

Distributed Systems Project Report

Live Video Streaming Service with Facial Recognition



April 30, 2023

YIKUN FAN 20099869

Table of Contents

[1. Introduction 2](#_Toc133873314)

[2. Architecture & Technologies 2](#_Toc133873315)

[2.1 Architecture 2](#_Toc133873316)

[2.2 Technologies 3](#_Toc133873317)

[3. Functional and non-functional requirements 4](#_Toc133873318)

[3.1 Functional requirements 4](#_Toc133873319)

[3.2 Non-functional requirements 4](#_Toc133873320)

[4. Implementation 4](#_Toc133873321)

[4.1 Design 4](#_Toc133873322)

[4.2 Implementation 5](#_Toc133873323)

[4.2.1 Installing and Configuration 5](#_Toc133873324)

[4.2.2 Development 6](#_Toc133873325)

[4.2.3 Integration & Key Code Walk through 7](#_Toc133873326)

[5. Conclusion 9](#_Toc133873327)

[5.1 Modification 9](#_Toc133873328)

[5.2 Summary 13](#_Toc133873329)

[6. Reference 13](#_Toc133873330)

# 1. Introduction

As a modern technological achievement, facial recognition technology has brought convenience to our lives. This project aims to use facial recognition technology to construct a web app that can recognize faces from live stream video.

So far, many mature face detection models have been developed worldwide, such as VGGFace designed and developed by the Computer Vision Group at the University of Oxford, FaceNet developed by Google, and OpenFace developed by Carnegie Mellon University. These models are very powerful and can perform tasks such as age estimation and emotion recognition in addition to facial recognition.

However, these powerful models require a significant amount of computing resources. After considering the application scenarios and requirements of this project comprehensively, I have chosen to use the face\_recognition library as the algorithm for facial recognition. face\_recognition uses the deep learning framework dlib for training. The library is easy to use, and its applications include detecting faces in images and video streams, extracting facial features, and matching them. It can identify the location of faces, provide visual outputs, and calculate the confidence of matches, which aligns well with the goals of this project.

In this project, I will also use other technologies such as OpenCV to ultimately create a web application that can identify the location of faces, label the name of the match, and provide a confidence score that can be accessed by multiple users simultaneously.

# 2. Architecture & Technologies

## 2.1 Architecture

When designing the architecture of this system, some requirements for system coupling and security were sacrificed in order to improve performance. As a result, data communication between modules is rather simple. The system is divided into several modules, as shown in the following diagram:

User Interface: This is the frontend of the web application where users can view the output from the server. It is built using technologies such as HTML.

Server: This is the web server and backend service of the web application, which are implemented to respond to frontend requests. It is implemented using the Flask web framework.

Face Processing Module: This module includes two sub-modules, Face Detection and Face Recognition. Face Detection is responsible for extracting faces from images or video streams using OpenCV. Face Recognition is responsible for functions such as identification, matching, and detection of faces, using the Face Recognition API.

Database: This module stores the information of faces for image comparison, and is simplified to a folder that stores images.

The overall architecture is shown in the diagram.

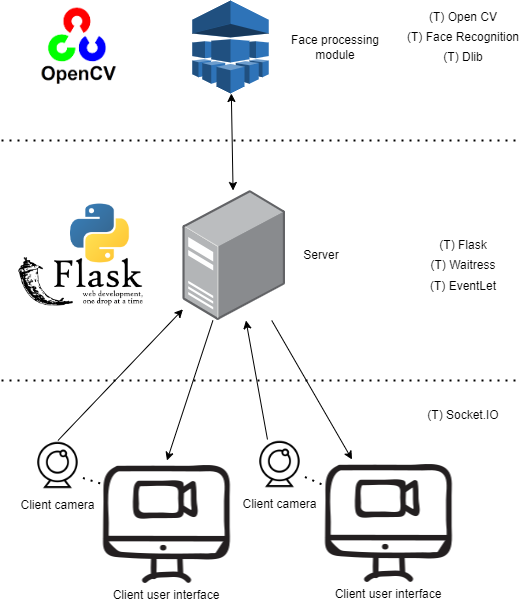


Figure 1 System Architecture

## 2.2 Technologies

In order to develop this system, various technologies such as Flask, OpenCV, are used, where they used are indicated in the diagram. Here, let me introduce the key technologies:

OpenCV: OpenCV is a cross-platform computer vision library that provides algorithms for image processing and machine learning. In this application, OpenCV is mainly used for image processing and face detection, as well as for locating facial features.

