CSE 546 — Project2 Report

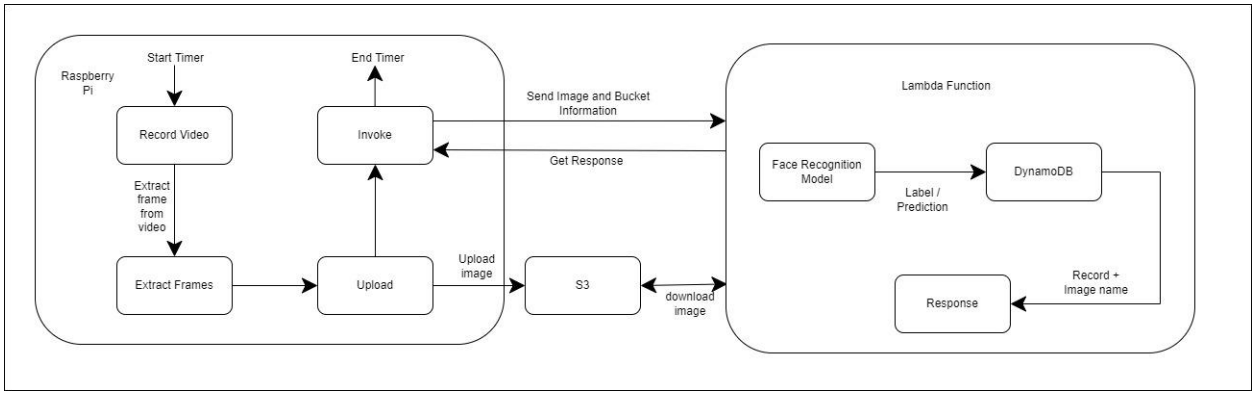
Sai Surya Kaushik Punyamurthula - 1220096111
Krisha Vijay Gala - 1222514124
Mrunalini Jitendra Dighe - 1219515557

1. Problem statement

To build a distributed application using PaaS services from Amazon Web Services and IOT devices which can perform face recognition in real-time from videos being recorded by the device in real-time. AWS Lambda is used for PaaS service and Raspberry Pi is used as an IOT development platform. This application records videos of people in Raspberry Pi and returns the resultant name of the person after performing face recognition by running a background thread. This problem is important since it addresses combining IOT and cloud resources which are two of the most in-demand industry resources. This problem is very similar to real-world problems where IOT and cloud-based applications are used.

2. Design and implementation

2.1 Architecture



2.1.1 Architectural Summary

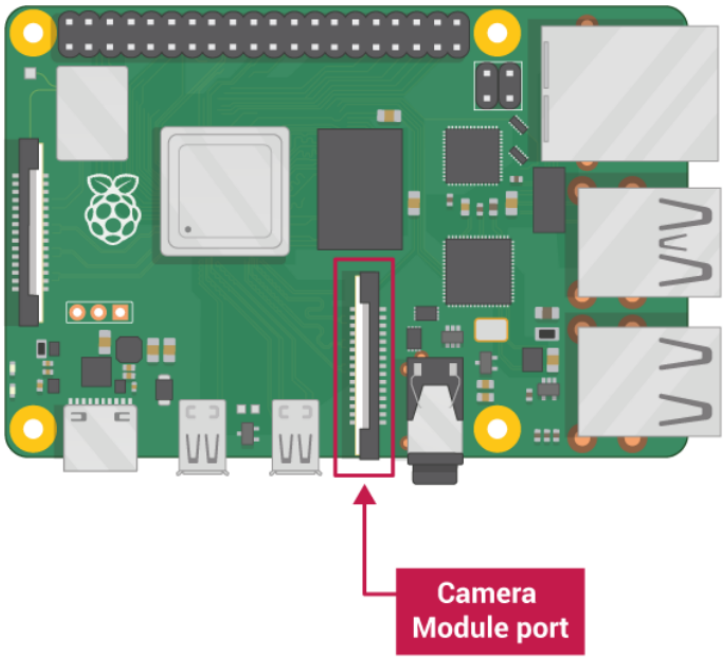
We have implemented a loosely coupled architecture where various components efficiently serve their purpose. To begin with we have the Raspberry PI module which is primarily responsible for capturing the images and videos. Pi runs on the concept of multithreading wherein different threads perform various tasks. There are 4 threads in this application and they perform 1 task each like recording of videos, extracting frames from these videos, uploading the extracted frames to S3 bucket and invoking the Lambda functions. Multiple iterations take place once the application is started. The first iteration records a 0.5 second long video, the second iteration extracts the frames from the first video and records a second video. In the third iteration it uploads the frames for the first video to the S3 bucket, extracts frames for the second video and records the third video. In the fourth iteration it invokes the lambda function on the frames of the first video to perform face recognition, uploads frames of the second video to the S3 bucket, extracts frames for the third video and records the fourth video. This keeps going on until all the processing is done.

