CSE 546 — Project Report

Debesh Mishra and Aakash Murari

General requirements:

- Strictly follow this template in terms of both contents and formatting.
- Submit it as part of your zip file on Canvas by the deadline.

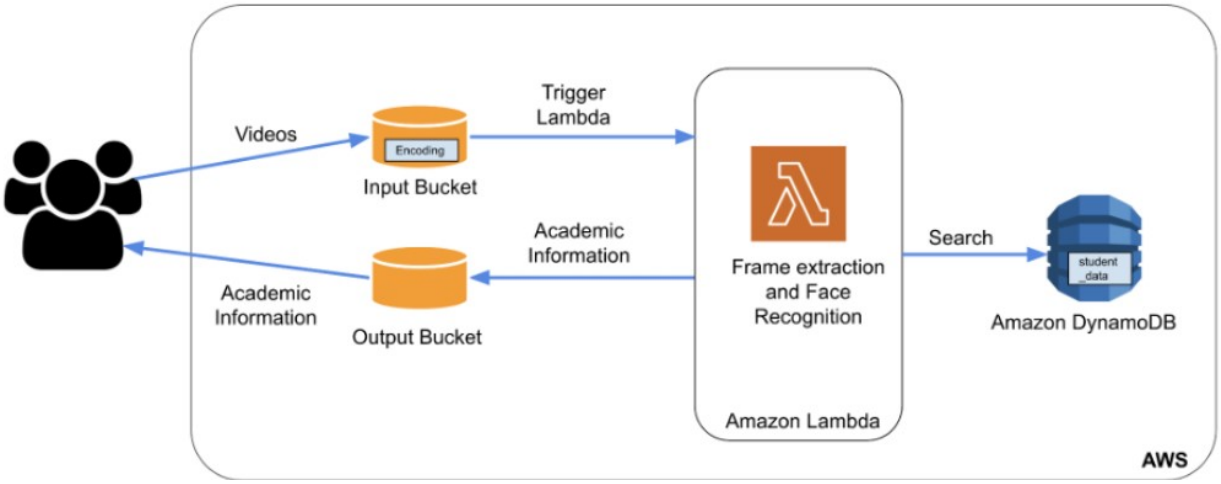
1. Problem statement

To provide a face recognition service by using the cloud resources to perform face extraction on the videos provided by the users. It is important because it allows the user to search something directly by the faces that appear in the videos and get the relevant results. For example - in case of preferring a particular instructor from various lecture videos, or watching documentaries on a particular person, etc.

2. Design and implementation

2.1 Architecture

We followed the similar architecture as provided on the assignment which is as follows:



1. Input Bucket: This receives the input directly from the users and stores those videos inside the bucket.
2. Output Bucket: This receives the input from the lambda function after the face recognition algorithm has been performed and stores the result in the bucket.
3. Amazon Lambda: This is a lambda function built-on a custom image which does the face recognition. This function has an S3 trigger with the input bucket, so whenever there is an input, this function will automatically trigger and send the output to the output bucket.
4. Amazon DynamoDB: Although this feature was optional for our group, we decided to implement it anyway. This a database we are maintaining with student data, the image recognition

performed by the lambda function will query the database and return the relevant results back to the lambda function which will in turn frame them in csv format to send to the output bucket.

2.2 Autoscaling

Since the application uses a lambda function to do the face recognition, AWS will automatically scale itself to handle the load. Nothing extra was implemented here.

We tested this by implementing a multithreaded workload generator by ourselves and the performance was significantly improved and the processing time for processing 100 images reduced to less than 2 minutes..

2.3 Member Tasks

1. Debesh Mishra
    - Created the initial code base along with the repository.
    - Created the resources on AWS such as a private repository in ECR and uploaded the dockerfile image to it. As well as the input and output buckets used for data transmission.
    - Created the main logic of getting the video file from the input S3 bucket and using ffmpeg to generate the amount of frames as well as printing the output of the face recognition that was generated from the first image.
2. Aakash Murari
    - Created the logic for reading student data from DynamoDb.
    - Created the application for bulk loading of data to DynamoDb.
    - Created a multi-threaded workload generator to upload multiple images to the input bucket in parallel for auto scaling.
    - Created the logic to upload the output csv data to the output S3 bucket.
    - Created the logic for iterating through the frames and stopping once a frame with a matching face is found.

3. Testing and evaluation

We performed load testing by sending 100 requests several times through the workload generator provided, and every time it finished the output within 5 minutes which is under the time limit provided.

As mentioned above, we also used our implementation of a multithreaded workload generator and the time taken to finish the output was within 2 minutes which is more than 50% faster than when we used the normal workload generator.

4. Code

Dockerfile is the implementation of the image which contains the needed libraries to perform facial recognition on aws such as boto3 and facial_recognition.

The **handler.py** is the main class where most of the logic is present. In this class we are first getting the input from the input bucket and then getting the key name and bucket name from the said input. After retrieving the video file, we call the ffmpeg library by using the following statement:

os.system("ffmpeg -i " + str(video_file_path) + " -r 1 " + str(path) + "image-%3d.jpeg")

The above statement creates frames out of the video which is provided and then we loop through these frames to find the image where a face first appears.

Then we call upon the face_recognition library to get the face encoding of the above frame and then compare it with the known faces that are present in the encoding file provided in the project.

After a match is found, the output is sent to S3 to be stored as a csv file and the program ends.

5. Running the application

To run the application, just run the python script for the workload generator which is provided in the project. However, to see the output, you would need to query the output S3 buckets using the credentials provided.