The face\_recognition library: The face\_recognition library is a Python package that provides face recognition capabilities using deep learning algorithms. In this application, the algorithm is responsible for implementing the face recognition functionality, by comparing the detected faces with known faces and determining the identity of the face.

Dlib: Dlib is an open-source C++ machine learning library that provides various algorithms in the field of machine learning. In this application, Dlib is the underlying library for the face\_recognition algorithm, providing the face recognition algorithm for the face\_recognition library.

Flask: Flask is a lightweight web application framework written in Python. In this application, Flask is used to build the web app, defining the application structure, handling client requests, and providing a web server interface.

Waitress: Waitress is a lightweight and reliable WSGI (Web Server Gateway Interface) server that can be used to build web servers. In this application, Waitress is used to build the web server, considering the limited hardware resources and small-scale application.

Eventlet: Eventlet is a Python asynchronous I/O library, similar to Waitress, that provides a simple and efficient way to write concurrent and high-performance web apps. In this application, Eventlet is used to learn how to build a web server.

Socket.IO: Socket.IO is a JavaScript library that provides event-based communication and can transmit data through WebSocket protocol, polling, and other transport protocols. In this application, Socket.IO is responsible for implementing bi-directional communication between the client and server.

# 3. Functional and non-functional requirements

## 3.1 Functional requirements

(1) The application must be accessible via a network server.

(2) The application must be able to capture video in real-time through a webcam.

(3) The application must be able to recognize and classify faces in video frames.

(4) The application must be able to recognize and classify at least three different people's facial features.

(5) The application must annotate recognition results on video frames.

## 3.2 Non-functional requirements

(1) The application's response speed should be fast enough, and the FPS should be stable at 64 or higher.

(2) The application's recognition accuracy should be high, with a success rate of 90% or higher for recognition and classification.

(3) The application should be user-friendly, allowing users to easily use the system for face recognition.

(4) The application should have high scalability to facilitate future refactoring.

(5) The application should have good maintainability and testability to facilitate system maintenance and updates.

(6) The application should have good security to prevent system abuse and information theft.

# 4. Implementation

## 4.1 Design

The workflow of the entire system is as follows:

(1) Use the camera to capture live video and send it to the backend.

(2) The backend server performs face detection on the received video.

(3) If a face is detected, it is processed for location and comparison.

(4) The recognition result is returned.

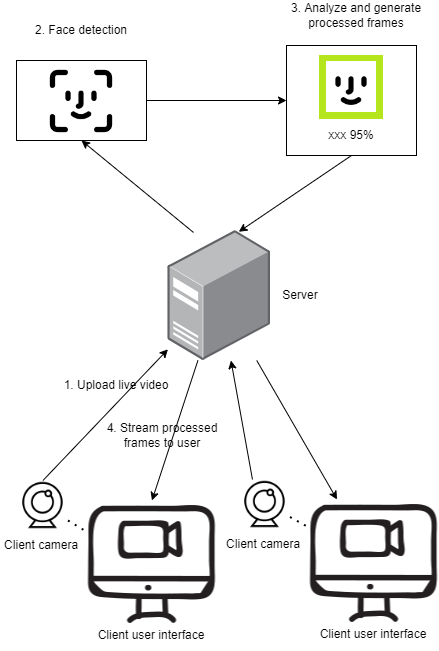
The workflow is illustrated in the following diagram:

Figure 2 System Design

## 4.2 Implementation

### 4.2.1 Installing and Configuration

Firstly, I referred to a previous schoolmate’s assignment to learn from his project. However, during trying his project, I encountered many difficulties, mainly in the installation of the required libraries.

(1) When installing Socket.IO, I encountered an error that prevented me from installing it. Based on the error message, I searched for a solution online and found an article [1] that suggested that my computer was missing C++. I resolved this issue by installing Microsoft Visual C++ Build Tools.