Lambda function plays the next major role in this application. The lambda function contains the docker container which runs the face recognition model. So, when the lambda function is invoked, it starts the container which performs face recognition and it helps in predicting and labeling the image. Thereafter the image with specified label predicted is cross-checked with the

results residing in the DynamoDB. The other image details such as the graduation year, major, image name and person's name are then sent as the response back to the request.

2.1.2 Raspberry Pi (IoT device)

We are making use of raspberry pi with a camera module to capture the images. The images and videos are captured via the Picam which can then be used in later stages of processing. We also attach an additional mouse and keyboard and a monitor for efficient usage of picam while running the program.Images are typically saved in Pi using the command "raspistill -o path_to_imagestorage/image.jpg". Similarly we can capture videos using the command "raspivid -o path_to_videostorage /video.h264". We are operating the PiCamera via the python code itself.



2.1.3 AWS Components

1. **Simple Storage Service (S3)**

   S3 is a storage service provided by AWS which is used to store the frames which are extracted from the videos in buckets.

2. **Lambda**

   Lambda is a function which provides a function as a service. It is a computing service which helps create functions to apply computation to data in the cloud. It is used for classifying images and fetching the students' information from DynamoDB based on the classification.

3. **DynamoDB**

   DynamoDB is a database system which enables key-value storage. It is used to store student information such that academic information of any student can be retrieved by using the student's name.

2.2 Concurrency and Latency

Concurrency: To implement concurrency we made use of multithreading. In the file Edge.py there is a code for running threads in parallel. The threads are so fashioned that when one thread is working on the Pi to capture the image, other works of frame extraction. While this is happening , the 3rd thread will work on uploading the images of the S3 bucket and the 4th thread can work on invoking the lambda function in a very short time of 0.5 seconds.

Latency: Latency refers to the time it takes to complete the process of taking the video as input for facial recognition and the time it takes to identify it correctly with all the detailed parameters enlisted. The latency is highly dependent on the concept of multithreading. By making use of multithreading we can parallely compute different operations like capturing image, frame extraction, uploading to S3 bucket and invoking lambda functionality to perform face recognition. The latency was further tried to be tweaked by using multiple and single lambda functions. The best results in our case were obtained when we used a single lambda function.

3. Testing and evaluation

Testing the lambda function :

We tested that the lambda function is publicly available and returns proper results.For that we installed postman application to hit the lambda function. We provided a json body to the request . The json request had the following data {

"ImageName" :"frame-2022-05-03 13:59:46.507871.jpg",

"BucketName" : "cc-2-37-bucket"

