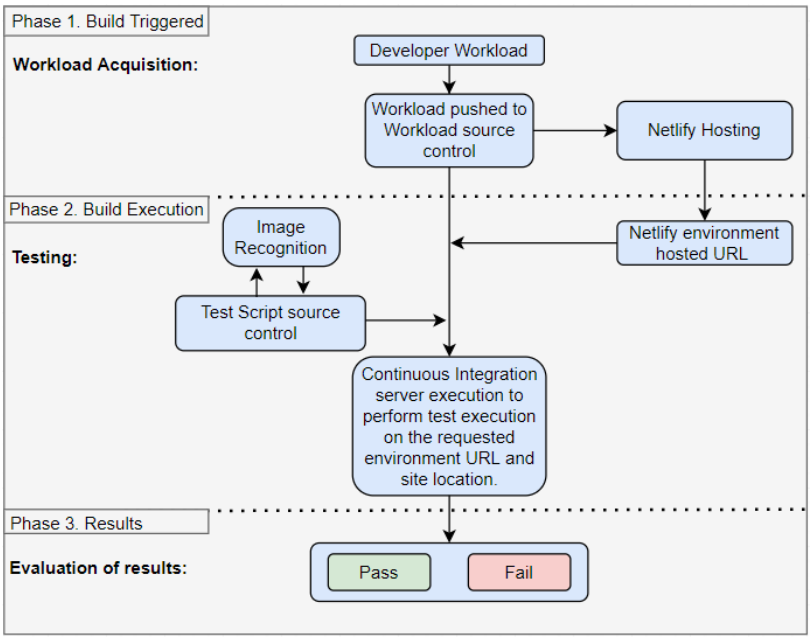


Fig. 2. A general, high level Architechture of the System based off the Use case in section 2.1.1.4.

Understanding the CNN architecture.

The most complex algorithm implemented was the Convolutional Neural network. CNN's are a recent development of machine learning technology, widely used in modern developments such to improve image classification and object detection (S. Ren, K. He, R. Girshick, and J. Sun). Image features are extracted by a CNN through and iterative learning approach. As mentioned in the executive summary, CNN's can be built from scratch, or pre-built models that that already have a very high success rate in classifying image features can be altered to suit your use case through methods of transfer learning such as feature extraction or fine-tuning, yielding high success rates for you too if the appropriate alteration methods are applied (Brownlee, J., 2020).

A regular CNN can be built with a Sequential Model architecture, which is a linear stack of neural network layers. Each layer is defined by weather or not it is fully connected to

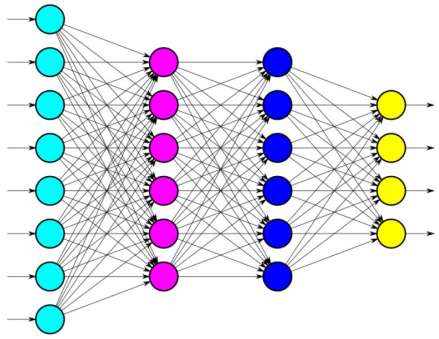


Fig. 3. Linear stack of nodes in a Sequential model (deeplizard, 2020.)

The layer are defined based on the following opperations:

Convolution – to mathematically extract the input pixels and sustain each pixel's relationship with its surroundings.

Polling – another common term for polling is subsampling. When computing through the layers of the CNN, a 'feature map' picks up the image features by navigating along the image. Once complete, this 'feature map' outputs strict special results between pixels and takes up a lot of computation of the network. Pooling is used to reduce the dimensions of each outputted feature map, so that we can then flatten this computation.

Flattening – by converting the above into a single column to be passed to the next layer in the network.

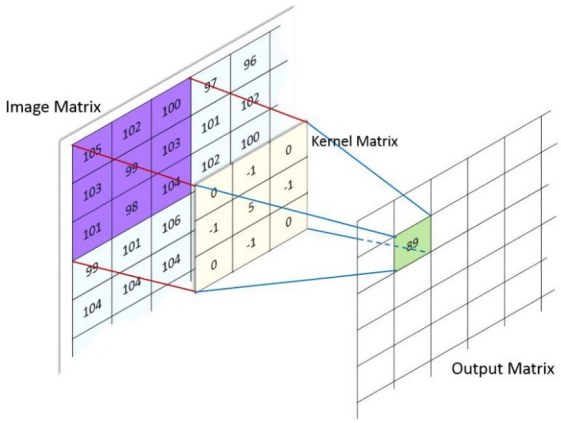


Fig. 4. Feature map applied to the specified feature map area (kernal/filter size), to produce an output matrix.

The kernel metric is also known as the filter which will be mentioned in the Implementation of the model, in terms of how I applied this where and why.

2.3. Implementation

Implementation outlines elaborates the mean by which requirements were met, adjusted, and evaluated throughout project development.

Understanding the BBD environment setup in the automated testing framework.

For BDD, I have used the Behave Python Library that is required for behave in a python environment. The behave library follows as specific project tree structure, which requires the PyCharm IDE professional edition. This licence was acquired by signing up through a student JetBrains Educational account membership.

BDD was chosen as the integrated workflow testing methodology for Auto-Trust early on in requirements specification, as a TDD would require an additional process of pre-defining tests before the workload is developed, not using the behave architecture and expected test results are updated before the new code development is tested. The question arose at the beginning of the project - “what happens when a new feature is added to the User Interface, the visual test will fail”. BDD overcame this obstacle through its heavy involvement in business logic before testing, where a testing engineer develops tests to cater for the coded logic, rather than a developer developing into the test logic. Since this framework is aimed at being integrated into an already existing environment, adjusting the framework to suit the behaviour of a site is considered BDD. Once fully integrated, new code releases will technically temporarily follow a test TDD approach in the sense that failed tests will be assessed before being released in most scenarios. However, any new developments that add change to the sites front-end interface, the automated tests will either be updated in parallel with the code development, or after the code development, both methods through communication between the test suite developer, the code developer, and the business. From this we can see how the test in BDD are developed to suit the code suite, whereas in TDD the code is developed to suit the test suite.

Behave runs in this environment using the Gherkin language, which takes on the form of translating from a business understanding to technical understanding, so that teams can transfer information with the aim to iron out many communication and knowledge barriers. The 'Scenario' or 'Scenario Outline', is defined in a Gherkin Feature file, and outlines the steps that the user is to take, and the expected result of these steps, as 'Given' and ''When'. Behave must also follow a file format for it to navigate between directories which is shown in Fig. 6 and explained further in Fig. 7. Below in Fig.5, is an example Scenario outlining the procedure of refunding items, and the expected result of each step in the Scenario. The expected results of this scenario would be assertions of the expected number of sweaters in stock against the current number of sweaters in stock (i.e., assert number-of-sweaters == 3). A failed assertion would result in a failed test step, which would be a failed Scenario.

```
Scenario: Refunded items should be returned to stock
  Given a customer previously bought a black sweater from me
    and I currently have three black sweaters left in stock.
   When he returns the sweater for a refund
   then I should have four black sweaters in stock.,
```

Fig. 5. A basic scenario outlining the expected result of stock status after refunding items [1].

```
features/
features/signup.feature
features/login.feature
features/account_details.feature
features/environment.py
features/steps/
features/steps/website.py
features/steps/utils.py
```

Fig. 6. An example of the tree structure that behave searches for to execute each .feature file [1].

A Scenario is produced after discussion on code features and deliverables, with the expected result in mind. My project outlines the steps taken to get to the required URL that needs to be tested, and The Document Object Model (DOM) elements that need to be tested. In a normal testing scenario these elements are located and compared by their direct expected outcome, for example testing if **'HelloWorld == HelloWorld'**. Each Below is the current implementation of this scripting approach in my example of a user's testing environment, which currently navigates to our example of a user's web domain. It also shows that by clicking on each of these steps, the Ghirkin language identifies these as steps and prompts us to create functions for these steps, that handle the parameters passed in by our Scenario. The parameters I am defining is the **<homepage_url>**, the **<heading_text>,** and **<screenshotted_page_location>.**

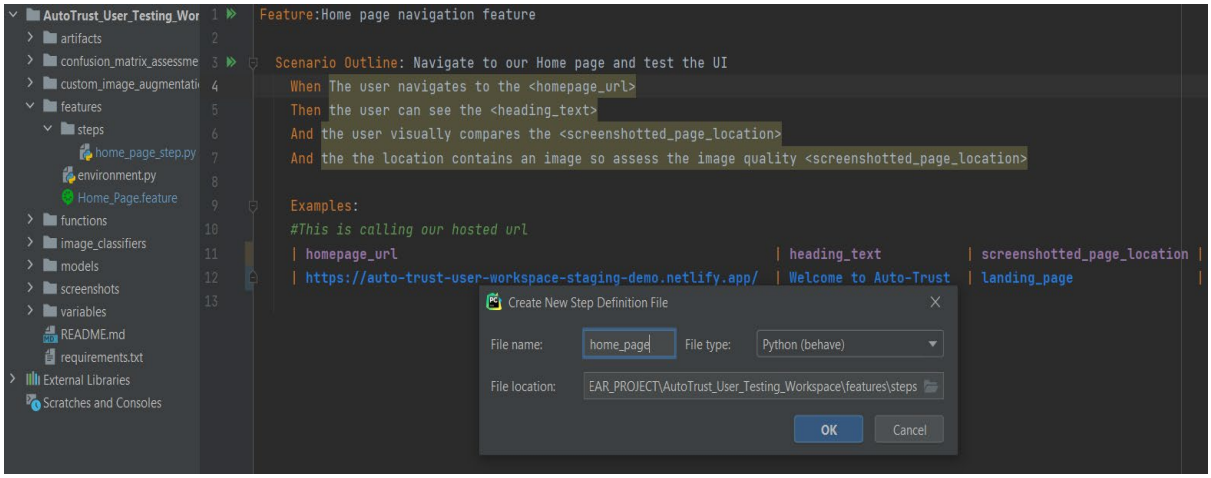


Fig. 7. The Hompage Gherkin feature file defining the steps to be taken and the parameters corresponding to these steps.

The scenario can then be extended to define other test cases by adding addition sought after values into the given parameters, as shown below:

```
Scenario Outline: Navigate to our Home page and test the UI
  When The user navigates to the <homepage_url>
  Then the user can see the <heading_text>
  And the user visually compares the <screenshotted_page_location>
  And the the location contains an image so assess the image quality <screenshotted_page_location>

  Examples:
  #This is calling our hosted url
  | homepage_url                                          | heading_text                | screenshotted_page_location |
  | https://auto-trust-user-workspace-staging-demo.netlify.app/  | Welcome to Auto-Trust       | landing_page                |
  | https://auto-trust-user-workspace-staging-demo.netlify.app/  | This where you can download | landing_page                |
  | https://auto-trust-user-workspace-staging-demo.netlify.app/  | Expand for more info        | more_info_section           |
  | https://auto-trust-user-workspace-staging-demo.netlify.app/  | If you have any queries     | landing_page                |
```

Fig. 8. The Hompage Gherkin feature file with added tests.

From the file structure shown above we can see that the steps directory is within our feature's directory, and this will allow Behave to execute these features steps. Once the steps file is created, we can then define the function that corresponds to this step. We can see in Fig. 9. how business logic defined in the feature is discovered in the steps file, where each step of the scenario is associated with function, and the business logic is transformed into back end developing logic.

```python
# This defines the step outlined from our feature file

@when("The user navigates to the (?P<homepage_url>.+)")
def homepage_navigation(context, homepage_url):
    """
        Description: navigates to the homepage url, to open for extension and de-couple operations in further
        development, simply change the 'homepage_url' param  to 'url', move to a file for global general steps,
        and you can then define this step in any scenario, in any feature for any specified url to navigate to.
        The same operations will be executed but on a different url.
        :param homepage_url: The url of the sites homepage.
        :return: executes execute_homepage_navigation() function which executes the request in teh browser.
    """
    execute_homepage_navigation(context, homepage_url)
    time.sleep(2)


@then("the user can see the (?P<heading_text>.+)")
def locate_welcome_text(context, heading_text):
    """
        Description: navigates to the page welcome text, using the imported
        welcome_text browser DOM Xpath defined in variables -> browser elements.
        :param heading_text: The expected heading/subheading to find in the browser, used for assertion.
        :return: test assertion of the found welcome_element text against the expected heading_text param.
    """
    element_found_by_xpath = find_element_by_xpath(context, welcome_element)
    time.sleep(3)
    text_found = element_found_by_xpath.text
    print(text_found)
    assert text_found == heading_text
```

Fig. 9. First half of the Hompage_steps file defining the fuctions that correspond to each step of the outlined user Scenario

```python
""" when an element is found, we then need to screenshot the current browser state and store it in our
/screenshots/browser_screenshot_outputs directory. Then we need to compare the image, executed in the
compare_page_location_similarity function, e.g., compare(pre_defined_homepage_image1, new_browser_homepage_image2)"""

@step("the user visually compares the (?P<screenshotted_page_location>.+)")
def compare_chosen_page_location_image(context, screenshotted_page_location):
    """
        Description: takes a screenshot of the current browser state at the specified page location.
        :param screenshotted_page_location: specified page location, used to tell the compare_page_location_similarity()
        function the name of the pre-defined screenshot that is to be used in comparison.
        :return: fail or pass if the compare_page_location_similarity() function fails or passes image comparison.
    """
    page_name = screenshotted_page_location
    screenshot(context, page_name)
    time.sleep(3)
    compare_page_location_similarity(context, page_name)


@step("the the location contains an image so assess the image quality (?P<screenshotted_page_location>.+)")
def assess_chosen_page_location_image_quality(context, screenshotted_page_location):
    """
        Description: uses the same screenshot of the page location captured in the last step, by defining the
        same pre-defined screenshot name, allowing the assess_and_classify_image_quality() to locate this image
        again in the browser_screenshot_outputs folder. Image Distortion Classification is then assessed.
        :param screenshotted_page_location: specified page location, passed to assess_and_classify_image_quality()
        function the name of the pre-defined screenshot that is to be used in comparison.
        :return: fail or pass if the assess_and_classify_image_quality() function fails or passes image comparison.
    """
    page_name = screenshotted_page_location
    screenshot(context, page_name)
    time.sleep(3)
    assess_and_classify_image_quality(context, page_name)
```

Fig. 10. Second half of the hompage_step file defining the fuctions that correspond to each step in the outlined user Scenario.

In Fig. 9 and Fig 10. above, the functionality behind each step can be seen in the description, accepted param, and return value.

To further elaborate an example of the **locate_welcome_text** step in Fig.9, in terms of the 'HelloWord' example mentioned earlier this is where are comparison of 'HelloWorld == HelloWorld' takes place. The comparison is '**assert text_found == heading_text'**, where the **heading_text** is the welcome text defined in our scenario, and the **text_found** is the text extracted from the browser. This text extracted is performed by calling the find_element_by_xpath function located in our main functions file, which utilizing Selenium's find_element_by_xpath function. Selenium Webdriver offers web-scraping functionality, which we perform on our driver of choice – **Chromedriver**.

As you can in Fig.11 below, we specify to Selenium that we would like to utilize the find_element_by_xpath function, and pass in desired element's XPath that would like to locate. This desired element's XPath is obtained manually through the browsers (DOM), as shown in Fig 12. below.

```
def find_element_by_xpath(context, element):
    by_xpath = 'find_element_by_xpath'
    try:
        """driver.implicitly_wait(10)
        found_element_value = WebDriverWait(driver, 10).until(
            lambda driver: driver.find_element_by(By.XPATH, wanted_element_value))"""
        found_element_value = WebDriverWait(context.browser, 10).until(lambda browser: getattr(browser, by_xpath)
        (element))
        context.exe = None
        return found_element_value
    except Exception as e:
        context.exe = e
        print('{} not found'.format(element))
```

Fig. 11. Finding a text element using selenium webdrivers by_xpath approach.

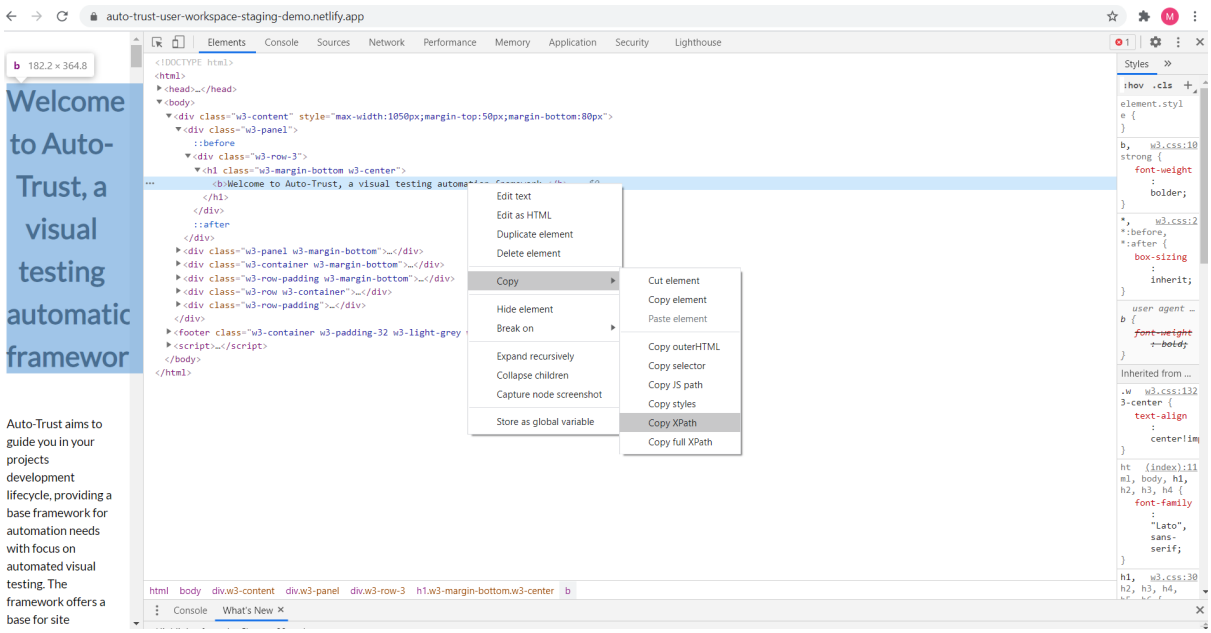


Fig. 12. Manually finding the welcome_element location in the DOM and copying its Xpath.

```
welcome_element = "/html/body/div/div[1]/div/h1/b"
```

Fig. 13. The browser XPath of the homepage welcome text.

Below are some of the other available functions offered by Senelium to locate web elements outlined in Seleniums doccumentation (Muthukadan, B., 2011.).

- To retrieve a single element by the requested element (finds the first instajnce of this element in the browser):
  - **find_element_by_id**
  - **find_element_by_name**
  - **find_element_by_tag_name**
  - **find_element_by_class_name**
  - **find_element_by_css_selector**
- To retrieve multiple elements by the requested element (finds all instances and their sub-tree element data):
  - **find_elements_by_id**
  - **find_elements_by_name**
  - **find_elements_by_tag_name**
  - **find_elements_by_class_name**
  - ***find_elements_by_css_selector***

Behave offers the ability to add an environmnet.py file to your features file, which will be discovered if present, which can be used to specify actions to take before or after the test executions. Below in Fig. 12. we have defined two behave funtions that are regognised by behave and therefore executed. In the before_all function we have told behave to to launch the chrome browser with the given arguments, such as the screen size, the port number to run on, or the 'headless' option argument which instructs the browser to run in the background and not display on the test executers screen. Removing this headless option is extreamly useful for testing debugging, allowing you to follow the actions of your automated tests such as what location of the page the test navigates to, if it inputs text into text fields correctly, or clicks the correct buttons.

```
# this is our environment file to work with our environment executables
from selenium import webdriver
from variables.app_variables import chrome_executable_path


def before_all(context):
    """ this will be called to set up our chrome driver environment before each test sequence """

    options = webdriver.ChromeOptions()
    options.add_argument("--test-type")
    options.add_argument("--headless")

    context.browser = webdriver.Chrome(chrome_executable_path, port=9515, options=options,
                                       keep_alive=False)
    context.browser.implicitly_wait(10)
    #context.browser.set_window_size(1920, 1080, context.browser.window_handles[0])
    context.browser.set_window_size(1528, 768, context.browser.window_handles[0])


def after_all(context):
    context.browser.quit()
```

Fig. 14. Chromedriver Installed on our local port 9515 for local testing.

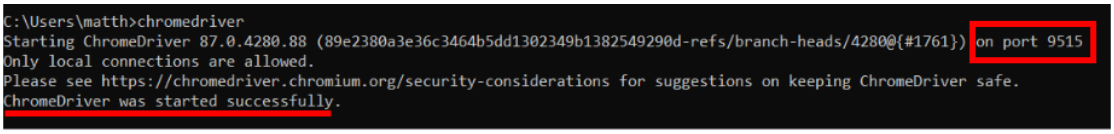


Fig. 15. Chromedriver Installed on our local port 9515 for local testing.