(2) Later on, I encountered another issue when installing dlib. After trying the recommended version of dlib suggested by the project specification, I was still unable to install it. However, this prompted me to do further research, and I came across an article [2] which suggested that I download the appropriate source file for dlib that matched my virtual environment's Python version and install it locally. For my Python version, the appropriate dlib version was dlib-19.22.0. Unfortunately, I still couldn't install it even after the local installation. After further research in article [3], I found that the issue was with setuptools. During the installation of Socket.IO, setuptools was set to use version 3.3, which was incompatible with the current version of dlib. Therefore, I created a new virtual environment and upgraded setuptools to the latest version, which allowed me to successfully install dlib.

### 4.2.2 Development

According to the main functionalities, I divided the application into multiple phases and developed them step by step.

(1) In the first phase, I created a server and build the application structure. I built a user-friendly front-end page and set up the corresponding routes and services on the back end. After testing, the server was able to receive client connections, and I confirmed that multiple connections could exist simultaneously, as shown in the figure:

Text

Description automatically generated

Figure 3 Web server running logs

As we can see, "(10796) accepted ('127.0.0.1', 55453)" and "(10796) accepted ('127.0.0.1', 55454)" indicate that two new clients have established connections with the server.

(2) In the second phase, I developed the part of the video stream processing. During this stage, I implemented the function of locating the position of faces in the uploaded video stream. Here is an example of the result:

A picture containing text, person, wall, indoor

Description automatically generated

Figure 4 Result of trying face detection

(3) In the final phase, I developed the functionalities of the facial processing module. In this phase, the system can detect and recognize faces in the captured images. The screenshot shows the result.

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

Description automatically generated

Figure 5 Result of trying face recognition

### 4.2.3 Integration & Key Code Walk through

At the end of the development, I integrated all the functional modules into the web application. Next, I will introduce the key code of the application in the order of the workflow.

(1) First, a server application ‘app’ and a Socket.IO application ‘socketioApp’ are created using the Flask framework, where ‘app’ is used to receive HTTP requests, and ‘socketioApp’ is used to receive WebSocket requests.

Text

Description automatically generated

Figure 6 Key code of creating web server

(2) Using the face\_recognition library, all the facial images in the faceImages directory were read and encoded, and then stored in the facesKnownEncoded list. At the same time, the file names of the facial images (excluding the extension) were stored in the names list.Text

Description automatically generated

Figure 7 Key code of loading known images

(3) Initialize the ‘VideoCapture’ object cap from OpenCV to read the video stream from the local camera.

Text

Description automatically generated

Figure 8 Key code of video capturing

(4) In the ‘processFrames’ function, each frame is read from the video stream using cap.read(). The face\_locations function from the face\_recognition library is called to find the locations of all faces in the frame. Then, the face\_encodings function is called to encode the feature vectors for each face.

Text

Description automatically generated

Figure 9 Key code of detecting faces in one frame

(5) For each detected face, first, the ‘compare\_faces’ function from the face\_recognition library is called to compare it with all the faces in ‘facesKnownEncoded’, resulting in a Boolean array of match results called ‘matches’. Second, the face\_distance function from the face\_recognition library is called to calculate the distance between the detected face and all the faces in ‘facesKnownEncoded’. Only when the smallest distance corresponds to a true value in matches is the face considered successfully recognized.

Text

Description automatically generated

Figure 10 Key code of recognizing faces in one frame

(6.1) If the recognition is successful in step 5, the ‘matchIndex’ is used to find the name of the matched face in the ‘names’ list, as well as the confidence level ‘matchPercentage’ of the match. Then, a rectangle is drawn around the face and the recognition result and confidence level are labelled.

A screenshot of a computer

Description automatically generated

Figure 11 Key code of dealing successful recognition

(6.2) If the recognition fails, a rectangle is drawn around the face in the original video frame, and the label "Unknown" is added to it.

A screenshot of a computer

Description automatically generated

Figure 12 Key code of dealing failed recognition

(7) After mark the fps of the processed video frame, the processed video frame ‘img’ is returned. This frame is transmitted to the front-end, and by continuously refreshing the image, a video-like effect is achieved.

