Journal of Visual Language and Computing

journal homepage: www.ksiresearch.org/jvlc/

A Case Study of Testing an Image Recognition Application

Chuanqi Tao[a,*], Dongyu Cao[a], Hongjing Guo[a] and Jerry Gao[b]

[a]Nanjing University of Aeronautics and Astronautics, China and [b]San Jose State University, USA

ARTICLE INFO

Article History:
Submitted 3.1.2021
Revised 6.1.2021
Second Revision 8.1.2021
Accepted 9.30.2021

Keywords:
Image Recognition
Testing AI Software
AI Software Quality Validation

ABSTRACT

High-quality Artificial intelligence (AI) software in different domains, like image recognition, has been widely emerged in people's daily life. They are built on machine learning models to implement intelligent features. However, the current research on image recognition software rarely discusses test questions, clear quality requirements, and evaluation methods. The quality of image recognition applications becomes more and more prominent. A three-dimensional(3D) classification decision table can help users to conduct classification-based test requirement analysis and modeling for any given mobile apps powered with AI functions in detection, classification, and prediction. This paper presents a case study of a realistic image recognition application called Calorie Mama using manual testing and automation testing with a 3D decision table. The study results indicate the proposed method is feasible and effective in quality evaluation.

1. Introduction

With the rapid development of big data analysis and artificial intelligence technology, AI software and applications have been widely accepted in our daily life including business, education, social media and so on. At present, AI software and applications are based on the most advanced machine learning models, and various artificial intelligence features are realized through large-scale data training.

The most important implementation of Artificial Intelligence is the imitation of human interactions—vision. Nowadays, there is an abundance of digital images captured by high-quality equipment. Most images are captured with phones. Artificial Intelligence is often used to process these images to extract knowledge, categorization, and labeling along with other advantages. Typical applications of image recognition include object recognition, face recognition, text parsing.

Detecting bugs and errors in software can be very costly. Sometimes bugs can be even deadly if it is a real-time application of software, such as some software that testing the software is very important to verify that the is used to help with surgeries in the hospital. Therefore, the product meets requirements and specifications. Software testing ensures the correctness, integrity, and high quality of the software by checking errors or bugs and fixing them in the initial design.

This paper focused on testing an image recognition application called Calorie Mama utilizing both manual testing and automation testing. Calorie Mama is a smartphone app that runs on Android and IOS devices. It uses deep learning to recognize food from food images and track nutrition based on the food in the image. It calculates the calorie based on that. We evaluated the performance, correctness, and quality of the app using both manual testing and automation testing.

This paper is written to provide our perspective views on image recognition software testing and quality evaluation. The paper is organized as follows. Section 2 discusses the review of AI software testing and image recognition. The third part elaborates methodologies of manual and automation testing. Then, the fourth part shows a case study of testing Calorie Mama APP using these two methods and presents the comparative results of test efficiency and coverage. At last, section 5 gives the conclusion.

*Corresponding author
Email address: taochuanqi@nuaa.edu.cn
ORCID: 0000-0002-0698-7307

DOI reference number: 10-18293/JVLC2021-N2-194

2. Related Work

An in-depth research is conducted in the field of AI testing. This research helped us in choosing our methodologies and impacted our testing approach as a whole.

Gao et al. [1] explained the various testing methods of the AI software testing wherein the authors mention context classification modeling, input and outcome classification modeling and decision tables. Various functional and non-functional quality parameters such as program correctness, system operations, performance, reliability and scalability are discussed to better understand the concept. In addition to this, the authors discuss the issues and challenges of AI testing. AI testing can be costly and time-consuming. There is a lack of adequate models and well-defined standards.

According to [2], AI can apply methods on data for software testing purposes like classifications, regression, clustering and dimensionality reduction. The paper also discusses the testing coverage containing requirement analysis, test planning, test development and execution.

In terms of test case generation, Zhu et al. [3] proposed a new method called datamorphic testing, which consists of three components: a set of seed test cases, a set of datamorphisms for transforming test cases, and a set of metamorphisms for checking test results.

AI software testing is different from traditional software testing, because AI software is characterized by dependence on big data, difficulty in predicting all application scenarios, and constant self-learning from past behavior. King et al. [4] discussed the issues and challenges in software testing. According to the authors, non-determinism is a huge issue. The same input to the system can produce different outputs. Testing has fuzzy oracles, i.e., determining the correctness can be a challenging task. The other challenges of testing include security, performance and scalability. Marijan et al. [5] stated that traditional systems have a fixed behaviour as they execute a set of rules and are typically pre-programmed. ML-based systems exhibit non-deterministic behavior as they use prediction algorithms or so. The quality parameters included correctness, robustness and reliability.

Metamorphic testing (MT) is a property-based software testing technique, which has been leveraged in many domains for addressing the test oracle problem and test case generation problem [6]. Chen et. al [7] introduced MT in 1998, which has been an effective AI-based software testing approach. MT has been applied to autonomous driving system [8,9], machine translation [10,11], ML classifiers [12,13], Google map App [14], search engines [15], facial age recognition software [16], and object detection system [17], etc., all of which have achieved good results.

GUI testing is also important in AI software testing. Rauf et al. [18] discussed the issues and challenges related to testing applications with a user interface. According to the authors, it becomes difficult to address the large number of states of GUI. Also, it is difficult to generate test cases each time the GUI changes. The testing can be platform-specific and thus shows a limitation. The methods prescribed and used in [18] did not address the problem of a huge number of states that even a small application's UI can have and thus can lead to a number of test cases. This paper has used particle swarm optimization (PSO), partition testing based on particle swarm optimization (PSO), test case minimization using an artificial neural network (ANN), Bayesian Network (BN).