The domain URL is web page for Auto-Trust, where a description of the automated framework in place is described, the source code is accessible for download, and the methods of image comparison and distortion classification are described along with their importance. The Graphical user Interface (GUI) for this web page is outlined in the next section 2.4, as a 'Users Workflow Example domain' of their development, staging, or production environments. Up until later stages of development, this web page was an extremely basic web page until image distortion classification was integrated. Due to some unknown difficulties of implementing the model, and which requirements would potentially need to be updated based on what model was developed for what specific classification measures. In this sense, these difficulties accounted for streamlined the process of updating requirements after the model integration. Having a basic web page containing a basic image enhanced performance testing and error troubleshooting in these early stages of development, opening the door for future scaling.

As outlined in the project proposal, the original classification measures were that of an object detection model to detect and highlight the main elements of a user's site such as buttons, integrated into a tool that allows a user to input their own site location images/site elements to trained to detect these elements. This method was proposed before I knew much about how image classification and object detection worked on a low-level with regards to classifying image features/objects/edges etc. The model can be trained to detect a user's sites web elements, but I soon realized this was in terms of detection rather comparison.

Under this this initially proposed sense, model approach would be:

1. Specific to each user which I did not want after a later judgement in requirements engineering. I wanted a to have a final deliverable that was in fact closed, yet more open for expansion to different scenarios than a model that needed to be re-trained on new data when a new site feature is added, as would be needed with the initial approach. Even a deep understanding of how to apply transfer learning and fine-tune the model on new element data would not be possible since the model would be encapsulated within a tool. Again, the purpose of a tool was to de-couple these operations from the user, therefore this approach was slightly counter intuitive when it came to the need for expansion with ease.

2. Requires a lot of initial image setup data which would be very tedious for the user, even if the tool offered image augmentation to augment the image.

3. Using an image classifier in this way would classify the image features, and alert that the site location being testing is in fact a part of the user's site, returning a passed test result. This would end up passing all new captures from the browser even if an element of the site is missing, as it is essentially saying 'This is your site location, or part of your site location'. Only completely different sites would be alerted as failed.

A more complex model would have to be trained to detect differences rather than classify similarities. This would have to be a general model provided and not a model specific to a user's site, trained on a larger data and not encapsulated within a tool.

During more research and requirements engineering, I decided to restructure this approach, removing the tool aspect, and providing solely the framework aspect as a final deliverable. Since I proposed to detect element, text, or image distortions, I set out to implement element and text distortion classification by directly comparing exact pixel values vs. expected pixel values of areas of a user's site. The new model that was set to be implemented was a CNN that classifies image distortions, learning to detect distortion features of a given image. The development of element and text distortion classifier is described later in this Implementation section under the heading 'Integrated Classification Scripts', and the CNN can be seen further in this section under the heading "CNN Distortion Classifier Integration", and both their results of integration into an end workflow is described in "Tying all developed processes together into an automated testing CI/CD workflow".

Initial Data gathering

Before the data for the model training was acquired, I was inspired to take on the approach of image classification on distorted images after being inspired by an article that outlined how image distortion was applied to local areas of images to detect small areas of locally distortion within these images (Ahn, N., Kang, B. and Sohn, K., 2018.). The conductors of this implementation used these local distortions in terms of object detection, highlighting a bounding box around the areas of an image that is distorted, with a label attached displaying the type of distortion detected. After training and testing different fine-tuned models, their final implementation was finalized on the VGG-16, a large-scale CNN trained on a dataset of 14 million images, classes into 1000 classes (Simonyan, K., & Zisserman, A. 2015.). By fine-tuning this model, they passed in a dataset containing 8 different distortion types, with three variances applied to each type (Ahn, N., Kang, B. and Sohn, K., 2018.).

To gather the distorted image data for training my own model, I began searching for distorted datasets. I could not find any that had availability for academic research, so I decided to create my own by collecting a small image dataset and applying various levels of distortion to these images inspired to take a similar image distortion approach to (Ahn, N., Kang, B. and Sohn, K., 2018.).

This was achieved using Python libraries such as skimage to apply the distortions, numpy to manipulate and pre-process the images before applying distortion metrics, and opencv-python to save the images. This pre-processing was needing as skimage accepts input as a numpy array, which is an array of a fixed sized that cannot be altered after creation, unlike arrays known to programming languages such as python or java (The SciPy community., 2008.). The skimage numpy array accepts data types within a range such as float types (of range -1 to 1 or 0 to 1) or unit8 (of range 0 to 255) and can be seen implemented below in Fig. 14. In Fig.13. you can see the different distortion types offered by skimage that were applied to these images, defined as Gaussian Blur, Gaussian Noise, Salt and Pepper, and Sparkle, with three different levels of variance.

- Gaussian blur is a filter with a variable Gaussian kernel.
- Gaussian noise is distributed additive noise.
- Salt replaces random pixels with 1 and pepper replaces random pixels with 0.

- Speckle is defined as noise, where the output is gaussian noise with addition mean multiplied by an image size and a defined variance.

(Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors.)

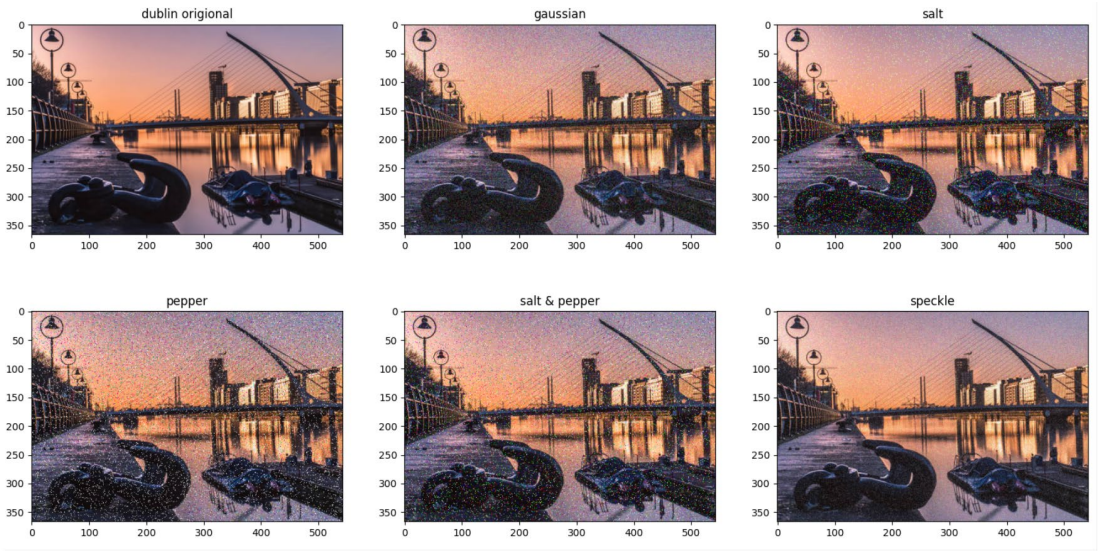


Fig. 16. Distortion applied to an origional high resolution image of dublin.

To apply this to a dataset, I collected the NRIA Holidays dataset which contained 516 images (Herve Jegou, Matthijs Douze, and Cordelia Schmid, 2008.)

This data was organized by labelling each image with a number, the type of distortion applied, and the applied level of distortion variance as they are written to a new folder, which can be seen in Fig.17. I used a command line argument parser to parse the arguments supplied in the command line when executing the command to run the script. This was implemented as a way of inputting a specified directory to retrieve the images from, and a directory to store the output images in, which allowed me to open this script up for use across my whole machine, which came in very handy later in development. I began processing 248 of the images from the dataset which gave me 12 levels of distortion for each image (4 types of distortion x 3 variances), and a total of 2976 images. The total time taken was two hours and 10 minutes. I did not capture this time with a screenshot as I did not add any time calculation output to this script. However, I did try speed up the process by adding multiprocessing workers which and the time is captured, explained further as part of testing in section 2.5.

```
saving prefix 2
saving prefix 3
processing image: 248
saving prefix 1
saving prefix 2
saving prefix 3
saving prefix 1
saving prefix 2
saving prefix 3
saving prefix 1
saving prefix 2
saving prefix 3
saving prefix 1
saving prefix 2
saving prefix 3
(env)
matth@LAPTOP-SUGFKJOK MINGW64
```

Fig. 17. Consol output for the itteration of applying distortion to 248 images.

```python
parser = argparse.ArgumentParser(description='Starting Image distortion generator')
parser.add_argument('-load_image_directory', default='./dataset/', type=str)
parser.add_argument('-directory_to_save_to', default='./distorted_dataset/', type=str)
args, _ = parser.parse_known_args()

if not os.path.isdir(args.load_image_directory):
    print(f'folder {args.load_image_directory} does not exist! '
          f'please provide existing folder in -load_image_directory arg!')
    exit()

if not os.path.isdir(args.directory_to_save_to):
    print(f'folder {args.directory_to_save_to} does not exist! '
          f'please provide existing folder in -directory_to_save_to arg!')
    exit()

print(f"Directory to retrieve the images: {args.load_image_directory}")
print(f"Directory to store new images: {args.directory_to_save_to}")

dataset_dir = args.load_image_directory
# may need change this later to save in separate directories specific to the distortion applied
#distorted_dataset_dir = "./distorted_dataset/"
distorted_dataset_dir = args.directory_to_save_to


# add various noise to an image with a specified variance
def add_noise(img, noise_mode):
    applied_noise_1 = None
    applied_noise_2 = None
    applied_noise_3 = None
    if noise_mode == "gaussian":
        # variance = rand(1,100)
        applied_noise_1 = util.random_noise(img, mode=noise_mode, clip=True, var=0.0125)
        time.sleep(0.02)
        applied_noise_2 = util.random_noise(img, mode=noise_mode, clip=True, var=0.025)
        time.sleep(0.02)
        applied_noise_3 = util.random_noise(img, mode=noise_mode, clip=True, var=0.00)
        time.sleep(0.02)
    elif noise_mode == "s&p":
        applied_noise_1 = util.random_noise(img, mode=noise_mode, clip=True, salt_vs_pepper=0.0125)
        time.sleep(0.02)
        applied_noise_2 = util.random_noise(img, mode=noise_mode, clip=True, salt_vs_pepper=0.025)
        time.sleep(0.02)
        applied_noise_3 = util.random_noise(img, mode=noise_mode, clip=True, salt_vs_pepper=0.05)
        time.sleep(0.02)
    elif noise_mode == "speckle":
        applied_noise_1 = util.random_noise(img, mode=noise_mode, clip=True, var=0.0125)
        time.sleep(0.02)
        applied_noise_2 = util.random_noise(img, mode=noise_mode, clip=True, var=0.025)
        time.sleep(0.02)
        applied_noise_3 = util.random_noise(img, mode=noise_mode, clip=True, var=0.05)
        time.sleep(0.02)
    return applied_noise_1, applied_noise_2, applied_noise_3


# gaussian blur with sigma variation
def gaussian_blur_noise(image):
    g_image1 = filters.gaussian(image, sigma=1.5, multichannel=False)
    g_image2 = filters.gaussian(image, sigma=3, multichannel=False)
    g_image3 = filters.gaussian(image, sigma=6, multichannel=False)
    return g_image1, g_image2, g_image3
```

Fig. 18. Method of adding three levels of variances distortions to each distortion type.

```python
# apply noise calls apply noise above
def distort_and_save_images(image_from_dataset, name):

    # gaussian blur
    gb_1, gb_2, gb_3 = gaussian_blur_noise(image_from_dataset)
    # gaussian noise
    gn_1, gn_2, gn_3 = add_noise(image_from_dataset, "gaussian")
    # salt and pepper noise
    sp_1, sp_2, sp_3 = add_noise(image_from_dataset, "s&p")
    # speckle noise
    s_1, s_2, s_3 = add_noise(image_from_dataset, "speckle")

    # apply_noise returns a floating-point image in the range
    # [0, 1] so we need to change it to 'uint8' with range [0,255]
    gb1, gb2, gb3 = convert_to_unit8(gb_1, gb_2, gb_3)
    # gaussian blur prefix and pathing
    gb_prefix = "_gaussian_blur"
    gb_path_to_save = distorted_dataset_dir + name + gb_prefix
    save_to_new_directory(gb1, gb2, gb3, gb_path_to_save)

    gn1, gn2, gn3 = convert_to_unit8(gn_1, gn_2, gn_3)
    # gaussian noise prefix and pathing
    gn_prefix = "_gaussian_noise"
    gn_path_to_save = distorted_dataset_dir + name + gn_prefix
    save_to_new_directory(gn1, gn2, gn3, gn_path_to_save)

    sp1, sp2, sp3 = convert_to_unit8(sp_1, sp_2, sp_3)
    # salt and pepper prefix and pathing
    sp_prefix = "_salt_and_pepper"
    sp_path_to_save = distorted_dataset_dir + name + sp_prefix
    save_to_new_directory(sp1, sp2, sp3, sp_path_to_save)

    s1, s2, s3 = convert_to_unit8(s_1, s_2, s_3)
    # speckle prefix and pathing
    s_prefix = "_sparkle"
    s_path_to_save = distorted_dataset_dir + name + s_prefix
    save_to_new_directory(s1, s2, s3, s_path_to_save)


# convert floating-point images
def convert_to_unit8(floating_1, floating_2, floating_3):
    # img = cv2.convertScaleAbs(image, alpha=255.0)
    converted_1 = np.array(255 * floating_1, dtype=np.uint8)
    converted_2 = np.array(255 * floating_2, dtype=np.uint8)
    converted_3 = np.array(255 * floating_3, dtype=np.uint8)
    return converted_1, converted_2, converted_3
```

Fig. 19. The process of calling distortions to be applied to a given image, and concerting the image to unit8 dta format.

```python
def save_to_new_directory(image_1, image_2, image_3, path_to_save):
    # designed to take three image distortions that were applied to a
    # single image, with the appropriate path and file name, then allocate
    # a final prefix to the appropriate image with a file extension
    prefix_1 = path_to_save + "_1.png"
    prefix_2 = path_to_save + "_2.png"
    prefix_3 = path_to_save + "_3.png"

    # saving the file with the original name, the new distortion
    # prefix to the new location, e.g "./distorted_dataset/image_1_gb_1.png
    # -> i.e the first variance of image one with gaussian blur applied
    cv2.imwrite(prefix_1, image_1)
    print("saving prefix 1")
    time.sleep(0.02)
    cv2.imwrite(prefix_2, image_2)
    print("saving prefix 2")
    time.sleep(0.02)
    cv2.imwrite(prefix_3, image_3)
    print("saving prefix 3")
    time.sleep(5)
    #cv2.waitKey(0)


i = 0

for image_in_dataset in os.listdir(dataset_dir):

    image_path = dataset_dir + image_in_dataset
    imported_image = Image.open(image_path)
    # convert to numpy array
    converted_to_np = np.asarray(imported_image)

    i += 1
    image_name = str(i)
    print("processing image: " + image_name)
    #image_name = os.path.splitext(image_in_dataset)[0]
    distort_and_save_images(converted_to_np, image_name)
```

Fig. 20. The process of looping through each file in a specified directory and calling distort_and_save_image.

I did not perform another iteration of these images as I also wanted to also perform data augmentation to these images. To do this, I created another script that applied various augmentation metrics, which can be seen in Fig.21. and the code implementation can be seen in Fig.22. which follows a similar approach to using an argument parser to specify directories to flow the data to and from but invokes different functions from the Keras library for data preparation and manipulation. The images are augmented with ImageDataGenerator from Keras which applied different augmentations to a specified number of batch iterations over the inputted data. Again, this data needed to be a numpy array of images.

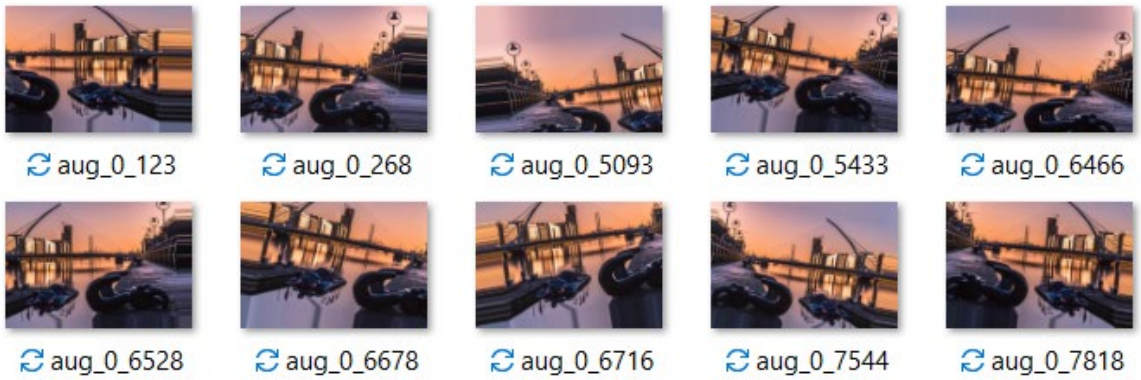


Fig. 21. Augmentations applied to an image of, labeled by the augmentation itterator with my own prefix of 'aug' for augmented and '0' as the image number in order or flowing from a directory.

```
parser = argparse.ArgumentParser(description='Starting Image distortion generator')
parser.add_argument('-load_image_directory', default='./dataset/', type=str)
parser.add_argument('-directory_to_save_to', default='./augmented_images/', type=str)

args, _ = parser.parse_known_args()

if not os.path.isdir(args.load_image_directory):
    print(f'folder {args.load_image_directory} does not exist! '
          f'please provide existing folder in -load_image_directory arg!')
    exit()

if not os.path.isdir(args.directory_to_save_to):
    print(f'folder {args.directory_to_save_to} does not exist! '
          f'please provide existing folder in -directory_to_save_to arg!')
    exit()

print(f"Directory to retrieve the images: {args.load_image_directory}")
print(f"Directory to store new images: {args.directory_to_save_to}")


dataset_dir = args.load_image_directory
# may need to save in separate directories specific to the distortion applied
#distorted_dataset_dir = "./distorted_dataset/"
distorted_dataset_dir = args.directory_to_save_to


def load_image(img):
    load_img(img)


# convert the loaded image to a 3d numpy array and expand samples
def convert_to_3d_numpy_array(loaded_img):
    data = img_to_array(loaded_img)
    sample_data = expand_dims(data, 0)
    return sample_data


# prepare iterator using .flow() to generate random batches of transferred images
def prepare_iterator(sample, directory, image_prefix):
    iterator = data_generator.flow(
        sample,
        save_to_dir=directory,
        save_prefix=image_prefix,
        shuffle=False)
    return iterator


data_generator = ImageDataGenerator(horizontal_flip=True,
                                    rescale=1/224.)
```

Fig. 22. Loading the image and preparingthe batch iterator for data augmentation.

```
# prepare iterator using .flow() to generate random batches of transferred images
def prepare_iterator(sample, directory, image_prefix):
    iterator = data_generator.flow(
        sample,
        save_to_dir=directory,
        save_prefix=image_prefix,
        shuffle=False)
    return iterator


data_generator = ImageDataGenerator(horizontal_flip=True,
                                    rescale=1/224.)

i = 0

for image_in_dataset in os.listdir(dataset_dir):
    time.sleep(0.02)

    image_path = dataset_dir + image_in_dataset
    imported_image = load_img(image_path)
    # convert to numpy array
    samples = convert_to_3d_numpy_array(imported_image)

    i += 1
    image_name = os.path.splitext(image_in_dataset)[0]
    print("processing image: " + image_name)

    prefix = image_name + "aug"
    prepared_iterator = prepare_iterator(samples, args.directory_to_save_to, prefix)

    # generate augmented images and save to augmented_images folder
    for i in range(10):
        batch = prepared_iterator.next()
        image = batch[0].astype('uint8')
```

Fig. 23. Invoking next batch of the itterator for the specified number of itteration (range(10)).

Final dataset Objectives

After building a suitable model for this, I was not happy with results, further elaborated in the next section – Building the Model. I re-visited requirements gathering in search for a distorted dataset again, and I finally found one that was available for academic research, the kadid19k dataset (Lin, H., Hosu V., Saupe D., 2008). The dataset contained 81 original images, with 25 different types of distortions applied to each image, and 5 levels of variance applied to each type. A total of 10,125 images in the dataset. Upon observation, the first two levels of very low variance applied did not show any difference, so I removed the first two levels of distortion variance for each type of distortion from the dataset. I also decided to group the 25 distortions applied of related distortion types into the 7 categories (classes): blurred, brightness, colour distortion, JPEG compression, noise, sharpness and contrast, and spatial distortion. This was achieved within Jupyter Notebook, using the Glob Python library to access only specified file name within a directory and then the Shutil Python library to move each of these files into a new directory. This data was then split up into train, validation, and test directories, for training, validating, and testing my model. The final number of images in each category can be seen in the table below.

Grouped by distortion	**Type of distortion**	**Number of images produced after arrangement**
blur	- Gaussian blur - Lens blur - Motion blur	729
Brightness change	- Brighten - darkness - mean shift	729
colour distortion	- Colour diffusion - Colour shift - Colour quantization - Colour saturation 1 - Colour saturation 2	1215

JPEG compression	- JPEG2000 standard compression - JPEG standard compression	486
noise	- Gaussian white noise - Gaussian white noise in colour component) - Impulse noise - Multiplicative noise - Denoise	1215
sharpness and contrast	-High-sharpen -Contrast change	486
spatial distortion	- Jitter - Non-eccentricity patch - Pixelate - Quantization	1215

Fig. 24. Organization of distortion by new category name given to a grouped set of distortion types, by similarity of type, condensed from Kadid10k dataset (Lin, H., Hosu V., Saupe D., 2008).

