KaiYu Chen

20100199

Assignment #3

Live Streaming Service - Facial Recognition Server

Course: Bachelor of Science (Honours) in Software Systems Practice



Continuous Assessment Report

Distributed Systems - 92021 - [2022-2023]

Table of Contents

[1. Introduction 1](#_Toc133808911)

[2. Requirements 1](#_Toc133808912)

[2.1 Functional requirements 1](#_Toc133808913)

[2.2 Non-functional requirements 1](#_Toc133808914)

[3. Environment 2](#_Toc133808915)

[4. System Architecture 2](#_Toc133808916)

[5. Directory Structure 2](#_Toc133808917)

[6. Technologies 3](#_Toc133808918)

[7. Installation 3](#_Toc133808919)

[7.1 Problems 4](#_Toc133808920)

[7.1.1 Could not find version that satisfies requirement cv2 OpenCV 4](#_Toc133808921)

[7.1.2 socketio 4](#_Toc133808922)

[7.1.3 Other errors 4](#_Toc133808923)

[8. Implementation 4](#_Toc133808924)

[8.1 Implementation 5](#_Toc133808925)

[8.1.1 Use separate threads for camera reading, model inference, and image display 6](#_Toc133808926)

[8.1.2 Skip some frames 7](#_Toc133808927)

[8.1.3 Compress the frames (to about 25% of the original size) before feeding them into the models 8](#_Toc133808928)

[8.2 Errors 10](#_Toc133808929)

[8.2.1 cv2.error 10](#_Toc133808930)

[8.2.2 socket.gethostbyname(hostname) error 10](#_Toc133808931)

[9. Conclusion 11](#_Toc133808932)

[10. Bibliography 11](#_Toc133808933)

[11. Appendix 12](#_Toc133808934)

Table of Figures

[Figure 1 System Architecture 2](#_Toc133808935)

[Figure 2 Folder tree of the project directory 2](#_Toc133808936)

[Figure 3 loads the face images 5](#_Toc133808937)

[Figure 4 computes the face encodings for these known images 5](#_Toc133808938)

[Figure 5 face\_distance\_to\_conf function 6](#_Toc133808939)

[Figure 6 create a camera object 6](#_Toc133808940)

[Figure 7 Use separate threads for camera reading, model inference, and image display 6](#_Toc133808941)

[Figure 8 Skip some frames 7](#_Toc133808942)

[Figure 9 process\_frame function 7](#_Toc133808943)

[Figure 10 Compress the frames (to about 25% of the original size) 8](#_Toc133808944)

[Figure 11 generates the processed camera frame sequence 8](#_Toc133808945)

[Figure 12 run server 8](#_Toc133808946)

[Figure 13 router 9](#_Toc133808947)

[Figure 14 client web page 9](#_Toc133808948)

[Figure 15 Skip the wrong file and empty images 10](#_Toc133808949)

[Figure 16 Get the local IP address by using Google Public DNS 10](#_Toc133808950)

# Introduction

Live streaming has gradually become a very important market in the online video industry in recent years, with more and more people sharing their lives, work, games etc. via live streaming. From the dedicated live streaming platform twitch to the video site YouTube, all have launched live streaming services, from which we can see how popular live streaming platforms have become. This report is an Assignment #3 report for the continuous assessment of Distributed Systems - 92021 - [2022-2023]. This CA looks at live streaming and face recognition, with the aim of building a live streaming site that allows hosts to stream their own cameras and share them for others to see. Additionally, the program used for sharing the camera supports facial recognition, which identifies the faces in front of the camera. If a recognized face is present in the training set, the program displays the name of the person in front of the camera. If the face is not recognized, the program will prompt "Unknown".

# Requirements

## Functional requirements

Functional requirements are the functions or features that a system must have in order to meet the needs of the user. For this application, the functional requirements include:

1. Live video stream processing: capturing a live video stream from the camera and processing the video frames.
2. Face detection: detection of face areas in video frames.
3. Face recognition: recognize detected faces, compare them to a pre-stored face library and determine if they match.
4. Online display: the processed video frames are displayed on the client's web page in real time, including drawing rectangular boxes around the recognized faces and displaying the recognized person's name and match.

## Non-functional requirements

Non-functional requirements are measures of system performance, such as availability, reliability, maintainability, etc. For this real-time face recognition application, non-functional requirements include:

1. Response time: The application should generate a live video stream as soon as possible after receiving a client request to ensure user experience.
2. Real-time: the processing speed of face recognition should be fast enough to ensure a smooth real-time presentation.
3. System stability: the application should have certain error handling capability for abnormal situations such as image file corruption.
4. Ease of use: the user interface should be simple and clear and easy to use.
5. Maintainability: The code structure is clear and modular in design to facilitate later maintenance and upgrade.

