SE 3XA3: Test Plan
OpenCameraRefined

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

April 6, 2020

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
February 29, 2020	1.0	Initial Document
April 6, 2020	2.0	Fixed feedback and added new test cases

1 General Information

1.1 Purpose

The purpose of this document is to describe the process and techniques that will be used to verify the correctness of the system.

1.2 Scope

The scope of the document is to describe the testing techniques that will be used to verify the additional features that will be implemented specifically. This includes ~~performing unit testing, integrated testing and automated testing~~ performing unit tests, integrated tests and automated tests on any additional functions that have been implemented as well as on the system as a whole.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: ~~Table of Abbreviations~~

~~Abbreviation~~	~~Definition~~
~~Abbreviation1~~	~~Definition1~~
~~Abbreviation2~~	~~Definition2~~

Table 3: Table of Abbreviations

Abbreviation	Definition
ML	Machine Learning
TFLite	Tensorflow Lite

1.4 Overview of Document

~~Our team~~ The development team will be developing two additional features to the open source camera app called OpenCamera. The first feature will allow the user to take a photo through the use a gesture. The second feature

Table 4: **Table of Definitions**

Term	**Definition**
OpenCamera	A open source camera application for Android
OpenCameraRefined	Our teams version of the OpenCamera application
Gesture	A movement of a body part used to express something
Filter	The process of changing the appearance of an image by manipulating its colours and shades.
Machine Learning	The process of using computer software to find a relationship between a input and a output typically based on provided input output pairs.
TensorFlow	A open source machine learning library
Real Time	A system is real time if it responds to input immediately as it happens (in other words live)

allows the user to apply a photo filter either before or after the photo has been taken. This document will outline the tests to be performed for both of these features.

2 Plan

2.1 Software Description

This software is designed to change the way we capture pictures through our phones. Using image processing algorithms and machine learning, the team is focused on enabling the software to process and recognize a certain gesture in real time. In addition, the software is designed to offer the user with real time filters, mainly using existing packages such as OpenCV. Some initial gestures recognized by the software can include Smile, Thumbs up or Wave. Future gestures can be added by retraining the Object Classification algorithm through machine learning frameworks such as TensorFlow. The purpose is to gain accessibility and functionality through the use of "hands-free" picture capture and real time filters. The Software is developed using Android Studio and written prominently in Java. The user is also given the opportunity to change the configuration of the automatic image capture. For example, the user can mandate the application to only take a picture when

a defined gesture is performed by the user for a certain period of time.

2.2 Test Team

The following team members are responsible for scheduling, planning and executing different test phases, covering the entire code:

1. Zayed Sheet
2. Pedram Yazdinia

2.3 Automated Testing Approach

The Testing process for each module is initiated by extracting the requirements that are affected by the test cases. These requirements should be recorded and verified. The team is then responsible to determine the testing level. The traditional testing levels include, Unit Testing, Integration Testing and System Testing. Developers group the tests into a certain level by looking at where they are added in the software development process or by checking how specific they are in the context of the entire implementation. Once the testing level is determined, the developers brainstorm on different testing methods based on the scope of the testing. The testing methods mainly include White-box, Black-box or Grey-box testing which is a combination of the other two. In the execution stage, the team first compares different testing tools available, and selects the ones that are most compatible with our level and method of testing. Finally, the tests run, and the defects are one by one reviewed. In the final stages, it is crucial that the team updates the Requirements Traceability Matrix that shows the completeness of our validation through Software Testing. Finally, the developers are responsible to revise and review the effected requirements if an error has been fund. The implementation will change accordingly to the test cases results and the new requirements.

2.4 Testing Tools

Open Camera is mainly developed using Android Studio and TensorFlow. Android Studio is the official IDE for android development which includes a comprehensive set of features used for unit testing, code coverage and debugging. As a result, the test team will mainly use Android Studio for

testing purposes as well. Other tools such as Epxeritest and Robotinum were considered which offer a more advanced way to test the code, however for time consuming purposes we have decided to use Android Studio.

2.5 Testing Schedule

See Gantt Chart at the following url Gantt-Proj

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 Input through gestures

Perform actions based on trained gestures

1. test-smile-capture

   Type: ~~Manual~~Manual, Static, Functional

   Initial State: ML model and label files in correct directory. Camera view is initialized. Preview width and height is known. Inception model is initialized.

   Input: Face in the view of the selected camera. (Image is the input)

   Output: Green bounding box around the face in the preview. Following a successful detection, a picture is captured after 2 seconds.

   How test will be performed: A tester will hold the Android device running our version of the OpenCamera application and verify that a green bounding box is displayed around the face on the preview and a picture is captured 2 seconds after the user smiles in the view of the camera.

