CMPUT 398 Optional Project
Client-Server Emotion Detection System

Tina Nguyen

1762833

ngocthuy

1. Introduction

Due to the rapid advances in machine learning, computer vision has become one of the most widely impactful areas. Visual applications such as face detection, identity verification, and environmental understanding depend heavily on deep neural networks, which require substantial computational resources. Among these visual applications, emotion detection has become increasingly important. By identifying whether a user is happy, sad, or neutral, machines can better evaluate engagement, interpret the surrounding atmosphere, and respond more effectively to users. It makes emotion recognition a paramount tool in interactive systems.

This project focuses on emotion detection as it is an excellent case of a computationally heavy machine learning task and therefore well-suited to a client-server architecture. DeepFace, as a facial recognition framework, uses a complex nine-layer neural network and hundreds of millions of weights [1], which usually demand specialized libraries, GPU acceleration, or optimized numerical computation. Running these models directly on phones, tablets, or lightweight laptops is often infeasible. Offloading the computation to a backend server allows almost any device with a web browser to access advanced ML capabilities without needing to install large dependencies or handle intensive processing locally on the device.

The goal of the project is to build a system capable of having a separate server to perform computation and a separate client to handle user interaction. The backend is implemented using FastApi and integrates the DeepFace library to both analyze images and video files. The frontend, implemented in React, provides an intuitive interface for uploading files, and viewing the analysis returned by the server. The separation enables scalability, maintainability, and the flexibility to update the model on the server without requiring any client-side changes or powerful client hardware. Therefore, a terminal-based approach was not considered due to its monolithic design.

To enable this separation between the client and server, a REST API was used for communication between the two components. REST provides a simple, stateless, and widely supported mechanism for sending requests and receiving responses, making it well-suited for standalone operations such as uploading an image or video and retrieving the corresponding emotion analysis. WebSockets were briefly considered as an alternative, particularly for low-latency video streaming, but they introduce unnecessary complexity for this project. WebSockets require maintaining persistent connections, handling bidirectional communication, and tracking session state. Since the system does not need to support live streaming, the additional overhead is not justified. For the scope of this project, a REST-based architecture offers the cleanest and most appropriate approach to illustrating core client–server principles while effectively supporting image and video uploads.

2. Analysis Implementation:

a. Emotion Analysis Engine:

Lies at the core of the system is DeepFace library, an open-source Python library that provides a unified interface for several popular models such as VGG-Face, FaceNet and OpenFace[2]. Instead of requiring developers to manually load TensorFlow, Keras or PyTorch models, process and align for inference, DeepFace is a single “framework” that acts as a black box to provide pre-trained facial analysis models capable of producing emotion predictions for a given image. The project does not retrain or modify any underlying network weights but rather treats DeepFace as a standalone encapsulated computational component that accepts an image (either as raw bytes or a NumPy array) and returns a dictionary of mapping emotions to corresponding confidence values. All emotion recognition in both images and videos is reduced to a single DeepFace call.

```
analysis = DeepFace.analyze(
    img_path=io.BytesIO(file_bytes),  # or cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    actions=['emotion'],
```

```
    enforce_detection=False
)
```

Only the `actions=['emotion']` parameter is enabled to limit the analysis to only emotion classification and to reduce computational load. The `enforce_detection=False` flag allows the system to return results even when a face is partially visible or difficult to detect, which is important for real-world video frames that may be low quality or motion blurred. The raw DeepFace outputs contain the mapping of emotions to confidence scores. Then we extract the most three predominant feelings based on its confidence levels from a single frame to be returned as JSON when dealing with single images or to be further processed when with videos. As a result, the DeepFace component serves as a foundational computational layer for the client-server overall system.

b. Other supporting Libraries:

Although DeepFace performs the actual emotion inference, several supporting libraries are also important in building the overall system. As a backend framework, FastAPI is used to provide REST service, chosen for its lightweight design, simplicity, high performance and support for asynchronous file uploads. In addition, to decode video files and iterate through all the frames effectively in the video, OpenCV library is utilized. Pillow (PIL) validates uploaded images, ensuring that image data is not corrupted nor malformed before processing to protect data integrity. For persistence, SQLAlchemy is chosen due to its compatibility and easy setup with FastAPI and it acts as the ORM layer, saving images, videos, and frame-level results as relational database entries into the local database. Finally, NumPy is used to calculate statistical summaries of emotions throughout the video such as averages, standard deviations, and extreme values for each detected emotion. Together, these tools provide the infrastructure required to build the client-server system around the DeepFace framework.

c. Video Aggregation and Temporal Weighting:

As DeepFace supports emotion detection in a single frame or image, emotion detection in videos requires a higher-level design. Building upon the smaller component of detecting in a single image, videos are essentially a sequence of images or frames. Instead of performing detection in every single frame in the video, we only sample every N-th frame (default: 30) to balance performance with accuracy. Each sampled frame is passed into the same DeepFace call used for single images. Once we gather a sequence of frame-level predictions, the challenge is to combine these predictions into a meaningful interpretation of the predominant emotion throughout the video.

