# Code Testing Report

ARC - Artificial Recognition of Cannabis

ENSE 400/477 2022/23

Gregory Sveinbjornson

Feras Daghmoush

# Introduction

This code testing report will provide a high level overview of the testing plans and processes our group, ARC, completed during the eight months of our capstone project. Our test plans determined the approaches that were taken to test the different sections of our project. Test results were either viewed as passed, failed, or unreliable. Overall, this document aims to demonstrate the code testing performed throughout the course of this project.

# Test Plans

**Front End - Angular**

Our testing plans in Angular were in part created with our partners at BudSense. There were a few main things we wanted from our front end:

1. Two images could be uploaded
2. The images would fit inside the upload boxes and not move the page
3. The images could be sent to the API
4. The page could get a response from the API
5. The response could be displayed in a readable format
6. The animation in the product identifier would play at default, then be removed when the response was given

7. The upload button would play an animation as the page was waiting for a response
8. The user could only upload 1 image into each box
9. The reset button clears the page

**Back End**

Our testing plans for our back end code would require three main test sections:

1. A way to test our machine learning model
2. A way to test Tesseract-OCR on our images
3. A way to test that we could send images and get a response from our API

# Test Results

**Front End - Angular**

The types of tests performed fall under the categories of functionality (F) and user experience (UX). Status of the tests fall under Pass (P), Fail (F), or Unreliable (U).

| Type | Description | Steps Taken | Expected Result | Status Mar. 1 | Status Apr. 1 |
|---|---|---|---|---|---|
| F | Upload Image | Upload image functionality and drag and drop | Image will appear on the screen and be usable | P | P |
| UX | Images Fit in boxes | Images are uploaded | The screen will stay stable and images will be in boxes | U | P |
| F | Images sent to API | Upload images and click upload button | API will receive the images | P | P |
| F | Page gets response | Upload images, click upload button, wait for response | A response is displayed | P | P |
| UX | Response is readable | Check that response is formatted | Response info is readable | F | P |
| UX | Barcode Animation | Check that the animation plays on startup and disappears on response | Animation works as intended | U | P |
| UX | Upload Animation | Check that the animation plays while waiting for a response | Animation works as intended | P | P |
| F | 1 image per box | Check that a user cannot upload an image in a box that already has an image | The upload field does not allow a second image | P | P |
| F | Reset Button | Check that the page resets to its original state | The page is as it is on startup | P | P |