King et al. [19] discussed the testing methods of Replication with Validation (RV) and Safe Adaptation with Validation (SAV). While testing a conventional application, we formulate creative test cases, manually explore the product, or write automated test scripts, testing AI-based products that focus on data and analytics. Testing supervised ML has two major phases: training validation and relevance testing. Training data is tested in the training validation phase which is verified during the validation phase. A genetic algorithm is used to generate test cases to cover all DU-pairs in [20]. As per the genetic algorithm, BP model is used which uses BP neural network.

Besides, other methods have been used to test AI software. The testing methods of RAP (Reconfiguration Automation Project) and FEID timeline set up were discussed in [21]. The automation of the several phases of the flight software testing procedure is the actual idea behind RAP. Also, it introduced Artificial Intelligence into the Space Shuttle flight software testing. Different models of AI systems were discussed in [22]. The authors discussed building testable AI systems, limiting the AI system to propositional logic and intervening variables in reducing testing. Ramanathan et al. [23] used symbolic decision procedures coupled with statistical hypothesis testing to validate machine learning algorithms for studying the correctness of intelligent systems. They also used an algorithm to analyze the robustness of a human detection algorithm built using the OpenCV open-source computer vision library.

In the field of image recognition, most researchers focus on recognition algorithms. Girshick et al. [24] proposed the R-CNN algorithm, which added selective search operations to the CNN network to identify candidate regions. The algorithm first divides the candidate regions of the input images and then extracts the characteristics of the candidate regions through the CNN network model for classification and recognition. He K et al. [25] proposed the SPP-Net algorithm, which reduced the process of image normalization and solved the problem of image information loss and storage. Girshick R [26] proposed the Fast R-CNN algorithm, which refers to the Region of Interest (RoI) and the

multi-task loss function method, and replaces SVM classification and linear regression with Softmax and SmoothLoss to realize the unification of classification and regression and reduce the disk space.

Moreover, Redmon et al. [27] proposed the YOLO algorithm, which can identify the categories and locations of multiple items in an image at one time, realizing end-to-end image recognition. First, the YOLO algorithm meshes the input images, calculates the confidence degree and classification probability of the existence of target objects in each grid, and removes the mesh without target objects by threshold. The YOLO algorithm runs fast, but has low recognition accuracy. Liu et al. [28] proposed SSD algorithm, which is a multi-target detection algorithm to directly predict the target category. It first extracts the feature map through the CNN network model, and then carries on the regression classification to the feature map. At the same time, SSD algorithm adds a multi-scale feature graph function, which can return the candidate boxes of different sizes on the feature graph of different levels, detect targets of different sizes, and improve the recognition accuracy.

However, the evaluation of image recognition system is relatively less but important. In [29], an implementation of Yolo-v2 image recognition and other test benches for a deep learning accelerator were presented. They converted the Yolo-v2 software to 16-bit floating point version and used it in the simulation and FPGA experiment during the chip development. Several other testbenches were designed and used to test various networks.

In [30], Yu et al. presented an image-driven tool, namely LIRAT, to record and replay test scripts across platforms, solving the problem of test script cross-platform replay for the first time. LIRAT recorded screenshots and layouts of the widgets, and leverages image understanding techniques to locate them in the replay process.

3. Methodology

The testing methodologies of the software ensure that the software meets the client's requirements. To do this, different types of strategies are used in accordance with the application to be tested. Every test methodology consists of an objective, method and results. A clear understanding of these terms is needed to conduct thorough testing of the application. We have used two different testing techniques to test image recognition applications using manual testing and automation testing.

3.1 Manual Test

The following are the test methods that we can use to efficiently and elegantly complete the software test process and ensure good quality.

1) Equivalence Partitioning Method

Equivalence class partitioning (ECP) is one of the software testing techniques that divides the data that is obtained as input of software into partitions of equivalent information with the help of which test cases are systematically derived. By principles, test cases are designed to provide coverage to each of these partitions once or at least once. The following table is an example of an equivalent partition for tree detection.

Table 1: Equivalence partitioning of tree detection

Equivalence Partitioning	
(1) The height of the tree	A) >20ft
	B) <20ft
(2) Picture completeness	A) The whole tree
	B) The trunk of the tree
	C) The root of the tree
	D) The branch of the tree
(3) Flowers	A) Without flowers
	B) With flowers

2) Boundary Value Testing

Boundary value analysis refers to the testing technique where tests are designed for validating software behaviors and functions by focusing on boundary values for each boundary in the system.

3) Category-Partition Testing

The Category-partition method is to divide the input domains of a component into N different disjoint partitions, and then select one value from each domain, combine them as one single test case. For example, it is used for vehicle detection and we choose to define the category of vehicle as follows:

A. Vehicle Type

1. Motor 2. Bus 3. Truck 4. Car

B. Vehicle Direction

1. Front 2. Side 3. Back

The result of the test cases is shown below.

Table 2: Test cases of the vehicle detection

Test Case #	Test Case	Description	Result
1	A1B1	Front of Motor	**Fail**
2	A1B2	Side of Motor	Pass
3	A1B3	Back of Motor	Pass
4	A2B1	Front of Bus	Pass
5	A2B2	Side of Bus	Pass
6	A2B3	Back of Bus	Pass
7	A3B1	Front of Truck	Pass
8	A3B2	Side of Truck	Pass
9	A3B3	Back of Truck	Pass
10	A4B1	Front of Car	Pass
11	A4B2	Side of Car	Pass
12	A4B3	Back of Car	Pass

4) Scenario-Based Testing

Scenario-based testing is one of the testing methods that consists of the business process flow tested end to end. Some test steps should be written in a way that completes and validates the positive flow of an application.