I then needed to acquire high resolution images to add as an addition class to the above classes. I found 800 high resolution images from the DIV2K dataset (EiriKu, A., Radu, T, 2017.). For consistency measures, I removed 81 images and added the 81 original high-resolution images from the Kadid10k dataset (Lin, H., Hosu V., Saupe D., 2008.). This consistency would allow me to have these images within each image quality class, so that the model would have a slightly difficult time classifying them, at time, but with a result that is more accurate regards to being able to classify the image quality of an image rather than the objects within the image. The DIV2K dataset also came with 100 images for validating the model.

Due to the large gap of data between classes, I then ran the necessary classes through my image augmentation script, applying one basic level of augmentation to each image – horizontally flipping each image, duplicating each image. For training the model which is further discussed in the next section. The data was split into batches of training, validation, and testing, where it was essential that there were no duplicates in each, bar testing batches which were randomly sampled from both training and validation. The training and validation were split based on the 20/80 split methodology provided by the DIV2K dataset on the high-resolution images. The 80/20 split is a widely used split that is closely coupled to the learning curve of Artificial Intelligence and echoes the Pareto principle, an aphorism which asserts that 80 percent of outcomes (or output) come from 20 percent causes (or results) (The Data Detective, 2020.). Testing was split into 10 percent of random samples from both training and validation. The table below shows the amount of data in each class after applying augmentation and splitting the data.

Distortion category/class	**No. of images after augmentation (100%)**	**Training split (80%)**	**Validation split (20%)**	**Testing split (10% random samples)**
blur	1458	1166	292	146
brightness change	1458	1166	292	146
colour distortion	1215	972	243	122
JPEG compression	972	778	194	97
noise	1215	972	243	100
sharpness and contrast	972	778	194	122
spatial distortion	1215	972	243	97
high resolution	1600	1600	200	160

Fig. 25. Data split after data augmentation was applied to ccategories.

Note that this table applied the 80/20 split for training and validation across different amounts of data but does not fully apply the 80/20 split across an even amount of data across all classes. This was a problem in implementation explained further in the next section that needed immediate requirements engineering which is why I have decided to include it in implementation, a key part of my requirements engineering process.

```
print('# moving all Blurs, with intensities 03, 04, 05')
for name in glob.glob('./kadid10k/*_0[1-3]_*[3-5].*'):
    print(name)
    shutil.move(name, 'organized/blur')

print('# moving all colour distortions, with intensities 03, 04, 05')
for name in glob.glob('./kadid10k/*_0[4-8]_*[3-5].*'):
    print(name)
    shutil.move(name, 'organized/color_distortion')
```

```
# moving all Blurs, with intensities 03, 04, 05
./kadid10k\I01_01_03.png
./kadid10k\I01_01_04.png
```

Fig. 26. Using glob wildcards to specify the type and level of distortion to be moved to a new directory location.

Building the model

The first model that was implemented was a custom-built Sequential CNN. The first step is to load the images from the train, validation, and test directories, using the Keras library to manipulate the image size, e.g. (224x224), and the images are once again stored in a numpy array which can be seen in Fig. as train_batches, valid_batches, test_batches. These batches are defined by the order of your classes within each directory as an array, i.e., blur =0, brightness change = 1, which gives the model information on what class data is in what directory, so it knows each class location when training, validating, and testing the data.

```
train_path = './train'
valid_path = './validation'
test_path = './test'

train_batches = ImageDataGenerator(preprocessing_function=preprocess_input) \
    .flow_from_directory(directory=train_path, target_size=(224,224),
        classes=['gaussian_blur', 'gaussian_noise',
        'salt_and_pepper', 'sparkle'], batch_size=10)

valid_batches = ImageDataGenerator(preprocessing_function=preprocess_input) \
    .flow_from_directory(directory=valid_path, target_size=(224,224),
        classes=['gaussian_blur', 'gaussian_noise',
        'salt_and_pepper', 'sparkle'], batch_size=10)

test_batches = ImageDataGenerator(preprocessing_function=preprocess_input) \
    .flow_from_directory(directory=test_path, target_size=(224,224),
        classes=['gaussian_blur', 'gaussian_noise',
        'salt_and_pepper', 'sparkle'], batch_size=10)
```

Fig. 27. Gathering and storing our training, validation, and test classes as batches to be passed to our model.

The first model built was a sequential CNN, invoking the Keras Sequential model. The first layer is a Convolutional layer (Conv2D) that accepts the image input, and we specify the input shape of our data, which is 224x224x3 because we specified the target size to be 224x224 when we created the batches, and they were saved in Red, Green. Blue (RGB) format which has 3 colour channels. We the specify the number of filters applied in this layer. We also specify the ReLU

activation function, which, as a high-level explanation, is a linear function that will output direct positive or negative results if the input data is positive or negative, i.e., 1, or 0. We want these images to have Zero Padding so that the dimensionality of the images is not reduced in this layer, achieved by adding pixels to surrounding areas of the models input when this input is greater than the accepted input width, allowing for integers of 'zero' to be added to the surrounding area while being processed. If padding is added, the dimensionality of the input data is not affected, whereas the as the counter approach would not add padding, but instead drop off the remaining input from the input data of the feature map that does not fit the specified input width, e.g., if the input data is 12 and the accepted input width is 10, the remaining 2 in ignored. Zero padding helps around image edges, where the feature map can very easily outbound the image size as it iterates over the image pixels.

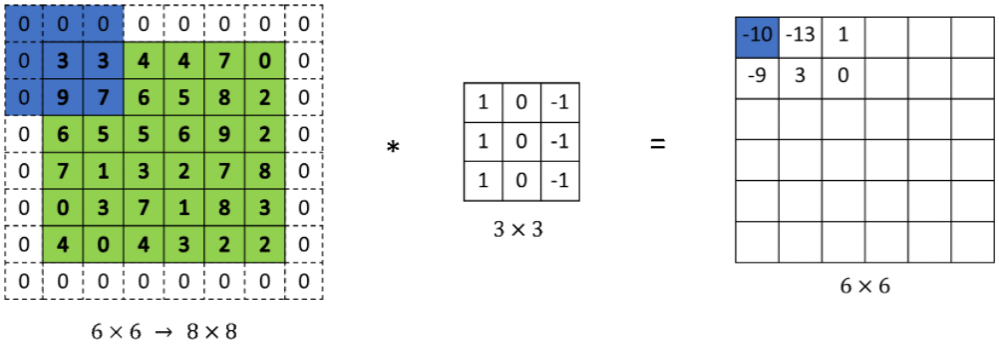


Fig. 28. A more comprehensive look at Zero Padding being applied using a 3x3 filter (unknown, datahackers.rs, 2018).

The next layer added was a Max Pooling (Pooling) layer that down sampled the dimensions after the data has been passed from the first Conv2D layer and cuts the dimensions in half, specified by the strides value equal of 2. Another Conv2D and Pooling layer is added, yet this time the Conv2D layer has 64 filters, doubling the filter size of the first Conv2D layer. Filter numbers does not have to be doubled but it is a common practice as more convolutional layers are added, the feature map on the first and early convolutional layers detects lines, and edges, and as the feature map increases it maps a larger area of the original image, detecting more global image values such as objects, faces, animals. (Ramesh, S., 2018.)

The next layer added was the Flattening layer, to flatten the data into a single container/one-dimensional tensor, which is then passed to our Dense layer. In Fig. below it was set to 4, when we had 4 classes within my original custom distortion dataset or gaussian blur, gaussian noise. The activation function is set to 'softmax' to convert this one-dimensional tensor of predictions of probability distribution (Brownlee, J., 2020). The number of units and the activation function are very important here.

```
model = Sequential([
    Conv2D(filters=32, kernel_size=(3, 3), activation='relu',
        padding = 'same', input_shape=(224,224,3)),
    MaxPool2D(pool_size=(2, 2), strides=2),
    Conv2D(filters=64, kernel_size=(3, 3), activation='relu',
        padding = 'same'),
    MaxPool2D(pool_size=(2, 2), strides=2),
    Flatten(),
    Dense(units=4, activation='softmax')
])

model.summary()

#specifying a low learning rate
model.compile(optimizer=Adam(learning_rate=0.0001),
    loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(x=train_batches,
    steps_per_epoch=len(train_batches),
    validation_data=valid_batches,
    validation_steps=len(valid_batches),
    epochs=10,
    verbose=2
)
```

Fig. 29. Sequential CNN layered architechture, summary function, compile and fit the model for training and validation.

The output of running **model.summary()** is the architecture of the model built.

```
Model: "sequential_28"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_69 (Conv2D)           (None, 224, 224, 32)      896
_________________________________________________________________
max_pooling2d_67 (MaxPooling (None, 112, 112, 32)      0
_________________________________________________________________
conv2d_70 (Conv2D)           (None, 112, 112, 64)      18496
_________________________________________________________________
max_pooling2d_68 (MaxPooling (None, 56, 56, 64)        0
_________________________________________________________________
flatten_28 (Flatten)         (None, 200704)            0
_________________________________________________________________
dense_34 (Dense)             (None, 6)                 1204230
=================================================================
Total params: 1,223,622
Trainable params: 1,223,622
Non-trainable params: 0
```

Fig. 30. Custom Sequention CNN outputted summary.

Compiling the model is then performed by **model.compile,** where the Keras optimizer was set to 'Adam', which is based on adaptive estimation, and the learning rate was set to 0.0001 (default is 0.001), a slow learning rate to yield better results. A larger learning rate is of course faster, but the trade-off is accuracy (Brownlee, J., 2019.).

To begin training, the **fit** function was invoked on the model. I specified the number of steps to take per epoch to specify when one full epoch is declared complete, which is how many batches of data need to be sampled from each of our classes within our batches, i, e., if you have 500 samples and 10 is the batch size your specified, the steps-per-epoch should be set to 10. A work-around for this was setting the steps-per-epoch to the length of the training batches. Setting the value for the validation steps is done in the same fashion.

When the model was run on my custom distortion dataset, the results were great to see on my first ever training and I was very happy with as it achieved a training accuracy of 99 percent and validation accuracy of 86 percent.

```
Epoch 2/10
300/300 - 738s - loss: 1.2338 - accuracy: 0.8760 - val_loss: 0.8621 - val_accuracy: 0.8889
Epoch 3/10
300/300 - 635s - loss: 0.4690 - accuracy: 0.9363 - val_loss: 0.8309 - val_accuracy: 0.9042
Epoch 4/10
300/300 - 624s - loss: 0.1433 - accuracy: 0.9720 - val_loss: 0.9415 - val_accuracy: 0.8833
Epoch 5/10
300/300 - 682s - loss: 0.1028 - accuracy: 0.9797 - val_loss: 0.5449 - val_accuracy: 0.9139
Epoch 6/10
300/300 - 708s - loss: 0.1191 - accuracy: 0.9767 - val_loss: 0.6898 - val_accuracy: 0.8944
Epoch 7/10
300/300 - 687s - loss: 0.1067 - accuracy: 0.9770 - val_loss: 0.4963 - val_accuracy: 0.9167
Epoch 8/10
300/300 - 652s - loss: 0.0728 - accuracy: 0.9833 - val_loss: 0.5640 - val_accuracy: 0.9194
Epoch 9/10
300/300 - 652s - loss: 0.0075 - accuracy: 0.9967 - val_loss: 0.4772 - val_accuracy: 0.9236
Epoch 10/10
300/300 - 756s - loss: 0.0307 - accuracy: 0.9937 - val_loss: 1.1670 - val_accuracy: 0.8639

<tensorflow.python.keras.callbacks.History at 0x1de10e2b3a0>
```

Fig. 31. Results of training the CNN for the first time.

However, when testing this model using a small training batch, the results were not as high as the validation accuracy shown in the model training output, which can be seen in the confusion matrix in Fig.32. below that was constructed by plotting the confusion matrix with the Python matplotlib.pyplot library. After research, I have concluded that this occurred because of overfitting, when the model begins to learn the detail and noise in the data that is passed in early layers of the model and learns these so well to the extent that it negatively impacts the performance of the model when new testing data is introduced (Brownlee, J., 2016.*).* This is when I reverted to requirements engineering and began looking for a new dataset, discovering the Kadid10k dataset described in the previous section.

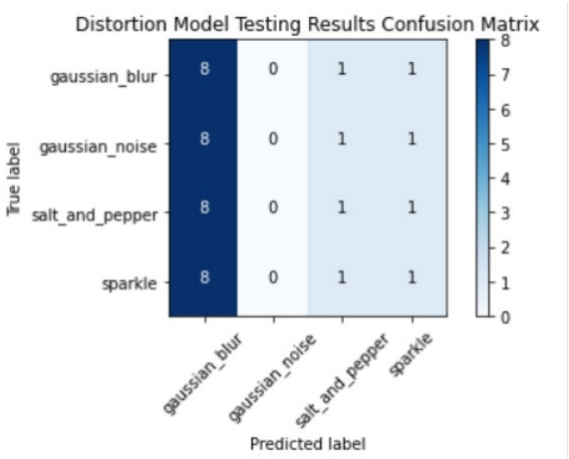


Fig. 32. Predictions confusion matrix output plotted with matplotlib.pypot.

```
predictions = model.predict(x=test_batches, steps=len(test_batches), verbose=0)
```

Fig. 33. How to invoke predictions from the model, specifying the batches of data to enavuate, the steps, and the verbosity of logging to log to the consol output.

Instead of running the Kadid10k dataset on the same model, I wanted to increase accuracy by applying transfer learning methods to a successful CNN. I originally planned to apply transfer learning to the VGG-16 model as this was the model that was used in the article that inspired me to classify direct image distortion, however I decided to apply fine-tuning to the MobleNet CNN over VGG-16 due to its architecture and definition as a low-latency, low-power model that can perform the same classification tasks as high-latency, high-power models (Wang, W., Hu, Y., Zou T., Liu, H., 2020.). VGG-16 has a size of 553 megabytes (MB) and a total parameter of 138 million, whereas MobleNet has a size of 17 MB, and 4.2 million parameters (deeplizard, 2020).

Organizing the batches of data was done in a similar fashion, except this time I utilized the MobleNet prepressing input function with the image data generator (Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H., 2017).

```
train_path = './train'
valid_path = './validation'
test_path = './test'

train_batches = ImageDataGenerator(
    preprocessing_function=tf.keras.applications.mobilenet.preprocess_input) \
    .flow_from_directory(directory=train_path, target_size=(224,224),
    classes=['blur', 'brightness', 'color_distortion', 'compression',
    'high_resolution', 'noise', 'sharpness_and_contrast',
    'spatial_distortion'], batch_size=32)

valid_batches = ImageDataGenerator(
    preprocessing_function=tf.keras.applications.mobilenet.preprocess_input) \
    .flow_from_directory(directory=valid_path, target_size=(224,224),
    classes=['blur', 'brightness', 'color_distortion', 'compression',
    'high_resolution', 'noise', 'sharpness_and_contrast',
    'spatial_distortion'], batch_size=32)

test_batches = ImageDataGenerator(
preprocessing_function=tf.keras.applications.mobilenet.preprocess_input) \
    .flow_from_directory(directory=test_path, target_size=(224,224),
    classes=['blur', 'brightness', 'color_distortion', 'compression',
    'high_resolution', 'noise', 'sharpness_and_contrast',
    'spatial_distortion'], batch_size=32)
```

Fig. 34. Different approach to generating batches of data from using the method of MobleNets pre-processing input.

MobleNet has a total of 88 layers (Wang, W., Hu, Y., Zou T., Liu, H., 2020.). This is a lot of layers, and additionally fine-tuning comes with experience and I gained a lot of experience from trial and error. I got the best output by freezing the sixth-to-last layer, and then specifying that I only want the last 23 layers in the model to be trainable. Luckily, I was able to save a lot of trail and error by taking this suggested fine-tuned architecture of MobleNet from an online tutorial (deeplizard, 2020).

```
# remove the sixth-to-last layer
x = mobile_model.layers[-6].output

# passing all previous layers up to the sixth-to-last
# layer as our output layer
output = Dense(units=6, activation='softmax')(x)

# using this new output, which also defines
# the previous layers, as our model output
model = Model(inputs=mobile_model.input, outputs=output)

# now freezing the last 23 layers
for layer in model.layers[:-23]:
    layer.trainable = False
```

Fig. 35. Fine tuning MobleNet as a feature extractor in a new model.

The new fine-tuned model output structure was:

```
_________________________________________________________________
conv_dw_13_bn (BatchNormaliz (None, 7, 7, 1024)        4096
_________________________________________________________________
conv_dw_13_relu (ReLU)       (None, 7, 7, 1024)        0
_________________________________________________________________
conv_pw_13 (Conv2D)          (None, 7, 7, 1024)        1048576
_________________________________________________________________
conv_pw_13_bn (BatchNormaliz (None, 7, 7, 1024)        4096
_________________________________________________________________
conv_pw_13_relu (ReLU)       (None, 7, 7, 1024)        0
_________________________________________________________________
global_average_pooling2d_2 ( (None, 1024)              0
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 6150
=================================================================
Total params: 3,235,014
Trainable params: 1,869,830
Non-trainable params: 1,365,184
```

Fig. 36. The last few layers of the new 82 layered model with the last 23 layers trainable. 1,869,830 total trainable params.

I was getting mixed results with initial runs, testing alterations such as two, or 5 epochs, increasing the learning rate, and batch size. After analysis these mixed results, I could see that many images with applied distortions; brightness change, sharpness, and spatial were getting mixed results and some were even being classified as high resolution. I realized that the unequal amount of data in each class was a factor, causing the model to naturally bias towards some distortion classes and causing incorrect classifications to smaller data classes such as sharpness and contrast (Chatterjee, S., 2018.)

I then discovered something else that changed my whole process for the better and streamlined the future implementation of building the right distortion classifier. Since I was following a similar implementation to the article that inspired the idea for direct image classification, who conducted their experiment specifying that they only applied noise, blur, and compression as distortion measures. I returned to requirements engineering and data requirements and decided began remove any instances of distortion that were not classified as noise, blur, and compression, by removing these classes and running small tests on the MobleNet model to inspect the difference. I removed brightness change, then sharpness and contrast, and I did not have to remove spatial distortion as the results immediately revealed very high accuracy. The approach of incrementally removing each class one by one allowed me to keep spatial distortion and colour distortion in my dataset as extra classes of distortion for my model to detect, and still yield very high accuracy. I then reduced the data in each class to contain for even amounts of data and following an exact 80/20 split between training and validation, totalling 980 images in each training class, 180 in each validation class, and 10 percent random samples totalling 90 images in each testing class. This data was run through my first model, my Sequential CNN, with 20 epochs and a low training rate again of 0.0001, training result in Fig.38., below. I also wanted

to retrieve more reproduceable results when testing, which can be offset by the fact that "neural network algorithms are stochastic", utilizing randomness in their operations such as initializing weights, dropouts, and optimizations, which naturally opens the case for different results to be yielded from different tests on the exact same data (Brownlee, J., 2019). The seed for the random number generation within these operations can be set to a fixed value to overcome this, so I set the random seed of my environments - the Python hashed operating system seed to 0, numpy's random seed and Python's random seed to 123, and TensorFlow's random seed to 1234. These values were obtained as standard recommended values from TensorFlow's documentation (TensorFlow. 2018.).

```
# operating system seed to 0
os.environ['PYTHONHASHSEED'] = '0'
# numpy's random seed to 123
np.random.seed(123)
#Python's random seed to 123
python_random.seed(123)
#TensorFlow's random seed to 123.
tf.random.set_seed(1234)
```

Fig. 37. Setting environment variables random seeds to creating more repoducable results.

```
Epoch 12/20
540/540 - 309s - loss: 0.1142 - accuracy: 0.9750 - val_loss: 0.2038 - val_accuracy: 0.9528
Epoch 13/20
540/540 - 308s - loss: 0.0803 - accuracy: 0.9817 - val_loss: 0.3289 - val_accuracy: 0.9111
Epoch 14/20
540/540 - 308s - loss: 0.0472 - accuracy: 0.9907 - val_loss: 0.1654 - val_accuracy: 0.9713
Epoch 15/20
540/540 - 308s - loss: 0.0155 - accuracy: 0.9969 - val_loss: 0.2264 - val_accuracy: 0.9519
Epoch 16/20
540/540 - 305s - loss: 0.0061 - accuracy: 0.9993 - val_loss: 0.1394 - val_accuracy: 0.9796
Epoch 17/20
540/540 - 308s - loss: 0.0016 - accuracy: 0.9998 - val_loss: 0.1525 - val_accuracy: 0.9769
Epoch 18/20
540/540 - 309s - loss: 0.1399 - accuracy: 0.9669 - val_loss: 0.2360 - val_accuracy: 0.9481
Epoch 19/20
540/540 - 307s - loss: 0.0612 - accuracy: 0.9867 - val_loss: 0.1700 - val_accuracy: 0.9602
Epoch 20/20
540/540 - 309s - loss: 0.0415 - accuracy: 0.9896 - val_loss: 0.1413 - val_accuracy: 0.9694

<tensorflow.python.keras.callbacks.History at 0x1ce64cb25b0>
```

Fig. 38. Training output new Sequential CNN on the final, most sucessful dataset structure, customized Kadid10k dataset of each class components: training:900, validation:180. A 96.94 percent validation accuracy.