2. test-face-detect

   Type: ~~Manual~~Manual, Static, Functional

   Initial State: ML model and label files in correct directory. Camera view is initialized. Preview width and height is known. Inception model is initialized.

Input: Face in the view of the selected camera. (Image is the input)

Output: Green bounding box around the face in the preview.

How test will be performed: A tester will hold the Android device running our version of the OpenCamera application and verify that a green bounding box is displayed around the face on the preview.

3. test-thumb-filter

   Type: ~~Manual~~Manual, Static, Functional

   Initial State: ML model and label files in correct directory. Camera view is initialized. Preview width and height is known. Inception model is initialized. OpenCV is initialized.

   Input: "Thumbs Up" in the view of the selected camera. (Image is the input)

   Output: Orange bounding box around the detected thumb following a filter displayed on the preview.

   How test will be performed: A tester will hold the Android device , at a distance of 1 meter, running our version of the OpenCamera application and verify that they are able to use the "live filter" when they show "thumbs up".

3.1.2 Live Filter

1. ~~test-cycle-through-filters~~test-cycle-filters

   Type: ~~Manual~~Manual, Static, Functional

   Initial State: ML model and label files in correct directory. Camera view is initialized. Preview width and height is known. Inception model is initialized. OpenCV is initialized.

   Input: "Thumbs Up" in the view of the selected camera. (Image is the input)

   Output: Every time a new "thumbs up" is detected, the system displays a new filter (cycles through the 4 filters available.)

   How test will be performed: A tester will hold the Android device running our version of the OpenCamera application and verify that

they are able to scroll through the "live filters" every time they show a "thumbs up".

2. test-button-filter

   Type: Manual, Static, Functional

   Initial State: User has the Open Camera app open

   Input: User touches the filter button.

   Output: Filters are cycled through in the preview.

   How test will be performed: A tester will hold the Android device running our version of the OpenCamera application and verify that upon touching the filter button, the user can toggle through all filters.

3.1.3 Modify Captured Image

1. test-save-filter

   Type: Manual, Static, Functional

   Initial State: User has set the filter on preview.

   Input: User smiles or touches the capture button)

   Output: Image with the filter is saved

   How test will be performed: A tester will hold the Android device running our version of the OpenCamera application and verify that upon taking an image in filter mode, the saved image has filter.

2. test-modify-picture

   Type: Manual, Static, Functional

   Initial State: User has captured the image and opens teh gallery.

   Input: User chooses a modifying tool.

   Output: User is able to modify the captured image.

   How test will be performed: A tester will hold the Android device running our version of the OpenCamera application and verify that upon taking an image, the gallery offers them a tool to modify the image with.

3.1.4 Simple Capture

1. test-save-picture

   Type: Manual, Static, Functional

   Initial State: User has opened the OpenCamera application.

   Input: User touches capture button

   Output: Image with the filter is saved

   How test will be performed: A tester will hold the Android device running our version of the OpenCamera application and verify that upon touching the capture button, the application captures an image.

2. test-switch-camera

   Type: Manual, Static, Functional

   Initial State: User has opened the OpenCamera application.

   Input: User switched the cameras

   Output: Camera view switches

   How test will be performed: A tester will hold the Android device running our version of the OpenCamera application and verify that upon switching camera views, the filters remain on view.

3.2 Tests for Nonfunctional Requirements

3.2.1 Usability

1. test-ease-of-use
   Type: Manual, Static, Functional

   Initial State: User has opened the OpenCamera application.

   Input/Condition: User is told: "smiling will automatically capture a photo, and a thumbs up will switch between filters"

   Output/Result: The system is able to detect the gesture and perform relevant action.

How test will be performed: A user will be selected, who hasn't used this application before, and will be instructed with the following: "smiling will automatically capture a photo, and a thumbs up will switch between filters". Test will pass once the user can successfully perform these actions.

2. test-ease-of-learning
   Type: Manual, Static, Functional

   Initial State: User has opened the OpenCamera application.

   Input/Condition: User smiles or shows thumbs up.

   Output/Result: The system is able to detect the gesture and perform relevant action.

   How test will be performed: A user will be selected, who hasn't used this application before and will be tested for intuitiveness of the feature before giving instructions.

3.2.2 Performance ~~of gesture detection~~

1. test-reliability

   Type: ~~Manual~~Manual, Static, Functional

   Initial State: Application being operated in an poorly lit environment

   Input/Condition: A poorly lit gesture is found in the image.

   Output/Result: The system is able to detect the gesture and perform relevant action.