# Environment

macOS Ventura 13.3.1

Python 3.9.16

# System Architecture

Graphical user interface, text, application, chat or text message

Description automatically generated

Figure 1 System Architecture

Fig.1 illustrates the basic system architecture of this program. Users access the server resource (this program) through the client (browser), and the server uses the Flask framework and provides services through Face Recognition. In other words, users can access the server resource through their browser, and the server shares the camera's live streaming content, which is streamed to the client. The streamed resources sent to the client are processed by the server and include the results of facial recognition. Therefore, the client can see the results of facial recognition. More specific details about the project structure will be introduced in the next section.

# Directory Structure

Fig.2 shows the Directory Structure of this project, which contains two *.py* files and one *.html* file as seen in the Folder tree. As mentioned in the previous section, in this application, the server service is provided by *server.py*, which is responsible for setting up and running the WSGI server to host the Flask application in the production environment. Face Recognition is provided by *stream.py*, which is responsible for processing camera data streams, displaying and processing facial recognition tasks with clients, and setting up routes, etc. These tasks include loading face images and generating face encodings, etc.

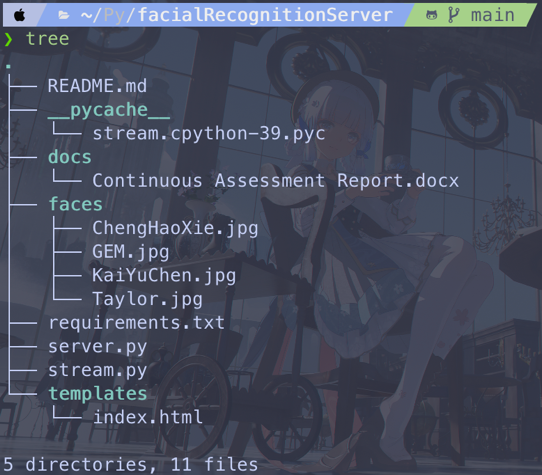


Figure 2 Folder tree of the project directory

# Technologies

The following technologies are mainly used in this application, they are:

* **opencv-python**: used to process images and video streams. In this application it is used to capture live video streams from the camera, resize images and plot the results of face recognition.
* **Flask**: used as a web server framework. In this application, Flask is used to create web applications and set up routing so that clients can access live video streams.
* **numpy**: used for numerical calculations. In this application, it is mainly used to find the index of the best match in the face recognition process.
* **python-socketio**: used for real-time communication. In this program it is used to create a socket.IO server.
* **Flask-SocketIO**: Integrates Socket.IO into a Flask application. In this application it is used to implement real time communication in the application.
* waitress: acts as a WSGI server in the production environment. In this application it is used to start a WSGI server to host the Flask application.
* **cmake**: is used to support the installation of dlib, a cross-platform C++ library for solving problems in machine learning, computer vision, image processing and more. In this application, cmake is used as a dependency to install dlib.
* **dlib**: is used to support the installation of face\_recognition. In this application, dlib is the underlying library used to implement the face recognition functionality.
* **face\_recognition**: used to perform face recognition. In this application it is used to detect faces in video frames, extract face features, compare face features and calculate matches.

# Installation

You can install the required packages for this program by running the following command in the root directory:

**pip3 install -r requirements.txt**

or

**pip install -r requirements.txt**

I encountered some errors when configuring my environment, and you may encounter these problems as well. You may be able to try to resolve the problem by using these commands:

**pip3 install --upgrade pip**

**python3 -m pip install --upgrade setuptools**

Here are some of the other problems I have encountered:

## Problems

### Could not find version that satisfies requirement cv2 OpenCV

**Error log:**

Could not find version that satisfies requirement cv2 OpenCV

**Solution:**

Using **pip install opencv-python** instead of pip install cv2 [1]

### socketio

**Why error happen?**

The package literally named "socketio" seems like it may simply be old and unsupported, while "python-socketio" seems alive and developed. [2]

**Solution:**

Using **python3 -m pip install python-socketio**

### Other errors

**Error log:**

fatal error: 'Python.h' file not found / Could not build wheels / metadata-generation-failed / setuptools>=49.6.0, but you have setuptools 3.3

**Solution:**

**pip3 install --upgrade pip**

**python3 -m pip install --upgrade setuptools** [3]

# Implementation

This section will describe the implementation of this program in detail. This section will first introduce the packages used, then the functional implementation with the implementation of the following three specific requirements:

* Compress the frames (to about 25% of the original size) before feeding them into the models. This already makes a huge difference, while recognition still works well!
* Skip some frames. Do the detection/recognition part only on every other frame.
* Use separate threads for camera reading, model inference, and image display.