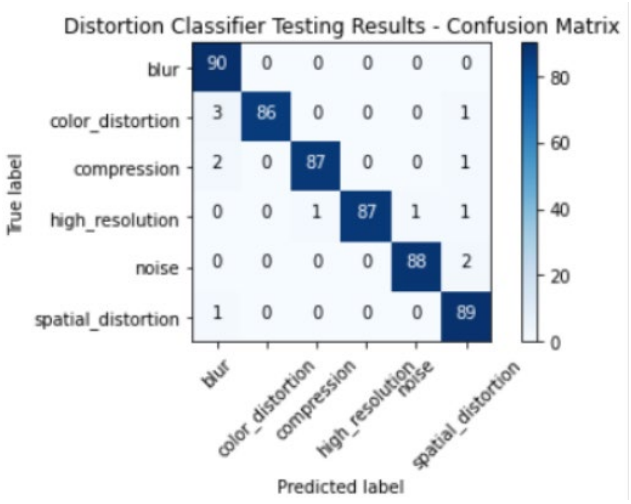


Fig. 39. Confusion matrix output for testing results of customized kadid10k data on custom Sequential CNN. 90 images in each distortion class.

```
image = load_img('./test/high_resolution/0801.png', target_size=(224, 224))
# convert the image pixels to a numpy array
image = img_to_array(image)
# reshape data for the model
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
# prep the image
image = preprocess_input(image)
```

```
# get extracted features
features = model.predict(image)
print(features.shape)

for i in features:
    print([i][0][3])
```

```
(1, 6)
0.999848
```

Fig. 40. An example of loading a single high resolution image, and retreiving a distortion classification from our final model implementation.

Integrating Classification Scripts

As mentioned earlier, the features file defines the value of the **site location** that the user would like to navigate to as a parameter, e.g., the landing_page. The last two steps shown outlines the process of taking a screenshot of the current browser location and then pass the name of this **site location** to the functions that encapsulate our image classification scripts. The process of both integrations is described below:

CNN Distortion Classifier Integration.

The process is as follows:

- The user specifies the page location within the scenario as a parameter.
- The next step then screenshots the page location which has been navigated to, and the page location parameter is passed to the **assess_and_classify_image_quality** function, which uses the specified page location to filter through the browser screenshot outputs directory. Essentially the page location parameter is a label used by the user in the scenario to is discovered the pre-defined screenshot by naming convention within the test execution

- The new screenshot I then passed into the distortion classification model and a prediction is returned. A plotted confusion matrix is then saved as an image offering visual evaluation of results to the user.

The code snippets below further elaborate this process with code comments. The variables used in this process are also shown for reference. These variables are stored in a separate folder for separation of concerns and global usage.

```
#chrome path for local testing
chrome_executable_path = 'C:/webdrivers/chromedriver.exe'
# chrome path for for circle-ci
#chrome_executable_path = '/usr/local/bin'
pre_defined_screenshot_path = './screenshots/pre_defined_screenshots/'
screenshot_results_path = './screenshots/browser_screenshot_outputs/'
failed_comparisons_path = './screenshots/failed_comparisons/'
failed_file_extension = "_failed_comparison.png"
png_file_extension = ".png"
image_quality_distortion_model_path = './models/Sequential_Image_Distortion_Classifier.h5'
confusion_matrix_assessment_path = './confusion_matrix_assessment/model_classes/'
assessment_high_resolution_assessment_path = './confusion_matrix_assessment/model_classes/high_resolution/'
confusion_matrix_output_path = './confusion_matrix_assessment/confusion_matrix_output/'
matrix_extension = '_confusion_matrix.png'
artifacts_path = './artifacts/'
```

Fig. 41. Global variables used within scripting.

```
# classify the image quality with the CNN Distortion classifier,
# called from our test script functions
def classify_image_quality(screenshotted_image_path, image_name):
    image_path = screenshotted_image_path
    # load image as RBG with defined dimensions for model
    loaded_image = load_img(image_path, target_size=(224, 224))

    img_array = im.img_to_array(loaded_image)
    img_array_expanded_dims = np.expand_dims(img_array, axis=0)
    image = tf.keras.applications.mobilenet.preprocess_input(img_array_expanded_dims)

    feature_prediction = model.predict(image)
    np.round(feature_prediction)
    print('feature prediction model output shape: ' + str(feature_prediction.shape))

    final_prediction = None

    for class_predictions in feature_prediction:
        prediction_list = []
        for class_prediction in class_predictions:
            prediction_list.append(class_prediction)
        max_prediction_location = prediction_list.index(max(prediction_list))

        classes = ['blur distortion', 'color distortion', 'JPEG compression',
                   'high resolution', 'noise distortion', 'spatial distortion']

        final_prediction = classes[max_prediction_location]
        print('image features are classified as {}'.format(final_prediction))

    if final_prediction != 'high resolution':

        # write image to the high resolution folder to be assessed in a confusion matrix
        new_image_path_for_confusion_matrix = os.path.join(assessment_high_resolution_assessment_path,
                                                           image_name + png_file_extension)

        image_to_store_in_high_res_folder = cv2.imread(screenshotted_image_path)
        cv2.imwrite(new_image_path_for_confusion_matrix, image_to_store_in_high_res_folder)
        create_confusion_matrix(image_name)

        assert final_prediction != 'high resolution', \
            "The captured image labeled '{}' in your pre-defined images has failed " \
            "image quality assessment for distortion. Distortion of type '{}' was " \
            "classified instead of 'high resolution'".format(image_name, final_prediction)


def assess_and_classify_image_quality(context, page_name):
    image_name = page_name
    screenshotted_image = os.path.join(screenshot_results_path, image_name + png_file_extension)
    # assess the image that was screenshotted from the browser
    classify_image_quality(screenshotted_image, image_name)
```

Fig. 42. Classifying the image quality with the CNN distortion classifier, desipher prediction and output this prediction.

```
def plot_and_save_confusion_matrix(image_name, cm,
                                   classes, title=''):
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.get_cmap('Blues'))
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    print('Performing Confusion matrix of image quality assessment')

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label for each feature class')
    plt.xlabel('Predicted label of image quality/distortion')

    path_to_save_matrix_plot = confusion_matrix_output_path + image_name + matrix_extension
    print('For local testing, saving matrix at the location: ' + path_to_save_matrix_plot)

    confusion_matrix_artifacts_path = artifacts_path + image_name + matrix_extension
    print('For CI/DC server integration, saving matrix at the location: ' + confusion_matrix_artifacts_path)

    plt.savefig(path_to_save_matrix_plot)
    plt.savefig(confusion_matrix_artifacts_path)


# crates the confusion matrix and calls plot_and_save_confusion_matrix above.
# the image name is passed for saving the
def create_confusion_matrix(image_name):
    test_path = confusion_matrix_assessment_path
    test_batches = ImageDataGenerator(preprocessing_function=preprocess_input) \
        .flow_from_directory(directory=test_path, target_size=(224, 224),
                             classes=['blur', 'color_distortion', 'compression',
                                      'high_resolution', 'noise', 'spatial_distortion'],
                             batch_size=10, shuffle=False)

    assessment_predictions = model.predict(x=test_batches, steps=len(test_batches), verbose=0)

    cm = confusion_matrix(y_true=test_batches.classes, y_pred=np.argmax(assessment_predictions, axis=-1))

    matrix_title = "Distortion Classifier Confusion Matrix Assessment for '{}'".format(image_name)
    plot_labels = ['blur', 'color', 'compression', 'high res', 'noise', 'spatial']
    plot_and_save_confusion_matrix(image_name, cm=cm, classes=plot_labels, title=matrix_title)
```

Fig. 43. Creating, plotting and storing the confusion matrix.

Element and Text Distortion Comparison Integration.

The process is as follows:

- The user specifies the page location within the scenario as a parameter.
- The current browser location is screenshotted, and the page location parameter is passed to the **compare_page_location_similarity**.
- The new screenshot taken by the browser is then located by naming convention and these two screenshots are compared by converting them to grayscale using the opencv-python library, and a similarity score is then calculated between the two images using the **structural_similarity** of the skimage Python library. If the similarity is less that 1.0 (1:1 ratio), then the co-ordinates of the then the browser screenshot image are used to insert a red bounding box around these co-ordinates and store the failed image in the failed comparisons folder.

The code snippets below further elaborate this process with code comments.

```
from variables.app_variables import screenshot_results_path, png_file_extension, \
    failed_comparisons_path, pre_defined_screenshot_path, failed_file_extension, artifacts_path


def compare_page_location_similarity(context, image_name):
    test_failed = False

    # temporarily using a distorted pre-defined screenshot to compare against the captured
    # screenshot, as the browser currently displays a non-distorted (true) image.
    pre_defined_screenshot = os.path.join(pre_defined_screenshot_path, image_name + png_file_extension)
    image1 = cv2.imread(pre_defined_screenshot)
    screenshotted_image = os.path.join(screenshot_results_path, image_name + png_file_extension)
    image2 = cv2.imread(screenshotted_image)

    # convert the images to grayscale
    image1_grayscale = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
    image2_grayscale = cv2.cvtColor(image2, cv2.COLOR_BGR2GRAY)

    # passing the two grayscale images into compare_ssim to find the
    # Structural Similarity Index (SSIM)
    # use structural_similarity instead of compare_ssim for skimage==0.18.1
    (ssim_score, diff) = structural_similarity(image1_grayscale, image2_grayscale, full=True)
    ssim_score_percentage = ssim_score * 1
    print("Browser element comparison Similarity Score: {}% image accuracy. "
          "Similarity score is not 100%.".format(ssim_score_percentage))
```

Fig. 44. Element and distortion classifier compare_page_location function to detect image difference and product a similarity score.

```
# if the similarity ratio is less that 1:1
if ssim_score < 1.0:
    test_failed = False
    # diff is represented as a floating point data type in the range [0,1]
    # as it was returned from compare_ssim, so we must convert the array to
    # 8-bit unsigned integers in the range [0,255] before we can use it with cv2
    image_diff = (diff * 255).astype("uint8")

    # threshold the difference image, followed by finding contours to obtain the regions
    # of the two input images that differ by comparing the difference
    threshold = cv2.threshold(image_diff, 0, 255, cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
    contours = cv2.findContours(threshold.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    contours = imutils.grab_contours(contours)

    for contour in contours:
        area = cv2.contourArea(contour)
        if area > 20:
            # compute the bounding box of the contour
            (x, y, w, h) = cv2.boundingRect(contour)
            # now drawing the bounding box to highlight the difference area,
            # un-comment image1 below for further debugging if necessary
            # cv2.rectangle(image1, (x, y), (x + w, y + h), (0, 0, 255), 2)
            cv2.rectangle(image2, (x, y), (x + w, y + h), (0, 0, 255), 2)

    # save the image
    failed_image_path = os.path.join(failed_comparisons_path, image_name + failed_file_extension)
    cv2.imwrite(failed_image_path, image2)
    time.sleep(1)

if test_failed:
    failed_comparison_to_artifacts_path = artifacts_path + image_name + failed_file_extension
    print('For CI/DC server integration, saving page element comparison at the location: '
          + failed_comparison_to_artifacts_path)
    cv2.imwrite(failed_comparison_to_artifacts_path, image2)
    time.sleep(1)
    # fail the current test
    assert ssim_score == 1.0, \
        "The captured image labeled '{}' in your pre-defined images has failed " \
        "image quality assessment. The binary image pixels do not match," \
        "an element on your site must not be visible.'".format(image_name)
```

Fig. 45. Actions to take if the similarity score is less than an exact comparison of 100 percent similarity (100 percent = 1.0).

Continuous Integration Server Implementation.

Continuous integration servers for CI/CD are used to speed up development by offering a pipeline for testing that executes tests and notifies on success or failures, allowing tests to be automated or even scheduled.

Jenkins as a Continuous integration server

In recent years Jenkins has become less popular, but it is still one of the most trusted open-source CI/CD tools for professionals with huge community help (Munesti, T., 2020.). For this reason, I chose to use Jenkins.

There were many challenges with Jenkins throughout development. Jenkins runs on a local server until it is provided with a public URL, such as applying one directly into the Jenkins application running locally or hosting it in a virtual machine instance on a public cloud provider like Amazon Web Services. To integrate with GitHub, I needed a publicly accessible domain that these two systems communicate through Secure Socket Shell (SHH) encryption keys. After many issues with this process, I got the two systems integrated by using a port tunnelling service which ran my local server as public Domain name System (DNS) by adjusting URL configurations within Jenkins. This service only lasted 8 hours or when the port tunnelling application is shut down, which was a problematic initialization to do each time I used Jenkins, as it additionally had to be the domain name set it GitHub for the 'GitHub webhook' to allow SHH communication. There was also a major issue with running Jenkins on Windows which I attempted to fix for a large portion of development, which was the communication between Jenkins as an old Microsoft Windows 32-bit system installation, and the new Windows environment variables such as my Python Path needed for my Jenkins' job execution. I tried everything that I could find from the community support and was determined to get this working, but I eventually moved this process to an Ubuntu Virtual Machine which worked instantly. Ubuntu is a distribution of Linux running on a Linux kernel. Jenkins' pipeline for continuous integration consists of a project build architecture, defining General Steps in the build such as environment variables to set, Source Control Management where your workspace GitHub would be defined with SHH credentials for the webhooks, Build Triggers such as defining GitHub code change Pull Requests to be accepted and execute the build, Build Environment defines the environment such as timeout after failed test etc, Build defines the commands to run such as install requirements or run behave, and Post Build Actions defines files to dump to Jenkins after the build, or to send email notifications.

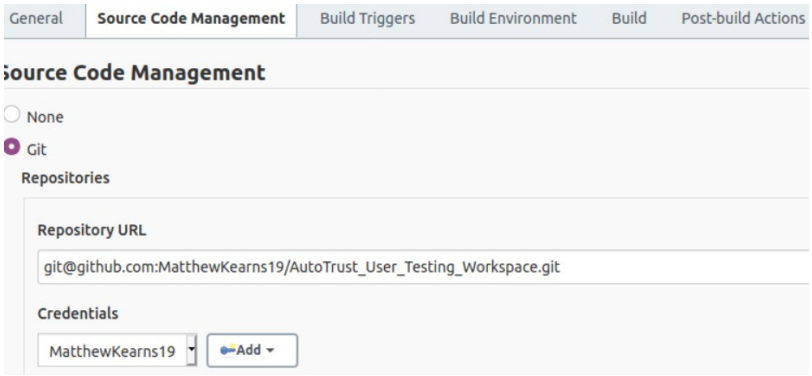


Fig. 46. Jenkins job architechture.

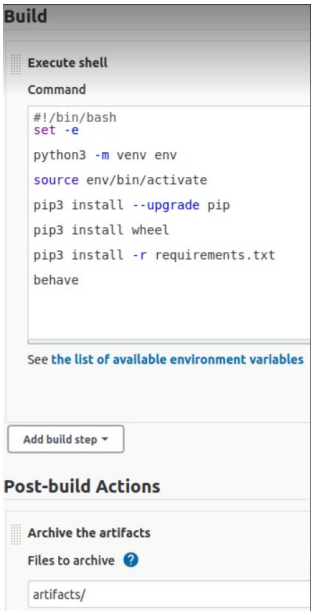


Fig. 47. Jenkins job command line steps to take during the build, to execvute the environmnet, and Post-Build Actions to dump all the artifact files.

Circle-Ci as a continuous integration server.

Because of the issues I was having with Jenkins, Circle-CI was introduced as a new way to optimize automated testing through a full CI/CD workflow. I chose Circle-Ci as it is a modern architecture, that support many interesting features such as a smooth interface, free-tier package and allows parallel job executions which I wanted to utilize. It also offers instant caching of dependencies to save load time and credit on each build, and the ability to configure your jobs to execute inside pre-defined docker containers. At a high-level definition, a Docker container is a lightweight architecture of dependencies encapsulated in a container offering access to its features at runtime (Merkel, D. 2014). The Circle-Ci build is also connected through GitHub, and the steps are defied in a YAML file, which is stored in your project and Circle-CI can locate this and execute the build. At the beginning of Circle-CI integration I used these Docker containers, but later in development I finalized with the following structure in my YAML:

```yaml
version: 2.1

orbs:
  browser-tools: circleci/browser-tools@1.1.3

workflows:
  main:
    jobs:
      - build-and-test

jobs:
  build-and-test:
    docker:
      - image: cimg/python:3.8

    steps:
      - checkout
      - run: curl -L -o google-chrome.deb https://dl.google.com/linux/ \
            direct/google-chrome-stable_current_amd64.deb
      - run: sudo apt-get update ; sudo apt-get install fonts-liberation \
            libasound2 libatk-bridge2.0-0 libatk1.0-0 libatspi2.0-0 libcairo2 \
            libcups2 libgbm1 libgtk-3-0 libpango-1.0-0 libxkbcommon0 xdg-utils
      - run: sudo dpkg -i google-chrome.deb
      - run: sudo sed -i 's|HERE/chrome\"|HERE/chrome\" \
            --disable-setuid-sandbox|g' /opt/google/chrome/google-chrome

      - run: python3 -m venv env
      - run: . env/bin/activate
      - run: python3 -m pip install --upgrade pip
      - restore_cache:
          key: -v3-requirements-{{ checksum "requirements.txt" }}
      - run: pip install -r requirements.txt
      - save_cache:
          key: -v3-requirements-{{ checksum "requirements.txt" }}
          paths:
            - "env"
      - run: behave

      - store_artifacts:
          path: root/AutoTrust_User_Testing_Workspace/artifacts
          destination: artifact-file
```

Fig. 48. Circle-Ci YAML file.

There was a lot of trail and error with this and the order in which to activate a virtual environment, and upgrade 'pip' (pythons instillation package manager) before installing the requirements.txt file which holds my dependencies for the project. Because of the download time of some of the larger dependencies such as TensorFlow, I configured a cache called -v3-requirements and I dump my dependencies in after each build., allowing them to be cached to speed up build time on subsequent builds.

Tying all processes together into an automated testing CI/CD workflow.

This section will describe the end deliverables as a framework to execute automated testing and discover passed or failed results. To begin, the basic scenario that was outlined in ''Understanding the BBD environment setup in the automated testing framework' had explained how the expected text on our landing page would be asserted. If the expected text is not equal to the text found by the script executing in the browser, the following assertion will fail, outputting a failed test.

```
  File "features\steps\home_page_step.py", line 27, in locate_welcome_text
    assert text_found == heading_text
AssertionError
```

Fig. 49. Assertion error resulting in test failure.

What happens if the above test passes the when the valid expected text is found, yet there is surrounding text, or a button that is not rendering correctly and the test still passes? That is why we need more in-depth comparison.

Let us see a solution – If we include the Element and Text Comparison script, we can see that the original screenshot captured from the browser has been stored in the directory 'browser_screenshot_outputs' and labelled by the appropriate parameter specified in the test scenario. We can also see the same label in the pre-defined screenshot directory. After comparison we were able to capture a similarity score of 99 percent which is a failed result. This highlighted a paragraph HTML tag that was not closed correctly in development with a red bounding box as it was not part of the pre-defined screenshot. An example of the accidental leaking of unclosed HTML tags into the deployment environment being captured by the test.

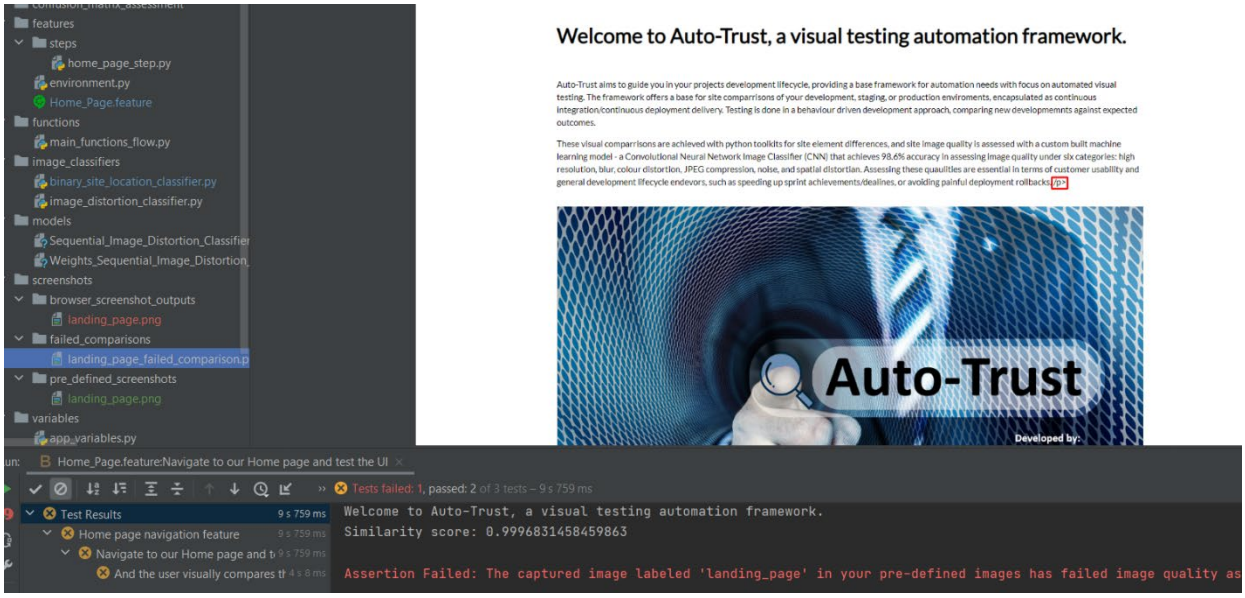


Fig. 50. IQA using the compare_page_location_similarity function, and the successfull capture of a text element that was leaked throught to developemnt by-passeing basic site assertion tests.