5) Decision Table

Testing is one of the best ways to deal with various combinations of input that produce different kinds of results. This is also referred to as Cause-Effect table. It provides a systematic way of stating complex business rules, which is useful for developers as well as for testers.

3.2 Automation test

Automation testing uses different types of tools, scripts, and software to perform test cases by reusing predefined actions. It is more reliable if the scripts are well written, because the machine can perform related tasks without error. Moreover, it is faster than manual testing which can be used when we have something repetitive to test.

It is impossible to automate all test cases. Therefore, we need to consider the test cause that can be automated. Different testing reasons can be automated, such as repetitive tasks and capturing results rather than manually collecting data and creating graphs. We can create a tool to capture the results for us so that we can save time and effort. In addition, the task of data entry also needs to be automated so that it is done automatically without having to enter the data manually or write it to the archive form.

We need to identify and create an automation plan by identifying the goal of the automation test. We need to know the type of test we want to do. After that, we will select the right tool that will help us with testing. It is important to pick the right tool to get a good result of automated testing. After selecting the right tools, we need to know the scope of the automation by selecting which tests to automate. It could be the features that are remarkable for the business, scenarios (which have a big amount of data), the technical feasibility of which business components are used, or the complexity of the test cases.

We usually use Appium as the main test framework. The main advantage of Appium over other test frameworks is that it is open-sourced and can be used to test native mobile apps. Another advantage of picking Appium is that it can be used across different platforms (mac and windows) and can test against various mobile operating systems such as Windows, iOS and Android. Appium comes with vendor-provided automation frameworks.

4. A Case Study

4.1 Test Setup

This paper took the test Calorie Mama APP as an example, using manual testing and automated testing respectively as seen in figure 1.

The Calorie Mama is an app designed to help the user achieve the weight goal he/she sets. In this app, one key function to do food tracking is to let the user take a picture of the food, and the app would recognize the food contents in the picture and display the food calories in the meal.

The test data is a mix of various sources: images from Google, images clicked in real-time using a smartphone camera. The experiments were performed with a high-resolution and high-quality camera.

Test coverage in software testing measures the factors including information about which part of the program is executed and how much code is utilized when running the test suite. Some of the benefits of test coverage are listed below. It assures the quality of the automation test and helps in identifying the exact portion of the code utilized. It makes sure that time, cost and complexity are under control and also identifies the gaps in requirements, test cases and errors.

The coverage of functional tests depends on the design of test cases. If the test cases are fully covered, the coverage of functional tests will be high. If N is the total test cases and M is the number of test cases for execution, the function-based test coverage percentage will be calculated as follows:

Test Coverage = M/N *100%

This paper focuses on function-based test coverage.

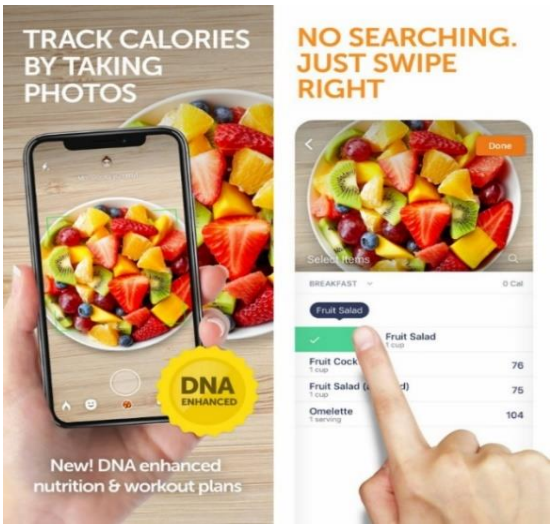

Figure 1: Information about Calorie Mama APP.

4.2 Test Experiment

(1) Manual Test

In this approach of manual testing, we selected conventional decision tables to test. A decision table is a table with various conditions and their corresponding actions. It is divided into four parts, condition stub,

action stub, condition entry, and action entry. We use decision tables to test manually from two aspects, namely detection of non-food and food items.

1)Detection of non-food items: Different non-food items are input into the Calorie Mama APP and the results were shown on the user interface. A summary of the detection of the non-food items can be seen in the following decision table. Bold content indicates that the test failed.

As we can see, the condition stub is designed as two conditions, including the state of the Internet and access to the Camera, which is essential for the image recognition software. When not turning on WIFI or Cellular, and not allowing access to the Camera, image recognition will not work. Besides, the application detected artificial pumpkin and artificial cake as food items. In contrast, it could not correctly identify the butter block. As a result, it failed in some of the cases.

Table 3: Decision table of the non-food items

	Turn on WIFI or Cellular	Allow access to Camera	Food item	Detected as food	Not detected as food
R1	T	T	Pen	F	T
R2	T	T	Apple	T	F
R3	T	T	**Artificial Pumpkin**	**T**	**F**
R4	T	T	**Butter Block**	**F**	**T**
R5	T	T	Banana	T	F
R6	T	T	Chicken Wings	T	F
R7	T	T	Clarified Butter	T	F
R8	T	T	**Artificial Cake**	**T**	**F**
R9	F	F	Glass of Water	-	-

2)Detection of food items: We divided the generic term of food items into four categories which are Indian cuisine, raw fruits and vegetables, a variety of apples and eggs, and food items in different backgrounds.

We sampled some food items under the Indian food category and fed the images to the application. The application was able to recognize some of the food items while it failed in many as seen in the table below.

