Software Engineering Project
Design Documentation - Team 4

TITLE : **Emotion Detection in Videos**

Team Members :
Rizwan Ali (20171167)
Manan Goel (20171102)
Priyank Modi (20171055)
Devesh Vijaywargiya (20171132)
Nitish Dwivedi (20182010680)
Aashna Jena (20171095)

Overview

Our aim is to build a web app on which users can upload their videos and in return, obtain a video in which the emotions displayed are highlighted. Our aim is to achieve good results in the shortest time possible, hence we're reducing time taken to process these videos by distributing the frames of the videos over multiple CPUs, achieving results 20 times faster than the sequential processing technique of these videos. Multiple users can have their videos processed asynchronously, due to our task queue thereby achieving overall horizontal scalability.

Technology Stack

Database : Postgres
Task Manager : Celery/Ray
Task broker (cache) : Redis
Production server : Gunicorn
Load balancer : nginx
Deployment : Docker containers
Python, Flask, Auth0, OpenCV

Goals

- User friendly and user specific interface
- Parallelizing the Processing of Video Frames
- Highlighting the emotions in each frame parallely
- Support Multiple Users processing their videos asynchronously and keeping the database in an available state
- Containing our product in Docker for ease of` deployment

Non-Goals

- Improving Face Detection accuracy
- Divide and Conquer type video writing on multiple processors
- Scope of the emotion detection is not extensible

Milestones

- Web application with User Authentication and a Dashboard
- Addition of Uploading and Playback mechanism
- ML and CV based model that returns distribution of emotions in a video
- Incorporate Multi-threading and Distributed processing
- Integrate the model with the Web application
- User specific features using custom backend in Postgres

Existing Solution

The existing solution processes the frames of videos one by one, hence it takes a lot of time. Moreover, this model does not allow multiple users to access the backend at the same time. Only one user can use this at a given instance of time.

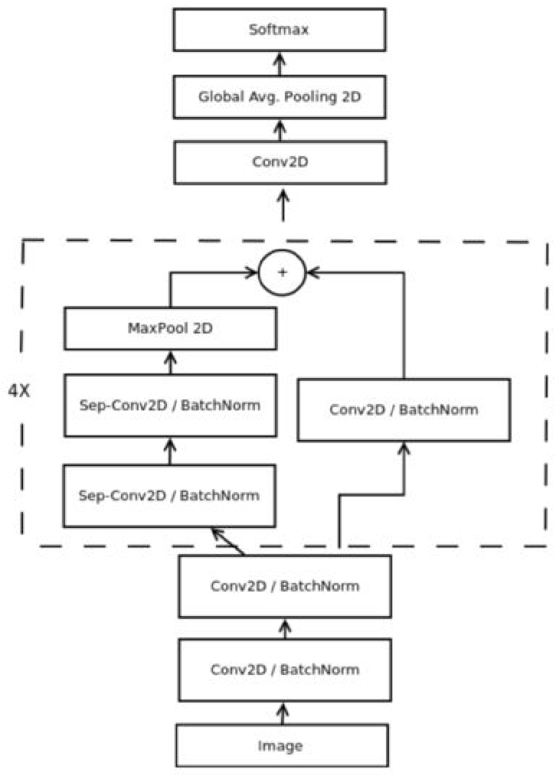
Proposed Solution

As mentioned in the existing solution section, as of now there are models which process videos and re-render them to show emotions displayed on each face in the video. However, each frame is processed sequentially i.e. one after the other, significantly increasing the amount of time taken. Moreover, there is no web-based interface with which regular users can interact. We propose to create a web-based interface using which users can upload a video to the server on which each video will be processed. Each user will also have a personalized dashboard showing only the videos he/she uploads, its status of processing and a link to the newly rendered video if rendering has been completed. The frames of each video will be distributed across CPUs to speed up the processing and once the emotions of each face in each frame are obtained, a new labelled video is rendered.

In order to achieve this, we first make an interface which allows users to sign up and upload videos using Flask. The user sign up and login is done using the Auth0 API, hence no local User database is maintained since any transactions with the User table can be done using API calls. To keep track of videos a database is maintained in PostgreSQL, which stores details of each video and the user who uploaded it. All transactions with the database are done using the SQLAlchemy ORM since it provides a pythonic way of making queries and removes the need to manually write SQL queries.

Once, a video is uploaded, it needs to be sent as input to the model to get the labelled video. However, this needs to be done asynchronously so as not to keep the user waiting for each video to be processed. This is done using celery which is an asynchronous task queue/job queue based on distributed message passing. The execution units, called tasks, are executed concurrently on a single or more worker servers using multiprocessing. The processing of each video is a task which is completed asynchronously. Moreover, queries can also be made for the progress of the task which will be shown on the frontend as well. Celery/Ray distributes tasks across CPUs again making the task completion faster. The message broker being used by celery is redis which is an extremely fast data structure to store key value pairs. Another reason for using these is the ease of interaction with Flask and well maintained python API. All writes to the database are also done through celery tasks to maintain consistency of the database and make sure there are no conflicting write operations.

