Seeing AI Mobile Application Testing Process

Anna Systaliuk, Chen Shi, Maytinee Apiwansri, William Wong

Abstract: The steps to improve human life and drive innovation go hand in hand with Artificial Intelligence applications which deliver numerous opportunities for developers to build upon. In an environment where real-world situations are evident, numerous AI applications have shown that it is possible to be used anywhere and everywhere needed. An infinite number of feasible scenarios are set for these applications to operate on. However, the ability to test these infinite scenarios has caused a critical problem for developers and test engineers to handle. The testing approach that has been introduced is Conventional testing, which is used to test the AI application, only allowing a restricted number of tests to be done which may lead to insufficient validation due to the limited context the test can handle. Hence, the ability to defeat this is to use 3D decision tables and classification-based AI software testing to produce distinct test cases for the application. [2] Furthermore, our team compared the four quality assurance metrics which consist of accuracy, correctness, reliability, and consistency for the AI functionality of our application. Our team chose one AI mobile application called Seeing AI to test four different features: Facial Analysis, Text Recognition, Outdoor Scene Analysis, and Indoor Scene Analysis. We compared these features by using conventional testing and AI validation methods as well as doing automation testing by creating scripts. Our case study shows that our generated test cases verify the feasibility and are reliable for test automation.

Keywords-Testing automation, conventional Testing, AI Mobile App

I. Introduction

Within the light of recent technological advances, the power of machine learning has become prominent within our lives. These advances made a significant impact on many in this field which in turn influenced a Microsoft team to release a free mobile application for Apple's iPhone called Seeing AI.

Seeing AI is an AI vision application for the blind and visually impaired. This application recognizes text, documents, people, common objects, colors, products, and many more while also supporting seven different languages. Seeing AI can analyze the scene and detect objects present. Users can take a picture of the scene and hear its description. As the user moves the finger around the screen, the app automatically detects and voice object names. Objects can be broken down into different categories such as what color the object is, what the object is known as, what location, etc. The app recognizes those objects and can display the item on your screen as text or use a voice assistant to tell the user what the object is. As the user scans the surroundings, the app detects how many people are around, how close they are, and what their facial expressions are. Users can also teach the app to recognize a particular person as well.

The testing of the AI mobile application constitutes numerous challenges and conventional methods of testing are not

quite enough to initiate any AI application quality assurance. Testing AI applications posses its own challenges which may include these but are not limited to how AI functions may be tested in the mobile application, identifying the quality assessment and validation criteria requirements for the AI functions, initiating the testing coverage requirements for the AI functions, as well as identifying a "well-defined" testing requirement for AI functions in a mobile application.

With these challenges in mind, our team began by performing automation testing; however, directed towards the bugs and failures discovered while using the conventional testing method. Section 2 discusses the related work regarding AI software testing, which incorporates the basic concepts, as well as the recent work in Conventional Testing and AI Testing. Additionally, section 2 will include the case study analysis of the AI-software testing for AI mobile applications is shown, which includes the conventional testing modeling, AI test modeling for AI features, test case generate an AI feature test case results, and presents an innovative model based on 3D Classification Decision Table for all four of the features [2]. In Section 3, will discuss the comparison of the conventional testing method and the AI testing method. Section 4, will discuss the automated applied with AI testing. Section 5 will discuss the bug analysis comparison for conventional testing, AI testing, automated testing. Section 6 will present the Conclusion.

II. Related work

A. Conventional Testing

Conventional Testing is a form of testing where test engineers will check the ins and outs of an application and ensure that the application is working as intended. There are different conventional test methods which can be used and leveraged to produce more effective testing of an application.

(1) Decision Table

One of the most effective methods used when testing SeeingAi was the Decision Table Testing. Decision Table Testing provides outcomes based on the various input given in the beginning. For example, in the case of Document Recognition, a decision table serves the many different combinations of inputs that can result in different outputs. Having large text with black in on a document may be much easier to transcribe than a document with very tiny font and black ink.

Document - Input Decision Table

			Rules									
			11.1	11.2	11.3	11.4	11.5	12.1	12.2	12.3	12.4	12.5
Conditions	Background	Black	T	T	T	T	T					
		White						T	T	T		T
	Color	Black	T									T
		White		T				T				
		Gray			T				T			
		Blue				T				T		
		Red					T				T	
	Font	Arial	T	T	T	T	T	T	T	T	T	T
		Caveat										
		Pacifico										
		Times New Roman										
	Font Size	Extra Large	T	T	T	T	T	T	T	T	T	T
		Large										
		Medium										
		Small										

Figure 1. Document Decision Table

(2) Scenario Testing

Scenario testing can provide test cases for very complex situations. For instance, in the context of Face Recognition, Decision Table testing is limited to showcasing one output attributes. When an object becomes complex (age of person, hair color, facial expressions, etc), recognizing a person becomes much more difficult. This is when

Scenario Testing comes into play as it can cover criteria as mentioned.