Table 4: Decision table of Indian cuisine

	Turn on WIFI or Cellular	Allow access to Camera	Food item	Exact detection or correct choices	Offered wrong choices
R1	T	T	Lentils	T	F
R2	T	T	Sev	T	F
R3	T	T	**Potato Capsicum**	**F**	**T**
R4	T	T	**Okra**	**F**	**T**
R5	T	T	**Prawns &Okra**	**T**	**F**
R6	T	T	**Rice&Fish**	**F**	**T**
R7	T	T	Lamb Curry	T	F
R8	T	T	**Mixed Lentil &Rice**	**F**	**T**
R9	T	T	Vegetable Biryani	T	F
R10	T	T	Samosa	T	F

For raw fruits and vegetables, the application was given an input of raw fruits and vegetables. It recognized a majority of the food items but failed in a few cases as seen in table 5 below.

Table 5: Decision table of raw fruits and vegetables

	Turn on WIFI or Cellular	Allow access to Camera	Food item	Exact detection or correct choices	Offered wrong choices
R1	T	T	Apple	T	F
R2	T	T	Fig	T	F
R3	T	T	Mango	T	F
R4	T	T	Okra	T	F
R5	T	T	Horse Radish	T	F
R6	T	T	**Oppo Squash**	**F**	**T**
R7	T	T	**Bitter Gaurd**	**F**	**T**
R8	T	T	Lettuce	T	F
R9	T	T	**Mustard Greens**	**F**	**T**
R10	T	T	**Loquat**	**F**	**T**

For a variety of apples and eggs, we decided to test the application under different varieties of the same food items. For this particular case, we considered the different varieties of apples. As shown in table 6, the application failed to recognize a majority of the apple varieties.

Table 6: Decision table of apples

	Turn on WIFI or Cellular	Allow access to Camera	Food item	Exact detection or correct choices	Offered wrong choices
R1	T	T	**Cortland**	**F**	**T**
R2	T	T	**Gala**	**F**	**T**
R3	T	T	**Golden delicious**	**F**	**T**
R4	T	T	Granny Smith	T	F
R5	T	T	**Fuji**	**F**	**T**
R6	T	T	**Honey Crisp**	**F**	**T**
R7	T	T	**Macintosh**	**F**	**T**
R8	T	T	Red Delicious	T	F
R9	T	T	Dry Apple	T	F

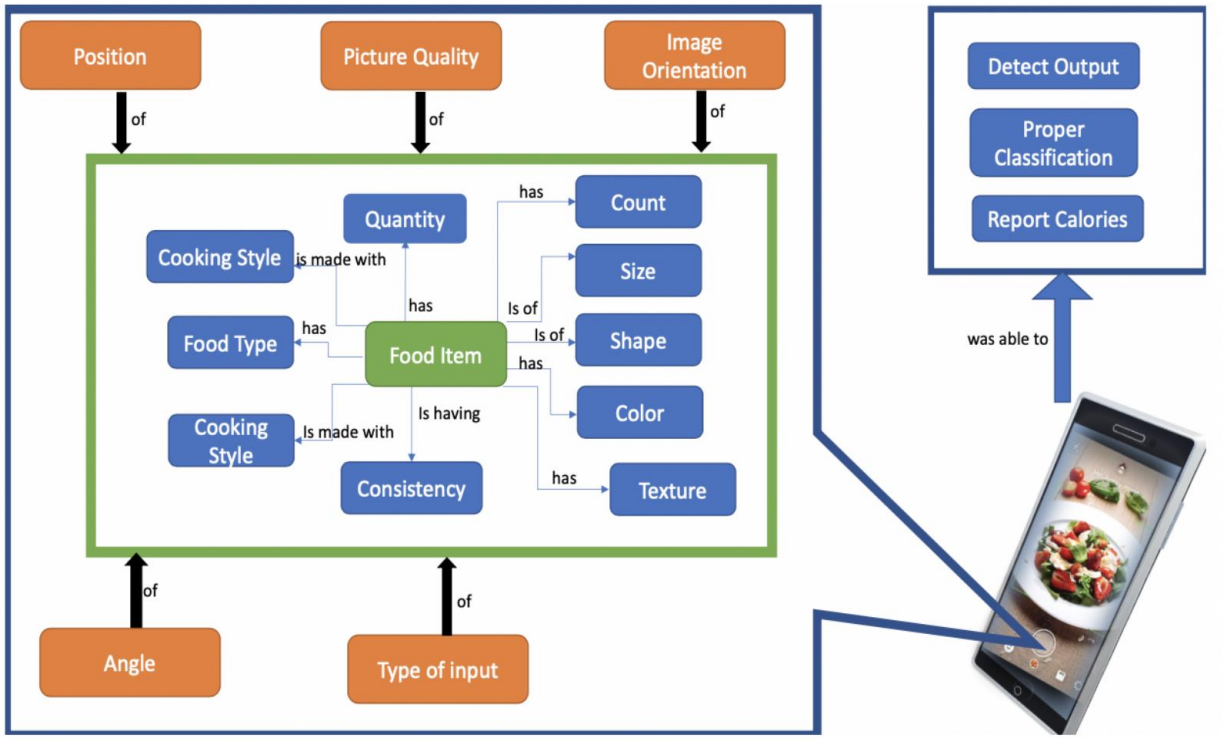

Figure 2: A logical data model.

Finally, for food items in different backgrounds, the background of food is a very important aspect and we decided to test the application with images of food items with different backgrounds. As seen in table 7, the Calorie Mama application was able to correctly recognize the food items when given inputs with red, blue, and wooden backgrounds. However, the application detected wrong when the egg is in a tray.