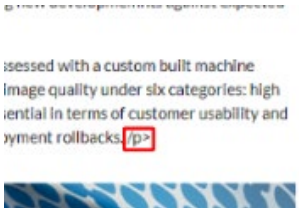


Fig. 51. A closer view of the failed unclosed paragraph tag result.

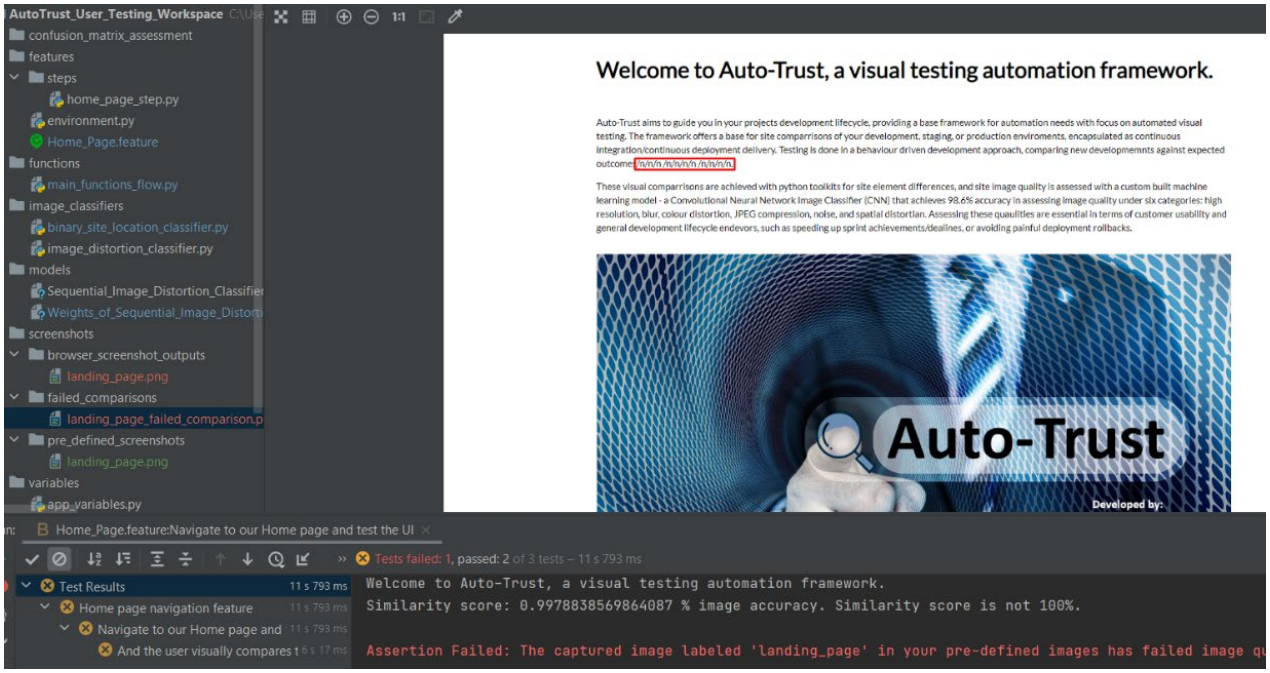


Fig. 52. Example of what poor decoding/encoding of text may look like that was not rendered in the environment. Our script also caturing this.

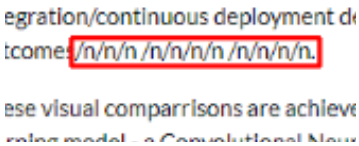


Fig. 53. A closer look at the result of poor decoding/encoding highlighted.

Now what happens if an image in the environment is marked as failed because its quality has been distorted, and a poor simmilarity score marked as failed under test inspection. We now know that the image is being highlighted but we may not be able to figure out why, or perhapse we can see that it is distorted but we do not know what type of distortion is occuring to begin troubleshooting potential root causes. If we invoke the script containing the process of our distortion classification CNN, we can yeild the following result:

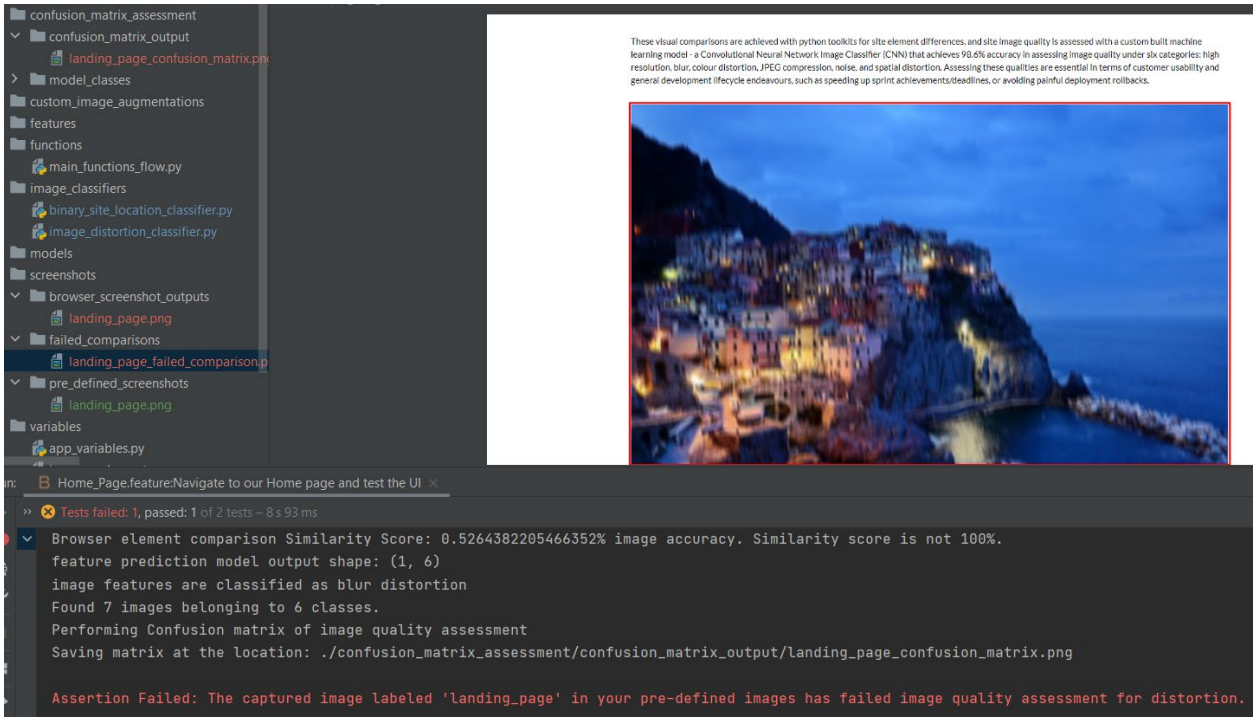


Fig. 54. IQA using the distortion classifier to predict image quality. Output highlighted as ‘blur distortion’.

This process was then integrated into the CI/CD pipelines of Jenkins and Circle-CI. By creating a single artifacts file in the final stretch of implementation, I was able to instruct my script to store the necessary images in the artifact’s directory, to be dumped by the continuous integration server after the build is finished. In Fig.55. below you can see the user-friendly interface for the Circle-CI dashboard, notifying builds that were executed after a new code change was pushed to the workspace environment on GitHub. If builds do not pass, the merging of these new changes will be disabled. If the build passed, a green tick would appear, and the changes are them okay to merge to be released.

Fig. 55. IQA using the distortion classifier to predict image quality. Output highlighted as ‘blur distortion’.

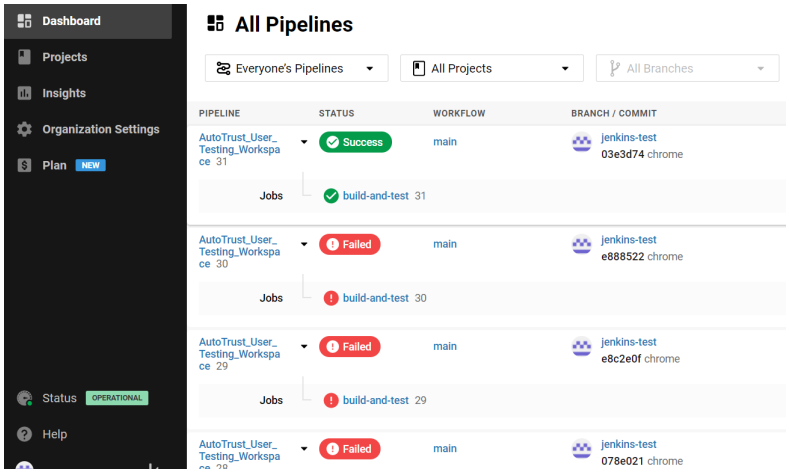


Fig. 56. Circle-Ci dashboard showing builds that have failed and builds that have passed.

I was unable to fully implplement this process fully as I was getting issues with a sub-library that was being used within the image grayscale conversion. Chromedriver was successfully installed and configuered in the behave setup, and Behave executing the test in the browser, taking a screeshot of the site but was getting stuck when the sub-library just mentioned was invoked. I was unfortunately unable to bypass this so for demonstratin purposes I just had the Image distoprtion classifier executed by specifying the pre-defined landing page image to be compared. The Job succeeded and it was of course classified as high resolution as can be seen in the build steps in Fig.57.

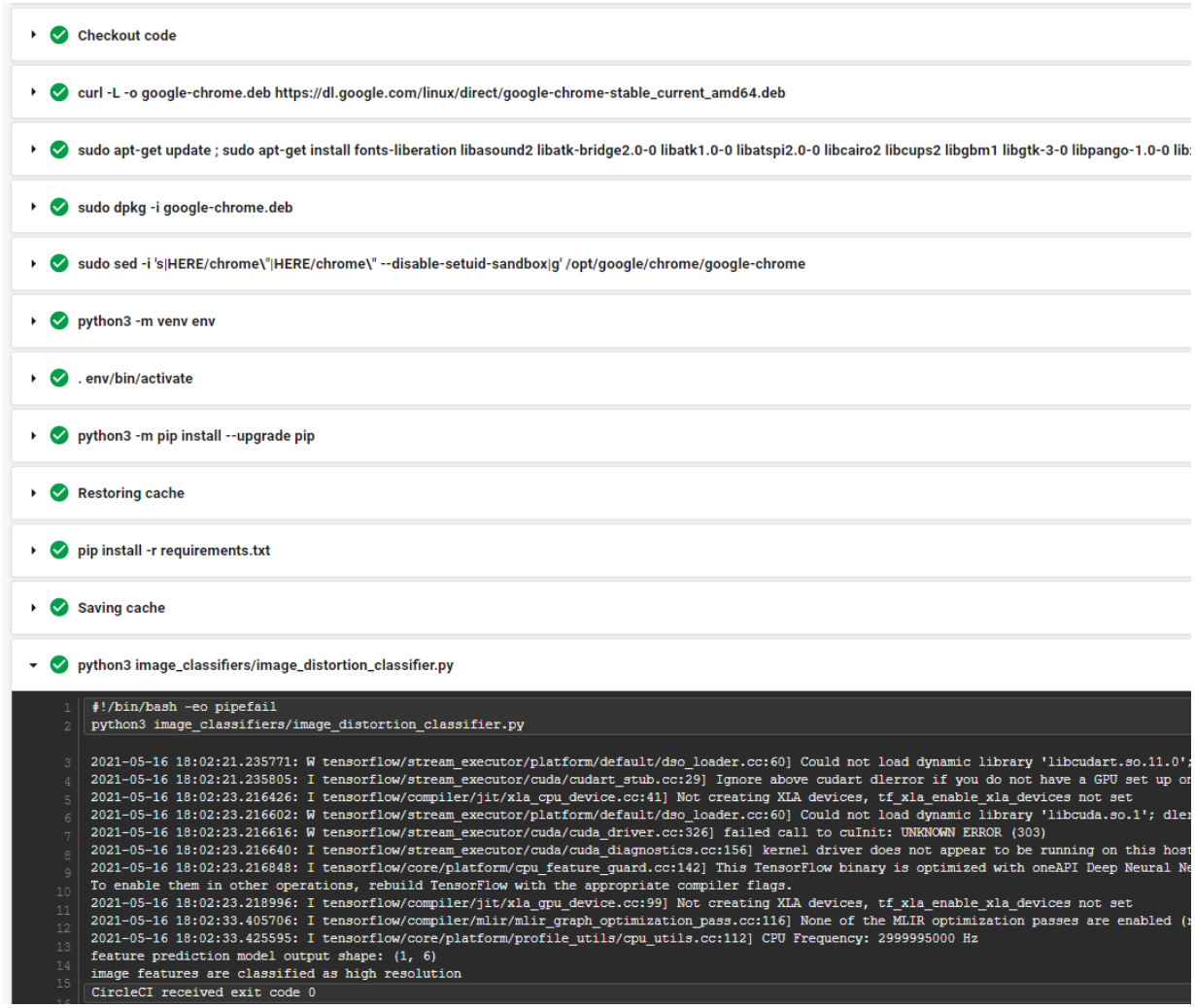


Fig. 57. Circle-CI showing the steps of the seccessful build.

Another implementation issue I ran into, the build did not pick up on the artifacts that were stored in the artifacts folder. After following tutorials and documentation, the examples or storing text files would success, but as soon as I applied the exact same configurations of approach to my case with images, they artifacts could not be found.

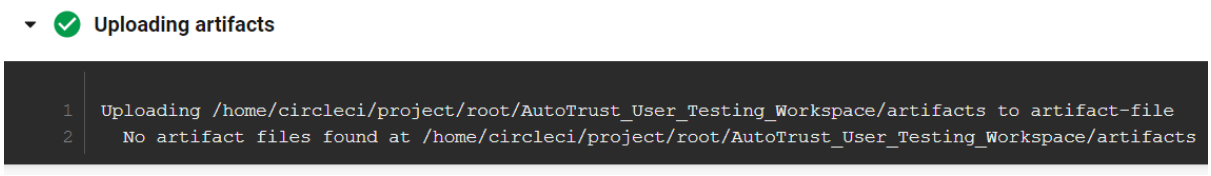


Fig. 58. Circle-CI showing the steps of the seccessful build.

Regards jenkins, I could not get the chromedriver environment variable path configured properly, however unlike Circle-Ci, the Post-Build Actions were able to store artifacts, tested by manually inputting a sample image into the artifcats folder (since the build would fail before it could launch the webdriver), as seen in Fig.59.

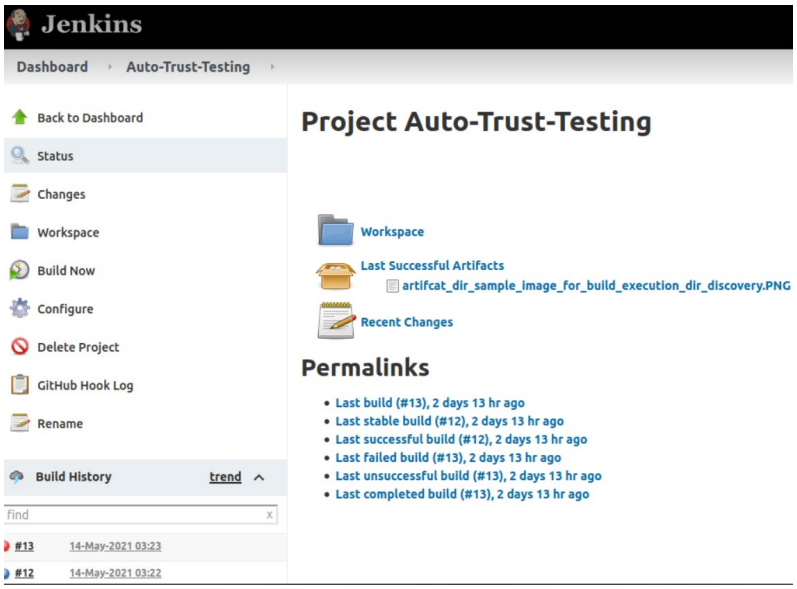


Fig. 59. Jenkins storing 'Last successful Artifacts' beimng stored, noted successful in terms of image uoload, not build step success.

Noticeable errors concluded

When a Larger area of the size is missing, or if the screenshot is taken too early by not invoking the function to sleep for a second between navigating to the page and taking the screenshot, it causes a whole page shift, which flags the whole image as differentiated from the original. This is not good for the project results but can be taken in as a future requirement to only highlight smaller areas and if a larger amount of distortion is detected, to ignore it, to be handled by another more complex script or CNN.

Fig. 60. Consusion in image difference asessement between two grayscale, exact images cause by a lareg shift in the page when an element was removed.

2.4. Graphical User Interface (GUI)

The GUI for this project outlines a user's typical workflow to achieve the features and results offered by Auto-Trust's automation framework. The GUI screens for the user consist of the user's workspace where their end application is hosted, their testing workspace where their automation framework lies, and their continuous integration and continuous delivery through GitHub, Jenkins, and Circle-CI. The GUIs in this section demonstrate the user's interactions with these tools and technologies of the framework. The Auto-Trust example workspace is demonstrated below, and can be found at: https://auto-trust-user-workspace-staging-demo.netlify.app/

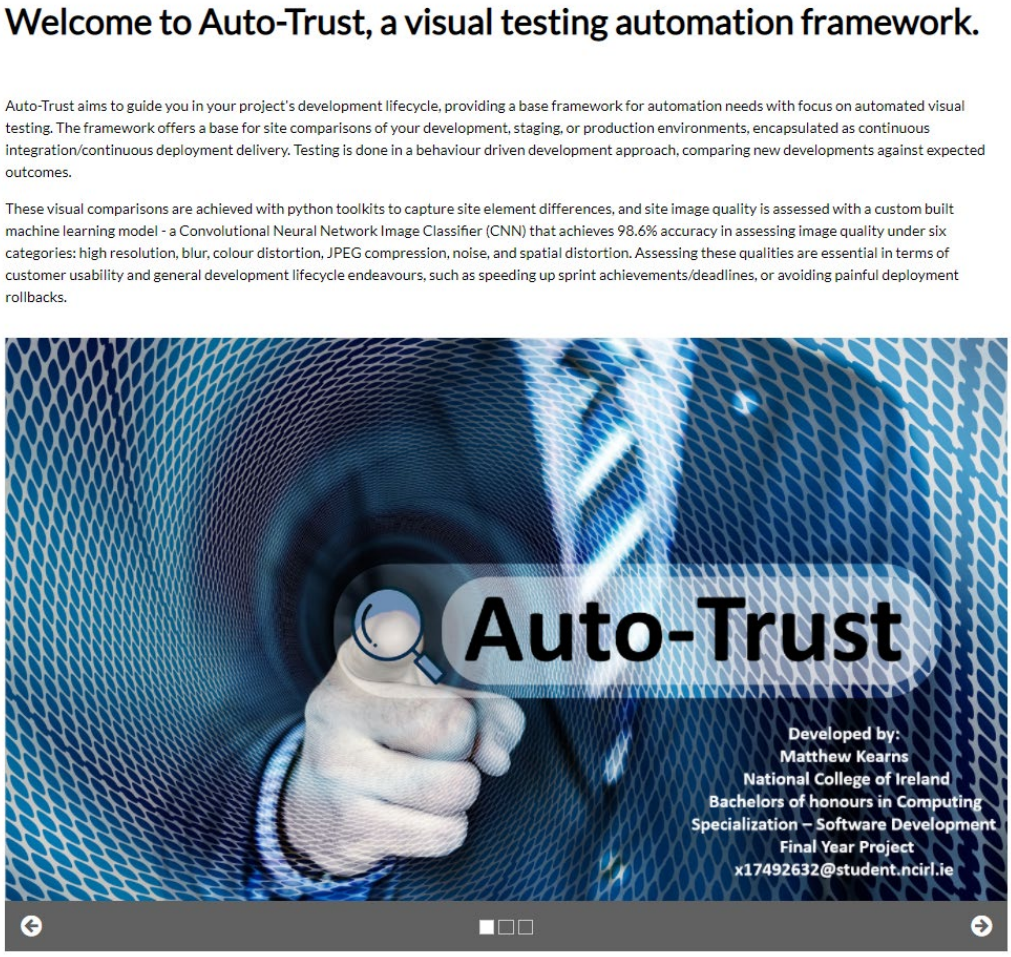


Fig. 61. Final site landing page of an end users workspace. This example is this is a doccumentation of Auto-Trust and the importance of what it offers.

Fig. 62. Final site of an end users workspace.

Fig. 63. Final site of an end users workspace.