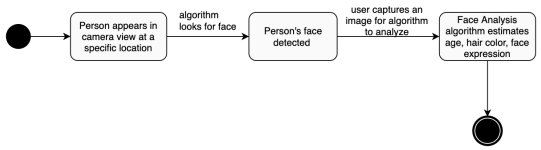


Figure 2. Face Recognition Scenario Test

(3) Equivalence Partitioning

Equivalence Partitioning helps with breaking down test cases into smaller categories that are valid or invalid. It helps break tests into partitions to help with reducing the amount of redundant testing. In the case of Face Recognition, this helps get better statistical data of the algorithm's behavior. For example, a person's age as a teenager can range from 12-18. SeeingAI will try its best to guess what age that person is. If the application guesses the person is a teenager, then that shows that SeeingAI was able to get into a good estimate of the person's age.

(4) Category Partitioning

Category Partitioning helps decompose different categories into functional units that can be tested. In the case of Scenario Testing, having different categories of a scene can help with classifying test cases. For example, if a scene is at a beach compared to at a park, you can categorize the two into two different categories and have test cases for those categories.

Category	Test Case No.	Street/road	Building	Vehicle	People	Animal	Plant	Small still objects
1	1						T	
2	2	T	T					
	3	T		T				
	4	T			T			
	5	T				T		
	6	T					T	
	7		T	T				
	8		T		T			
	9		T			T		
	10		T				T	
	11			T	T			
	12			T		T		
	13			T			T	
	14				T	T		
	15				T		T	
	16					T	T	
3	17	T	T	T				
	18	T	T		T			
	19	T	T			T		
	20	T	T				T	
	21	T		T	T			
	22	T		T		T		

Figure 3. Scene Recognition Category Partitioning

Looking at Figure 4. The test cases are pretty accurate for all features except Face Recognition. Many of the test cases run for Face Recognition had a variety of results. The discrepancy comes from SeeingAI being unable to differentiate certain expressions and ages of a person. For instance, SeeingAI could sometimes capture a person frowning as a person smiling or a person who is 20 years old as someone who is 40 years old. These little differences make up a majority of the test failures seen for Facial Recognition.

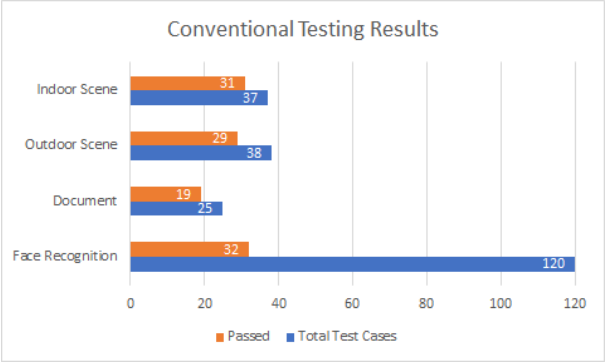


Figure 4. Conventional Testing Results

B. AI Testing

With the fast advance of artificial intelligence and big data computing technologies, more and more software and applications are being developed by using machine learning models, making business and intelligent decisions based on intelligent features such as face recognition, document/text identification, object detection, scene description, and so on[3]. Conventional testing might not be adequate to test those features and functionalities. For this reason, this paper explored AI testing methodology in testing those AI-based features.

The main objectives of the AI testing are:1) Increase efficiency and productivity for a test process; 2) Optimize a test process using AI techniques; 3) Apply optimized test strategies; 4) Increase bug detection effectiveness and speed; 5) Reduce software testing cost[4].

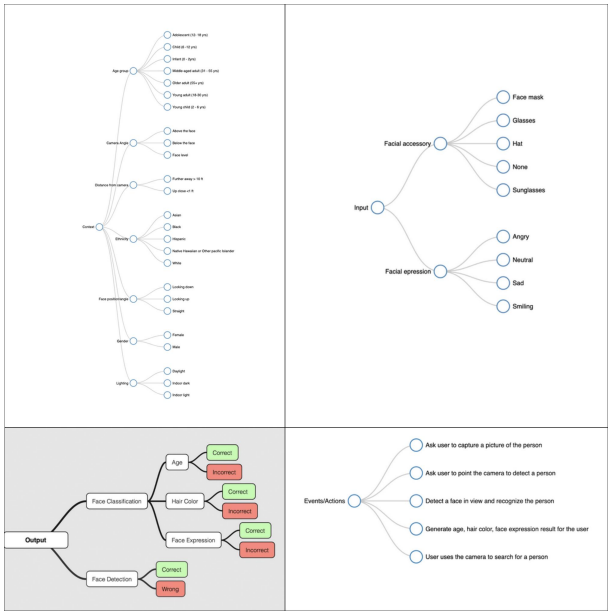
The main focuses of the AI testing in this paper are: 1) Testing four AI functional features in the Seeing AI app to assure their adequate quality in accuracy, consistency, relevancy, timeliness, and correctness; 2) Testing Seeing AI's quality of system service parameters based on performance, and reliability. 3) Apply data-driven AI techniques to facilitate AI testing processes and future test automation.