Text

Description automatically generated

Figure 13 Key code of returning one processed frame

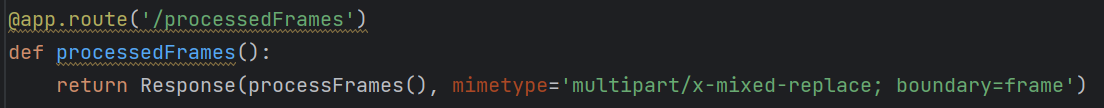


Figure 14 Key code of returning frames to the clients



Figure 15 Key code of using processed frames to simulate live video

# 5. Conclusion

Completing this project has given me a deeper understanding of the machine learning techniques involved in facial recognition. Throughout the process, in addition to performing some more operational tasks, I also discovered some interesting phenomena. In the following, I will summarize these phenomena and the entire project.

## 5.1 Modification

During the development of this application, I found that the video composed of images returned by the backend is smooth when no face is detected. Therefore, in order to maximize the fps when a face is detected, I tried to improve the application and observe the effect of the improvement.

(1) Compressing images

I followed the reference article [4] and compressed the original images before detecting faces. Before compressing the images, the fps of the returned video was stable at 1.9. After compressing the images with a compression rate of 0.5, the fps was stable at around 2.3. Further increasing the compression rate to 0.2, the fps was stable at around 2.45. Therefore, I finally decided to set the compression rate to 0.25, and the fps was stable at around 2.4.

Below are the main code and the results of implementing this modification:

Graphical user interface, text

Description automatically generated

Figure 16 Key code of compressing

A picture containing text, person, indoor

Description automatically generated

Figure 17 Compression optimization effect

(2) Multi-Threading

Perhaps my code wasn't well enough, disappointingly, the performance of the application did not improve after using multiple threads, instead, it even decreased. After recognizing a face, although the FPS still remained stable around 2.4, the video latency became more noticeable. Here is the key code I tried to use multiple threads:

Text

Description automatically generated

Figure 18: Key code of modifying app using threading

I divided the previous ‘processFrames’ function into 3 parts: reading images, detecting and recognizing faces, and generating frames for display. Each part is running in a separate thread, and the first two functions temporarily store the processing results in a common queue for the later function to use.

After referring article [5], I found that although multi-threading is more recommended for Flask compared to multi-processing, in reality, due to the existence of the Global Interpreter Lock (GIL), the Python interpreter is limited to executing only one thread at a time. That is to say, both multi-processing and multi-threading are not suitable for this modification, which aims to improve execution efficiency. At the same time, the article also provides me two alternative solutions: ‘Celery’ and ‘Rq’, which I will try to use in future improvements. In the end, I did not use the code to improve the program with threading, and only stored it in multiThreadProcessFrames.py.

(3) Skip some frames

I choose to process every other frame, which means I process every other frame while skipping the others, because I found that skipped frames are not drawn with the face bounding box and recognition result, and skipping too many frames would greatly affect the user experience. Currently, I have not found a solution without bugs for drawing the face bounding box and recognition result on skipped frames, but I will try to improve it in the future.

There are two changes made to the key code. First, a counter is added to process only every other frame for detection and recognition. Second, the method of calculating fps is modified. In fact, the fps of the frames that are not skipped remains around 2.4, but we need to consider the overall fps of the video stream, so here we calculate the fps using the time of processing both the processed frame and the skipped frame.