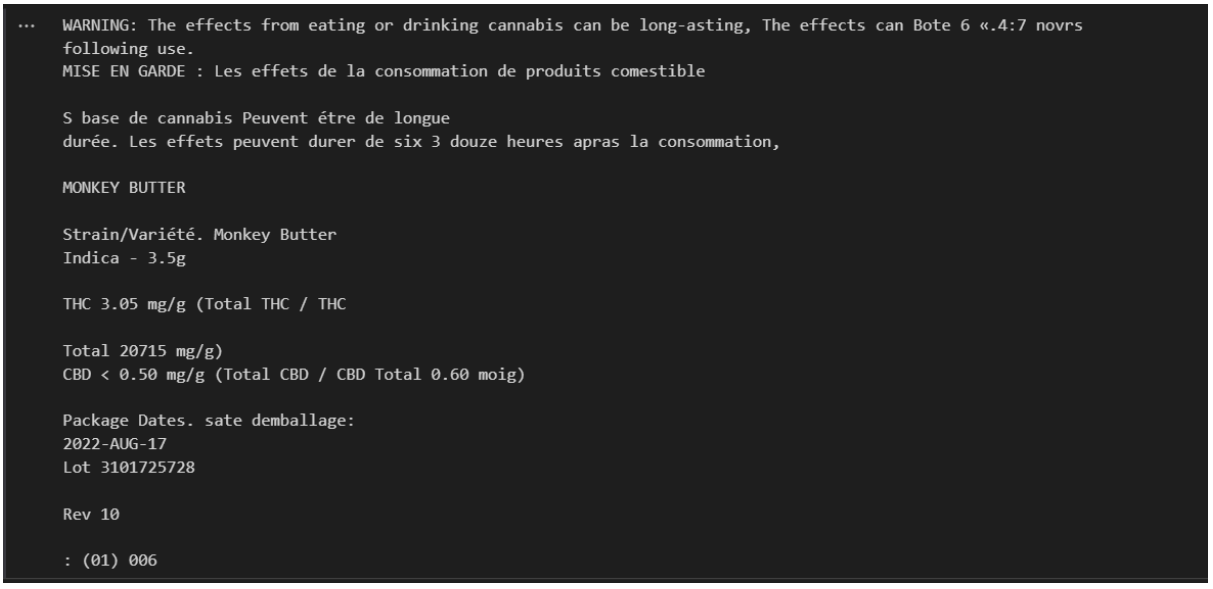
**Back End**

Our back end testing took place in 3 files:

1. modelTest.ipynb was used to test our model
2. tessTest.ipynb was used to test Tesseract-OCR on our images
3. apiTest.py was used to test the connection to our API

In modelTest.ipynb, we loaded in our machine learning model to test its accuracy and usability. We also tested how to best preprocess the images to get the fastest and most accurate results. This is where we also tested our product database and gathered all the relevant information from a single prediction by using the predicted value to search our database for the correct match. We also tested the confidence value to determine what value of confidence from the model was too low; ultimately we decided on 80%, as below 80% predictions were starting to be incorrect. We tested this by using pictures with an increasing level of difficulty to predict, such as being blurry or off centre, and seeing if our model could predict them and with what confidence value it would return with. All incorrect predictions had a confidence value of less than 80%, which is how we chose the value. In the figure below is what a returned value looked like:


In tessTest.ipynb, we tested Tesseract-OCR on our images to see if we could get an accurate response. In our tests, we discovered that while .jpg images returned gibberish, .png images would return a string that looked like the image below:



We still have no idea why this occurred, and made sure that before any image was sent to Tesseract-OCR, it was converted to a .png file.

Once a string was returned from an image, we had to look through the text for the relevant values. To do this, we used a regex expression to look for the total THC and CBD values. Once gathered from the string, they were gathered and converted to a percentage. An example of a successful response is pictured below:

```
    print("THC total not found")

if matchCBD:
    cbd_total = round(float(matchCBD.group(1)) / 10, 2)

    if cbd_total > 50 or thc_total < 0:# Check twice as the decimal point may be missing
        cbd_total /= 10
    if cbd_total > 50 or thc_total < 0:
        cbd_total /= 10

    print("CBD:", cbd_total, "%")
else:
    print("CBD total not found")
```

```
[114]
...  THC: 20.715 %
     CBD: 0.06 %
```

Finally, in apiTest.py, we made a connection to our AWS EC2 instance to connect to and send sample images and receive a response. This way we could quickly test the functionality of our API while our user interface was still being developed. The code was simple and would display a json response from the API if successful. The code for apiTest.py is pictured below:

Project Files > Tests > apiTest.py > ...

```python
import requests

# Define the URL of the API endpoint
# Testing on localhost
url = "http://127.0.0.1:5000/predict"

# Read the front and back image files
front_image = open("gsf.JPG", "rb")
back_image = open("gsb.JPG", "rb")

# Define the multipart/form-data fields for the request
data = {'front_image': front_image, 'back_image': back_image}

# Send the images to the API for prediction
response = requests.post(url, files=data)

# Print the response from the API
print(response.text)

```

# Conclusion

The testing done throughout the course of our capstone project was vital in completing it. Our testing plans allowed us to quickly test code and functionalities, while allowing the other group member to be kept in the loop by just looking at the tests and their status. While our user interface is simple, it is also robust. Our API is reliable, and because of our testing we got a response every time.  We are happy with the quality standards that this project achieved.