The main steps of AI testing are: 1) AI function context classification modeling. Identify and classify diverse context conditions and parameters and present the classification using a context classification model, or context classification tree. 2) AI function input classification modeling. Identify and classify diverse input data in terms of its category classes and their sub-classes. Use an input classification model (or input classification tree) as an analysis and test model to facilitate and represent diverse input data. 3) AI function outcome classification modeling. Identify and classify diverse AI function outputs, including texts, audio, video, image, or events/actions. An output classification tree model is generated as the outcome of this step. 4) Generate a 3D classification decision table for each under-test AI function feature to identify three dimensional mappings among disjoint classified context conditions, disjoint classified inputs, and disjoint classified outputs. [1]

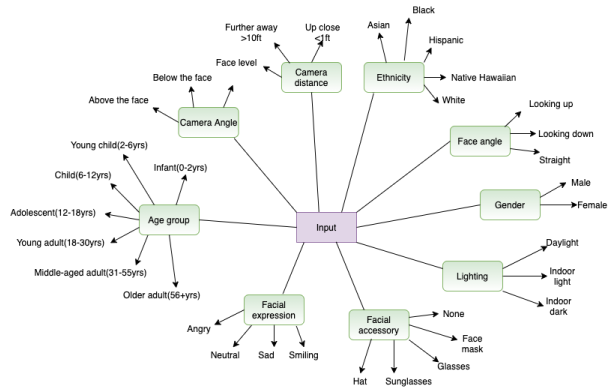
In this section, AI testing modeling process and results for four AI-based functions (Facial Analysis, Text Recognition, Outdoor Scene Analysis and Indoor Scene Analysis) in Seeing AI App are summarized as below.

- ***Facial Analysis***

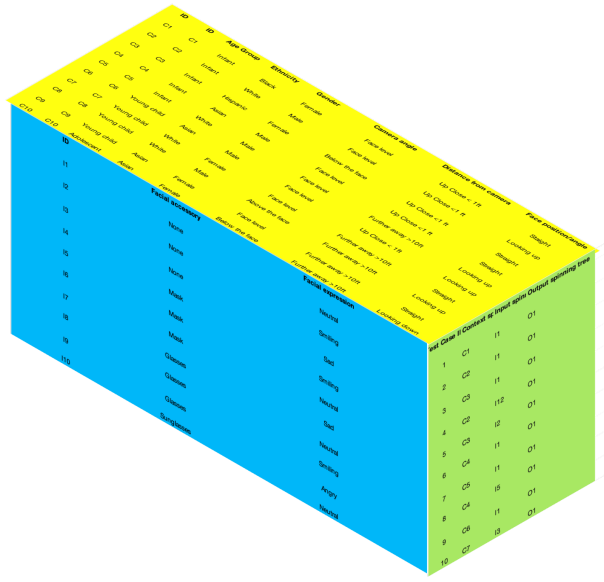
AI testing modeling trees
(context, input, output and events)



AI testing data model



AI testing 3D decision table

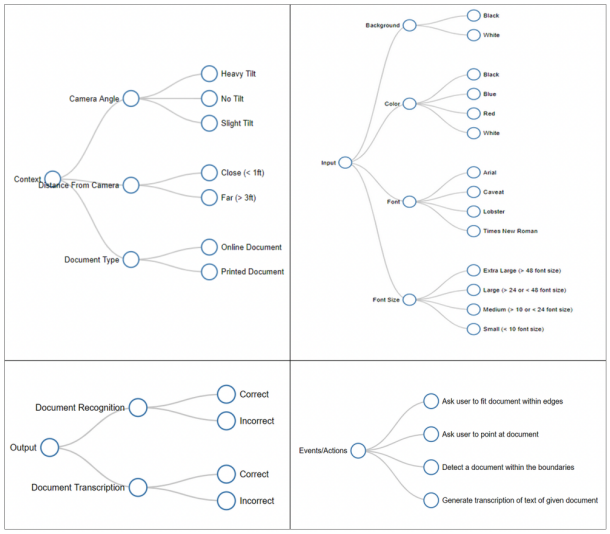


AI testing results

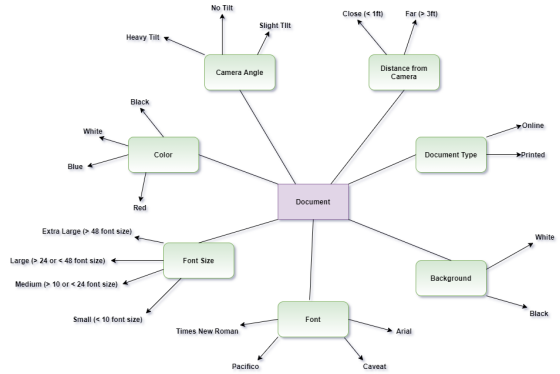
Facial Analysis	Context	Input	Total
Total Test Cases	22	17	39
Passes	18	6	24
Pass Rate	81.82%	35.29%	61.5%