A naive approach, with which I have tried previously, was to average all confidence values of given emotions and select the highest values. However, this approach implicitly assumes that all frames represent equal amounts of time and hence should be weighted equally. In practice, this is not the case. If one sampled frame appears for 0.15 seconds and the other appears for 0.20 seconds later given both having the same confidences, then the emotions of the latter frame should implicitly contribute more to the result. Treating these two frames as "equal" would incorrectly imply that both durations contribute equally to the overall emotional content, even though one frame spans a far longer portion of the video timeline. This becomes especially important in realistic videos where facial expressions or emotions fluctuate.

To take into account the duration, after all frames are processed, the final time-weighted average for each emotion is obtained by dividing the accumulated weighted sum by the total weight or $avg = \frac{\Sigma(confidence\ x\ duration)}{\Sigma\ duration}$

```
emotion_weighted_scores[emotion]["weighted_sum"] += confidence * duration
emotion_weighted_scores[emotion]["total_weight"] += duration

"average": round(data["weighted_sum"] / data["total_weight"], 2)
```

In addition to weighted average, the system also computes simple averages, standard deviation, max, min and overall percentage value, which reflects how much of a video an emotion occupies relative to the total duration. The result is sorted by time-weighted average,

enabling the system to identify the dominant emotions throughout the video. Through the video analysis, we start with single frame analysis and construct richer behaviour by combining and aggregating them over time. This shows component reuse and composition of a smaller part towards a larger system.

3. API Design

The system implements REST interface to mediate all communication between the React frontend and the computational backend. Since emotion detection is inherently request-driven, stateless nature where users upload images or videos to receive emotion analysis as responses, REST is the optimal choice. In our project, we expose *GET, POST, DELETE* endpoints for retrieval, creation and deletion of resources as emotion analysis. The API is designed to uphold statelessness, clear resource boundaries, separation of concerns, and consistency. For statelessness, since each request contains all necessary information, the server does not need to keep user state between calls. To deal with resource boundaries, single images, videos, and individual video frames act as separate model resources with unique identifiers. In our models defined in the database, we have three categories: **ImageAnalysis** for a single-frame emotion classification result, **VideoAnalysis** for a collection of frame-level results and aggregated statistics and **VideoFrame** for a per-frame emotion prediction associated with a specific video. By defining them separately and uniquely with integer ID stored in SQLAlchemy-backed relational databases, each analysis is persistent and independent for retrieval. Also, while metadata and results are returned as JSON, binary media of original frames and extracted frames of videos are provided through different dedicated file endpoints. This enhances the separation and special functionality of each endpoint. Last but not least, both image and video analysis follow parallel endpoint structures, reducing design complexity and maintaining consistency for easier frontend implementation. For both the image and the video uploaded for

analysis, make sure that the name of the file is unique to ensure idempotence in POST requests. Below is the endpoint overview:

Table 1: REST Endpoints provided by the Backend

Resource	Method	Endpoint	Description
Image	POST	/image	Upload an image and perform emotion analysis Argument: - Required: file (PNG/JPEG)
Image	GET	/images	Retrieve all stored image analysis
Image	GET	/image/{id}	Retrieve metadata and results of the image with corresponding id
Image	GET	/image/{id}/file	Retrieve the image file stored
Image	DELETE	/image/{id}	Delete the image information including analysis and file
Video	POST	/video	Upload a video and perform frame-based analysis Argument: - Required: video (MOV/AVI/MP4) - Optional: frame_interval
Video	GET	/videos	List all stored video analyses
Video	GET	/video/{id}	Retrieve metadata, aggregated statistics, and frame summaries
Video	GET	/video/{id}/frame/{n}	Retrieve emotion predictions for a specific frame
Video	GET	/video/{id}/frame/{n}/file	Retrieve the stored frame image
Video	DELETE	/video/{id}	Delete a video analysis and all associated frames

4. Empirical Evaluation

The section shows examples of the functionality of the emotion-detection using direct HTTP requests through curl commands and through the React frontend. The goal is to

demonstrate correctness of the API, and expect returned data structures, and verify that the system performs DeepFace inference reliably for both images and videos.