## Implementation

In ***stream.py***, the program first loads the face images (Fig.3) stored in the faces folder via the **face\_recognition.load\_image\_file** function and stores the images and the corresponding names (filenames) in the **face\_images** and **face\_names** arrays respectively.

Text

Description automatically generated

Figure 3 loads the face images

The **find\_face\_encodings** function (Fig.4) computes the face encodings for these known images. This function takes a set of images and calculates a face code for each image. The facial encoding is a 128-dimensional vector that represents the features of the face. These encodings will be used in subsequent facial recognition tasks. In this function, we traverse the list of input images, convert the images from BGR format to RGB format and then extract the face encodings using the **face\_recognition.face\_encodings()** function. Finally, these encodings are added to a list and returned.

Text

Description automatically generated

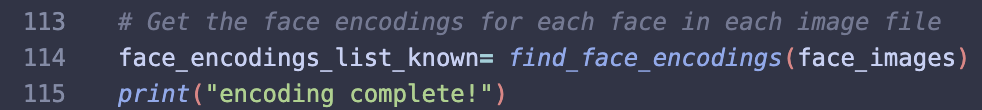


Figure 4 computes the face encodings for these known images

**face\_distance\_to\_conf** (Fig.5) function is used to convert the face distance calculated during face recognition to a confidence score. This score represents the level of confidence of the face match. This function takes two parameters: **face\_distance** and **face\_match\_threshold**, with a default threshold of 0.6.

A screenshot of a computer

Description automatically generated

Figure 5 face\_distance\_to\_conf function

The **VideoCapture** class from OpenCV is used to create a **camera** object for capturing the real-time video stream from the webcam. (Fig.6)

Text

Description automatically generated with low confidence

Figure 6 create a camera object

### Use separate threads for camera reading, model inference, and image display

The **CameraThread**, **ProcessThread**, and **DisplayThread** classes are used for asynchronously processing video frames. (Fig.7) The **CameraThread** is responsible for capturing video frames from the camera, the **ProcessThread** is responsible for processing the captured video frames (including face detection and recognition), and the **DisplayThread** is responsible for displaying the processed video frames.

Text

Description automatically generatedText

Description automatically generatedText

Description automatically generated

Figure 7 Use separate threads for camera reading, model inference, and image display

### Skip some frames

In the **ProcessThread**, we can see that the **Skip some frames** requirement is achieved by counting the frames in the loop using **frame\_counter** (Fig.8). This helps to reduce the computational demand while maintaining vstreaming.

Text

Description automatically generated

Figure 8 Skip some frames

The **process\_frame** function (Fig.9) is used to detect faces in a video frame and compare them with known face encodings. If a matching face is found, a rectangular box is drawn around the face with the name and match score displayed.

This function is responsible for processing captured camera frames. It takes an image (camera frame) as input and performs the following operations:

* Resizes the image to speed up processing.
* Converts the image from BGR colour space to RGB colour space.
* Detects the position of the face in the image.
* Calculates the feature code for each detected face in the image.
* Compare the feature codes of the faces in the current frame with the known face feature codes to find the best match.
* Draws a rectangular box on the image showing the face name and the confidence level of the match.
* Return the processed image.

Text

Description automatically generated

Figure 9 process\_frame function

### Compress the frames (to about 25% of the original size) before feeding them into the models

The compression of the **frame** is also implemented in the **process\_frame** function by using the **SCALE\_FACTOR** constant, which controls the compression ratio of the frame. (Fig.10)

A picture containing text, outdoor, sign

Description automatically generated

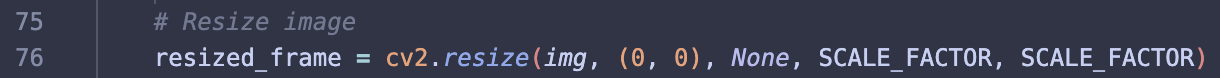


Figure 10 Compress the frames (to about 25% of the original size)

The **gen\_frames** function (Fig.11) then generates the processed camera frame sequence for real-time display on the webpage.

This function is used to generate a sequence of processed camera frames for live display on a web page. It performs the following operations:

* Fetches the processed image from the processed\_queue queue.
* Encodes the image into JPEG format.
* Converts the image to a byte string.
* Generates a sequence of frames and sends it to the client via an HTTP response.

