Facial recognition algorithm

Author: Tuan Dung Lai

1. Introduction

Hi people, my name is Tuan and today I would like to share and capture the use of a fast, easy-to-use and portable facial recognition algorithm. This is the core algorithm that I have used through out my wonderful summer internship at the Applied Artificial Intelligent Institute (A2I2), Deakin University, Burwood Melbourne, Australia. This algorithm can recognize a person by their face in a relatively small amount of time, in our use case, i.e. virtual reception, the processing time to recognize one person is less than 1 seconds with 95% accuracy.

2. The face recognition

2.1. Portability

Our facial recognition can work independently inside a container. In our case, we used Docker as the tool to containerize the application. Docker is a tool designed to make deployment and software creation easier. A container allows a developer to package up an application with all necessary libraries and dependencies and finally ship them in one final package. After that, the application will run on any other machine regardless of any customized settings that machine has. Docker is somewhat similar to a virtual machine, however, rather than creating a while virtual operating system, docker allows application to use the same Linux kernel as the system that they are running on and this gives Docker a big advantage in term of performance and application size.

2.2. Overall process

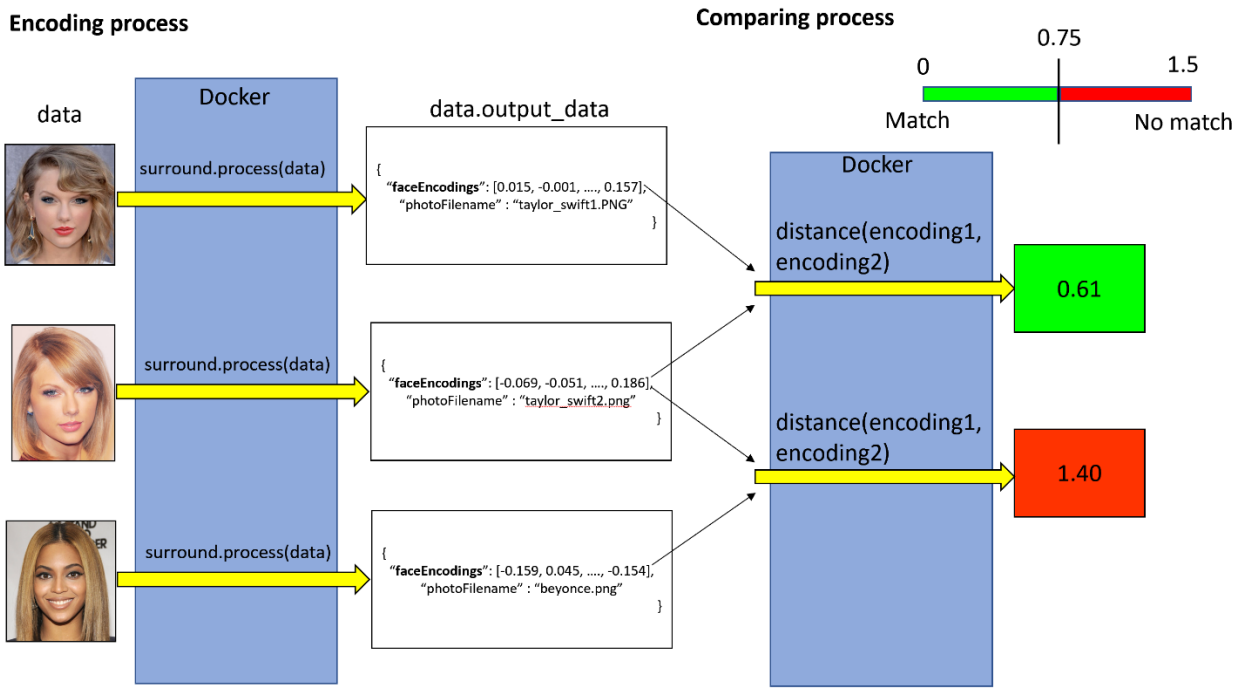

Figure 1: Overall structure and workflow

We used Flask (a python web framework) to host a local server can process HTTP POST request from users. This web server can process an individual image and return a face encoding that uniquely represent the image data as seen figure 1. The face encoding is bundled together with other attributes such as photo file name in a JSON format. Now each image has 1 unique face encodings, each encoding is an array of 128 float numbers ranging from -1 to 1.

In order to check the similarity of 2 faces to decide whether they are the same person or not. We compare its unique encodings, this is also can be done through the web server via a HTTP POST request. The web server takes the 2 JSON format string containing the face encodings and comparing them by calculate the Euclidean distance between 2 face encoding vectors. 2 Faces are considered the same if the distance between 2 encodings is less than 0.75. The less the distance the more similarity the 2 faces have. This threshold can be adjusted; however, high threshold is more prone to mismatched faces.