Table 7: Decision table of food items in different backgrounds

	Turn on WIFI or Cellular	Allow access to Camera	Food item	Exact detection or correct choices	Offered wrong choices
R1	T	T	Blue Back-ground	T	F
R2	T	T	Red Back-ground	T	F
R3	T	T	Wooden Background	T	F
R4	T	T	Egg in a bowl	T	F
R5	T	T	Egg on a plate	T	F
R6	T	T	Egg on a pan	T	F
R7	T	T	Egg in the glass	T	F
R8	T	T	Egg in a jar	T	F
R9	T	T	**Eggs in a tray**	**F**	**T**

After conducting the manual testing, we experienced its various drawbacks, and it is time-consuming. Also, load testing and performance testing are not possible under manual testing. Besides, regression test cases are very costly. Due to these drawbacks, we decided to shift to automation testing.

(2) Data Modeling

The three-dimensional (3D) classification decision table is influenced by the concept of conventional decision tables to conduct classification-based test requirement analysis and modeling for any given mobile apps powered with AI functions using a 3D tabular view.

As seen in figure 2, a logical data model is created after brainstorming and observing the various possibilities for the input image of food along with the context in which the image was clicked. This information was further utilized to create a 3D classification table.

The major testing focus for a 3D classification table is the mappings among classified disjoint context conditions, classified input selections, and classified AI function outputs. These mappings are known as image recognition function classification rules. Each of them represents the conjunction among three different views.

Test case design and generation based on a 3D classification decision table must cover these image recognition classification rules. Adequate image recognition function testing coverage could be assessed. Next, we introduce the construction of each one-dimensional model in the 3D decision table.

1) Input Modeling

The input classification refers to the parameters and their values that represent different test case scenarios.

Each parameter has multiple possible values which when combined with context values gives us the final set of test cases. The following figure shows Calorie Mama's input classification tree, which contains information about the type of food being clicked, such as what the food is, and the physical appearance of the food, such as quality, size, shape, consistency, etc.

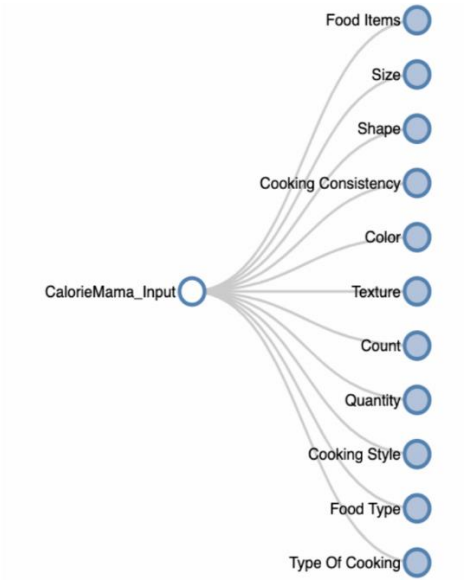

Figure 3: A sample input classification tree.

2) Context Modeling

The context classification tree contains information about the image context. It is basic information about the image itself and not specifically about the item in the image. For example, the context classification tree contains information like if the image is blurry or not well illuminated, what is the angle of the camera while clicking the image, if the image is rotated or so, etc. The following figure shows Calorie Mama's context classification tree.

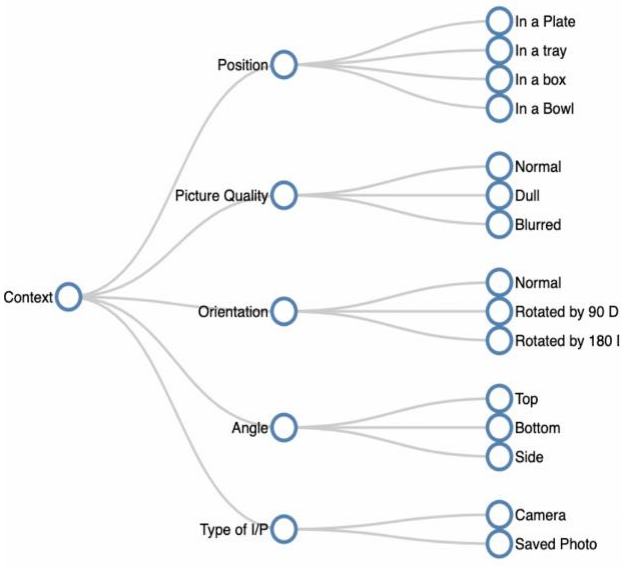

Figure 4: A sample context classification tree.

3) Output Classifications

The output classification tree contains information about the output. Various parameters regarding the output obtained from the application will be considered. This can be modified based on the requirements and results expected from the application. The following figure shows Calorie Mama's output classification tree.

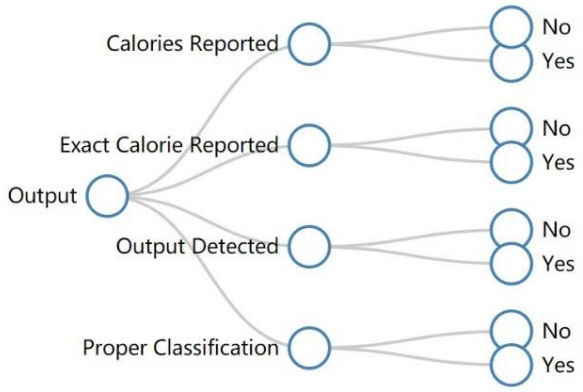

Figure 5: A sample output classification tree.