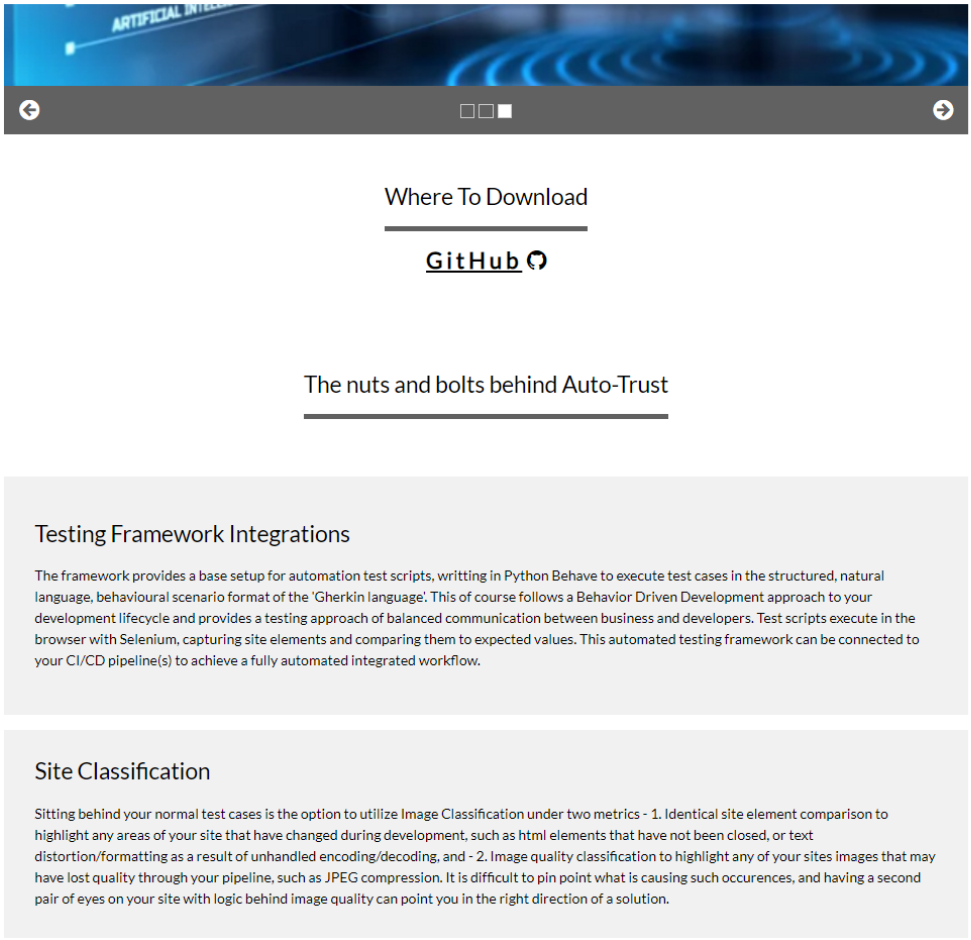


Fig. 64. Final site of an end users workspace – 'Where to download' the framework, and 'The nuts and bolts of Auto-Trust'.

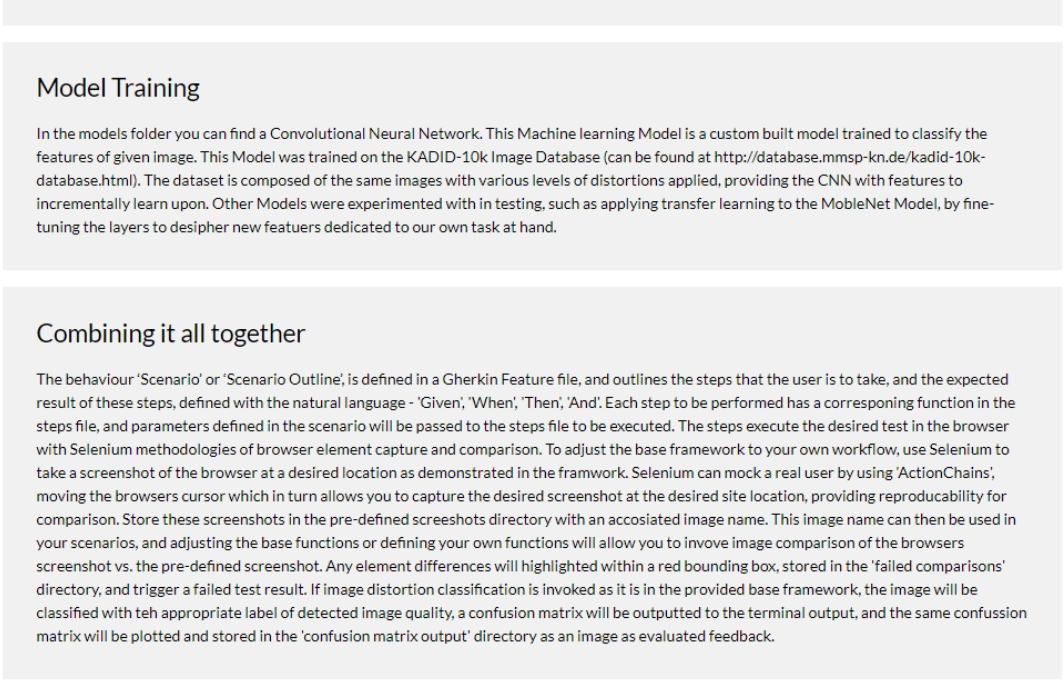


Fig. 65. Final site of an end users workspace - 'The nuts and bolts of Auto-Trust'.

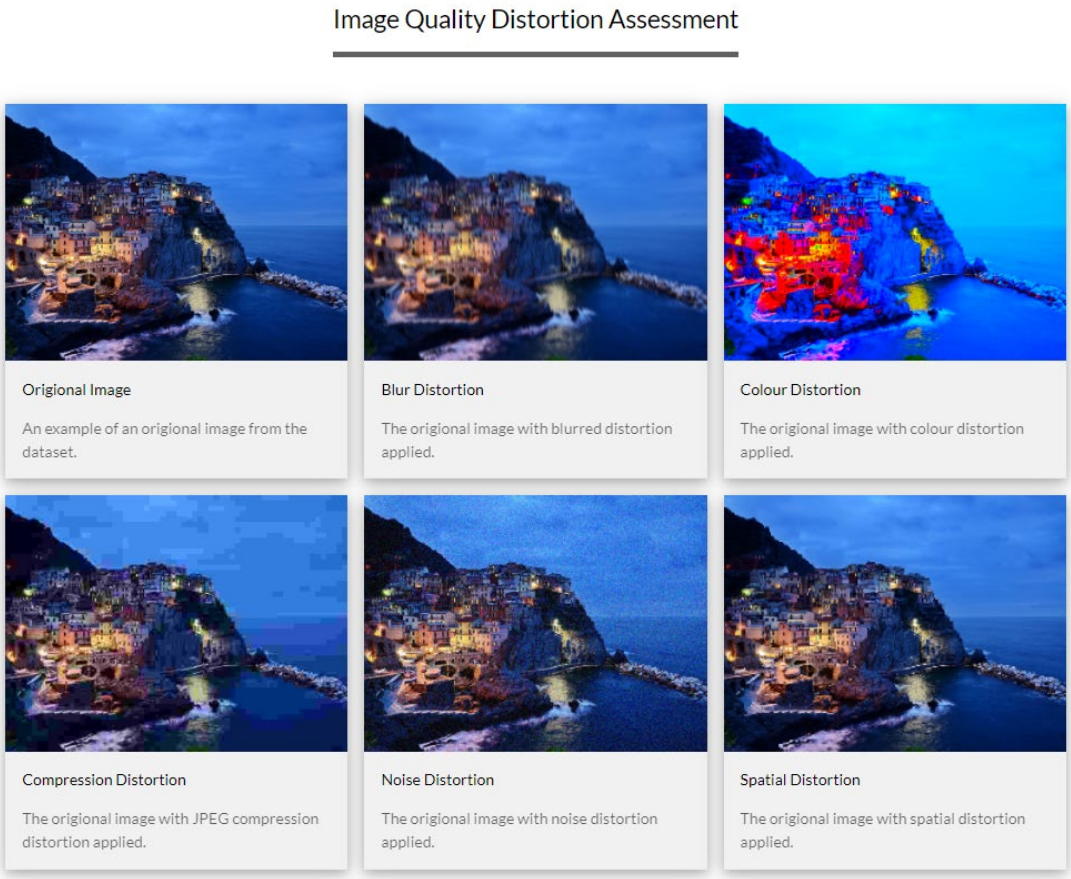


Fig. 66. Final site of an end users workspace – Image quality distortion sssessment categories.

Blurs:

- Gaussian blur
- Lens blur
- Motion blur

Color distortions

- Color diffusion
- Color shift
- Color quantization
- Color saturation 1
- Color saturation 2

Compression

- JPEG2000 standard compression
- JPEG standard compression

Noise

- Gaussian white noise
- Gaussian white noise in color component)
- Impulse noise
- Multiplicative noise
- Denoise

Spatial distortions:

- Jitter
- Non-eccentricity patch
- Pixelate
- Quantization

Back To The Top

Citation of the Dataset used in training the Convolutional Neural Network:

Lin, H., Hosu V., Saupe D., 2008. KADID-10k: A Large-scale Artificially Distorted IQA Database. [online] Visual Quality Assessment (VQA). Available at: "http://database.mmsp-kn.de/kadid-10k-database.html [Accessed 01 May 2021], pp. 1-3.

Fig. 67. Final site of an end users workspace – The different distortions applied to each image distortion category.

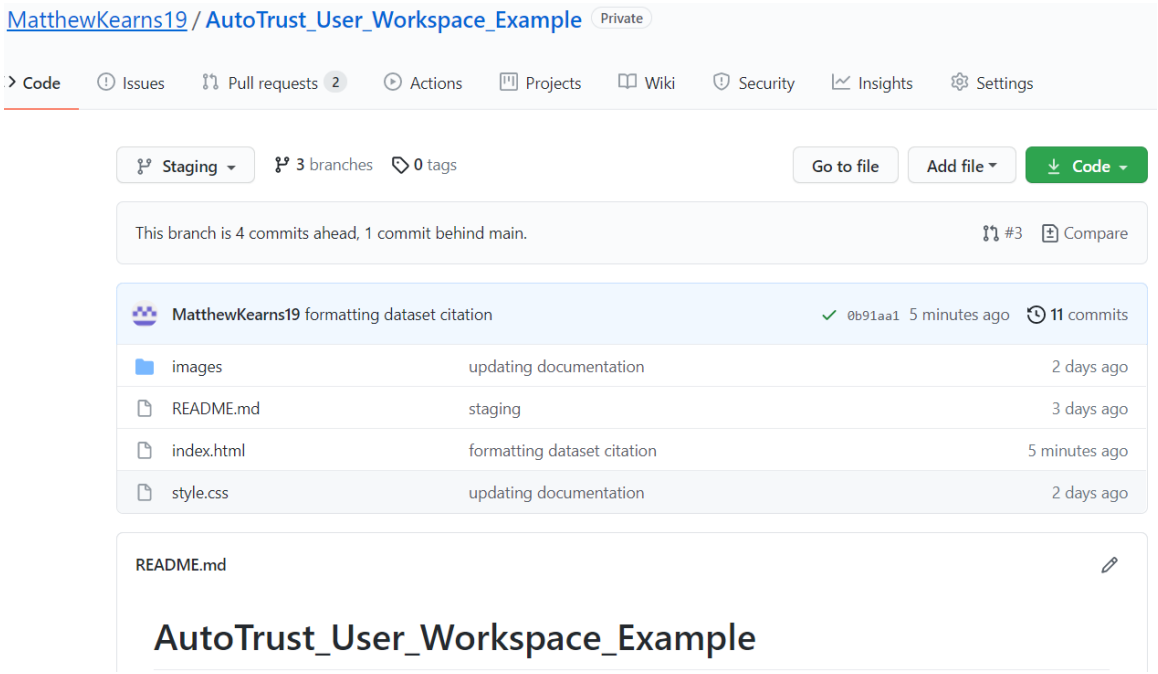


Fig. 68. An example of the GitHub source control for the end application workspace. A view of the Staging barnch.

Results will be returned to the user after they perform a push event to an open Pull Request (PR) on GitHub. A pull request is a code comparison between two areas of the workspace that are being developed, where one is 'requesting' to merge their changes into the other. Once a PR is open between two code developments, a new code development that is pushed from a local workspace to GitHub will trigger the Jenkins and Circle-CI build executions. This is a rule set in GitHub Actions settings.

The User's Source control for their testing scripts:

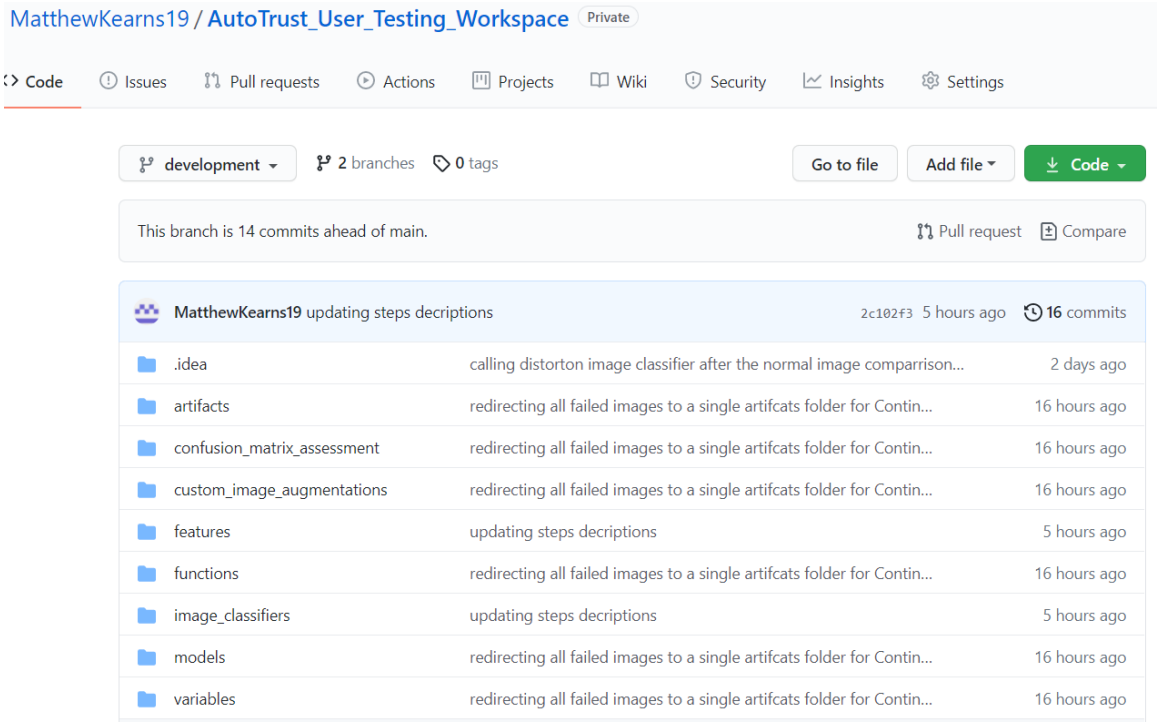


Fig. 69. An example of the GitHub source control of the full autamation workspace of a user. This holds the full automation testing environment used in this project.

2.5. Testing

Since my Project is a testing framework itself, testing the features of these tests was through iterative integration tests which were essentially executing my test scripts themselves throughout development. Additionally, any developments that were developed outside of this test architecture and integrated in later, such as applying image distortions, augmentation, and external development of the CNN, had performance testing applied. These performance testing measures are explained and evaluated in the next section 2.6, under the heading performance evaluations.

The main testing was during development of the CNN, testing each of the models developed after training was complete. Which was outlines in Section 2.3 – Implementation, and further evaluation in the next Section 2.6 – Evaluation.

2.6. Evaluation

Performance Evaluations

When applying image augmentation, this process began very slow, so I immediately stopped the process and decided to set up Multiprocessing workers to execute the same tasks but in parallel with the objective of task completion within a shorter time frame. I used command line

argument parser to specify the number of multiprocessing worker that I would like to run, testing different approaches to sought after the best result. To measure the time taken I set up a function that recorded the time that the execution started, when it finished, and calculated the difference. After many attempts the results were very mixed, with minimal difference in milliseconds and a complete plateau after 8 multiprocessing workers. In Fig.70. below you can see the time difference between one iteration with no multiprocessing workers, and another with 5 workers. This was the best time difference I received through testing time execution, and I could not re-produce this result. Due to this, I ended up not using multiprocessing workers for the rest of my implementation of data augmentation.

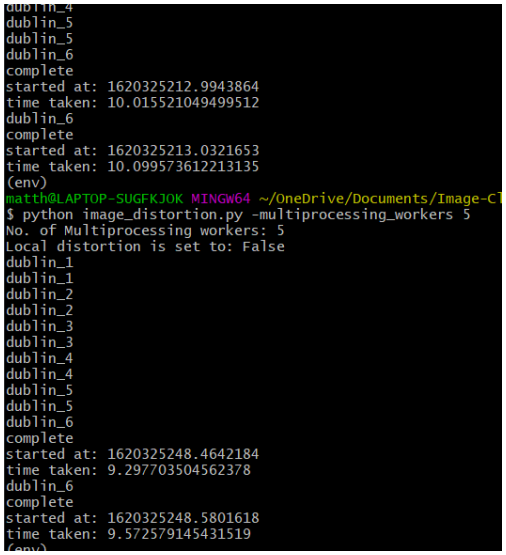


Fig. 70. Two Iterations of augmenting data, the first itteration with no multiprocessing workers, and the second with 5 multiprocessing workers.

Throughout development of each CNN, I evaluated the model's distortion classification. I outputted a Classification Report using the sklearn which gave me a visual representation of important metrics such as accuracy and precision.

Classification Report

	precision	recall	f1-score	support
blur	0.94	1.00	0.97	90
color_distortion	1.00	0.96	0.98	90
compression	0.99	0.97	0.98	90
high_resolution	1.00	0.97	0.98	90
noise	0.99	0.98	0.98	90
spatial_distortion	0.95	0.99	0.97	90
accuracy			0.98	540
macro avg	0.98	0.98	0.98	540
weighted avg	0.98	0.98	0.98	540

Fig. 71. Classification Report on validation data executed on customized kadid10k data on custom Sequential CNN. 180 images in each distortion class.

As mentioned in Implementation, I set the random seeds in the task-specific environment paths to fixed values to gain reproducibility. After introduced, I wanted to re-test the validation batches used to train the data to demonstrate a more conceptually correct larger scale reproducibility test than that of the smaller test data. After re-running the validation batches through the model using the model.evaluate() function, the exact same result was returned as the first time that validation classes were evaluated during training the model. I was extremely happy with this accuracy as it was a clear post-development test of reproducibility.

```
loss, accuracy = model.evaluate(valid_batches)
print('Accuracy on validation dataset:', accuracy)
```

```
108/108 [==============================] - 52s 486ms/step - loss: 0.1413 - accuracy: 0.9694
Accuracy on test dataset: 0.9694444537162781
```

Fig. 72. Weighted validation accuracy results from validation.

At the beginning of local testing, I was getting very accurate results, but they suddenly plateaued. This high accuracy was in terms of classification between other classes, but not high accuracy in terms of the prediction itself. For example, in Fig.72. you can see the output of a high-resolution image was being detected (as the fourth object in the array, the order in which it was trained upon defined classes), however the prediction itself is only 17.7 percent. As you can see from the warning output, I do not have a general processing unit (GPU), so it was unable to access all layers of the model under the constrains of my current CPU processing power. For this reason, I was unable to produce the same results that I could captured while developing the model in Jupyter notebooks with the provided IPython kernel. I was able to luckily capture one of the early successful tests of distortion classification within a full workflow execution, as demonstrated in Implementation as final deliverables that satisfied my project concepts importance in visual testing.

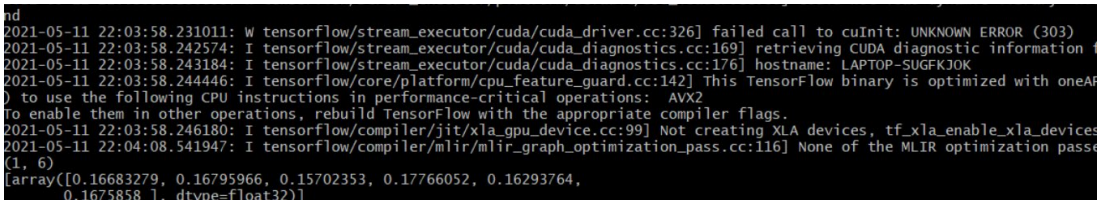


Fig. 73. Local tensorflow GPU warnig .

Correctness

As mentioned in Implementation of data gathering, I added the 81 original files into the high-resolution class to further demonstrate that the CNN was in fact detecting distortion image features instead of object detection, since each image was in each class, with multiple levels of distortion variance. To evaluate if this act of correctness held up in training the dataset, evaluating the data with only the 81 images in high-resolution folder within the test batches yielded a 95 percent accuracy, which I was very happy with. In Fig.73. you can see the results of this test plotted on a confusion matrix.