- ***Text Recognition***

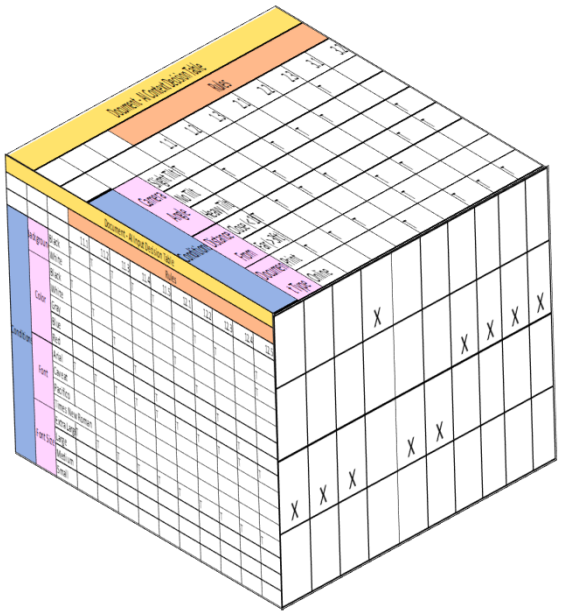
AI testing modeling trees
(context, input, output and events)



AI testing data model



AI testing 3D decision table



AI testing results

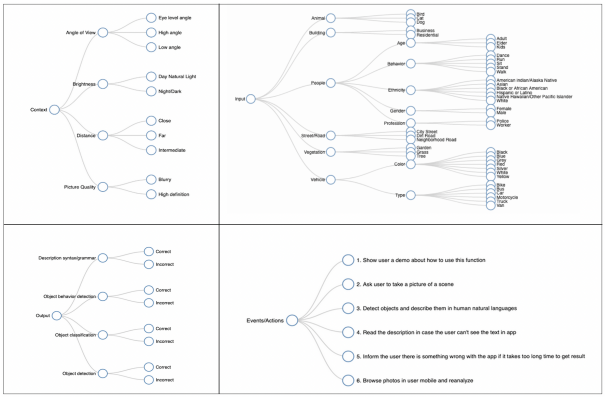
AI testing 3D decision table

AI testing results

Document	Context	Input	Total
Total Test Cases	7	14	21
Passes	1	10	11
Pass rate	14.29%	71.43%	52.24%

- ***Outdoor Scene Analysis***

AI testing modeling trees
(context, input, output and events)



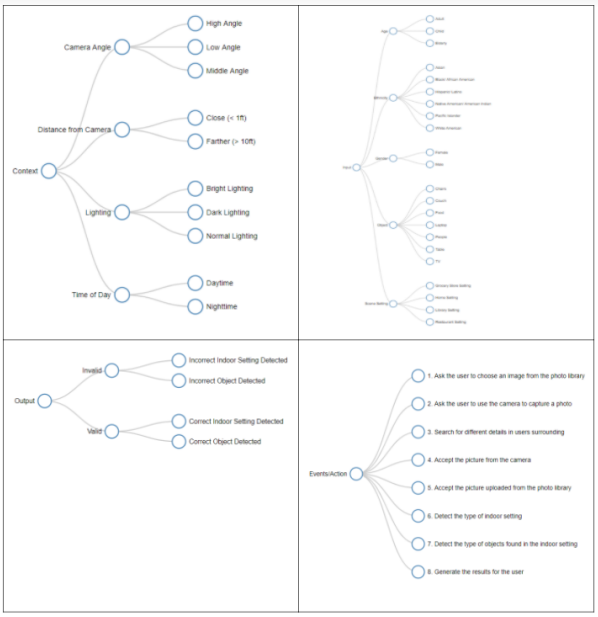
AI testing data model

AI testing results

Outdoor Scene	Context	Input	Overall
Total Cases	8	50	58
Pass	5	28	33
Pass rate	62.5%	56.0%	56.9%

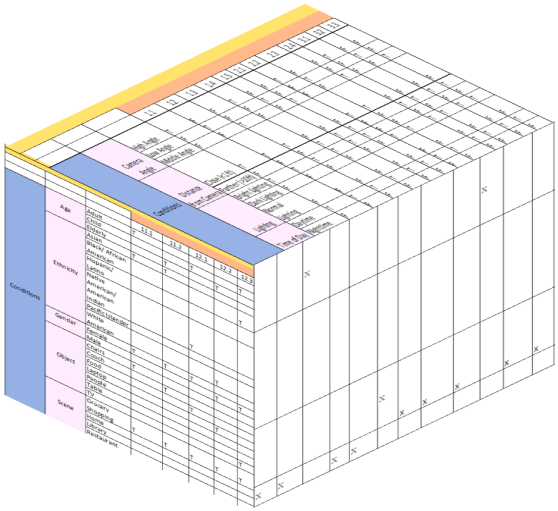
- ***Indoor Scene Analysis***

AI testing modeling trees
(context, input, output and events)