2.3. Building the pipeline

Our machine learning pipeline is served by Surround framework, an open-source light weight framework designed to be flexible and easy-to-use. After an image data initialization, we then used Surround framework to process the data process 8 stages as seen in figure 3, each stage has the responsibility to process the data, create output and prepare input data for the next stage. Surround framework allows us to control the work flow of the machine learning pipeline, if error occurs in any stage during the process, the stage will finish, and the output data will include the error description.

Here is some snippet showing how the pipeline is used:

Step 1: Initialize surround object with specified stages

```
surround = Surround([PhotoExtraction(), DownsampleImage(), RotateImage(), ImageTooDark(), DetectAndAlignFaces(), LargestFace(), FaceTooBlurry(), ExtractEncodingsResNet1()])
```

Step 2: Set up configuration for the pipeline

```
config = Config()

config.read_config_files(["/opt/theia/src/main/python/config.yml"])

surround.set_config(config)
```

Step 3: Initialize the stages, the initialize function of each stage will be executed before the operate method.

```
```

surround.init_stages()

```

**Step 4: Create input data**

```

data = PipelineData(image_path)

```

**Step 5: Process the data**

```

surround.process(data)

```

**Step 6: Access the output**

If error occurs during a stage, error description is accessed using `data.error`.

When there is no error, Json output is accessed using `data. output_data_dict`.

To access any warning if existed, use `data.warnings`

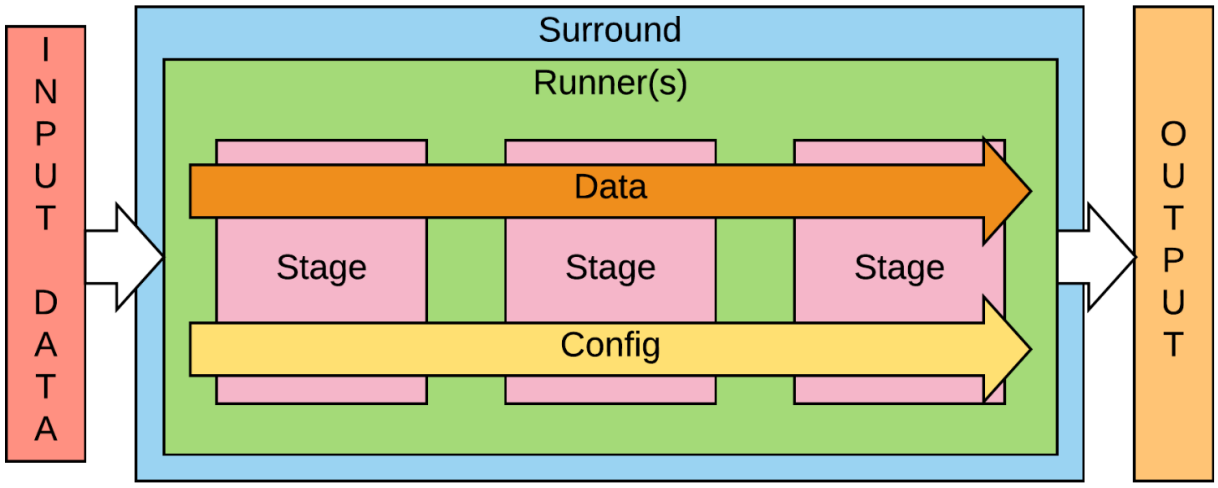


Figure 2: Surround framework workflow
```