The model for emotion detection has two parts: Face Detection and extracting emotions from the faces. Face Detection is done using the face_recognition python library which is an implementation of the FaceNet model developed by Google in 2015. The model for emotion extraction is as follows

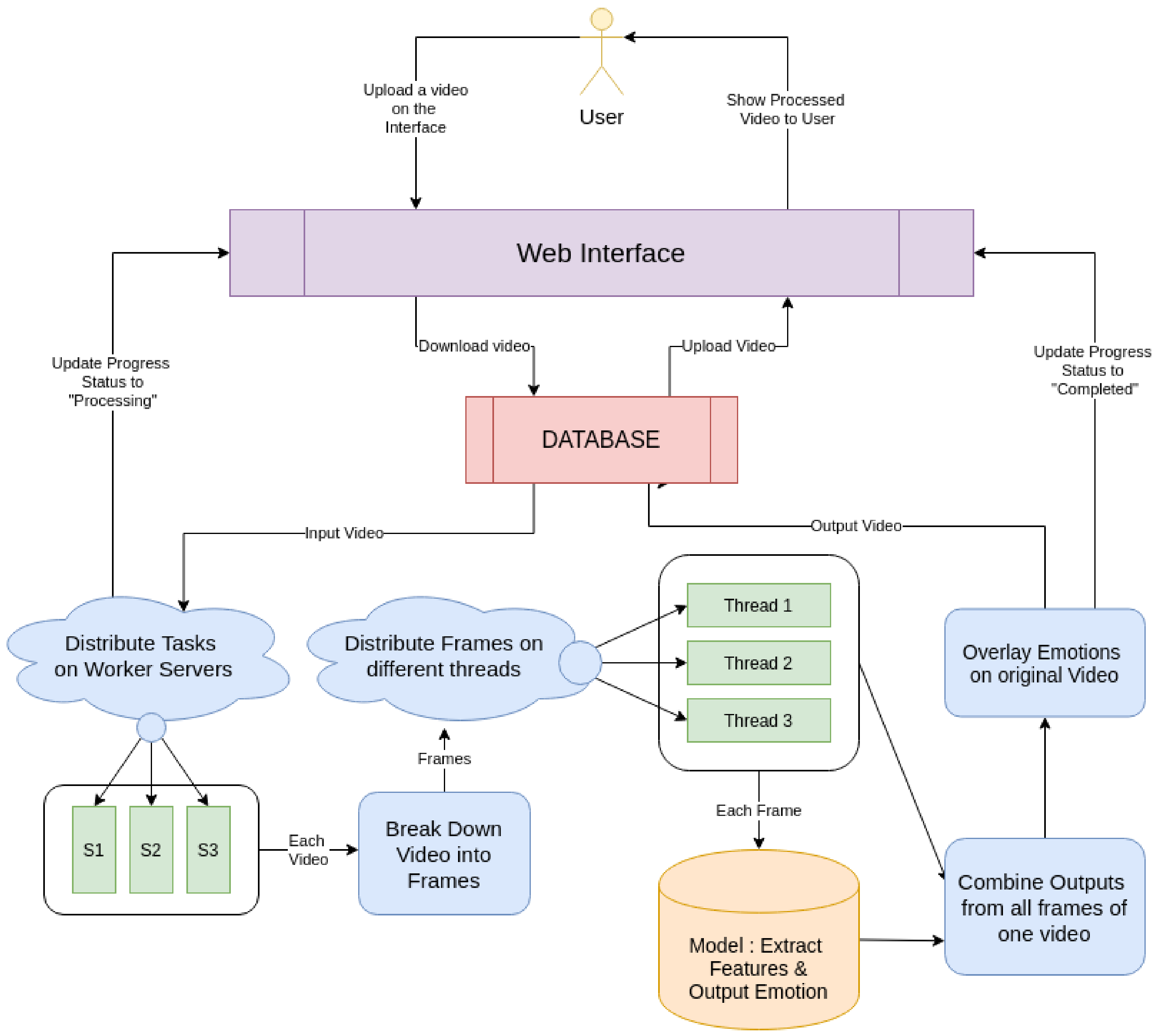


As of now the model predicts 7 emotions i.e. “angry”, “disgust”, “fear”, “happy”, “sad”, “surprise” and “neutral”. The model is trained on the Facial Expression Recognition Challenge dataset using keras with a tensorflow backend. The trained model is then deployed as a part of the web app in which frame data is extracted using OpenCV following which it is passed through the model to obtain a labelled frame. Each of these frames is processed parallely which is taken care of by the Ray/Celery library. Ray/Celery distributes each frame to different processors giving an almost 20x speed up.

The labelled video after rendering is stored and it’s path is added to the database. The status of the video is also updated on the dashboard showing percentage of completion if incomplete and link to the rendered video if completed.