(3) Automation Test

After data modeling, we performed automation testing with minimal human assistance on top of the model. Automation testing can increase coverage for test data and come up with more concluding test results for the selected mobile app. We used Appium as an automation tool to perform automation on the mobile app. Appium acts as a server that launches the app into the simulator or a real device and can access the elements for processing the actions triggered by the automation script which we wrote in Java.

Steps to perform the automation were:

1) Install Appium server.

Appium is an open-source test automation framework used to run automation scripts on mobile apps. To install Appium on our development machine, we installed NodeJs and Node Package Manager which is required for installing Appium.

We had Homebrew installed already which is a smart package manager for installing packages on Mac machines. With the help of brew, we installed node by running the below command,

brew install node

This installs the Node and Node Package Manager.

Below command installs Appium.

npm install Appium

After the appium is installed, we can just run Appium command to launch the Appium server.

By default, Appium starts on port 4723.

2) Create the automation environment for Android.

After Appium is successfully installed and launched, the next step is to create the automation environment for the mobile operating system which we are using to automate our mobile app. This section discusses creating an automation environment on Android while the next section takes iOS into account.

To create an automation environment for Android, we install Android Studio.

Android Studio enables us to create Android emulators with customizable hardware specifications. As we launch a new emulator, it will behave as a real device connected to the machine and we can actually use it to launch and automate our target app.

3) Create the automation environment for iOS.

We install XCode for creating an automation environment for iOS mobile app. iOS lets us create iOS simulators that behave exactly like an iOS mobile device.

We can launch iOS simulators and can change the specifications of the OS and hardware as required. We can run any iOS app on these simulators using XCode or launching simulators after they are once initialized by XCode.

4) Launch simulator/ Connect a real device.

Either we can connect our real device to run the automation scripts on our app or we can use the simulators. If we are using a real device, then we need to install Android Debug Bridge to get the device IDs for proceeding with the automation.

We need device IDs to enter in the script so that Appium can connect with the connected device.

5) Install Eclipse.

The next step after creating the automation environment is to install Eclipse, as Eclipse will be the used IDE to write the automation scripts.

Download link: https://www.eclipse.org/downloads/ Before installing Eclipse, we need to have Java Development Kit installed in our machine. Download JDK:https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

After JDK is installed, we install Eclipse and are ready to start with our automation.

6) Create a maven project in Eclipse to write and run the automation script.

We start automation by creating a Maven project in Eclipse. Maven is a build automation tool that is used to automatically install all the dependencies involved or required by our project. We don't need to worry about installing the dependent libraries one by one. In a maven project, we use the Project Object Model (pom.xml) where we write all the names of the required dependencies in a fixed XML format and then Maven sets the platform for us thereafter.

We provide the dependencies of Appium, Selenium, TestNG in the Project Object Model and then start writing the scripts. We use TestNG to run our automation tests. Soon after the execution of tests, test results are visible in the Eclipse console.

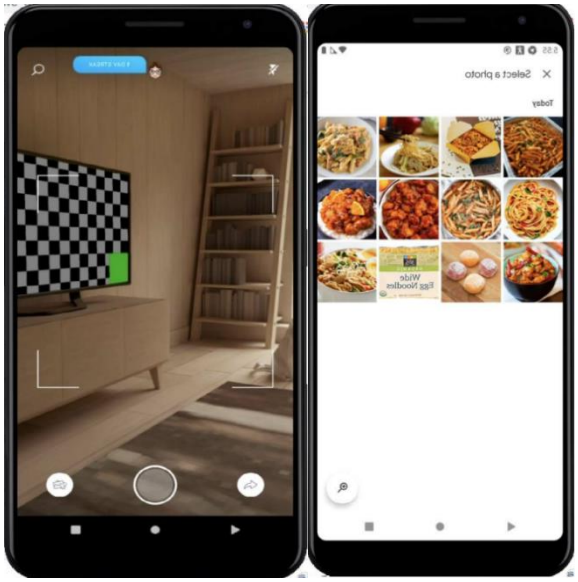

Figure 6: App can take input either through camera or from the gallery.

For the algorithm of the app automation, one image which is selected from the gallery of the phone is fed as an input into the target app as seen in figure 6, and the result of the execution is compared with the expected output. If the output from the target app is as expected, then the test case is displayed as passed or else failed. Also, when the app produces the output, more options, as provided by the app are taken into account. While showing the output to the user, there is an option to see more options from the suggestions coming from the app. The algorithm considers all those options as the output from the app and then decides if the test case is passed or failed.

4.3 Test Results

After applying manual testing and automation testing, we compare the coverages for both manual and automation tests. In manual testing, the coverage of the test case was limited due to timing. It was difficult to cover a larger set of data without the use of tools or scripts. On the other hand, automation testing has higher coverage because the tools and script helped us to cover more test cases. Figure 7 below shows that in automation testing we were able to cover more test sets of data than the manual testing over the same time. Approximately, in the automation testing, we were able to cover twice of what we covered in the manual testing.

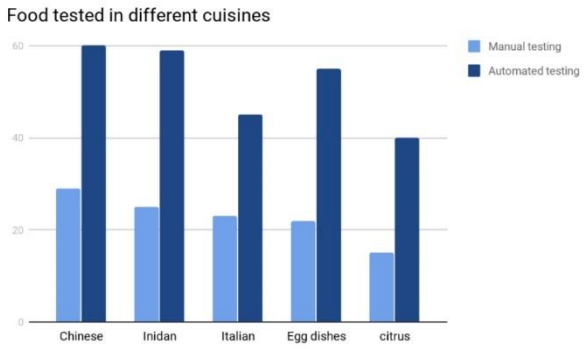

Figure 7: Test coverage for manual and automation test.