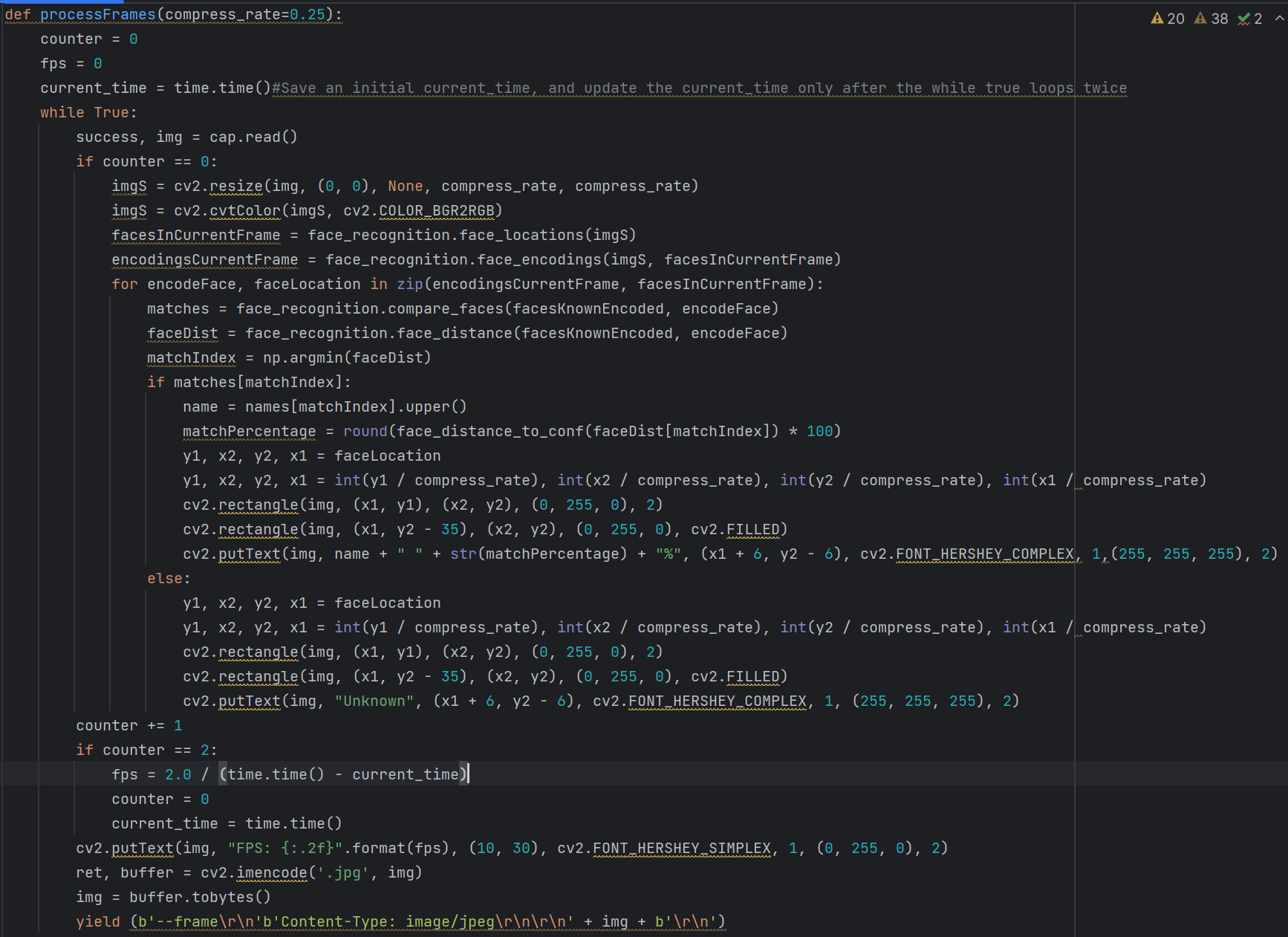


Figure 19: Key code of modifying app using skipping some frames

I observed that after the modification, the fps has significantly improved.