AI testing data model

AI testing 3D decision table



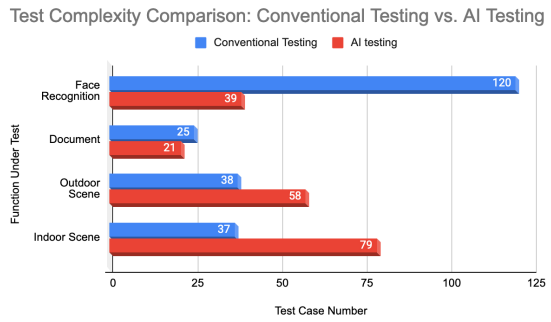
AI testing results

Total	Seeing AI
Pass Rate	47 / 79
Pass Percentage	59.50%

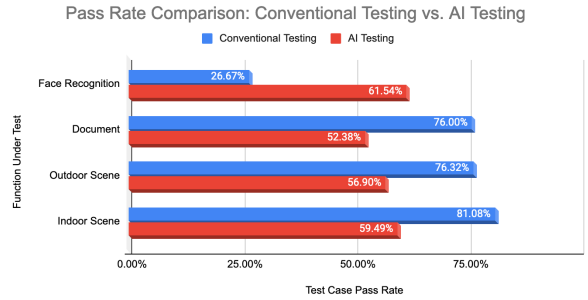
III. Comparison of Conventional Testing & AI testing

Results from conventional testing and AI testing for the four AI functions under test for the Seeing AI app were summarized and analyzed in Figure (a) and (b). Figure(a) shows the test complexity comparison between Conventional Testing and AI testing. We can see that both Outdoor Scene and Indoor Scene testing has a higher complexity compared with that of conventional testing, which should be because more test cases are generated by adopting AI modeling. Document Recognition has a similar test complexity as that of conventional testing and Face Recognition's test complexity decreased. This could be the result of different characteristics of each function and/or how or which conventional testing method was conducted. For AI testing, the test complexity for four functions are more similar to each other, showing how a systematic method can reduce the variety in AI funcion testing results from different conventional methods.

Test case pass rate comparison for conventional testing and AI testing is shown in Figure(b). For 3 out of 4 AI functions under test, test case pass rate decreased from 76%-81% to 52%-59%. More bugs were found after adopting AI testing for those three AI functions. Results from those three functions showing a very uniform trending of passing rate.



Figure(a)



Figure(b)

From the results above we can see that conventional testing is not accurate and adequate enough in testing AI functions because the variety of different conventional methods might generate variety in results and conventional testing methods can't provide enough test complexity to show enough sources of potential problems.

IV. Automation applied with AI Testing

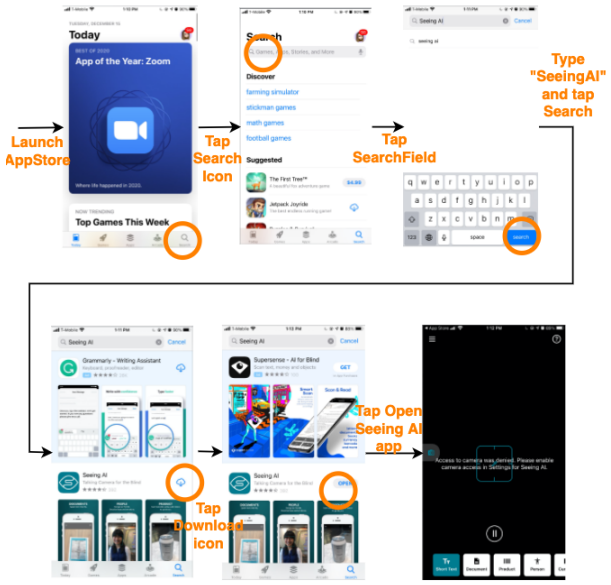
- **Automation Tools and Environment**

Because we do not have access to the source code of the Seeing AI application, we were limited in the scope of white-box testing. Hence, during our research for the automation tools we can use, we focused on finding the most optimal tool to perform UI testing that is used to test only the parts of software that users can come to contact with.

In order to perform UI Test Automation, we decided to use XCUITest - Apple native UI testing framework. XCTest is used for Unit testing and CXUITest is used for UI testing and CXUITest is built on top of XCTest to support a lot of its functionality, like XCTAssert statements used to detect bugs in our tests. XCUITest also runs within Xcode IDE and our team used Swift to write UI tests for automated testing of our iOS application as this language supports most up-to-date Apple testing functionality. [5](Mischinger, 2019)

The Xcode testing environment was straightforward to set up, update the tests and specify targets. Because both application and scripts are native iOS, we were less likely to have any compatibility issues and were able to easily read the test results and give UI tests directions to follow. Furthermore, we had a lot of flexibility in configuring the tests and controlling the flow of UI Tests, like keeping track of XCTAssert failures and letting tests continue even after they do not pass some of the XCTAsserts.