a. Example 1: Analysis of Shutter Island trailer video with the frame interval of 15

Curl request:

```
curl -X POST "http://localhost:8000/video" \
--header 'Accept: */*' \

--form 'file=@"/Users/tinanguyen/Downloads/Shutter Island - Official Trailer [HD].mp4"' \
--form 'frame_interval="15"'
```

Curl response:


```
{
  "id": 2,
  "upload_date": "2025-11-18T01:34:13.545920",
  "video_info": {
    "total_frames": 3411,
    "analyzed_frames": 114,
    "duration_seconds": 142.27,
    "fps": 23.98
  },
  "frame_by_frame": [
    {
      "frame": 0,
      "timestamp": 0.00,
      "top_k_emotions": [
        {"emotion": "sad", "confidence": 99.99},
        {"emotion": "fear", "confidence": 0.01},
        {"emotion": "happy", "confidence": 0.00}
      ],
      "dominant_emotion": "sad",
      "dominant_confidence": 99.99
    },
    ...
  ],
  "aggregated_emotions": {
    "dominant_emotion": "disgust",
    "dominant_average_confidence": 49.02,
    "emotions": {
      "disgust": {
        "average": 49.02,
        "simple_average": 49.02,
        "max": 97.38,
        "min": 0.00,
        "std": 39.76,
        "presence_percentage": 2.65
```

```
      },
      ...
    }
  }
}
```


For the *Shutter Island* trailer, using a frame interval of 15, the aggregated results identify *disgust* as the dominant emotion with the average confidence of 49.02%, *angry* with 45.06% and *sad* with 41.36% in the video. This suggests that over time, the model consistently interpreted the facial expressions in the video as conveying a predominantly negative emotional tone, which is consistent with the tone of the trailer in my opinion.

b. Example 2: Analysis of the below image

Curl request:

```
curl -X POST "http://localhost:8000/image" \
--header 'Accept: */*' \

--form 'file=@"/Users/tinanguyen/Downloads/female-model-near-window-with-rain-drops.jpg"'
```

Curl response:


```
{
    "top_k_emotions": [
        {
            "emotion": "fear",
            "confidence": 72.76898956298828
        },
```

```
        {
            "emotion": "sad",
            "confidence": 26.92816734313965
        },
        {
            "emotion": "surprise",
            "confidence": 0.2107897400856018
        }
    ],
    "dominant_emotion": "fear",
    "dominant_confidence": 72.76898956298828,
    "id": 1,
    "upload_date": "2025-11-18T02:03:52.648717"
}
```


These results suggest that the model interprets the subject's facial expression as strongly fearful, with some secondary indications of sadness.

c. Example 3: Analysis of the Frame 36 for the trailer video Pharrell Williams - Happy (Official Video).mp4 of frame interval 30

Curl request:

```
curl -X GET "http://localhost:8000/video/2/frame/1050/file" \
--header 'Accept:
image/webp,image/avif,image/jxl,image/heic,image/heic-sequence,video/*;q=0.8,image/png,image
/svg+xml,image/*;q=0.8,*/*;q=0.5' \
```

Curl response:

Curl request:

```
curl -X GET "http://localhost:8000/video/2/frame/1050/" \
--header 'Accept:
image/webp,image/avif,image/jxl,image/heic,image/heic-sequence,video/*;q=0.8,image/png,image
/svg+xml,image/*;q=0.8,*/*;q=0.5' \
```

Curl response:


```
{
   "frame_number": 1050,
   "timestamp": 35.04,
   "dominant_emotion": "happy",
   "dominant_confidence": 98.5277328491211,
   "emotions_data": [
       {
           "emotion": "happy",
           "confidence": 98.5277328491211
       },
       {
           "emotion": "surprise",
           "confidence": 0.8993520140647888
       },
       {
           "emotion": "neutral",
           "confidence": 0.3744669556617737
       }
   ]
}
```


These api endpoints return the analysis of the single frame in the video and the video.

5. Conclusion

This project implemented a client–server emotion detection system that uses FastAPI and DeepFace to analyze both images and videos. The results demonstrate that separating computation from the frontend provides scalability and keeps the client lightweight. While the current system relies on CPU processing and a pre-trained model, it forms a strong foundation for future improvements such as GPU support and real-time video analysis.

6. References

[1] RoX. (2025, June 17). DeepFace: Revolutionizing facial recognition technology. *AICompetence.org*. https://aicompetence.org/deepface-recognition-technology/

[2] *Medium*. (n.d.). Medium. https://medium.com/@p4prince2/understanding-deepface-and-its-powerful-models-for-face-recognition-a5541370683