}

The results we got included the details about the image -

{

"Graduation_Year": "2023",

"Major": "Computer Science",

"Image_Name": "frame-2022-05-03 13:59:46.507871.jpg",

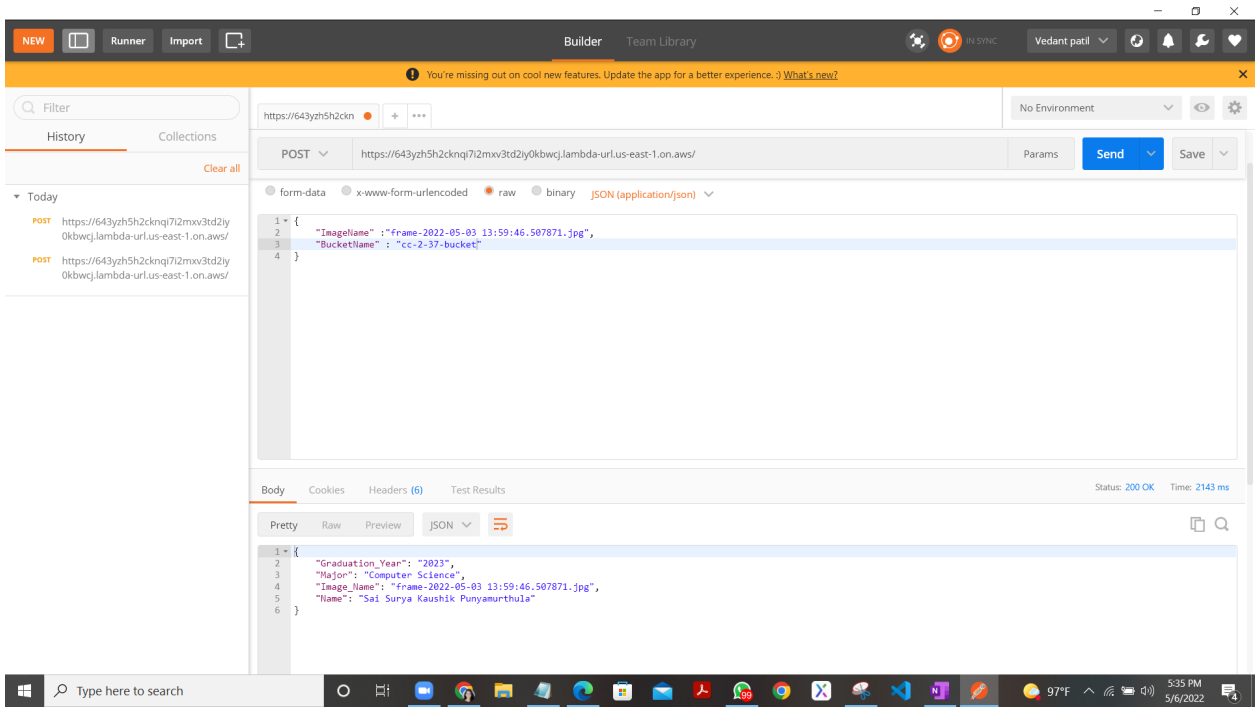
"Name": "Sai Surya Kaushik Punyamurthula"

}

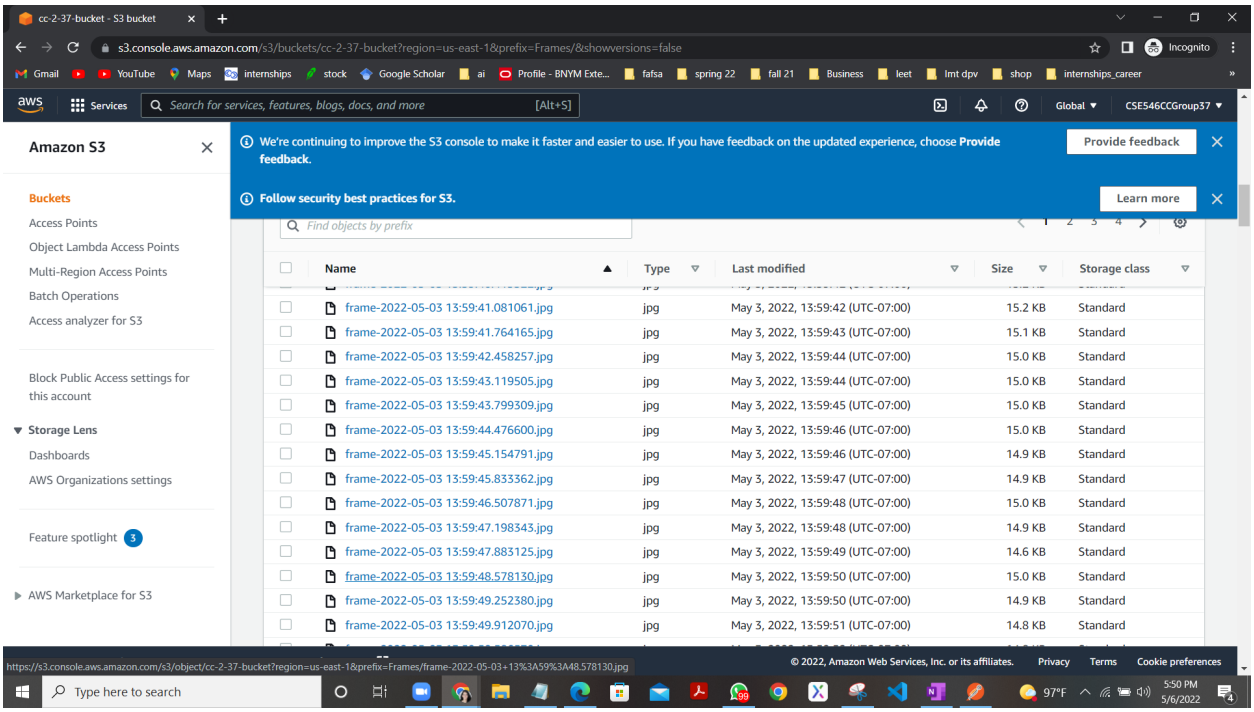
We have tested the lambda functions with different configurations like memory, timeout and other parameters that affect the performance of lambda. The execution is faster with larger memory size and the timeout has to be changed to accommodate the cold start of the lambda when the model is first downloaded and loaded. By trying out various values of the parameters we have up with possibly an ideal solution

