Team Justice League

Image Quality Detection

Nadav Fedida - 220548228

Table of Contents

	Page
1 Introduction	**1**
2 Aim	**1**
3 Pain assessment pipeline	**2**
4 Setting up a working environment	**2**
5 Getting started with code	**3**
6 Unit testing	**4**
7 Coverage report	**4**
8 CSV output	**5**
9 Current issues	**6**
10 Future upgrades	**6**

1 Introduction

Previously to this feature addition the pipeline for pain assessment was directly sending the image each user is taking, and processing it regardless of what was in the image. All data sent to the back-end on the Google Cloud Platform (GCP) for processing was raw from the user and therefore could create a bottleneck point. This is the scenario of which the *image Quality* feature is trying to overcome.

2 Aim

The aim of the *image Quality* feature is to remove images which do not pass a series of tests. These tests are designed to sift out the images which fail the following:

- Focus ➢ Checking if an image is in focus and is over a preset threshold.
- Brightness ➢ Checking if an image brightness levels are acceptable to allow for best image processing.
- Face exists ➢ Ensuring a face is available the image.
- Face count ➢ Determining how many faces are in the image.

- Distance of face from camera ➢ Processing how far the user is from the camera, rejecting if too close or too far.
- Nose location ➢ Determining the nose 'X' and 'Y' coordinates in the image to check for relative position from the canvas centre.

All the tests are executed in such way that each test relies on the previous test to pass in order to operate. This ensures redundant testing is eliminated and allows for faster processing.

3 Pain assessment pipeline

As per the introduction, the existing model did not account for several of possible aspects which would render an image useless for pain assessment. Having all the raw data directly sent to be computed for pain is resource intensive and could create a backlog of images to process, increasing the processing times.

The existing model can be represented as such:

```
User takes image on mobile phone
  └─Image send to GCP for processing
      └─Face detection
          └─Face landmark detection
              └─Face Alignment
                  └─Pain detection
                      └─Publish output
```

From the above tree diagram, we can see that users were able to send images from their phones to be processed for pain detection, even if the images did not contain a face in them.

New Pain assessment pipeline:

```
User takes image on mobile phone
  └─New Image Quality Feature      ← New feature
      └─Image send to GCP for processing
          └─Face detection
              └─Face landmark detection
                  └─Face Alignment
                      └─Pain detection
                          └─Publish output
```

4 Setting up a working environment

First step is to ensure Git is installed on the local machine, this can be downloaded from [1]: https://git-scm.com/downloads

Navigate to a folder in the terminal and initialise it:

```
$ git init
```

In a web browser, navigate to: https://github.com/Gopher-Industries/Team-Justice-League and **fork** the repository to your own GitHub.

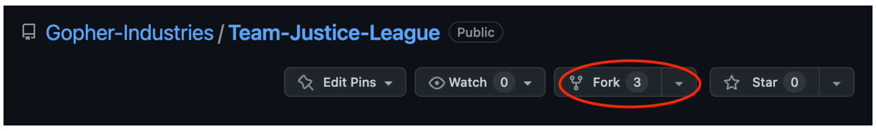


Figure 1: Gopher-Industries Github fork

[1]Guide will follow the process for Mac users. However, Windows installation will be very similar

Navigate to your own GitHub page and find the recently forked repository. Inside click on **Code** and copy the URL [2].

In the terminal, type the following to clone the forked repository to your local machine:

```
$ git clone URL
```

Once cloned, you can now create a working environment with Anaconda suitable for this project.

Download Anaconda , this can be done from:
https://www.anaconda.com/products/distribution.

Once downloaded and installed on a local machine, an environment can be created with the following command from the terminal (can be done from inside Anaconda directly)

```
$ conda create --name <env> --file requirements.txt
```

Example (will create an environment called "ImageQuality" using the requirements needed):

```
$ conda create --name ImageQuality --file requirements.txt
```

The requirements.txt file is in the clone repository:

```
Team-Justice-League
    └── RnD
          └── Face Quality
                └── Requirements
                      └── Requirements.txt
```

After the environment is created, activate it with the command:

```
$ conda activate <env>
```

5 Getting started with code

At the time of creating this document, the front-end application is not able to send an image to this environment and therefore this feature code is setup to read sample images that are saved locally and their paths stored in a file [3]:

/RnD/Face Quality/faces.csv

Save images into a selected folder (preferably not inside the git initialised directory, so when using source control, you do not push those images)

Change the paths in "faces.csv" file to suit your sample images.

