Cloud Computing(CS6847) Course Project

Face Recognition using Fog plus Cloud Computing Report

By:

Gaurav Arora(CS18M021)
Kumar Vaibhav(CS18M030)
Niketan Shahapur(CS18M036)

Submitted to:

Dr. D. Janakiram

Abstract - **Raspberry pi acts as a fog layer as well as end device in the process of face recognition using fog plus cloud computing. As an end device Rpi captures the frame using webcam and as a fog layer Rpi preprocesses the frame data such as gray scaling, auto scaling and keeping in mind its low power Rpi uses MQTT for data transportation. The main idea is to send the frame data to the cloud for face recognition through fog layer, where all the preprocessing steps are done in the Rpi on the fly before reaching to the cloud. The design can be modified to achieve three different best effort parameters. First, capture the frame through webcam on Rpi and send the data to the cloud where it detects the face and recognises it, in this way we achieve face recognition with minimal latency. Second, capture the frame through webcam, detect the face and send the face detected frame data to the cloud for face recognition, by designing this we achieve face recognition with minimal network bandwidth overhead. Third, capture the frame through webcam on Rpi, preprocess it and send the preprocessed data to the cloud where it detects the face and recognises it, in this case we achieve balanced network bandwidth overhead and latency.**

I. INTRODUCTION

Raspberry Pi is a mini computer with microprocessor, which are used widely in IOT networks, these are reliable, low energy, and affordable devices which provides freedom in programming hence are popular. This project aims to achieve capturing live stream of frames through webcam and streaming of frame data to cloud computing using Raspberry Pi for further frame data processing and face detection keeping in mind about the low energy supply to the IOT network.

II. LITERATURE SURVEY

The initial plan of using laptop as a fog layer was dropped due to high configuration and performance of laptop. Also laptop is not feasible to fix in any place where webcam surveillance is required for security purpose. The major drawback of using laptop is it consumes more power as compare to Rpi also its battery lifetime is also very less. Laptop costs very high so its not affordable to place anywhere which contains webcam for security purpose. Raspberry Pi and/or its kit are easily available in many online retailers and offline shops. Also it costs roughly 3,000/- INR it is affordable for project purpose. Webcam was bought from amazon, since the shipment was guaranteed within 24hrs [1][2].

Raspberry Pi is popular and affordable, hence many online tutorials to setup and use Raspberry Pi with webcam [3]. We referred one of the tutorial available in the website, which shows the way webcam should be connected to Raspberry Pi and how to read data from webcam using motion libraries. Before we move on to design and development part, we had to know more about AWS EC2 instance, as we have to design the streaming and recognition which should be supported by EC2 instance. AWS provided us some useful information. This website gave us brief idea about EC2 instance of type t2X large[9] its architecture and working. Initially we tried TCP connection between Raspberry Pi and Cloud server, soon on realization of energy consumption in TCP protocol, we switched to MQTT protocol using Mosquitto Broker [4], we referred several resource and chosen an example from github to setup and use the MQTT protocol [5].

III. DESIGN

The webcam captures the video frames continuously. These video frames are then converted from RGB to grayscale image, before sending these video frames to the cloud we first check the changes in the current frame with respect to the previous frame and if there are some movements in the frame then only we will send that frame to the cloud for face detection and recognition. We are only sending the selected frames to the cloud and this will save additional network overhead. We are using opencv library for image, video processing and detection of positions where faces are present on the frame. To send the frames to the cloud we are first converting the two dimensional numpy array frame to one dimensional numpy array and then converting that array to string, then we are using MQTT(message queuing telemetry transport) to send these strings to the cloud. On the cloud side the face recognition algorithm will use the received video frames to detect and recognise the faces present in the frame data and finally send the result back to the raspberry pi using MQTT broker.

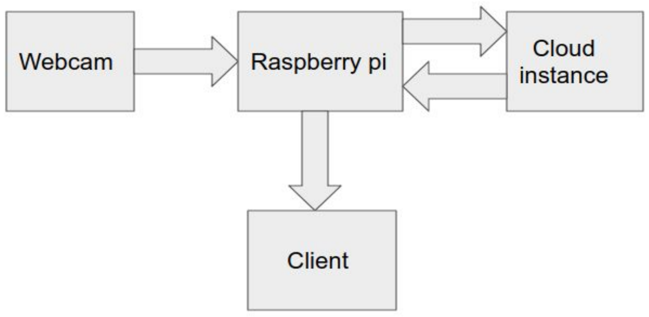

Fig.1 Architecture

III. IMPLEMENTATION

First task was to install proper well supported Operating System into Rpi. We initially chose Raspbian to be installed in it, using Raspbian Noob application we installed Raspbian into it [6]. But Raspbian OS did not support installation of opencv due to Rpi architecture so we searched other OS which should support installation of opencv also it should be compatible with Rpi architecture. We found Ubuntu Mate and rebuilt the opencv on top of it from scratch. Then we installed motion library for reading frame data from webcam. This motion supports both frame wise and live streaming of frames.