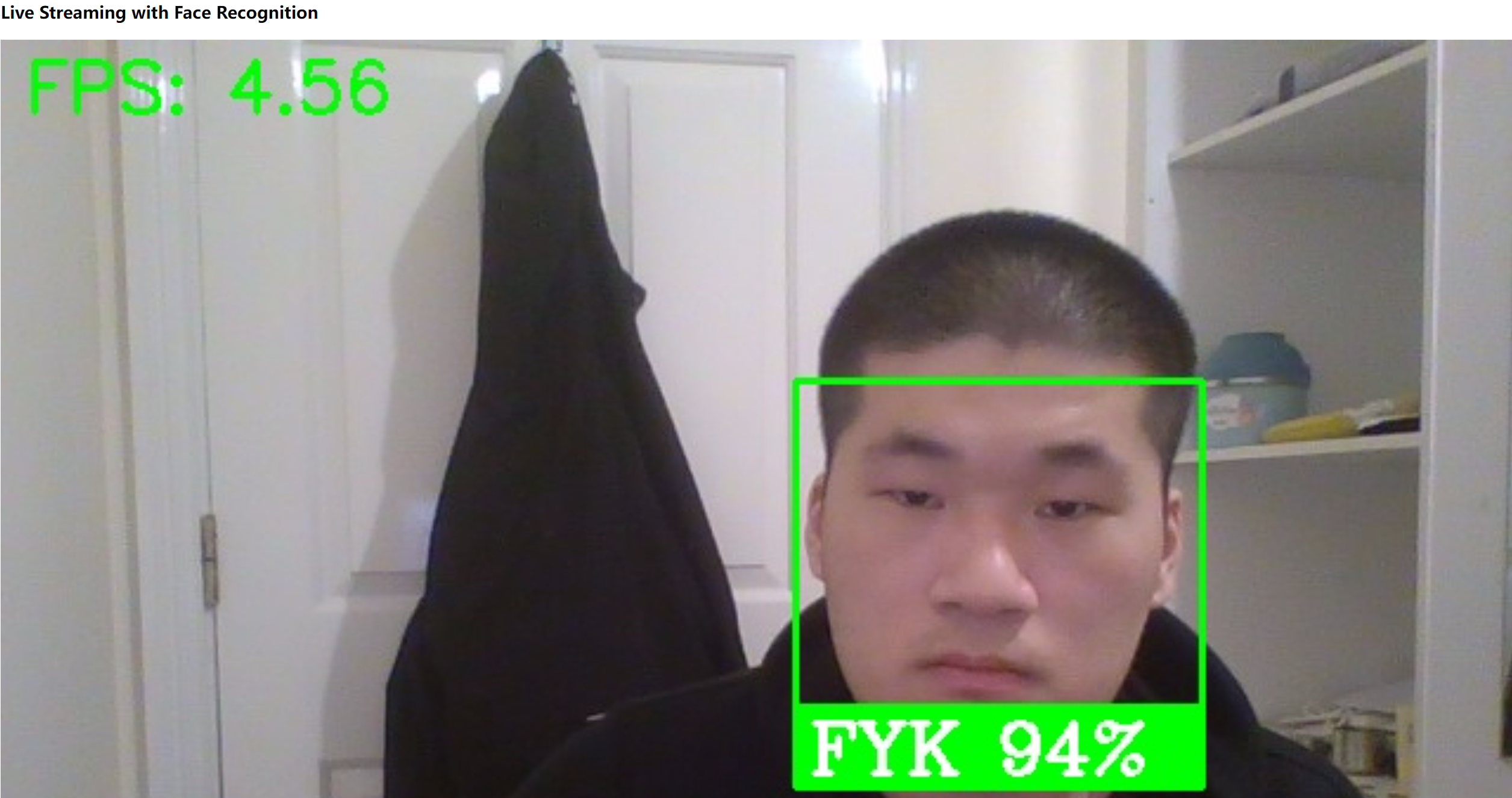


Figure 20: Skipping frame optimization effect

(4) Limiting the number of test images for learning facial detection

Previously, there were two images used for learning facial detection were placed in the same folder as other known facial images, causing these two to be included in the comparison. To avoid this, the two images were moved to a different location, reducing the total number of known facial images from 5 to 3. However, this did not result in a significant decrease in the frame rate, which remained stable at 2.4 frames per second.

In conclusion, the stuttering observed during using application was due to the high demand for computing resources required by the recognition model. Compressing the images before detection and recognition would have a significant impact on improving performance. Although skipping some frames can significantly improve the frame rate, it has some side effects. Other modifications have little effect.

## 5.2 Summary

Through the use of appropriate technology and persistent efforts, I have successfully developed a web application capable of recognizing faces in live video. Although my application may not be perfect, the process of learning, practicing, and optimizing has given me a rough idea of how to further improve it. In the future, I plan to deploy my application on a cloud platform for public use. However, the limited computing resources of the cloud platform require further optimization of my application, and I also need to make more efforts to enhance the security of the system.

# 6. Reference

[1]<https://stackoverflow.com/questions/16469086/npm-cant-install-socket-io>

[2]<https://stackoverflow.com/questions/41912372/dlib-installation-on-windows-10>

[3]<https://stackoverflow.com/questions/35991403/pip-install-unroll-python-setup-py-egg-info-failed-with-error-code-1>

[4]https://learnopencv.com/image-resizing-with-opencv/

[5]https://stackoverflow.com/questions/68126489/what-is-the-recommendation-about-using-multiprocessing-or-multithreading-insid