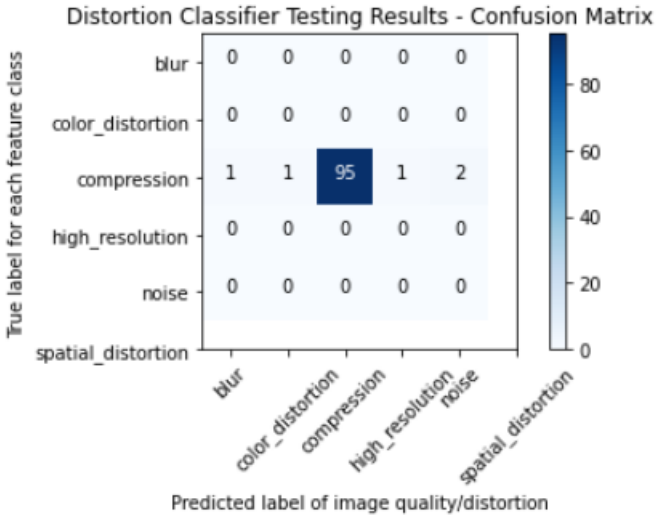


Fig. 74. 81 origional images fom the kadid10k dtaabase.

3.0 Conclusions

From this project I have learned a great deal about various technologies, the main ones being Behave, Selenium, and Machine Learning with focus on Convolutional Neural Networks, and configuring CI/CD integrations. All of which I had no experience with at all.

Through the project scope there were many pitfalls in implementation, requiring immediate attention to requirements engineering. During the early stages of development, my proposal of how I would evaluate an end users' sight with image recognition was uneducated in terms of testing and in terms of the methods that a CNN uses to classify images, projecting aims such as classifying user's sites to be 'similar'. Classifications in this manner would pass any image assessment tests defined to that use case of being 'similar', rather than classifying image differences. Throughout the project I learned handfuls of reasons why most initial ideas of mine were not suitable implementations to the task at hand, which on reflection emphasises the learning curve and knowledge achieved from undertaking this project.

Another instance that demonstrates this learning curve is how I struggled for a roughly a month being stubborn with Jenkins running on a modern windows environment and the effects it has with environment paths such as the PYTHONPATH. I was determined to overcome this when I should have pivoted after revisiting this in requirements. When I re-visited the issue later and decided to run Jenkins within an ubuntu Virtual machine, I was able to scale it with ease to the necessary configuration, with a port tunnelled URL for GitHub webhook connections, all in less than a 15-minute time frame, emphasizing the learning curve from this obstacle. I continuously learnt about CNN's and made large mistakes that lead to great lessons, such as not evenly distributing my data of distortion categories, leaving room open for bias resulting in inaccurate results on validation and training data, with poor reproducibility in results.

Since I was following a similar implementation to the article that inspired the idea for direct image classification, who conducted experiment on local image distortion rather than global image

distortion, mentioning that local distortion is a not a regular occurrence. Distortion usually effects the whole image, at a global level, and in fact, their approach was focused on classifying image distortion in areas of multiple image assembly such as picture collages (Ahn, N., Kang, B. and Sohn, K., 2018). When I originally read this, I did not realize this approach would apply to me when smaller images are captures from the browser. This was a mistake as my implementation does deal with local distortion of the final representation of the site location. When I realized that this article was only applying noise and compression to images, I adjusted my approach to lean towards only noise and compression distortions, since I was originally attempting to apply a similar approach all along. Through iteratively removing the categories that did not apply, such as brightness change, sharpness and contrast, and the results of my classification accuracy was immediately, significantly better. This accuracy was so good straight away, achieving 96 percent after training was complete, that I did remove any more instances of distortion even if they were not strictly blur or compression distortions. As a result, to my knowledge, I am one of the only people why have an image CNN that can decipher between features in not only the popular noise and compression categories, but also colour and spatial distortion, which I think is a pleasant achievement to conclude with.

Additionally, the main results and implementations summarized, automated testing as a method of invoking a systematic tracking coupled to the behaviour of your site's user scenarios described in its offerings of better communication between test engineers, developer, and business. The ease of implementation was also demonstrated, and how the benefit of automating scenarios is to inspect and assert site features before they reach production, reducing the element of manual error that occurs in manual testing. Further assertion was implemented through methods of image classification, 1. Element and Text Comparison through grayscale conversions of exact pixel values, and 2. CNN Image Distortion Classification through feature detection. It was demonstrated that automated scripts are effective in asserting errors, but anything out of the assertion scope of the scenario will be open for attacks of visual distortion, examples of such; text distortion in terms of HTML tags that have leaked through HTML files or unhandled encoding/decoding of data, and image distortions such as noise or JPEG Compression. Where these assertion tests may fail, the implementation of Element and Text Comparison through grayscale conversions of exact pixel values can detect occurrences of any shift in image difference. Furthermore, when it comes to image distortion, a CNN Image Distortion Classifier provides more accurate, more descriptive feedback, enabling the distortion in subject to be labelled, and a root cause can troubleshooted in said direction of distortion.

4.0 Further Development or Research

For automated systematic approaches, with further research I would like to adhere to a TDD approach, combined with the implementation of this framework with other languages such as Java and ruby. During this project time frame, I was enrolled in another module 'Introduction to Cloud Computing', where I learnt Ruby on Rails. I was amazed at the available architecture to scale Integration, and System tests into the Rails application architecture. The configuration of dependencies such as web drivers was an effortless operation, and I would have loved to have incorporated this into this project at a later stage in development as part of requirements engineering in terms of scaling my image distortion classifier into more widely available architectures. The process would be to encapsulate requests to this classifier within a ruby gem, where the classifier is hosted on a cloud instance.

I would further develop the CI/CD integration using Jenkins running on an Amazon Web Services, and Microsoft Azure cloud instance virtual machine, and compare their performances with certain tasks at hand through speed and cost analysis. I also am disappointed that I did not get my Model hosted on a Cloud Provider, utilizing GPU for more accurate results and a more distributed system. I am also going to investigate more into why the Circle-CI post build actions would not word for Portable Network Graphics (PNG) files. I have already requested a demo of the enterprise version to analyse its features, as I really liked Circle-Ci as a service.

With additional time and research, I would love to yield more local testing accuracy by running a GPU. Additionally, by training and testing them while a GPU was installed and measure the time difference in completion of these tasks and observe if better accuracy was achieved during validation and testing.

In the end, I had developed but did not get to run a VGG-16 that was fine-tunned to my distortion classification use case, and in the future, I will be testing this implementation to yield better results.

If you did not notice, during demonstration of image comparison, this version of the user's example workspace had various spelling mistakes. I would like to develop a CNN's that recognises misspelled words or poor grammar in English text and highlights these areas with feedback with regards to what was classified after observation. Similarly, I would like to add more to implement detection of HTML tags that have leaked through to the front-end environment UI, like demonstrated in my Implementation as a potential problematic occurrence within a UI. To classify these with feedback after observation and classification would be the next step in development of this Project, an extra layer deeper into more visual, user-friendly feedback within visual testing.

5.0 References

S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards realtime object detection with region proposal networks," in Advances in Neural Information Processing Systems (NIPS), 2015, pp. 91–99

Unknown, Behave.readthedocs.io. 2020. '*Behavior Driven Development — Behave 1.2.6 Documentation*.' [online] Available at: <https://behave.readthedocs.io/en/stable/philosophy.html#the-gherkin-language> [Accessed 10 November 2020].

Brownlee, J., 2020. *Transfer Learning in Keras with Computer Vision Models*. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/how-to-use-transfer-learning-when-developing-convolutional-neural-network-models/> [Accessed 1 May 2021].

Gulli, A. & Pal, S., 2017. *Deep learning with Keras*, Packt Publishing Ltd. [Accessed 1 May 2021].

TensorFlow. 2018. *Get Started | TensorFlow*. [online] Available at: <https://www.tensorflow.org/get_started/> [Accessed 1 May 2018].

Implementation – dataset
Ahn, N., Kang, B. and Sohn, K., 2018. *Image Distortion Detection using Convolutional Neural Network*. [online] Arxiv-vanity.com. Available at: <https://www.arxiv-vanity.com/papers/1805.10881/> [Accessed 1 May 2021].

Simonyan, K., & Zisserman, A. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR, abs/1409.1556*. [Accessed 1 May 2021].

Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors, 2014. *scikit-image: Image processing in Python*. PeerJ 2:e453 . [Accessed 1 May 2021].

Herve Jegou, Matthijs Douze, and Cordelia Schmid, 2008.
"Hamming Embedding and Weak geometry consistency for large scale image search"

Muthukadan, B., 2011. *4. Locating Elements — Selenium Python Bindings 2 documentation*. [online] Selenium-python.readthedocs.io. Available at: <https://selenium-python.readthedocs.io/locating-elements.html> [Accessed 12 May 2021].

Ramesh, S., 2018. *A guide to an efficient way to build neural network architectures- Part II: Hyper-parameter…*. [online] Medium. Available at: <https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d7> [Accessed 4 May 2021].

Brownlee, J., 2020. *Softmax Activation Function with Python*. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/softmax-activation-function-with-python/> [Accessed 4 May 2021].

Brownlee, J., 2019. *How to Configure the Learning Rate When Training Deep Learning Neural Networks*. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/> [Accessed 4 May 2021].

Brownlee, J., 2016. *Overfitting and Underfitting With Machine Learning Algorithms*. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/> [Accessed 4 May 2021].

Wang, W., Hu, Y., Zou T., Liu, H., 2020. A New Image Classification Approach via Improved MobileNet Models with Local Receptive Field Expansion in Shallow Layers. Computational Intelligence and Neuroscience. 2020. 1-10. 10.1155/2020/8817849.

unknown, 2020. *Fine-Tuning MobileNet on Custom Data Set with TensorFlow's Keras API*. [online] Deeplizard.com. Available at: <https://deeplizard.com/learn/video/Zrt76AIbeh4> [Accessed 4 May 2021].

Chatterjee, S., 2018. *Deep learning unbalanced training data?Solve it like this.*. [online] Medium. Available at: <https://towardsdatascience.com/deep-learning-unbalanced-training-data-solve-it-like-this-6c528e9efea6> [Accessed 4 May 2021].

The Data Detective, 2020. *Finally: Why We Use an 80/20 Split for Training and Test Data Plus an Alternative Method (Oh Yes…)*. [online] Medium. Available at: <https://towardsdatascience.com/finally-why-we-use-an-80-20-split-for-training-and-test-data-plus-an-alternative-method-oh-yes-edc77e96295d> [Accessed 3 May 2021].

The SciPy community., 2008. *numpy.array — NumPy v1.20 Manual*. [online] Numpy.org. Available at: <https://numpy.org/doc/stable/reference/generated/numpy.array.html> [Accessed 16 May 2021].

Lin, H., Hosu V., Saupe D., 2008. KADID-10k: A Large-scale Artificially
Distorted IQA Database. [online] Visual Quality Assessment (VQA). Available at: <https://
http://database.mmsp-kn.de/kadid-10k-database.html> [Accessed 16 May 2021], pp. 1-3.

unknown, 2021. *#004 CNN Padding*. [online] Datahacker.rs. Available at: <http://datahacker.rs/what-is-padding-cnn/> [Accessed 8 May 2021].

deeplizard, 2020. *#004 CNN Padding*. [online video] MobleNet Image Classification with Tensorflow's keras API. Available at: < https://www.youtube.com/watch?v=5JAZiue-fzY> [Accessed 8 May 2021].

Brownlee, J., 2019. *How to Get Reproducible Results with Keras*. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/reproducible-results-neural-networks-keras/> [Accessed 12 May 2021].

Munesti, M., 2020. What is Jenkins good for. [online] OpenLogic. Available at: < https://www.openlogic.com/blog/what-is-jenkins-used-for> [Accessed 12 May 2021].

Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, *2014*(239), 2.

Eirikur, A., Radu, T., 2020. NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study. [online] Agustsson_2017_CVPR_Workshops. Available at: < https://data.vision.ee.ethz.ch/cvl/DIV2K/ > [Accessed 1 May 2021].

Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *ArXiv, abs/1704.04861*.

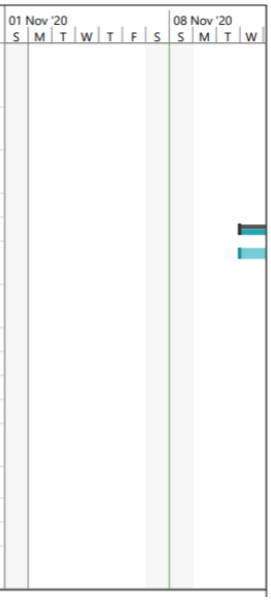
6.0 Appendices

This section should contain information that is supplementary to the main body of the report.

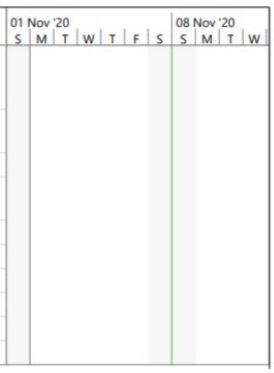
6.1. Project Plan

ID	Task Mode	Task Name	Duration	Start	Finish
1		**Semester 1 (Phase 1)**			
2		**Project proposal**	6 days	Mon 02/11/20	Sun 08/11/20
3		Deeper Market Analysis after project approval	3 days?	Mon 02/11/20	Wed 04/11/20
4		Deeper Target Market Analysis	3 days	Mon 02/11/20	Wed 04/11/20
5		Deeper Competitor Market Analysis	2 days	Tue 03/11/20	Wed 04/11/20
6		Objectives, and Background write-up	2 days	Thu 05/11/20	Fri 06/11/20
7		Technical Approach, Technical Details, and Evaluation write-up	2 days	Fri 06/11/20	Sun 08/11/20
8		Project Proposal Submission	1 day	Sun 08/11/20	Sun 08/11/20
9		Post project proposal reflection and research	2 days	Mon 09/11/20	Tue 10/11/20
10					
11					
12		**Mid Point Presentation**	30 days	Wed 11/11/20	Tue 22/12/20
13		Implementation	28 days?	Wed 11/11/20	Fri 18/12/20
14		Re-plan and settle on approach with current technologies and languages specified for architechture	4 days	Wed 11/11/20	Sat 14/11/20
15		Setup source control	2 days	Fri 13/11/20	Sat 14/11/20
16		Install necessary software architechture	2 days	Fri 13/11/20	Sat 14/11/20
17		Begin practice Selenium scripting	6 days	Sun 15/11/20	Fri 20/11/20
18		Begin Behaviour driven development scripting	7 days	Sat 21/11/20	Sat 28/11/20

ID	Task Mode	Task Name	Duration	Start	Finish
19		Begin mapping out/deeper study and then practice of Machine learning Image recognition practices	7 days	Sun 29/11/20	Sat 05/12/20
20		Implement of current knowledge to project Architechture (Implementation Increment 1)	11 days	Sun 06/12/20	Fri 18/12/20
21		Re-think and reflect on need for ethics at current point of the project	2 days	Sat 12/12/20	Sun 13/12/20
22		Due date for Ethics form 1st re-visit, if needed	1 day	Sun 13/12/20	Sun 13/12/20
23		Doccumentation	29 days?	Wed 11/11/20	Mon 21/12/20
24		Regularly document Implementaion - ongoing from Implementation start date	28 days	Wed 11/11/20	Fri 18/12/20
25		Documentation write-up for Mid-Point presentation	2 days	Fri 18/12/20	Mon 21/12/20
26		Doccumentation Finalization	2 days	Fri 18/12/20	Mon 21/12/20
27		Video Presentation	2 days	Fri 18/12/20	Mon 21/12/20
28		Mid Point Presentation Finalization	2 days	Sat 21/11/20	Sun 22/11/20
29		Mid Point Presentation Submission	1 day	Tue 22/12/20	Tue 22/12/20
30		Implement of current knowledge to project Architechture (Implementation Increment 2)	15 days	Wed 23/12/20	Tue 12/01/21
31		**Semester 2 (Phase 2)**	99 days	Wed 13/01/21	Sat 29/05/21
32		**Final Implementation & Doccumentation**	**84 days?**	**Wed 13/01/21**	**Sun 09/05/21**
33		Implementation	80 days?	Wed 13/01/21	Tue 04/05/21
34		Reflection of Increment 2 undertakook over christmas break	3 days	Wed 13/01/21	Fri 15/01/21



ID	Task Mode	Task Name	Duration	Start	Finish
35		Implement of current knowledge to project Architechture (Implementation Increment 2)	78 days	Sat 16/01/21	Tue 04/05/21
36		Continue developing, building on pre-established progress	78 days	Sat 16/01/21	Tue 04/05/21
37		Doccumentation	**84 days?**	**Wed 13/01/21**	**Sun 09/05/21**
38		Regularly document Implementaion - ongoing from Implementation start date	78 days	Sat 16/01/21	Tue 04/05/21
39		Documentation write-up for submission	4 days	Wed 05/05/21	Sat 08/05/21
40		Doccumentation Finalization	1 day	Sun 07/02/21	Sun 07/02/21
41		Due date for Ethics form 3rd re-visit, if needed	1 day	Sun 07/03/21	Sun 07/03/21
42		Due date for Ethics form 4th re-visit, if needed	1 day	Mon 12/04/21	Mon 12/04/21
43		**Video Presentation**	6 days	Mon 10/05/21	Sun 16/05/21
44		**Project Showcase**	6 days	Mon 24/05/21	Sat 29/05/21



6.2. Reflective Journals

Using Driscoll's (2000) Model of Reflection:

Reflective journal 1 – October 2020

I will be reflecting on the progression in the month of October, what I have already learned and what can I now learn because of my reflection. At the beginning of the month, I had a decent understanding of my project idea and how I was going to approach it, an AI CNN image recognition package or API hooked up to an automation server such as Jenkins, which will compare two screenshots as part of your testing phase to make sure they are tested on a visual level.

I had done some previous brief research during summer after thinking of this idea, to see if the idea had been implemented already, and I could not find anything made for software developing testing. I found a company that was doing something similar for games, based on when a game is already complete, and It seemed this company was the only company testing visually with visual automation.

I was extremely confident in the idea and did not look any further, until the start of October when we had to pitch our idea. After more market research I quickly found a company that was doing the same for apps. Followed by a finding which almost shattered my idea, a company was doing this same idea since 2018, comparing two screenshots by CNN image recognition.

I was shocked, as I could not believe I missed this in previous research when I would have had plenty of time to adjust to a new idea. I investigated this company's approach and could not find anything that tells me how exactly they implement their visual testing into the testing phase, and it appeared to be done as a web platform, which allows developers to move their images along the testing life cycle and be compared by the visual AI, approved by a colleague, and then pushed to live. This was different to how I imagined it but based on my knowledge on how their platform could possibly compare these images, I figured they must be hooked up to a build that runs in the background with the same concept as I pictured, or a very similarly, an API or NPM package plugged into an automated build.

To my understanding, although I did not know exactly how they operate their background testing, the idea still felt like it was taken and unoriginal. I then spent the following week trying to come up with other ideas, but some were too far-fetched, and more were already done. I had not enough time to research and solidify these new ideas, so I decided to go with my original idea, as it was still new in the eyes of my approach, and I knew that I could adapt to a new or similar idea if needs be, an idea with a different business approach to a different problem, while using the same tools. I know that an idea does not have to be original to offer original value to someone or something. In the second half of the 20th century almost every new idea is an adjustment to the wheel rather than re-inventing the wheel. I hope to continue with these tools and technologies and see where I can pivot.

I now know that I should have trusted but verified my findings early on, and this would have saved me a lot of time, and indecision. It was out of excitement that I turned a blind eye to digging deeper into market research, and instead dove straight into the idea generation and how I was going to implement it.

Reflective journal 2 – November 2020

I will be reflecting on what I have already learned and what can I now learn because of my reflection of the month of November. I began the month producing my project proposal after my project idea was approved. I began performing more market analysis of competitor products and then began writing up the Objectives, and Background sections of this proposal. This gave me more insight into the idea and some second guessing in areas where there held room for improvement. I then dove into the technical approach, technical details, and project plan where my vision as it stood was put on paper in details. The room for improvements were considered and a realistic vision was evaluated, with future, hopeful projections also mentioned and the planning for this implementation under current knowledge. I also created a Gantt chart as part of this submission.

After submission, on the 8th of November, I had a Gantt chart created for planning, implementation, and reflective progression, in increments. I was happy with this Gantt chart as it was very strict but allowed room for backlog and unexpected delays without causing destruction to the plan put forth. I then began planning more on the architecture after a post-project-proposal-reflection. I also met with my project supervisor and discussed the ability to use TensorFlow and Keras in my project, leaving me with some leverage to explore more technologies to create an interesting framework rather than spending most of the focus re-inventing the wheel. I have got my source control in place but have not yet added any files. I have mainly been focusing on behaviour driven development (BDD) and creating scripts that will navigate to the URL that needs to be tested. I have installed the necessary software for this as it stands, and further software that compliments BBD have not yet been added but are being researched and mapped to my architecture. I have not yet begun Selenium scripting and task has been added to the backlog. This brings me up to the current date and current project implementations as projected in the Gantt chart.