Started working with face detection on Rpi i.e. capture the frame data then preprocess it and send it to the cloud for face recognition using TCP connection component. TCP connection component contains two modules. First, client module where it creates a socket of type stream including cloud IP address and port number, using this it connect to the cloud and whenever the data frame is ready it send to the cloud and also it receives result from the cloud. Second, server module where it creates a socket with only port number and keep it in a listen state, when a client request comes it accept it and receive data frame it. It also send result to the Rpi.

Later on suggestion from one of the CS6847's TA, the TCP/IP connection is replaced with MQTT as the TCP connection consumes more power. Where as MQTT is a machine-to-machine (M2M)/"Internet of Things" connectivity protocol. It was designed as an extremely lightweight publish/subscribe messaging transport. It is useful for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium.

MQTT has two modules. First, client module where it subscribes for a topic say up and publishes for a topic say down. Where in publish topic it pushes the frame data from Rpi side and in subscribe topic it pulls the result data. Second, server module where it subscribes for a topic say down and publishes for a topic say up. Where in subscribe topic it pulls data frame which is pushed by client and in publish topic it pushes result data from server side.

Face detection is taken care of by pre-trained HAAR cascades available in opencv [7]. After we get the bounding boxes for detected faces, each face image is transformed into a 128-d feature vector by dlib facial recognition network [8]. All of the faces in training dataset are also encoded in this way. For a test image euclidean distance is calculated between the image encoding and all labeled training face encodings. For each training face encoding a boolean value (T/F) is generated based on its distance from test image encoding. Among all training labels, the label having maximum true values is determined to be the test image label.

To determine the effects of fog computing on the task three implementations were tested:

- No work on device apart from image capture(Impl. 1).
- Image preprocessing on device before sending to cloud(Impl. 2).
- Image preprocessing and face detection on device before sending to cloud(Impl. 3).

IV. FUTURE SCOPE

The project is designed in such a way that, it can be extended to support multiple fog layers where each fog layer contain a webcam to capture the frame and Rpi can be placed anywhere in the campus/office area, in this way the surveillance can be easily extended from a single room to entire campus. Where each fog layer work irrespective of other fog layers so there is no need of change in our design. At cloud side there is a small change i.e. instead of receiving a single request from fog layer it should be designed to accept multiple requests. Also enable auto scaling and load balance at cloud side to cope up with burst traffic to work efficiently.

V. INDIVIDUAL CONTRIBUTION

We all three worked closely to make the project work properly. Even though all the team members were together during entire process of the developing the project, some individual contribution were made which were unique and are mentioned below.

Gaurav Arora & Kumar Vaibhav :

- Installation of openCV on Raspberry Pi. As we can't directly install the openCV on Raspberry Pi using pip because it has different architecture. So to install openCV on Pi we have to rebuild its binary from its source code.
- Researching various methods of face detection and face recognition, using one of classic methods due to better performance and low computation required as compared to neural networks and non requirement of pre-training or dataset.

Gaurav Arora :

- Implemented face detection python program using opencv.
- Performed some preprocessing like color to grayscale on the Raspberry pi.
- I have used a pretrain face features in the form of a XML file for best results in Face Detection.

Kumar Vaibhav :

- Implemented python program to encode training faces to create training dataset.

- Implemented a kNN based recognition technique for face recognition.
- Wrote several implementations of distributed face recognition (Mentioned in implementation section) between cloud and device.

Niketan Shahapur :

- To get familiar with the Raspberry Pi(RPi) I explored the documentation and tutorials for the same and set up the Raspberry Pi with required configuration.
- For the lightweight OS other standard OS drivers won't fit and also additional drivers were required such as motion driver to run webcam, it was tedious task to set the RPi with proper hardware drivers due to mismatch of RPi architecture with standard OS drivers.
- Implemented TCP/IP client/server socket program at Raspberry Pi/cloud such that it should collect the captured frame/resultant frame data in it's specified format and deliver the data reliably with zero loss at the receiver/sender side without altering the frame data format so that the cloud/RPi side program should not face inconsistency.
- Later on to reduce the RPi's energy consumption due to TCP/IP communication protocol, I have explored other IOT protocols such as MQTT, CoAP, AMQP, Websocket and Node. I found MQTT broker to support RPi's less energy requirement and very efficient. Due to RPi's unique architecture I have installed MQTT from scratch by rebuilding MQTT from its source code on ubuntu MATE OS.
- Implemented MQTT publish/subscribe program at Rpi/cloud such that it should consume less energy and provides same communication service as of TCP/IP protocol.