As a part of automated environment setup, we've also created a separate "setup script" that testers run before starting the testing process to make sure that the correct application is installed and launched on their testing device [6](Khamaru, 2020) as displayed on a chart below.



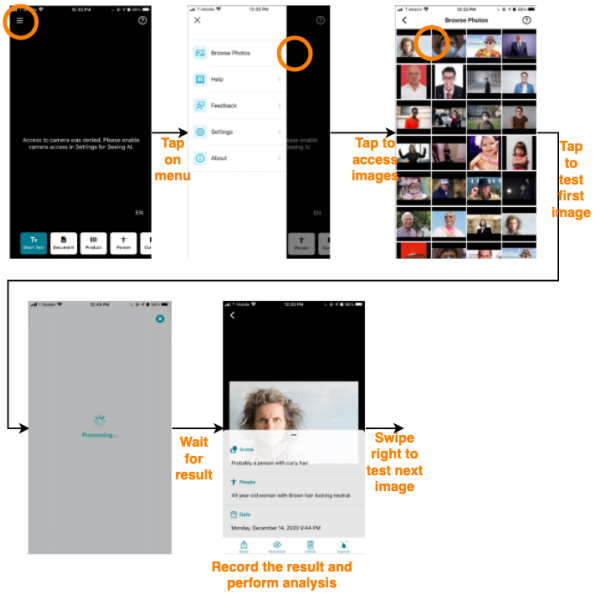
- **Testing Script Design and Scenarios**

The goal of the test automation process was to find the best way to optimize the time spent testing, recording results and bug analyzing during the manual AI testing process. Hence, our automation testing strategy was to introduce automation to the following actions in the testing process:

- Performing testing on the selected test cases
- Recording the testing results (the output of each AI function)
- Detect the bugs during texting using XCAssert statements
- Test result analysis generation
- Bug analysis generation

For each of the AI functions being tested, we have designed a test case model using a 3D classification table and came up with corresponding test case images. We repeated the testing process that was conducted manually before with additional help of test automation tools throughout multiple steps and compared the results. The automated testing scenario is shown in a Diagram and is as following the steps :

1. First, the testing automation script launches the Seeing AI application. It taps on the menu icon and after the new view opens, it taps on the “Browse Photos” section to open the user's photo library. After the photo library screen appears, the script taps on the last top left photo to start the testing process.
2. When the user opens the photo, the script waits for the algorithm to produce the output results. After receiving the results, script filters output data based on the AI function being currently tested and analyzed by comparing with expected results.
3. This process is then being repeated for the next image accessed by script swiping to the right to access the next photo.

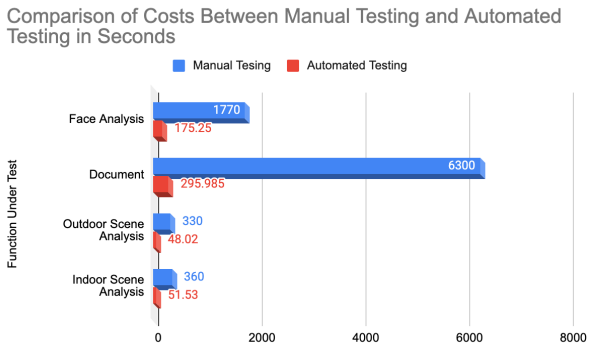


After all the test iterations were complete, a bug report was generated, which includes

information about the number of bugs detected, which type of failures occurred and which test cases had failed for each type of bug category.

- **Automation vs Manual AI Testing Cost**

Automating the testing, result recording, result verification and bug analysis processes helped drastically reduce the overall testing time in comparison with manual testing. Testing time was reduced by 90% on average for all AI functions. As shown on a diagram below, the difference is most drastic for Document AI function, as recording and verifying short texts take a lot more time if done manually. When we applied an automation script to record and compare the text results of the Document algorithm, it reduced the time from 6300 sec to 295.9 sec - 96% reduction in time cost!

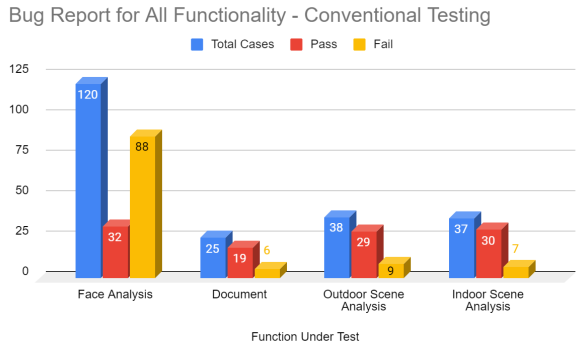


V. Bug Analysis Comparison for Conventional Testing, AI Testing, Automated Testing

- **Conventional Testing**

The bug analysis for conventional testing consists of the four functionality our team tested: face detection, document analysis, outdoor scene analysis, and indoor scene analysis. The face detection had a total of 120 test cases and only 32 passed the conventional test method test and 88 test cases failed. The document analysis had a total of 25 test cases with only 19 test cases passing and 6 failing. The outdoor scene analysis functionality had a total of 38 test cases and 29 test cases passed and 9 failed. The indoor scene analysis had 37 total test cases with only 30 test cases passing and 7 test cases failing.