The app was able to detect objects, recognize them, and classify them with its name. However, it does not tell the count or sub-classification of the food item. Moreover, testing Calorie Mama App, required a lot of time to do both manual testing and automation testing. Manual testing needs to take more time to generate all decision tables, analyze different test causes and test manually. On the other hand, in automation testing, we spend days getting the script working correctly and program it to do the testing automatically.

The following figure shows the results of the manual testing and automation testing of the Calorie Mama APP. In manual testing, the total test food item across different cuisines was four hundred items and each cuisine has eighty food items. For example, of the 80 Chinese cuisines, 26 were detected as errors and 54 were detected as passes. As a whole, 132 of them were wrongly detected they were bugs in the app. This gives us a 33 failed percentage and the passing percentage is 67. The diagram below shows the failing and passing results.

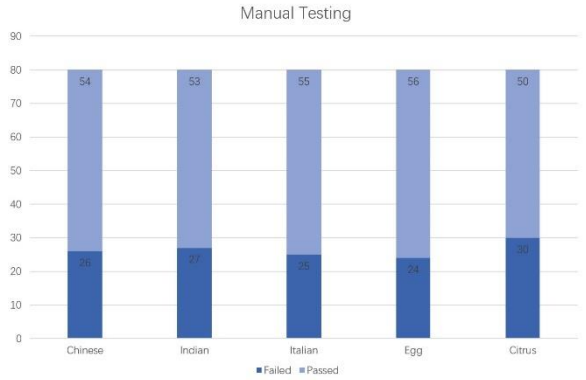

Figure 8: Manual testing.

In Automation testing, we tested four hundred different images in different cuisines similarly. We found out that out of the 400 images, 175 failed and 225 passed. This gives us a failure percentage of 43.75 and a passing percentage of 56.25 as shown in figure 9.

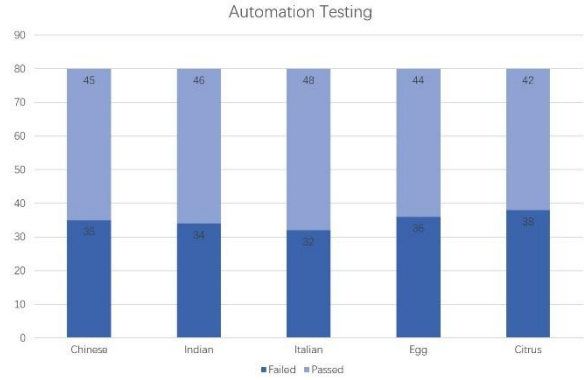

Figure 9: Automation testing.

Comparing the manual testing with automation testing, we can see that the errors that were found in the automation testing are higher than the errors that were found by the manual testing because the automation test allows us to test different inputs in a short time.

Also, in manual testing, it is more likely to make human mistakes because doing repeated tasks over time generates more errors by humans. Besides, manual testing can be expensive and time-consuming.

However, doing a repeated test using automation by writing a script and let the machine discover the error is more efficient. It helps them find errors without the need of performing redundant tasks. However, it needs talented and experienced people to do that, which is expensive. Besides, it is difficult to automate all kinds of testing where not everything can be redundant and reusable.

5. Conclusion

To sum up, we mainly leverage two methods to test the image recognition system, namely manual testing, and automation testing. We found that automation testing discovers more errors than manual testing.

In manual testing, the test is conducted by human testers inputting the use cases one by one, and observing the results. It is subject to human error; therefore, it is not one hundred percent accurate. On the other hand, in automation testing, the testers use tools and scripts to help them conduct the test among the image recognition software, which can save labor and time cost, thus improving testing efficiency.

For future work, we will evaluate more image recognition mobile apps with more datasets. Moreover, we plan to implement an automatic testing tool for detecting errors.

References

[1] Gao J., Tao C., Dou J. and Lu S., 2019, “Invited paper: What is AI software testing? and why,” 13th IEEE International Conference on Service-Oriented System Engineering, SOSE 2019, San Francisco, CA, USA, April 4-9, 2019, pp 27-36. doi: 10.1109/SOSE.2019.00015.

[2] Hourani H., Hammad A. and Lafi M., "The Impact of Artificial Intelligence on Software Testing," 2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT), Amman, Jordan, 2019, pp. 565-570. doi: 10.1109/JEEIT.2019.8717439.

[3] Zhu H., Liu D., Bayley I., Harrison R. and Cuzzolin F., "Datamorphic Testing: A Method for Testing Intelligent Applications," 2019 IEEE International Conference On Artificial Intelligence Testing (AITest), Newark, CA, USA, 2019, pp. 149-156. doi: 10.1109/AITest.2019.00018.

[4] King T. M., Arbon J., Santiago D., Adamo D., Chin W. and Shanmugam R., "AI for Testing Today and Tomorrow: Industry Perspectives," 2019 IEEE International Conference On Artificial Intelligence Testing (AITest), Newark, CA, USA, 2019, pp. 81-88. doi: 10.1109/AITest.2019.000-3.

[5] Marijan D., Gotlieb A. and Ahuja M. K., "Challenges of Testing Machine Learning-Based Systems," 2019 IEEE International Conference On Artificial Intelligence Testing (AITest), Newark, CA, USA, 2019, pp. 101-102. doi: 10.1109/AITest.2019.00010.

[6] Chen T. Y., Kuo F.C., Liu H., Poon P., Towey D., Tse T., and Zhou Z., "Metamorphic testing: A review of challenges and opportunities," ACM Computing Surveys 51(1), 4:1-4:27 (2018).