Text

Description automatically generated

Figure 11 generates the processed camera frame sequence

Then in ***server.py***, the **waitress** library is used to host the Flask application and run the WSGI server. (Fig.12)

Text

Description automatically generated

Figure 12 run server

Finally, in ***stream.py***, the routes are bound by using the **@app.route** decorator, providing web page services to the users. (Fig.13)

Text

Description automatically generated

Figure 13 router

The ***index.html*** file in the **templates** folder defines the structure of the client web page. By using **{{ url\_for('video\_feed') }}** in the **src** attribute of the **<img>** tag, the video stream can be transmitted from the server to the client in real-time. (Fig.14)

Text

Description automatically generated

Figure 14 client web page

## Errors

### cv2.error

**error log:**

cv2.error: OpenCV(4.7.0) /Users/runner/work/opencv-python/opencv-python/opencv/modules/imgproc/src/color.cpp:182: error: (-215:Assertion failed) !\_src.empty() in function 'cvtColor'

**solution:**

Skip the wrong file and empty images. (Fig.15)

Text

Description automatically generated

Figure 15 Skip the wrong file and empty images

### socket.gethostbyname(hostname) error

**error log:**

socket.gaierror: [Errno 8] nodename nor servname provided, or not known

This error is mainly caused by the attempt to obtain the local IP address of the computer through socket.gethostbyname(hostname). Specifically, there may be problems with resolving the hostname. This issue may be caused by DNS resolution issues or firewall settings in the operating system (for example, macOS may have measures that prevent me from obtaining the local IP address). The solution is to try another method to obtain the local IP address.

**solution:**

Get the local IP address by using Google Public DNS. [4] (Fig.16)

Text

Description automatically generated

Figure 16 Get the local IP address by using Google Public DNS

# Conclusion

This report summarises and analyses my Facial Recognition Server application which recognises faces of known people from live video streams and displays the recognition results. The application was built using the Flask framework and SocketIO library and written in Python. The file structure of the project contains face images, Python scripts, web templates and the required dependency libraries.

The facial recognition functionality is mainly implemented through the **process\_frame** function in the **stream.py** file. This function utilizes the **face\_recognition** library to detect faces in input images and compares each detected face with the facial features of known individuals. If a match is found, a rectangular box is drawn on the original image, with the matched individual's name and matching percentage displayed below the box.

In the **server.py** file, a web server is launched using the **waitress** library, which associates the generated real-time video stream with the Flask application. Meanwhile, the **index.html** file provides users with a web interface for displaying the live video stream.

The project's dependencies are listed in the **requirements.txt** file and can be installed by running **pip install -r requirements.txt**. To enhance system performance, the project employs multithreading for processing real-time video streams, as implemented in the **CameraThread**, **ProcessThread**, and **DisplayThread** classes in the **stream.py** file.

Within the project, the **faces** folder stores face images of known individuals, which are used to recognize individuals in the video stream. New facial images can be added to this folder, allowing the application to recognize more individuals. Currently, the project includes facial images of four known individuals.

In conclusion, this project implements a real-time facial recognition application capable of identifying and labeling known individuals' faces. The web application, built using the Flask framework and SocketIO library, offers a user-friendly interface, while multithreading technology ensures smooth processing of the real-time video stream and enhances performance.

# Bibliography

|  |  |
| --- | --- |
| [1] | B. Hadzhiev, "Could not find version that satisfies requirement cv2 OpenCV | bobbyhadz," [Online]. Available: https://bobbyhadz.com/blog/python-could-not-find-version-that-satisfies-requirement-cv2-opencv#:~:text=The%20error%20%22Could%20not%20find,not%20supported%20by%20the%20package.. |
| [2] | ti7, "• python - Installing socketio module on python3 seems to be corrupting pip - Stack Overflow," 26 Feb 2021. [Online]. Available: https://stackoverflow.com/questions/63385158/installing-socketio-module-on-python3-seems-to-be-corrupting-pip. |
| [3] | BigchainDB, "How to Install the Latest pip and setuptools - BigchainDB 0.6.0 documentation," [Online]. Available: https://bigchaindb.readthedocs.io/projects/server/en/v0.6.0/appendices/install-latest-pip.html. |
| [4] | UnkwnTech, "networking - Finding local IP addresses using Python's stdlib - Stack Overflow," 3 Oct 2008. [Online]. Available: https://stackoverflow.com/questions/166506/finding-local-ip-addresses-using-pythons-stdlib. |

# Appendix

GitHub: [cky008/facialRecognitionServer: Live Streaming Service - Facial Recognition Server (https://github.com/cky008/facialRecognitionServer)](https://github.com/cky008/facialRecognitionServer)