Inside an Integrated Development Environment (IDE), navigate to the git initialised folder to view the code.

to run the program, inside the Setup.py file, adjust the path to suit your relative path. Then execute the Main.py file. This will

[2] More information about cloning a repository can be found on:
https://docs.github.com/en/repositories/creating-and-managing-repositories/cloning-a-repository

[3] Option to change the reading method to inspect a specific folder and read the images directly, although the code will need to change to handle this operation

6 Unit testing

In order to be sure the code is not broken after new changes and or new features are added, the unit tests which will provide feedback about the new whole program, showing if any parts have failed and if so, where.

To run the tests, ensure the correct environment is activated *'see section 4'*.

Navigate inside the IDE to the working directory of the code.

Run the following command to execute and view the simplified unit tests:

```
$ pytest
```

Intended results:

```
RnD/Face Quality/tests/test_FaceQualityUtils.py ........      [ 66%]
RnD/Face Quality/tests/test_faceDetector.py ........          [ 75%]
RnD/Face Quality/tests/test_setup.py ........                 [100%]
```

7 Coverage report

The coverage report which will show what parts of the code is tested, which line of code do not get tested and what percent of the program passes.

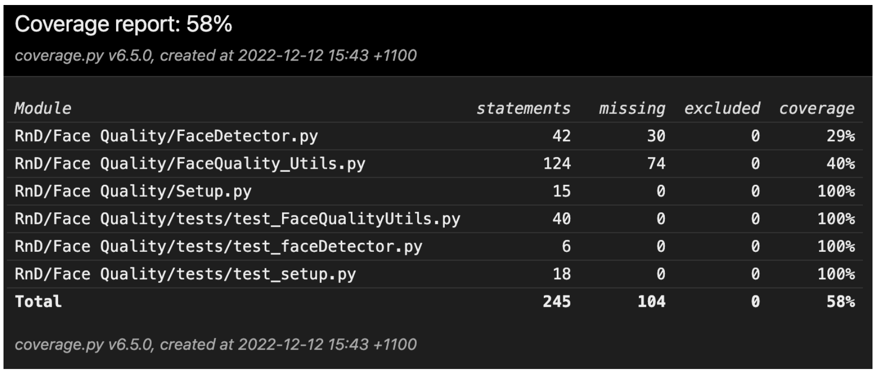
To run the coverage report, ensure the correct environment is activated *'see section 4'*.

```
$ pytest --cov . --cov-report=html
```

To view the results, find the folder **htmlcov** in the relative path, and view the **index.html** file.

```
Team-Justice-League
  └─RnD
      └─Face Quality
          └─htmlcov      ← new folder created for coverage test
              └─index.html     ← this file
```

Intended results:



Coverage report: 58%

coverage.py v6.5.0, created at 2022-12-12 15:43 +1100

Module	statements	missing	excluded	coverage
RnD/Face Quality/FaceDetector.py	42	30	0	29%
RnD/Face Quality/FaceQuality_Utils.py	124	74	0	40%
RnD/Face Quality/Setup.py	15	0	0	100%
RnD/Face Quality/tests/test_FaceQualityUtils.py	40	0	0	100%
RnD/Face Quality/tests/test_faceDetector.py	6	0	0	100%
RnD/Face Quality/tests/test_setup.py	18	0	0	100%
Total	**245**	**104**	**0**	**58%**

coverage.py v6.5.0, created at 2022-12-12 15:43 +1100

Figure 2: 58% testes at the time of creating this document

8 CSV output

As the tests progress throughout the program, each result gets saved in a JSON format and at the end converted to a CSV file. The intended use for these results is to allow pattern recognition and machine learning abilities in the future (*Added to section 10*).
CSV output

JSON format

```
{
   "Image Path": "Test1.png",
   "Brightness": {
      "Status": "Good",
      "Level": 133.28
   },
   "Focus": {
      "Status": "Sharp",
      "Level": 1032.37
   },
   "Face": {
      "Status": "Good",
      "Distance": 26.44,
      "Count": 1,
      "Confidence": 86.68,
      "Nose": {
         "X_loc": 1289,
         "Y_loc": 1611
      }
   },
   "PASS": true
}
```