We have also tried the lambda invocation using various methods like configuring a trigger on S3 file upload. Lambda S3 triggers configure the prefix and suffix of the files to restrict which images/videos will invoke lambda and which will be disregarded. Another approach is to use the API Gateway to configure an API call to invoke the lambda function. This same functionality has been newly introduced as a direct feature of lambda called Function URL. We can configure a fixed URL for a lambda function which can then be used to make HTTP Requests to invoke the lambda. It is also possible to make the URL accessible publicly or privately by restricting the access to the URL based on various IAM resources like IAM Users and IAM Roles. We have configured the function to be publicly available for this project as we faced a few issues with restricted access which couldn’t be mitigated in time. Using the function URL approach has helped decrease the latency by a decent amount.

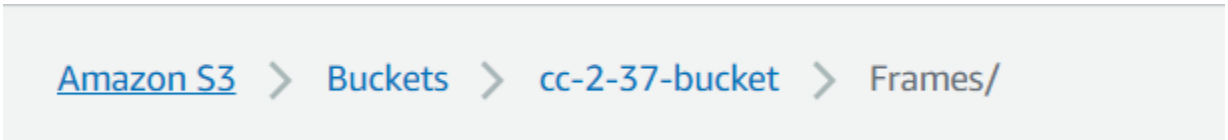
Below is a screenshot of the same in postman for better visibility.



Test case : Testing if images frames are getting saved to S3 bucket correctly



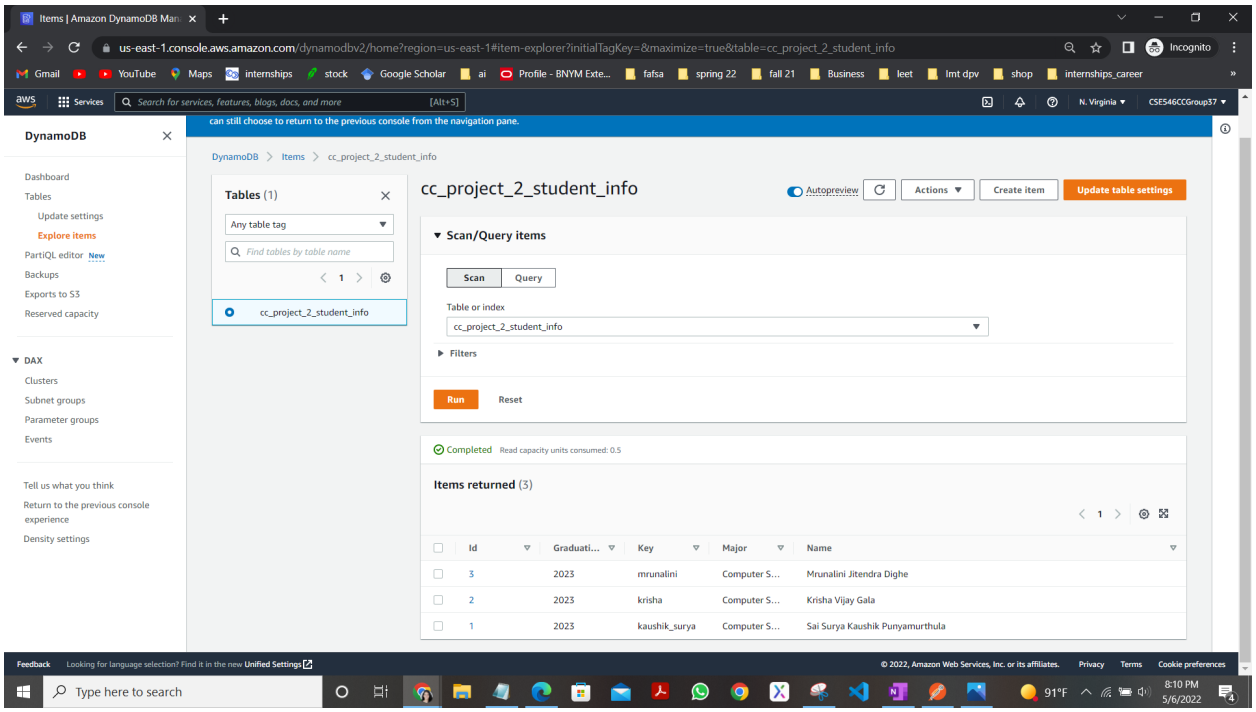
The frames are stored at a location inside of the S3 bucket -