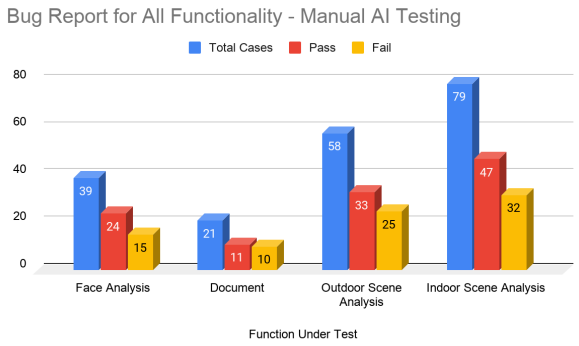
The highest bugs came from face analysis which is because of the higher test cases that were generated. The second highest is the indoor scene analysis, then the outdoor scene analysis, and lastly, the document analysis. The conventional testing bug analysis is shown below.



- **AI Testing**

The bug analysis for AI testing consists of the four functionality our team tested: face detection, document analysis, outdoor scene analysis, and indoor scene analysis. The face detection had a total of 39 test cases and only 24 passed the AI test method test and 15 test cases failed. The document analysis had a total of 21 test cases with only 11 test cases passing and 10 failing. The outdoor scene analysis functionality had a total of 58 test cases and 33 test cases passed and 25 failed. The indoor scene analysis had 79 total test cases with only 47 test cases passing and 32 test cases failing.

The highest bugs came from indoor scene analysis which is because of the higher test cases that were generated. The second highest is the outdoor scene analysis, then the face analysis, and lastly, the document analysis.The AI testing bug analysis is shown below and can also be seen in Section 2.

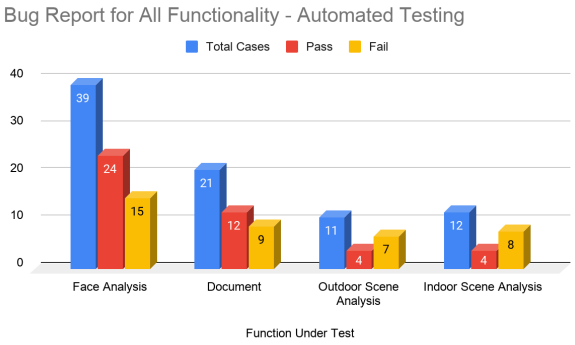


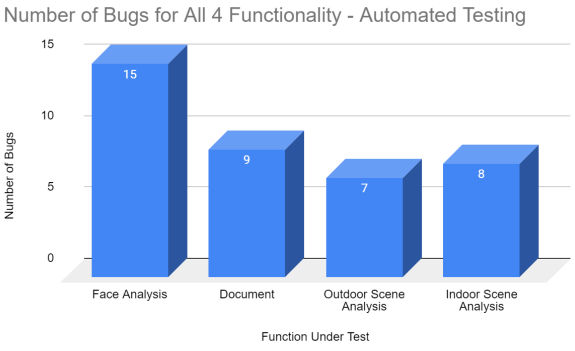
- **Automated Testing**

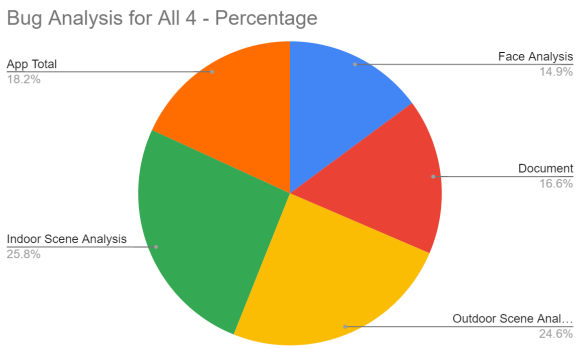
The bug analysis for automated testing consists of the four functionality our team tested: face analysis, document analysis, outdoor scene analysis, and indoor scene analysis. The face detection had a total of 39 test cases and only 24 passed the automated test method test and 15 test cases failed. The document analysis had a total of 21 test cases with only 12 test cases passing and 9 failing. The outdoor scene analysis functionality had a total of 11 test cases and 4 test cases passed and 7 failed. The indoor scene analysis had 12 total test cases with only 4 test cases passing and 8 test cases failing.

The most challenging part in applying automation to processes that were done manually before is results verification and bug classification. Results and bug count seem to stay unchanged only for the Face Analysis function, because the algorithm output for it is relatively easy to analyze and verify. However, in case of Indoor and Outdoor Scene Analysis, a lot of bugs were missed by automated verification scripts due to the challenges of being able to define what constitutes the correct answer. Unlike Face Analysis function, where function output can be easily distinguished by classified category keywords (age number, hair color, etc), with Scene function there is no way to evaluate the correctness of the result and compare with the expected values in a script without applying Natural Language Processing technologies. These factors have led to the number of bugs detected in automation testing to be lower for Scene AI functions.