The finally obtained web app is deployed using docker containers to remove issues with deployment and setting up local development environments. We use 4 containers that share resources, one each for 4 microservices: Flask Server, redis backend, celery worker and PostgreSQL server. In production the WSGI used is gunicorn to make sure the server can process as many requests as possible at the same time. We also plan to use a three tier architecture i.e. add nginx as a load balancer on top of gunicorn. However, this is an extended goal and will be done only if time permits.

Workflow Diagram


User
Upload a video on the Interface
Show Processed Video to User
Web Interface
Download video
Upload Video
Update Progress Status to "Processing"
Update Progress Status to "Completed"
DATABASE
Input Video
Output Video
Distribute Tasks on Worker Servers
Distribute Frames on different threads
Thread 1
Thread 2
Thread 3
Overlay Emotions on original Video
S1
S2
S3
Each Video
Frames
Break Down Video into Frames
Each Frame
Model : Extract Features & Output Emotion
Combine Outputs from all frames of one video

TESTING AND MONITORING

Unit Testing

Each individual component will be tested in this step. The purpose is to validate that each **unit** of the software performs as designed.

Test_name	**Component to test**	**Description**
TestU101	Authentication	Verifying if valid user can login using correct credentials
TestU102	Authentication	Verifying if invalid user cannot login using incorrect credentials
TestU201	Video Upload	Verifying if an entered video path correctly uploads the entire video without unnecessary compressions trimming
TestU202	Video Upload	Calling the Celery task asynchronously and then making the code wait until the task is ready to fetch the result and evaluate the test assertions.
TestU301	Dashboard	Verifying if past videos are visible after starting a new session
TestU402	Parallel processing of videos	Verifying if parallel processing(on multiple machines) leads to a reduction in total processing time
TestU402	Parallel processing of videos	Verifying if number of machines being used is equal to a predefined number n>1
TestU501	Face Detection	Verifying that the variance of the coordinates on multiple runs is low
TestU502	Face Detection	Verifying if multiple faces are detected in a video
TestU601	Emotion Detection	Verifying if training/validation set is labelled correctly

*more unit tests to be added for each function depending on the final implementation

Application Testing

The motive of this step will be finding errors in software. It'll deal with **tests** for the entire **application**. It'll help to enhance the quality of the **applications** while reducing costs, maximizing ROI, and saving development time.

Test_name	**Component to test**	**Description**
TestA101	Authentication	Use different combinations of strings to check for the loopholes in the logging in process. This will include using SQL injections and other known norms
TestA102	Authentication	Add multiple users with same details and check for appropriate error messages
TestA201	Video Upload	Trying to upload multiple videos at the same time
TestA301	Dashboard	Verifying if dashboard is visible if opened on multiple windows
TestA401	Emotion/Face Detection	Checking final output against pre-labelled video

DETAILED SCOPE

The main goal of this project is to extract emotions from videos by capturing facial features and output a distribution of emotions.

Deliverables

- Frontend:
    - User authentication
    - User Dashboard
- Backend:
    - PostgreSQL database for storing information about the uploaded videos and specific user related information.
    - Auth0 based authentication system incorporated in the Flask framework.
- Model:
    - Machine Learning based model that takes a video as input and returns a video tagged with faces and emotions.
    - Using Computer vision to enclose and label faces with emotions.

Features

- Login/Signup
- Video uploader and downloader
- Video player/recorder
- Celery/Ray based task queueing to support asynchronous video uploading
- Model that supports distributed processing of multiple frames
- Processing progress bar
- Support for detecting multiple faces in the same frame

Timeline

20th - 23rd March	Decide on the project topic Discuss project requirements

24th - 25th March	Solidify project requirements Go through existing solutions
26th March	Discuss project overview Project abstract submission
27th - 30th March	Subteam-1 : - Build a web-app for uploading and playing videos. Subteam-2 : - Research on available annotated datasets - Think about requirements of the model
31st March	**Scrum meeting** for discussing progress
1st - 4th April	Subteam-1 : - Add Auth0 authentication (Login/signup) - Work on initial dashboard Subteam-2 : - Train the baseline model - Check the accuracy of the model
5th April	**Scrum meeting** for discussing progress
6th - 9th April	Subteam-1 : - Integrated Docker for easy deployment - Decide the database schema for the backend - Explore use of Ray to support parallel processing of frames within a video Subteam-2 : - Embed the model's emotion outputs on the initial video to generate the final output - Research about the use of celery to process multiple videos at a time - Test the improvement in processing time while using celery
7th April	**Submit Design Document**
10th April	**Scrum meeting** for discussing progress

11th April - 14th April	Subteam-1 : - Complete working on the dashboard meeting all the requirements - Set up the API requests to be used - Work on the mechanism to store/download the videos Subteam-2: - Incorporate model with the Docker-Flask application - Integrate Ray to support distributed processing of frames and test the improvement in processing time of one video
15th April	**Scrum meeting** for discussing progress
16th - 19th April	Subteam-1 : - Store and display user history - Work on a video library Subteam-2: - Test the entire application and work on any additional features (eg. record a live video)
20th April	**Final meeting and Code submission**
28th April	**Demo and Evaluation**