Test case : Testing if the S3 bucket is accessible publicly.

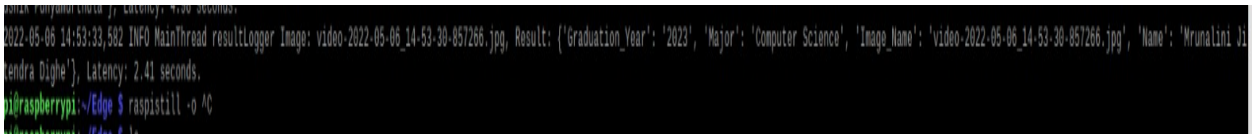
No, the S3 bucket is not publicly accessible

Test Case : Check if DynamoDb is storing the information properly.



The above screenshot tells us that the information is stored correctly into the DynamoDb with correct Graduation year, names, keys and major.

Test Case : Checking the results for the necessary details like graduation year, major, name



The results included all the necessary details like graduation year,major,name.

Test Case : Calculating the latency
The latency was calculated by noting the time difference between the start of the video capture till the time the lambda response was received.

4. Code

Installations:

We need to install Raspbian on the Raspberry Pi. Reference:https://docs.google.com/document/d/1MaHuP5qyA29oy2vhYwPEdCB6vR5qJtOu5nQp2tVIJaM/edit

The Raspbian image is flashed onto the microsd card and plugged into the Raspberry PI. In this way the Pi is ready to boot up. We access the pi for the first time by using the HDMI cable and thereafter we use the ssh functionality to connect to the monitor.

Functions:

Handler.py*:* This script is responsible for deciding what should happen when the lambda function is invoked. Handler.py calls the eval_face_recognition function and sends the result as the response. The frames which are in S3 are downloaded to the docker container with bucket information and image information such as from where the image is being downloaded and what image is being downloaded. This information is given as an event as the REST API call has information from this event as the input. Image name is sent from this function to DynamoDB with the name of the person as the key and it returns the values such as graduation year, major and image name and person's name.

Eval_face_recognition.py*:* In this script, face recognition is performed. The path to the image is provided and the face recognition is done based on the training data.

Edge.py*:* The raspberry pi uses 4 threads to manage the functionality of recording the videos, extracting the frames, uploading it to S3 bucket and invoking the lambda function on it. The code works through 4 iterations.For instance, consider in the 1st iteration the video is recorded. In the 2nd iteration again the video is recorded and frames are extracted from video 1. In the 3rd iteration the video is recorded, the frames from the 2nd iteration are extracted and frames from the 1st iteration are uploaded. In the 4th iteration the video is recorded, the frames are extracted from the 3rd iteration, frames from the 2nd iteration are uploaded and the lambda function is invoked on the first iteration's video.

We are using OpenCV to read the frames from the videos captured. We are using BOTO3 to upload videos to S3 bucket. We need to use AWs credentials to access the lambda functions. The Boto3 library is used to upload to the S3 bucket and similarly download from Dynamodb. The code also specifies the path where the videos need to be uploaded.

getFaceRecognitionResult(queue, framePath): This function is used to run the face recognition model on the lambda function. It takes in input the queue and framepath. It also has the payload of the bucketname and image name which is sent as a request payload to the lambda function.The BucketName specifies the path from where it is downloaded. While the Imagename tells what image is getting downloaded.

We also decide on the latency by subtracting the start time from the end time when the results are rendered. The code is as follows :

```
latency = endTime - videoMap[imageName]['startTime']
```

We use a video map for keeping track of the start and end time for the videos.