The highest bugs came from face analysis which is because of the higher test cases that were generated. The second highest is the document analysis, then the indoor scene analysis, and lastly, the outdoor scene analysis. The automated testing bug analysis is shown below.







VI. Conclusions

A. Conventional testing and AI testing

Conventional testing as a testing methodology mostly used for applications or software before the AI era, can't continue fulfilling the mission to test more and more AI applications/software nowadays. The need to develop a new AI testing methodology is urgent. The work in this paper shows that AI testing has better test complexity and provides new opportunities for finding more bugs. Hence it is a better option for AI software and application testing.

B. Important Bugs uncovered for AI functions

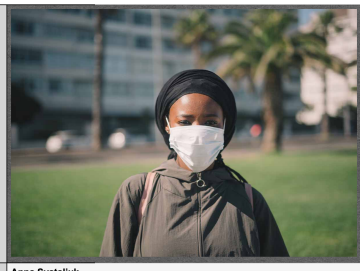
Example #1: Some cases from Outdoor Scene testing show that the app can't tell if a vehicle is driving or parked. It will confuse people who are vision impaired when they are walking outdoors, which could be a serious safety issue for them (see Figure (c)(d)).

(c) Output from app: "a group of cars on the road"

(d) Output from app: "a group of cars parked in a parking lot in front of buildings"

Figure(c)(d). Outdoor Scene Case: the app can't tell the state of vehicles correctly - a safety issue

Example #2: A consistent issue with Face Analysis function was that it was not able to recognize people who were wearing a facial mask. This is an important issue as a large number of people are wearing face masks in COVID-19 pandemic, and the algorithm is not able to assist visually impared people by recognizing people wearing masks around them.

Test case input:				
Performed by:	Anna Systaliuk			
Execution date:	11/03/2020			
Output categories	Face detection	Face classification		
		Age	Hair color	Face expression
Expected result:	TRUE	18-30	Hat	Neutral
Actual result:	FALSE FAIL	—	—	—

Example #3: Several Indoor Scene Analysis test cases that failed can be seen below. The mobile application could not differentiate children at certain photo angles. The application's actual output had detected the children as people which could result in confusing the users when they are in an indoor setting.

C. Challenges in testing Seeing AI

Throughout the testing process, we have encountered a significant amount of important bugs that could potentially be dangerous for a visually impaired person using Seeing AI. This issue indicates the importance of intelligent software testing that could help uncover such areas for refinements of intelligent algorithms. As we have shown in this paper, manual testing is very costly in terms of labor and time and hence, it cannot be relied on to test such complex intelligent functions.

When we attempted to apply automation to the AI Testing process, we have struggled to be able to find automation testing tools that could perform the result verification and bug classification tasks as effectively as manual labor. Even though we were able to significantly cut down time through automation, it also resulted in a number of bugs being missed and hence, compromising the testing effectiveness. Modern software is increasingly relying on AI algorithms, pushing the need for intelligent automation testing technologies to support the quality assurance maintenance and refinement of such systems.

References

[1]C. Tao, J. Gao and T. Wang, "Testing and Quality Validation for AI Software–Perspectives, Issues, and Practices," in IEEE Access, vol. 7, pp. 120164-120175, 2019 | Accessed on Nov 12, 2020

[2] J. Gao, C. Tao, D. Jie and S. Lu, "Invited Paper: What is AI Software Testing? and Why," 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE), San Francisco East Bay, CA, USA, 2019, pp. 27-2709, doi: 10.1109/SOSE.2019.00015. [Accessed on Nov 14, 2020]

[3]J. Gao, "AI Testing - Testing Quality Assessment", presented to Class, San Jose State University, San Jose, CA, USA, June 4th, 2018. [PowerPointslides]. Available: https://sjsu.instructure.com/courses/1371329/files/60033802?module_item_id=11272257, Accessed on: Dec 15th, 2020.

[4]J.Gao, "AI Testing", presented to Class, San Jose State University, San Jose, CA, USA, June 4th, 2018. [PowerPointslides]. Available: https://sjsu.instructure.com/courses/1371329/files/60033704?module_item_id=11272259, Accessed on: Dec 15th, 2020.

[5]S. Mischinger, "Appium vs. XCUITest for Automated iOS Testing," Bitbar, 18-Nov-2019. [Online]. Available: https://bitbar.com/blog/appium-vs-xcuitest-for-automated-ios-testing/. [Accessed: 05-Dec-2020].

[6] S. Khamaru, "iOS: XCUITest-How to Automate Tests for Apps without having access to Dev Source Code?," Medium, 23-Aug-2020. [Online]. Available: https://super-tester.medium.com/ios-how-to-automate-tests-for-apps-without-having-access-to-development-source-code-a35824abd5e3. [Accessed: 12-Dec-2020].