CSV format

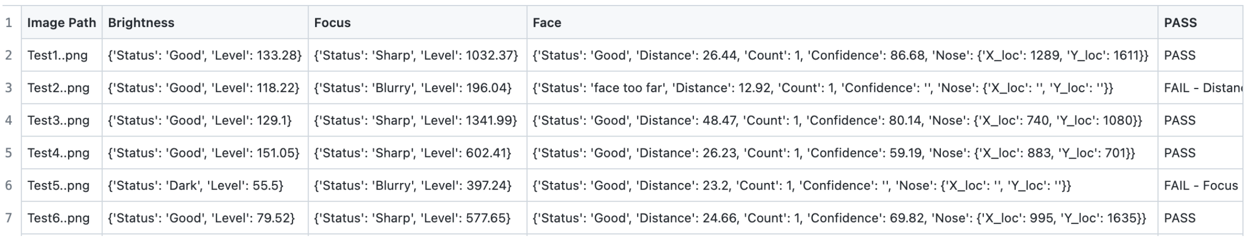


Image Path	Brightness	Focus	Face	PASS
Test1..png	{'Status': 'Good', 'Level': 133.28}	{'Status': 'Sharp', 'Level': 1032.37}	{'Status': 'Good', 'Distance': 26.44, 'Count': 1, 'Confidence': 86.68, 'Nose': {'X_loc': 1289, 'Y_loc': 1611}}	PASS
Test2..png	{'Status': 'Good', 'Level': 118.22}	{'Status': 'Blurry', 'Level': 196.04}	{'Status': 'face too far', 'Distance': 12.92, 'Count': 1, 'Confidence': '', 'Nose': {'X_loc': '', 'Y_loc': ''}}	FAIL - Distan
Test3..png	{'Status': 'Good', 'Level': 129.1}	{'Status': 'Sharp', 'Level': 1341.99}	{'Status': 'Good', 'Distance': 48.47, 'Count': 1, 'Confidence': 80.14, 'Nose': {'X_loc': 740, 'Y_loc': 1080}}	PASS
Test4..png	{'Status': 'Good', 'Level': 151.05}	{'Status': 'Sharp', 'Level': 602.41}	{'Status': 'Good', 'Distance': 26.23, 'Count': 1, 'Confidence': 59.19, 'Nose': {'X_loc': 883, 'Y_loc': 701}}	PASS
Test5..png	{'Status': 'Dark', 'Level': 55.5}	{'Status': 'Blurry', 'Level': 397.24}	{'Status': 'Good', 'Distance': 23.2, 'Count': 1, 'Confidence': '', 'Nose': {'X_loc': '', 'Y_loc': ''}}	FAIL - Focus
Test6..png	{'Status': 'Good', 'Level': 79.52}	{'Status': 'Sharp', 'Level': 577.65}	{'Status': 'Good', 'Distance': 24.66, 'Count': 1, 'Confidence': 69.82, 'Nose': {'X_loc': 995, 'Y_loc': 1635}}	PASS

Figure 3: Current CSV output [4]

[4]Image taken from Github page:
https://github.com/Gopher-Industries/Team-Justice-League/tree/main/RnD/Face%20Quality

9 Current issues

At the time of making this document, the following are the outstanding issues with the code which needs to addressed:

- **Face detection accuracy** - Even when checking the same image twice, the confidence level will vary and be lower than expected.
- **Face count** - When more than one face is included in the image, we are not able to extract only the larger face from the image, face count works and the code for determining the larger head is available, yet when printing the snippet of the larger head, only several pixels come up.
- **Nose X & Y accuracy** - Sometimes it marks the cross exactly on the tip of the nose, other times it marks it higher and off to a side.

10 Future upgrades

This feature at the time of making this document is completely working, yet still could have upgrades to complement it. These are some of the upgrades that should be added in the near future:

- **Add image to JSON** - This is the addition of the image to the JSON file so at the end of the test, the image *NParray* is sent to the back-end with all the additional data. This allows us to store all the data remotely off the users device and potentially perform pattern recognition with the stored data.
- **Encrypt the image and data** - Before sending the JSON file to the back-end, an industry standard encryption should be added to ensure user information is protected.
- **Return instructions to user if not close to image centre** - The nose *X & Y* coordinates need to be used to determine how far from the centre or how close to the image edge the user is, and from the relative position, calculate which direction to tell the user to move towards. This can be used to help guide the users to take better photos rather than guessing many times.
- **CSV output** - Fixing the CSV output to display all the JSON sub-objects in their own respective columns rather than having multiple sub-objects in one column.
- **Implement JSON exporting** - Sending the final JSON file to the GCP to connect the front and back ends together.
- **Unit tests** - Adding more unit tests to achieve 100% coverage in the HTML report (*see figure 2*)