   How test will be performed: A tester will hold the Android device in a poorly lit room running our version of the OpenCamera application and verify that the application is able to perform the corresponding actions relevant to the gesture. In this case test-smile-capture, test-face-detect and test-thumb-filter should pass.

2. test-speed

   Type: ~~Dynamic~~Manual, Dynamic, Functional

Initial State: ML model and label files in correct directory. Camera view is initialized. Preview width and height is known. Inception model is initialized.

Input: Bitmap image of a gesture

Output: Detection of the gesture in the image under 200ms.

How test will be performed: After initializing the Inception model, we feed in an image with a gesture. Then measure the time it takes for the model to make a classification.

3. test-garbage

   Type: Manual, Static, Functional

   Initial State: User has opened the OpenCamera application.

   Input: User points camera at random objects

   Output: Only smiles and thumbs up are the only 2 classes detected

   How test will be performed: A tester will hold the Android device running our version of the OpenCamera application and verify that upon pointing the camera at different objects, only the trained objects are classified.

3.2.3 Operational and Environment

1. test-different-devices
   Type: Manual, Static, Functional

   Initial State: User has opened the OpenCamera application.

   Input/Condition: User performs the actions in the Functional Requirements.

   Output/Result: The user should be able to successfully use the features.

   How test will be performed: A tester will perform this test on devices with the following configurations: [5 inch phone running Android API 23, 5.5 inch phone running Android API 25 and 6 inch phone running Android API 27]. The tester shall confirm the successful use of all features.

3.2.4 Maintainability

1. test-newer-versions
   Type: Manual, Static, Functional

   Initial State: User has opened the OpenCamera application.

   Input/Condition: User performs the actions in the Functional Requirements.

   Output/Result: The user should be able to successfully use the features.

   How test will be performed: A tester will perform this test on devices with the following configurations: [5 inch phone running Android API 23, 5.5 inch phone running Android API 25 and 6 inch phone running Android API 27]. The tester shall confirm the successful use of all features.

3.3 Traceability Between Test Cases and Requirements

3.3.1 Functional Requirements

FR Trace	Test ID
~~BE1.1~~REQ1. REQ2. REQ12.	test-smile-capture
~~BE1.2~~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

3.3.2 Non-Functional Requirements

NFR Trace	Test ID
PR1	test-speed
PR2	test-speed
OE1	test-reliability

4 Tests for Proof of Concept

4.1 Area of Testing

~~Title for Test~~

1. ~~test-id1~~

   ~~Type: Functional, Dynamic, Manual, Static etc.~~

   ~~Initial State:~~

   ~~Input:~~

   ~~Output:~~

   ~~How test will be performed:~~

2. ~~test-id2~~

   ~~Type: Functional, Dynamic, Manual, Static etc.~~

   ~~Initial State:~~

   ~~Input:~~

   ~~Output:~~

   ~~How test will be performed:~~

   Proof of concept testing will primarily be focused on validating the detection of a smiling face and thumbs. These are the core requirements in our project, on which the other features are built on top. Therefore, majority of proof of concept testing will revolve around the validation of the detections in different situations.

5 ~~Comparison to Existing Implementation~~

6 Unit Testing Plan

6.1 Unit testing of internal functions

Internal functions that return, filter or parse data and have objective outputs which can be manually predicted or derived will be tested in the Java

environment using the JUnit 5 library. This will involve testing individual methods by providing them with an input or data to be used, and comparing the output with the expected output.

Internal functions that don't have objective outputs will be tested manually by making sure the output makes sense.

The inputs used to test these methods will either be proper data that is expected to be used in a real-life scenario, or improper/unexpected data that is meant to test the robustness of the program.

This project does not require any outside drivers, or stubs to be installed for the purposes of testing. All required drivers or stubs will already be available within the individual classes or project as a whole.

In terms of the metrics used to ensure adequate testing of internal functions, we will be using coverage metrics with a goal of 75% code coverage

6.2 Unit testing of output files

Output files will be manually tested by making sure they make sense. This means they will visually be looked over by software testers to be approved.

For the filter output pictures, we will have example photos of what it should look like and compare it to the output. For gesture based actions we will have a user perform the gesture in a natural manner and ensure the software captures the gesture.

Per filter and gesture, we will take 10 or more pictures, using at least three different users and ensuring at least one photo is taken in a dark background environment and one in a light background environment. In addition, these tests will take place with all ~~four~~ software test developers present to ensure the picture quality adheres to the standards of all ~~four~~ software testers. Each filter and gesture must have a 100% success rate, meaning all tests will have the approval of all ~~four~~ software testers.