SE 3XA3: Test Report
OpenCameraRefined

Team #211, CAMERACREW
Faisal Jaffer, jaffem1
Dominik Buszowiecki, buszowid
Pedram Yazdinia, yazdinip
Zayed Sheet, sheetz

April 7, 2020

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
4/2/2020	1.0	Creation of Test Report
4/6/2020	1.1	Completion of Test Report

1 Functional Requirements Evaluation

1.1 Input through gestures

1. Test Name: test-smile-capture

   Results: The application successfully displays a green box around the user's face in the preview and captures the image when the user smiles in the view of the camera.

2. Test Name: test-face-detect

   Results: The app successfully displays a green box around the user's face in the preview when the user's face is in the view of the camera.

3. Test Name: test-thumb-filter

   Results: The user was successfully able to make a filter appear when presenting a "thumbs up" in the view of the camera

1.2 Live Filter

1. Test Name: test-cycle-filters

   Results: The user was successfully able to switch to the next filter when presenting a "thumbs up" in the view of the camera.

2. Test Name: test-button-filter

   Results: The user was successfully able to switch to the next filter by pressing the "filter" button.

1.3 Modify Captured image

1. Test Name: test-save-filter

   Results: The user was successfully able to verify the saved image has a filter.

2. Test Name: test-modify-filter

   Results: The user was successfully able to verify the saved image with a filter offers them a tool to modify the image in the gallery.

1.4 Simple Capture

1. Test Name: test-save-picture

   Results: The user was successfully able to capture an image by pressing the "filter" button.

2. Test Name: test-switch-camera

   Results: The user was successfully able to switch the camera view from the back facing camera to the front facing, and from the front facing camera to the back facing camera while retaining the filter.

2 Nonfunctional Requirements Evaluation

2.1 Usability

1. Test Name: test-ease-of-use

   Results: Users with an android phone were able to successfully capture photos and take video with filters, without filters, and using the smile to capture feature with no difficulty and without any additional information aside from being told "smiling will automatically capture a photo, and a thumbs up will switch between filters"

2. Test Name: test-ease-of-learning

   Results: Users with an android phone required the additional learning assistance of being told that "smiling will automatically capture a photo, and a thumbs up will switch between filters." Despite this, users claimed this is not difficult to learn and rated the difficulty an average of 1 out of 10.

3. Test Name: test-look-and-feel

   Results: 80% of users concluded that the application has that common "look and feel" that a typical camera application has, with one user claiming that the side bar is a-typical of a camera application. We concluded 4/5 was a satisfactory percentage.

2.2 Performance

1. Test Name: test-accuracy

   Results: The classifier was able to successfully detect images with 96.7% accuracy (training accuracy). To test the classifier on images not in the training data (real world samples), a data set of 20 images were created (validation set). After testing on the validation set, a validation accuracy of 95.4% was achieved. With the validation and training accuracy being over 95%, we concluded that the ML model performed well.

2. Test Name: test-speed

   Results: The inception model was able to accurately make classifications in an average time of 98ms, with 168ms being the maximum time it took to classify an image. This ultimately beats the maximum classification time that was previously set in the SRS document and test plan.

3. Test Name: test-garbage

   Results: When the application camera was presented with random objects in the view, it did not make false detections for a smiling face or a thumbs up. In addition, objects that were similar to the physical features of the trained classes, it was able ignore such objects 96% of the time.

2.3 Operational and Environment

1. test-different-devices

Results: The testing individual was able to confirm that the application successfully ran on all the several different android devices outlined in the test plan.

2.4 Maintainability

1. test-newer-versions

   Results: The testing individual was able to confirm that the application successfully ran on all the newer versions of android outlined in the test plan, demonstrating that the application is forwards compatible.

3 Comparison to Existing Implementation

In comparison to the existing implementation of this application, the refined application is more accessible because it allows the user to perform actions they previously were not able to. The primary difference between Open Camera Refined and the original Open Camera application is the smile to capture feature and the ability to use filters. Previously, these features were inaccessible to users, however with the new application users can easily capture images and apply filters without even needing to physically touch the phone themselves. Overall, the new application is a well rounded solution to Android consumer's photo and video capturing needs, as it still maintains a clean, easy to use UI while providing all the features that consumers may need.

4 Unit Testing

The objective of Unit Testing is to segregate individual functions and test their behaviour in isolation. Because of the nature of the features we had developed, some functions became difficult to unit test, as the ML model becomes a Black Box once it is trained.

In our case, some of the functions that were directly dependant on the output from the ML model were only tested for exceptions. Functions that performed intermediate calculations were the main focus of unit testing for our application and were thoroughly tested for correctness. Other functions were tested for base cases and exceptions, and then were more thoroughly

tested using other testing techniques such as integration testing and system testing. All unit tests were performed using JUnit and each function had a minimum of 4 test cases to test for regular and edge cases. In terms of the basic functionalities of the original camera application, the original application already had test cases that thoroughly covered these and we make sure to include this in our automated testing process.