[7] Chen T. Y., Cheung S., and Yiu S., "Metamorphic Testing: A New Approach for Generating Next Test Cases," Technical Report HKUST-CS98-01, Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong (1998).

[8] Tian Y., Pei K., Jana S., and Ray B., "DeepTest: Automated Testing of DeepNeural-Network-Driven Autonomous Cars," Proceedings of the 40th International Conference on Software Engineering (ICSE), pp. 303-314. Gothenburg, Sweden (2018).

[9] Zhang M., Zhang Y., Zhang L., Liu C., and Khurshid S., "DeepRoad: GAN-Based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems," 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 132-142. Montpellier, France (2018).

[10] Sun L. and Zhou Z. Q., "Metamorphic Testing for Machine Translations: MT4MT," 2018 25th Australasian Software Engineering Conference (ASWEC), 2018, pp. 96-100, doi: 10.1109/ASWEC.2018.00021.

[11] Pesu D., Zhou Z. Q., Zhen J. and Towey D., "A Monte Carlo Method for Metamorphic Testing of Machine Translation Services," 2018 IEEE/ACM 3rd International Workshop on Metamorphic Testing (MET), 2018, pp. 38-45.

[12] Murphy C., Kaiser G. E., Hu L., and Wu L., "Properties of machine learning applications for use in metamorphic testing," Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE), pp. 867872. San Francisco, CA, USA (2008).

[13] Murphy C., Kaiser G. E., Hu L., and Wu L., "Properties of machine learning applications for use in metamorphic testing," Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE), pp. 867872. San Francisco, CA, USA (2008).

[14] Brown J., Zhou Z., and Chow Y., "Metamorphic Testing of Navigation Software: A Pilot Study with Google Maps," 51st Hawaii International Conference on System Sciences (HICSS), pp. 1-10. Hilton Waikoloa Village, Hawaii, USA, (2018).doi: 10.24251/HICSS.2018.713.

[15] Zhou Z., Xiang S., and Chen T. Y., "Metamorphic testing for software quality assessment: A study of search engines," IEEE Transactions on Software Engineering 42(3), 264-284 (2016). doi: 10.1109/TSE.2015.2478001.

[16] Tao C., Gao J. and Wang T., "Testing and Quality Validation for AI Software–Perspectives, Issues, and Practices," IEEE Access, vol. 7, pp. 120164-120175, 2019. doi: 10.1109/ACCESS.2019.2937107.

[17] Wang S. and Su Z., "Metamorphic object insertion for testing object detection systems," Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE '20). Association for Computing Machinery, New York, NY, USA, 1053–1065.

[18] Rauf A. and Alanazi M. N., "Using artificial intelligence to automatically test GUI," 9th International Conference on Computer Science Education, Vancouver, BC, 2014, pp. 3-5. doi: 10.1109/ICCSE.2014.6926420.

[19] King T. M., Arbon J., Santiago D., Adamo D., Chin W. and Shanmugam R., "AI for Testing Today and Tomorrow: Industry Perspectives," 2019 IEEE International Conference On Artificial Intelligence Testing (AITest), Newark, CA, USA, 2019, pp. 81-88.

[20] Ji S., Chen Q. and Zhang P., "Neural Network-Based Test Case Generation for Data-Flow Oriented Testing," 2019 IEEE International Conference On Artificial Intelligence Testing (AITest), Newark, CA, USA, 2019, pp. 35-36. doi: 10.1109/AITest.2019.00-11.

[21] DeMasie M. P. and Muratore J. F., "Artificial intelligence and expert systems in-flight software testing," IEEE/AIAA 10th Digital Avionics Systems Conference, Los Angeles, CA, USA, 1991, pp. 416-419.

[22] Liu G., Liu Q. and Zhang W., "Model-based testing and validation on artificial intelligence systems," Second International Multi-Symposiums on Computer and Computational Sciences (IMSCCS 2007), Iowa City, IA, 2007, pp. 445-449.

[23] Ramanathan A., Pullum L. L., Hussain F., Chakrabarty D. and Jha S. K., "Integrating symbolic and statistical methods for testing intelligent systems: Applications to machine learning and computer vision," 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, 2016, pp. 786-791.

[24] Girshick R., Donahue J., Darrell T. and Malik J., "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, 2014, pp. 580-587. doi: 10.1109/CVPR.2014.81.

[25] He K., Zhang X., Ren S. and Sun J., "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," in IEEE Transactions on Pattern Analysis and Machine Intelligence,2015,37(9):1904-1916.

[26] Girshick R., "Fast R-CNN," 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 2015, pp. 1440-1448. doi: 10.1109/ICCV.2015.169.

[27] Redmon J., Divvala S., Girshick R. and Farhadi A., "You Only Look Once: Unified, Real-Time Object Detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.

[28] Liu W., Anguelov D., Erhan D., et al., "SSD: Single Shot MultiBox Detector," Welling M. (eds) Computer Vision – ECCV 2016. Lecture Notes in Computer Science, vol 9905. Springer, Cham.

[29] Kim C., Jeon I. S., Kwon Y., Kim H., Lyuh C. et al., "Implementation of Yolo-v2 Image Recognition and Other Testbenches for a CNN Accelerator," 2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin), Berlin, Germany, 2019, pp. 242-247. doi: 10.1109/ICCE-Berlin47944.2019.8966213.

[30] Yu S., Fang C., Feng Y., Zhao W. and Chen Z., "LIRAT: Layout and Image Recognition Driving Automated Mobile Testing of Cross-Platform," 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), San Diego, CA, USA, 2019, pp. 1066-1069. doi: 10.1109/ASE.2019.00103.