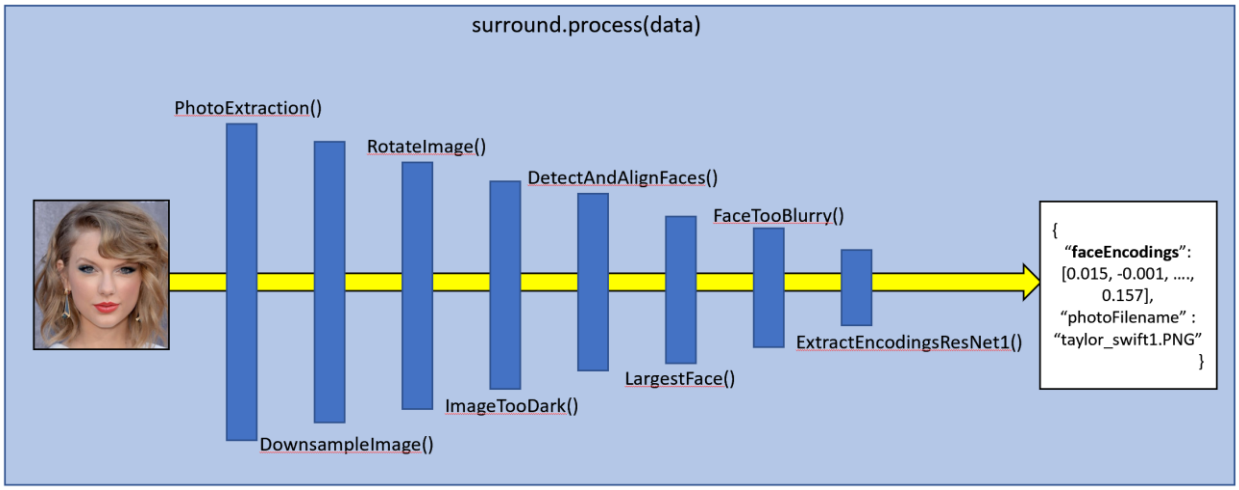

Figure 3: Facial recognition stages

2.4. Testing in local web server

A web page is created for quick testing and demonstration purposed as shown in figure 4. The website URL can be access after the docker is run. This website let users choose their own images in their local machine and get the corresponding encodings or errors.

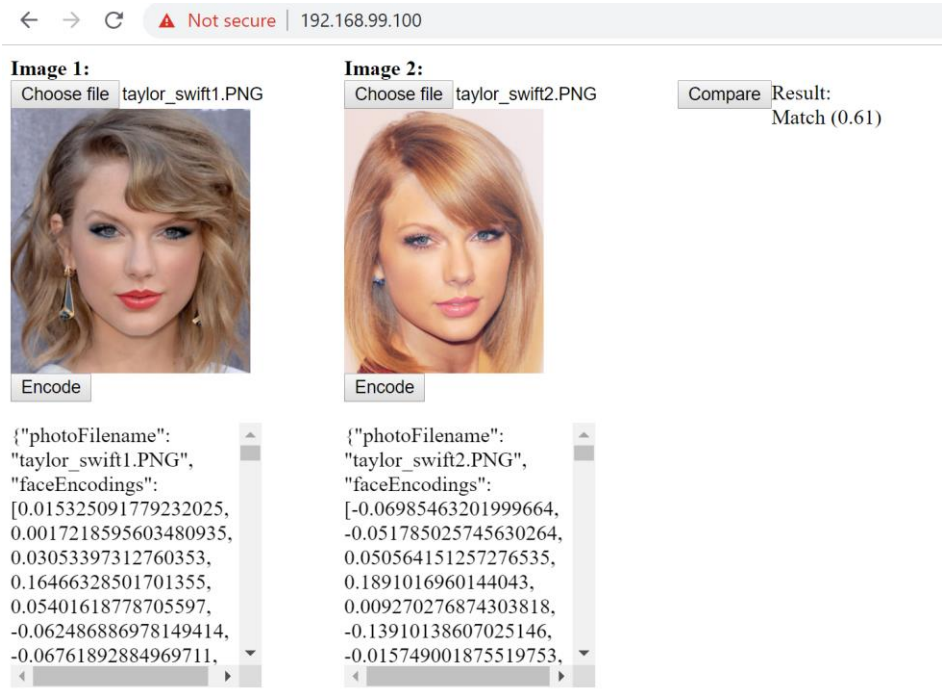

Figure 4: Web page for face comparison

3. Application in virtual receptionist

We have successfully integrating the facial recognition into a virtual receptionist application; the algorithm can recognize the person standing in front of the camera by send the image of their faces captured by the webcam and send it to the local server powered by Docker. The face is then encoded, and that unique encoding will be compared to all existing encodings we have in our database. If there are more than 1 face that match the face on the screen by having the encoding distance less than the threshold, we choose the one that has the smallest distance.

Figure 5: Virtual receptionist interface

4. **Limitation**

The recognition model is a deep learning model, this means the more quality input data, the more accurate and robust it will be. The recognition model we used was trained by face data taken from Hollywood celebrities as this is a large amount of data that is easily accessible. However, as white people dominating the Hollywood, our training data is mostly consisting of faces from the white race, these people have different facial feature from Asian and other races. For this reason, our current system works best when recognizing people with bright skin and tends to have bad results for people with darker skin tone like Asian and Africa.

Another drawback is that our system is currently not capable of detecting liveness, that means photo of other people can trick our system into believing that they are real.

These problems pose potential security threats and needs to be addressed or considered in case developers decide to use our algorithm in production.

5. **Future work**

The algorithm generates a unique encoding for each face and then it will be used to compare with other encodings in order to find the best match. This method can cause the accuracy decrease if the database has a massive number of different encodings which are ready to be

compared. In the future, we would want to test our current system with large database and examine the result.