Requirement.txt*:* This file contains all the necessary libraries that we need to download for the code functionality.

Configuration.properties*:* This file contains the configurations for S3 bucket, SQS queue and Lambda function's url.

Individual Contribution

Mrunalini Jitendra Dighe - 1219515557

This project involves edge computing and cloud computing as the underlying concepts. We use raspberry pi to capture videos of human subjects which are then processed into frames. A predefined face recognition model is used to evaluate the images and predict the person. We then get the details of the person stored in DynamoDB to get related information which is then sent back to raspberry pi to view.

I was involved in various implementations in raspberry pi which is an IoT (Internet of Things) device. We use multithreading to implement the various functionalities of raspberry pi in parallel. There four major functionalities of raspberry pi which are as follows:

1. Recording Videos: I used the python picamera module to access the camera that is attached to the raspberry pi on the camera port. I customized the picamera to record the videos in h264 format which is an encoded format and hence takes very little space.
2. Frame Extraction: The videos are then extracted into frames using the OpenCV-Python module. The OpenCV module enables us to open the captured video using the VideoCapture functionality and read every single frame of the video. I then resize the frame into the dimensions required by the Face Recognition Model which is (160x160). I also convert the image from RGBA to RGB format which is the acceptable format to the model.
3. Uploading Frames: The extracted frames are now uploaded to an S3 bucket, which is a scalable storage service offered by AWS.
4. Lambda Invocation: Lambda function is invoked using the customized Lambda function url as REST API call. The lambda function requires the image name and bucket name to be able to download the image from the S3 bucket. These details are sent as request payload in the REST API call. We use multiple threads for executing lambda invocation as the implementation time of lambda functionalities are higher than the other threads and would not be completed by the time an iteration is completed in the main working thread and stop each of the threads only on successfully completing the process.

These threads are created and started in a sequential manner so that we can achieve maximum parallel computing possible. Frame extraction threads are only created after successfully recording a video. To ensure this we first run an iteration with just the recording thread. The second iteration meanwhile will record a new video and process the first video into frames. Similarly in the third iteration the first recorded video's corresponding frame is uploaded to the S3 in addition to processing and recording. From the fourth iteration, the four threads will record, extract, upload and invoke lambda in the latest result from the previous stage execution in parallel. I used various thread-safe data structures to be able to get the results from the various threads. This is important for the sequential run and enables us to process the right data in each of the subsequent threads. It should also be noted that when the time duration for which the whole process should run is met, the recording of the videos is stopped, however there are intermediary results from each of the threads that are left to be processed further. I have configured the main loop in such a way that the iterations can continue for the individual threads to complete execution on all the inputs generated by the previous stage.

Time is noted before a video is recorded and stored into a dictionary of video file names and their respective process start and end time. The time is then recorded after the response from the lambda function REST API call is received. I now calculate the latency of the end-to-end process by finding the difference between the start and end time of the respective video which is displayed in the PI along with the face recognition information received as part of the lambda response. I was also actively involved in DynamoDB items creation, Dockerfile setup to make sure the end-to-end requirements were met and designing and documentation of the architecture of the project.

Individual Contribution

Sai Surya Kaushik Punyamurthula - 1220096111

I was actively involved in various stages of the project development. I was keen on understanding the project requirement and then devising a strategy to effectively manage the work . One of the main components that I developed and configured was the “lambda” function. The lambda function lies at the core functionality of the project as it was used to download the images from the S3 bucket. It also had the code logic for evaluating the image . It also has the logic to fetch the records from DynamoDB which stores information of the human subjects upon which the face recognition model is pretrained. The various attributes of the items in the DynamoDB table are name, graduation year, key and major.

For this to achieve I divided the work into segments and worked on them. By referring to the resources provided by the professor and TAs, along with the detailed documentation helped while facing any issues. The docker needs to be installed on the local machine. In the case of Windows we download software called ‘Docker Desktop’ which enables us to create and run docker images.While installing docker I configured the wsl settings and the proxies on the local to make sure we had access to the images created. Docker in its capacity can create containers from predefined images. In this project a background thread is run to invoke a lambda function. Lambda functions can be of different types based on the underlying platform it provides for instance Python, Node JS, and Docker containers etc. I implemented different lambdas, one for face recognition model evaluation and another for fetching records from DynamoDB based on the evaluation result. And later decided to opt for a single lambda to decrease the communication time between the two lambda functions. As the lambda function is provisioned to be able to run a docker container with complicated computation which would in general not be executed as part of a single script is now computable. A docker image is created which has the pre-trained model and evaluation script. The image is uploaded to the ECR repository and then this image in the ECR repository is linked to lambda. A lambda container is created for every lambda instance that is created based on the docker image linked to it. The docker image lambda handler is responsible for getting image and bucket information from the event and downloading the image from the S3 bucket. The face recognition model evaluates the downloaded image to classify the human subject into the predefined labels. The result is stored in the DynamoDB as a key along with other information about the person. These details are fetched from the DynamoDB which will then be the response to the REST API call that has invoked the lambda function.