5 Changes Due to Testing

5.1 Input Through Gestures Evaluation

Due to the test "test-thumb-filter" we noticed the classifier was positively recognizing hands in general as a thumbs up. This is because the input we used to train our model originally did not have many hand inputs unless they were a thumbs up. In order to fix this we had to re-train the machine learning model with more pictures of hands (that aren't a thumbs up) as input.

5.2 Live Filter Evaluation

Due to the test "test-cycle-filters", we discovered that in the camera api, the camera preview would be displayed upside down if the front facing camera is selected. This meant that the the filter's bitmap would display upside down when a filter was on. We previously did not encounter this issue because we were mostly testing through the rear facing camera. The fix for this issue was to conditionally rotate the filtered camera preview in the ImgFilterController module if the front facing camera is being used.

5.3 Modify Captured Image Evaluation

There have been no changes made due to testing in this category.

5.4 Simple Capture Evaluation

There have been no changes made due to testing in this category.

5.5 Usability Evaluation

There have been no changes made due to testing in this category.

5.6 Performance Evaluation

Originally our results for the test "test-speed" was an average time of 214ms. Because this did not meet our maximum classification time we found the most efficient way to reduce this speed was to lower the size of the input image going in to reduce the latency of the classification. Ultimately, through cropping the image and changing the resolution we managed to reduce inputs to 264x264 pixels.

5.7 Operational Environment Evaluation

There have been no changes made due to testing in this category.

5.8 Maintainability Evaluation

There have been no changes made due to testing in this category.

6 Automated Testing

Due to the nature of this project, we found that it was unfeasible to rely heavily on automated testing to ensure our program works correctly. This is because our application relies heavily on machine learning models, which do not provide objective outputs (for example, confidence levels) that can be can be tested automatically. In addition, our application interacts with our user outside of the environment in which it is programmed, meaning the best way to validate our application is to test it visually on our actual mobile devices and its hardware.

However, automated testing still had a part to play in the development of this project. Because our application required some calculations, we created test cases that ensure that the functions providing these calculations were correct. In addition to this, the original open camera application included an extensive number of unit tests. With the addition of every new feature we

ran these unit tests, including the unit tests we added to ensure the addition of this new feature did not affect the existing properties of the application.

7 Trace to Requirements

This section shows a traceability matrix between the test cases and the requirements that these test cases apply to.

FR Trace	Test ID
REQ1. REQ2. REQ12.	test-smile-capture
REQ5.	test-thumb-filter
REQ5.	test-face-detect
REQ7.	test-cycle-filters
REQ6.	test-button-filter
REQ8. REQ9. REQ10.	test-save-filter
REQ4.	test-modify-picture
REQ11. REQ3. REQ13.	test-save-picture
REQ10.	test-switch-camera
EU	test-ease-of-use
ARL	test-ease-of-learning
LR	test-look-and-feel
SL. PA.	test-speed
RR	test-garbage
EP, PRE	test-different-devices
ARM, ARS, MR	test-newer-versions

8 Trace to Modules

This section shows a traceability matrix between the test cases and the modules that these test cases apply to.

Test Case	Module
test-smile-capture	Classifier, GestureController
test-face-detect	Classifier, ClassifierConstants, Recognition
test-thumb-filter	Classifier, GestureController, Filter, Constants
test-cycle-filters	Classifier, GestureController, Filter, Constants
test-button-filter	Classifier, GestureController, Filter, Constants
test-save-filter	Classifier, GestureController, Filter
test-modify-picture	GestureController
test-save-picture	GestureController
test-switch-camera	GestureController
test-ease-of-use	Classifier, Classifier Constants, Recognition
test-ease-of-learning	Classifier, ClassifierConstants, Recognition
test-look-and-feel	Classifier, Classifier Constants, Recognition
test-reliability	Classifier, ClassifierConstants, Recognition
test-speed	Classifier, Recognition
test-garbage	Classifier, ClassifierConstants, Recognition
test-different-devices	GestureController
test-newer-versions	GestureController

9 Code Coverage Metrics

Through unit testing, we managed to achieve 100% code coverage. However, since our unit tests don't ensure complete correctness of our application since many outputs cannot be objectively tested, we realized this did not accurately depict code coverage.

Therefore, in order to accurately ensure that we covered a large percentage of our code, when doing our test cases we added debug logs throughout our code and checked which ones were displayed in the logcat. To do this, we used the command log.d("TAG","Message") (where d stands for debug and TAG is the name of the message) in sequential and conditional pieces of code. We then used the logcat search tool to search the messages and see how many were logged. Overall, 134/136 of the messages were logged resulting in a code coverage of 98.5%.

In addition to the high code coverage, the test cases tracebility matrices show that each module has been thoroughly tested through multiple test

cases. Ultimately we can conclude that our tests sufficiently covered most of our code.