If I were to re-do the month of November, I would have spent less time researching and more time implementing, as I knew beforehand that I would learn a lot (most of the workload) by trial and error and practice, while staying within the bounds of the planned scope. However, I kept falling into the trap of trying to set up everything perfectly beforehand, to try save valuable time in the future by working smarter and more strategic. This was a downfall as I knew the only way to really tackle this and get a feel for the architecture was to slowly trial and error some approached, get a feel for the methodologies, and avoid getting caught up in a deep hole of related study, although necessary in the future, but not at this time of the project.

Reflective journal 3 – December 2020

This reflection will outline the most practical work produced out of all months prior, as this I undertook implementation of my background research. As I tackled this month, some areas of my project plan were pushed back, and moved to my backlog as I iterated through each small print. However, these tasks were not left aside for long as I was able to delegate more to these tasks when working on the next task in the sprint. This backlog accumulation was a result of

December's workload across all other modules. December is always a busy month, but this year were especially. There were many assignment deadlines, which added an increased probability for unforeseen issues related to these deadlines. Time management was key, as it has been all semester, and the ability to adapt to new circumstances by re-visiting the drawing board was the critical difference between achieving the goals set out for this project or not.

After last month I knew that a large portion of the time was taken up by researching instead of implementing. This was due to the large number of technologies that I will be implementing, and I did not want to go down a path where I found myself stuck, running out of time. I was able to dive into more practical work this month and got a large portion of the end users testing architecture in place. I did not get the Jenkins framework set up and this is still in my backlog as of today. I continued practicing BDD scripting with the necessary packages and requirements and managed to get a demo in place for my mid-point presentation. This side of the project is for demonstration purposes to display the results of my image recognition feature, as well as showing the acquired knowledge to this testing framework. I reflected more on the image recognition requirements, dependencies, and concepts, yet did not get any hands-on implementation of this. As I incremented through these tasks, I constantly filled in segments of the mid-point presentation report. This mid-point presentation was heavily focused on requirements engineering, so although portions of my backlog were not complete, it still okay and very important to talk about in topic of reflection, delegating, and adjusting the approach to engineering some of these into the framework in the future.

Due to the way this month fell in terms of the workload for other modules on top of this module, closely coupled deadlines, backlog build-up and so on, I would not change the way I approached this month at all. I am happy with the work that I have got done this month. I can only take forward what I have learned, which is to always set to timeframes, unless certain tasks are outweighed by other priorities, but try to keep them at a minimum. I say this because This module is very heavily weighted, and the end goal here comes from a passionate around this concept. I have really enjoyed this projects workload so far, yet I have spent a lot more time on other modules that are not as heavily weighted. With this knowledge, and the knowledge acquired from my mid-point post analysis, I am very excited to focus more time on this next semester.

Reflective journal 4 – January 2021

This reflection will outline the month of January. The workload across other modules over the Christmas period took up all my time, and although I mentioned in my last report that I would re-factor deadlines and aim to stick to them no matter what, I still found myself prioritizing my other modules over the exam season period. This will be a short reflection as I have not made as much physical progress to show for compared to what was projected. This month has been the worst month for progress in terms of sticking to the Project Plan and physical evidence of progress, and I am disappointed in myself for this. I took an extended break over Christmas by not focusing on getting ahead like planned, and I have left myself further behind. I still do not have Jenkins set up, which is the one thing I really hoped to move forward with after splitting up for the Christmas break after exams. However, I have been practicing more behaviour driven development scripting which has come along a lot in the last month.

I have begun writing up additional requirements specifications which tie into what I had mentioned a lot in the mid-point presentation; **I have realized my approach to this can be done in very little Gherkin scripting itself but maintaining the behaviour** driven development approach. After more reflection on the requirements and refactoring from the project implementation, I have decided to keep this scripting to show what I have learnt on the Selenium-testing side of things as it was in my project plan from the start as a new language and tech framework that I wanted to learn.

The ideal end goal would be to showcase this in a domain that acts as a User Interface, breaking down the behaviour driven development testing approach, the project architecture etc. and contains user interaction that triggers a GitHub build which distorts a segment of the same page, or an image on the same page. This will allow me to display live visual distortion and return a failed test result to the page, which I think would be a cool implementation. After some brief research on this, it seems to be difficult to integrate the GitHub webhook into a webpage with access keys etc., but It should not be a major challenge. Pressing the same button when made available after some time (maybe 2 minutes), could trigger a background script that runs a merged code rollback to GitHub, to fix the page, and then return a passed test result. I think this will be cool to implement and an additional touch that will create user-interaction for the reader of this documentation and promote a more efficient demonstration.

Reflective journal 5 – February 2021

This reflection will outline the month of February. This month I have been working on the CI/CD pipeline for my project. I have been working on integrating Jenkins with GitHub. Jenkins runs on a localhost port, and to integrate it with GitHub you need to configure the port with a public DNS server. I have been back and forth with this process, testing various port tunnelling services such as SocketXP and ngrok. The issue is that these only last several hours, or until I close the port forwarding tunnel. I have had to configure the port using auth secrets every time I want to work on my Jenkins -> Git configuration. This has been a bit messy because I then must change the reverse proxy settings of the Jenkins URL to allow return requests to the Jenkins server, since the server is changing every time that I boot up this tunnelling process. I then must go to GitHub and change the webhook URL that I am using too. This is not too time consuming, but I am moving towards connecting this localhost to a public DNS, to speed up this process and to have added security. The port tunnels do offer me a http and a https each time, and I have been using https for the added encryption in the meantime until I finally decide on the best solution for this setup. I will need this to be publicly hosted for my project, I have just not decided on what approach to take yet. I am looking at configuring it with an Azure VM, allowing me to streamline this pipeline with git. This seems to be the best approach and the cost will not be much compared to an AWS ec2 instance.
Integrating Jenkins with GitHub webhooks has also took longer than expected (nothing new to the world of development). I have used several credential methods for this integration and the authentication is failing. Between GitHub passwords, personal access tokens, and SSH keys, the SSH key is the only method that is giving a larger output and more information to work off. Executing the same command from my local terminal is working, which is strange. I have read that it may be due to the new SSH format that GitHub accepts. I will recover from this quickly and hopefully be able to set up more details for the builds that will be triggered by the Jenkins server.

Reflective journal 6 – March 2021

This reflection will outline the month of February. This month I managed to successfully integrate my Jenkins local server with my GitHub webhooks using DNS-tunnelling. I have still not set this up to run on a public DNS. I have still not configured this with an Azure VM as this was another solution mentioned last month. It still seems to be the best approach and it would be of interest to me since this requirements specification adds an interesting new deployment pipeline into the project scope. I am disappointed that I have not got this implemented as it would be nice to get out of the way early on.

This month I have been very distracted with other modules and I got most of the monthly project progression during the beginning of the month, and I have not re-visited it since in detail. However, I have gathered more information on a roadblock that I faced with the concept of the CNN image classifier – It became clear to me that my previous approach of using Data augmentation to generate thousands of versions of the same 'expected result image' for the test would is essentially testing images to be 'like' the expected images, and returning passed results based on similarity rather than pinpoint accuracy. Therefore, these would pass even if there were some slight UI bugs. This is conceptually correct, but it is still not accurate in terms of testing. I needed to find out a way around this and I did not was to resort to implementing a more accurate, yet similar approach of using some packages belonging to the python Pillow module to compare images in binary format pixel by pixel. This was thought of in the original idea generation and I have explored comparing two images this way and this process is straight forward, but it is so accurate that it does not allow any room for error such as screenshot error, different browsers etc., Due to this and the fact that I purposely took on Image Classification to explore the technologies, languages, and neural networks associated, I have decided to combine demonstrations of both methods, and perhaps integrate them together with some external libraries for object detection to allow me to highlight the areas of a test result image that has failed. This combination should highlight warnings/errors based on the accuracy. If I can get these both integrated, I will be extremely happy with that result. From this reflection I have once again realized that I have been prioritizing other modules and I plan to keep this at bay throughout the final stretch of implementation.

Reflective journal 7 – April 2021

At the beginning of the month, my Jenkins server was completely wiped for some unknown reason, and I had to install and configure system setting, GitHub credential and keys, instal the necessary plugins all over again, but this did not take long it was just a minor setback that was frustrating.

Last month I mentioned that I successfully integrated my local Jenkins server with GitHub, and although these were integrated, and each build configured successful requests and responses from the webhooks, I still ran into an issue with the build itself. Jenkins has a build-step to define the steps of this build in the pipeline which allows for various scripts – the two widely used being shell and bat scripts. I began setting up the steps to set up a virtual env at the start of the build, download my requirements and then execute my sample script for a test image comparison. This script is the script mentioned in my last monthly report as part of an adjustment to my requirements specification, and it compares two images using the pillow library and OpenCV.

If I made a push to my GitHub project repo, it would successfully trigger a Jenkins job to build, but unfortunately, I could not get the environment in the build-step to execute successfully, as it was having issues with the dis-connection between my 64-bit windows system and the fact that Jenkins lies in my 32-bit windows system. Both shell and bash scripts in the build step ran but could not find basic paths such as my python path because of this system disconnection. I attempted to create some of my own bat scripts within the Jenkins folder of the 32-bit system that will launch shell distributions that I have available on my system such as git bash and windows subsystem for Linux, which ran and opened successfully but the basic system paths could still not be found. I attempted many solutions online and concluded that I really should not be running Jenkins from windows anyway, and I should be doing it within a Linux or mac system. Reason being – most tutorials and community help for Jenkins are for these systems, my final documentation needs to be clean and cannot contain several awkward workarounds making it over complicated. I always refer to this quote when it comes to coding, and the same stands in the scenario where I would end up making the documentation to complex, and my stubbornness to not pivot has also made implementation more difficult on myself; "An intellectual says a simple thing in a hard way. An artist says a hard thing in a simple way." (Bukowski, C.). Jenkins is also not very modern which may be an issue in the documentation, as I want it to show a modernized implementation. Having Jenkins on a Linux system also ensures more overall compatibility with development and production environments.

The next step is to set this up, which will not take long at all. In the meantime, I decided to take a dive into Circle-CI because it is a more modern approach which I mentioned very early in the project. I have managed to set up an integrated pipeline that runs from each GitHub push request, executing my sample python image comparison script. This runs off a sample docker image supplied by Circle-CI, and I plan to utilize this with my own docker credentials and containers in the next stretch of the project. Jenkins will still be set up on Linux VM for demonstration purposes, but I will not spend much focus on this but instead chip away at it in the background now that I have Circle-CI integrated instead. For demonstration purposes/extra options in my tutorial, it would be grated to modernize this approach by connecting to Azure webhook or AWS CodeDeploy which both offer modernized approaches to integrating your Jenkins pipeline into a Cloud instance VM. I also cleaned up my BDD script, which now takes a screenshot of the desired page on the web browser and stored it the desired folder to be used for image comparison. I am also currently working on Image Classification by Augmenting a single image to train the dataset with. In this sense, everything is piecing together nicely and working in their own way, I just need to develop more extensively towards the final pieces and plan my time well. My biggest self-induced pitfall has been not re-evaluating and pivoting when needs be, or at least suspending development on one project aspect and focus on another more important aspect – time management and developers' fallacy in a nutshell.

Journal References

Bukowski, C., n.d. *A quote by Charles Bukowski*. [online] Goodreads.com. Available at: <https://www.goodreads.com/quotes/83729-an-intellectual-says-a-simple-thing-in-a-hard-way> [Accessed 1 May 2021].

6.3. Other materials used

N/A

6.4 Project Proposal

6.4.1 Objectives

The goal of this project is to bring to use my current knowledge of software development along with undertaking the challenge of new tools, technologies, and frameworks, bringing light to the importance of visual testing when releasing code during the software development lifecycle. I plan to develop a plug in for automated testing, along with a simple, user friendly setup tutorial that will integrate with ease into a testing environment to provide clarity to testers and developers, offering the ability to compare images of their release and return results.

To give an example of the problem I plan to address, if a test automation script is built to test a company's login, when run it would pass if a 200 was returned and the script could find the 'welcome user' text on the homepage. However, if the script is only checking for the 'welcome user' text, and there is some text bellow which is not rendering properly, this would be overseen by the test. A passed result would be returned to the developer and they would release the code to the live site, which could be extremely hazardous. In terms of the total run-time, it takes to run scripts before releasing, it is unrealistic and not business efficient if these scripts are to check every line of code on a page for rendering issues.
As a result of this room for error, developers must still run manual tests alongside automated tests, and I believe this is a bump in the road for testing and releasing.

As I mentioned above, to overcome this I plan to build a process/or plug-in that takes screenshots of the pages within the product and use **'Convolutional Neural Network (CNN)** machine learning image processing' to compare the screenshots to pre-defined screenshots/expected

outcomes. This will bypass the need for manual tests in many areas where there are ‘trust issues’ with automated testing, by removing human error, and speeding up the testing and releasing phase of development. Hence the Project title – Auto-Trust.

When a developer wants to test their code, they can run their tests in a testing environment such as Jenkins, which will fail if their code does not pass it on a visual level. This process of continuous delivery and the expectations of results in the most cost-effective way possible, especially in a work environment incrementing in an agile methodology, is narrowing every year. To overcome this, the need for reliable visual testing is a must.

6.4.2 Background

To give some context to the background behind the idea generation itself, I was reflecting on my 3rd year internship and a real problem that I faced when developing at one stage, was the companies Quality Assurance Department had developed automation tests that were used to perform tests on a developers code before releasing, and a perfect result would present that all passed on a code-based, status-code level, and returning the 'go ahead' to push this code live, yet these tests did not pass on a visual level when I ran them manually myself. This visual impurity/bug was a template rendering issue causing text distortion on a portion of the page.

This was extremely hazardous as and I was not confident trusting the automated scripts at that level in future testing without thoroughly running my own manual tests of the system at a more extensive granularity. Being uncomfortable with this issue and the automated systems performance/reliability to catch bugs on a visual level due to lack of a visual testing architecture, an idea sparked to map out a potential solution. Leading me to a solution which encapsulated a full scope of the different technologies the company already had in place within their current architecture, along with the addition of methods and new technologies that I visioned would act as a complimentary within this architecture and it’s established workflow, as it is what made sense to me and my knowledge at the time. I mention this as this visioned approach still seems the most logical after research, and I will get into the technical aspects bellow in the following sections of this proposal.

To follow suit, I performed a brief investigation during summer after thinking of this idea, to see if the idea had been implemented already. At first, I could not find anything on a commercial level and in the form of architecture I had mapped out, but a while later discovered a company that was doing the same for apps. Followed by another finding, a company was doing this same idea since 2018, comparing two screenshots by CNN image recognition.

I could not believe I missed this in my early investigations. I investigated this company's approach and it appeared to be done as a development lifecycle web-based platform, which allows developers to move their images along the testing life cycle and be compared by their visual AI. If there was an issue it would be flagged on this platform, and otherwise approved by a colleague. This was different to how I imagined it but based on my knowledge on how their platform could possibly compare these images, I figured they must be hooked up to a build that runs in the background with the same concept as I pictured, or a very similarly, a plug-in package that contains the machine learning image recognition logic, and possibly hooked up to an automated build.

This still inspired me to continue with my idea as the tools and technologies for visual testing are well known but rely on the person implementing this to be familiar with Quality Assurance testing frameworks. At best I discovered the company outlined above that offers a web-based iteration platform, that the whole workplace must adjust to and integrate into their development lifecycle. I want to make this easy and as non-technical as possible with a simple plug-in and tutorial.

6.4.3 Technical Approach

There is a lot of knowledge out there about CNN's and using them, or similar packages within your organization, but there is nothing that stands out to me that a basic beginner, non-dev-op, or anyone without previous automation knowledge can understand, and that is what I would like to change. For these people or businesses, the information is there but the accessibility to bring this into your environment with basic knowledge at a very high granularity is not.

During research and requirements capture, I decided to use **Jenkins (a continuous integration automation server)** as my continuous integration server and proof of concept. I plan to build automated scripts that can run in Jenkins alongside my machine learning scripts. I aim for these scripts to be executed by Jenkins when a Jenkins build is triggered from a GitHub code merge. Jenkins will gather this main scripting execution workflow from a GitHub repository that is it pointing at. These scripts will use behave, which is a library for behaviour driven development scripting to open and control the browser.

To test my CNN, I must gather data to test its accuracy. For this process I will not be training the dataset, but instead testing its ability to recognise image consistencies. I will not be training the CNN on a large dataset to recognise popular consistencies that occur across user interfaces, but instead build the CNN so that it is trained on a set of pre-defined images that the user has

selected, and only these images. This way the Use-Case is specific to each user's application that they are testing. The images will be stored in a database, or within file pathing closely associated to the main scripting execution workflow.

During research there are different software approaches and methodologies for building a CNN image recognition system and training it on a dataset. These networks can be built in layers of nodes called neurons and these layers can increment on each other as many layers deep as needed for your specific project. Since I will only have to compare images 1 to 1, so the layers within the CNN should not get too deep or complex for a complete beginner within the timeframe of this project. So once complete, the concept of training this on a dataset will be implemented to ensure its Non-Functional and Functional requirements are met. I have no prior knowledge on machine learning or building a CNN, but there seems to be a lot of useful resources and technologies out there for building a CNN for image recognition such as **Keras**, and **TensorFlow**. The main building steps of the CNN are:

Convolution (extract the input pixels and sustain each pixel's relationship with the other surrounding pixels or empty out of bound space.)

Polling – using a 'feature map' that picks up the image features and outputs results between pixel values. It is used to reduce the dimensionality of each outputted feature map.

Flattening – converting the above into a single column to be passed to the next layer in the network.

6.4.4 Special Resources Required

No special hardware or books will be required for this project, all information for learning and implementation will be studied online. A professional license for the **Pycharm** IDE will be needed for automation scripts and acquired for free with my specified student email.

6.4.5 Project Plan

Highlighted in Appendix Section 6.2.

6.4.6 Technical Details

I will be using **PyCharm** as my IDE for this project. I will also be touching on shell commands to that will be defined to execute some of the build steps within the Jenkins build.

The user's application will be stored on GitHub, and GitHub will also be our source control for project development. The user's application will be

connected to Jenkins, a Continuous Integration environment as outlined in previous sections and this will execute the two environments in a build, one stage of the build pulling the application with the new code to be released, and the next step will be to pull the scripting workflow, execute it, and return results based on this visual execution.

To access the browser and create automated navigation through the browser I will use the behave library. This library is designed for teams to collaborate, creating automated testing of their development features. It is designed to be define scripts by the business, or a developer with no automation knowledge, and to be understood by a Quality Assurance engineer with automation knowledge and transformed into an automated script that can test the application features defined. I will use this to demonstrate the faults that may occur in automated testing on a code-based level and reveal the power of automated testing on a visual level. I will also use this to access the browser and navigate through the pages specified to be tested by the user.

JavaScript will be used for the development of the Node Package Manager (NPM) package. I plan to use the pyExecJs library which will allow me to integrate my python scripts with node.js for the NPM package.

I plan to use Flask framework to create our API and Postman to test it.

I will also be working with the Selenium Integrated Development Environment and will possibly be using its closely related technology WebDriver, which will help me in testing as outlined in the next section bellow.

To build the CNN I will need to use Keras, a library providing a Python interface for artificial neural networks. The following libraries will need tone installed for Keras: sequential model, Convolutional2D, MaxPooling2D, Flatten, Dense. These are related to the Convolution, Polling, and Flattening steps outlines in the Technical Approach above. I will also be looking at using TensorFlow, which I can integrate with my Keras model. Keras, is a high-level API that is built on top of TensorFlow and is more user friendly.

6.4.7 Evaluation

Unit tests will have to be written to test my API/ and or NPM package calls as my process of a ‘plug-in’. This will include all fetched functions and methods specified/available with the API/package. The following Non-Functional requirements will be required: Usability, Robustness, Performance, Security, and Reliability. To test the API, I plan to use Postman, which is a GUI for testing your API requests. With an NPM package this comes with the security already offered by the package and additional command line regular security tests can be run using the command ‘NPM audit’, which will be ran throughout production and included in documentation.

Constant regression testing will be performed after any new additions to the project. If any major architectural changes occur regression tests will be performed and documented, and any defects will have to be refactored before being merged with the source code. If these changes are evaluated and deemed to be non-compliant with existing features after analysis, changes will be rolled back, and new test case regression tests will be performed.

To test the initial and gradual development of the image recognition in place, I will have to use some self-made secondary datasets, or public secondary datasets to test the CNN on.

For the actual project system/framework itself, once this CNN is plugged in, I can run tests using pre-defined images stored within the automated scripts, and located/specified by a file path, I can run my machine learning function calls against this file path, which will test the new build against these pre-defined screenshots. The screenshots being tested will be of a website created by myself, and/or secondary data screenshots from other open-source websites.

If I decide to use WebDriver, I will be able to test this without releasing code and generating a new build, but instead specifying the website URL that I wish to test, capturing the screenshots, then inserting some JavaScript that can capture page elements and hide them for test purposes. Running the test again will capture these defects.

Regards evaluating the system with an end user, If I decide to bring forward the project idea to my current manager in my workplace to propose the conception of using this in our work environment, this will most likely have to go through a process of meetings etc. Occurring over a period after the final project submission, so in this sense It will be on a conceptual level and have no ethics dependency. If I decide to integrate this at an earlier stage to gather feedback for testing, I will add this disclosure use case to further documentation in later reports, along with an ethics application form submitted on the next due dates for Ethics, Dec 13th, Feb 7th, March 7th, or April 12th.