The detailed setup instructions were provided in the docker image which could be further used to create containers using the Dockerfile. The docker can be run in two ways: either the single stage setup or the multi stage setup. In our case, after discussing with my team, we decided on going with the multi step setup. Having a multistage setup helped in that artifacts from one image could be selectively used in different stages and thus saving much on the time. This also allowed the use of the same images in multiple steps . The main advantage of this was cutting down the time and easing the complexity of the work. The entry points were also clearly defined which usually is execution of a file/script which includes all the instructions to be performed when a docker container is created. Lambda function was integrated with the docker such that the image built on the instructions from docker file was pushed to the ECR. Docker is a very feasible way to be able to use a platform or machine which has the prerequisites taken care of and be executed almost instantaneously. In this, I got a very good chance to brush up the basics of the AWS lambda function and at the very same time implement those skills to bring a project to realization. My future goals align to this domain and this project has been of incremental importance to me. Not only did it make me grow with the technicalities but the team work and experience laid the groundwork for opportunities in the tech world. The experience with various cloud resources like Lambda functions, DynamoDB, S3 and other resources will enable me to perform better as a developer when I am responsible to execute any operations as part of DevOps.

Individual Contribution

Krisha Vijay Gala - 1222514124

I was involved in the design of the application, implementation, training and fine-tuning of the model and end-to-end testing as well.

Firstly, I was involved in the environmental setup of the Raspberry Pi, creating a lambda function on AWS, creating S3 bucket and also DynamoDB. I then contributed in deciding the end-to-end flow of the applications and what kind of functions would be required to record videos, extract frames, upload the frames to S3, apply face recognition, fetch the result from the database on the basis of recognition and provide the response.

I worked on preloading data into DynamoDB in the form of key-value pairs such that the key is the name of the person and the value contains information such as graduation year, major and the image name. I also worked on connecting DynamoDB with the handler function which is responsible for deciding what happens when the lambda function is invoked. This function is also made to call the face recognition functions. After face recognition is performed I fetched the results from the database to display it as a result.

I performed training on the model by performing a lot of trial and error. I created the code for recording videos and performing frame extraction of those videos. First, I performed extraction of the frames from videos in multiple ways. One way was by extracting it in cloud and the other was by extracting it directly in Raspberry Pi and then sending the frames to cloud for face recognition. I recorded various types of videos and images for training the model and made changes accordingly. I trained the model using images taken from different cameras and then trained it using images solely captured from the Raspberry Pi camera. I was involved in the setup and fixing of the camera in Raspberry Pi as that caused a lot of detection issues.

I performed end to end testing of the application from start to finish which involved recording a video, extracting the frames from the video, integrating the pi with the lambda function and helping in the creation of the docker container with the pre-trained model and evaluation script. I ensured that the flow from start to end is working without hurdles by making multiple changes throughout the testing of the model.

During the initial testing the latency and accuracy were very poor. To improve the latency and enable concurrency I changed the number of threads I was using. I increased the number of threads to four such that the first thread would record the video, the second thread would extract the frames, the third thread would upload the frames to the S3 bucket and the fourth thread would invoke the lambda function to perform face recognition. I also changed the number of lambda functions by increasing the number of lambda functions and also decreasing it. First I tried using only a single lambda function which would perform face recognition, fetch the label from DynamoDB and give out the response. I also tried using multiple lambda functions such that one function would perform face recognition and other functions would be responsible for fetching the person’s name from the database and providing the response as the result.

To improve accuracy, instead of taking single images of a person as training data, I recorded videos in different settings and then extracted frames from it to make the process faster and more efficient. I ended up improving the latency by a high margin and ensured smooth working of the application without any errors. I was also actively in the report making and creating the architectural flow.