VI. CONCLUSION

A. No work on device apart from image capture

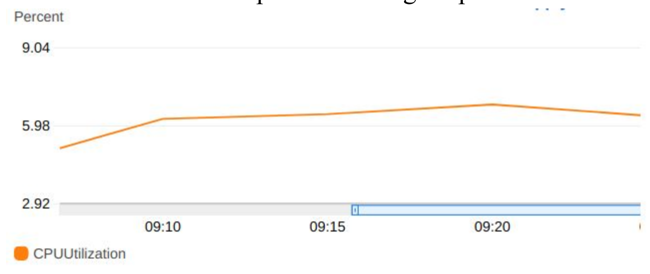

Fig.2 CPU utilization

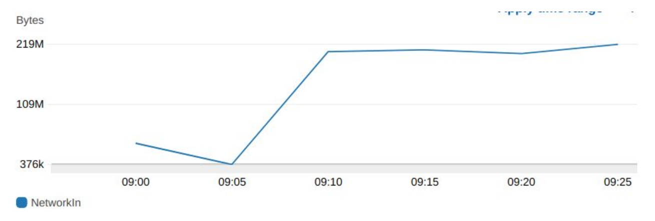

Fig.3 Network utilization

An average cpu utilization of around 6% is noticed in this implementation with around 40 MB/min bandwidth utilization.

B. Image preprocessing on device before sending to cloud

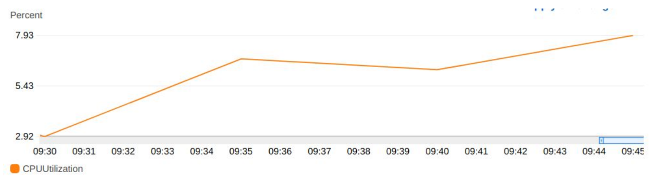

Fig.4 CPU utilization

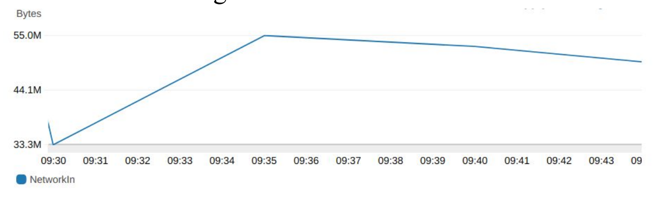

Fig.5 Network utilization

CPU utilization is again around 6-7 %. But there is a significant decrease in bandwidth utilization which turns out to be around 11 MB/min. This can be attributed to rescaling of image on device before transfer to cloud.

C. Image preprocessing and face detection on device before sending to cloud.

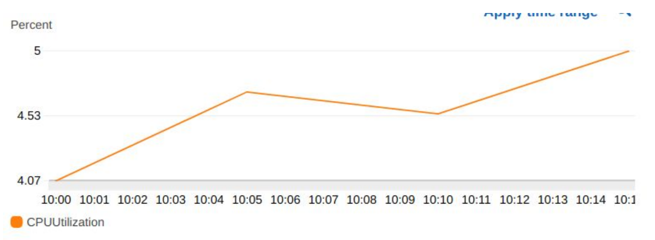

Fig.6 CPU utilization

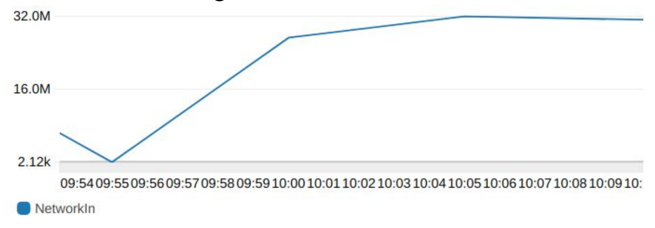

Fig.7 Network utilization

Here a lower CPU utilization is observed (~ 4-5 %) due to face detection being offloaded to device. Furthermore the device is observed to be not being able to handle face detection efficiently. Due to this the rate at which the device sends the frames to cloud along with the detected faces is lower than previous two cases. This is a case of offloading too much computation to the device. An even lower bandwidth utilization id observed ~6MB/min, due to device processing bottleneck.

Below table mentions the frames per second recognition performance achieved on the device for the three approaches to distributing computation.

	Impl. 1	Impl. 2	Impl. 3
FPS	0.96	1.38	0.93

Fig.8 Comparison of three designs

The above table shows that offloading some work from cloud to device certainly shows performance improvement for the task. But if too much work is offloaded to device the device might become a bottleneck to cloud performance and no improvement could be gained.

VII. ACKNOWLEDGMENT

We wish to acknowledge the contributions of Professor D. Janakiram, CSE, IIT Madras for his constant guidance and support. We would like to acknowledge Mr. Akshay Dhumal, MS Scholar, CSE, IIT Madras for her continuous support, motivation and valuable discussions and suggestions. We would also like to acknowledge Mr. Ravichandran S, Technical Officer, CSE, IIT Madras for his continuous support, suggestions, motivation and providing lab infrastructure.

VIII. REFERENCE

[1]
https://www.amazon.in/gp/product/B0725Z9JSV/ref=oh_aui_detailpage_o01_s00?ie=UTF8&psc=1

[2]
https://www.amazon.in/Quantum-QHM495LM-25MP-Web-Camera/dp/B00L5AWKUM/ref=sr_1_5?keywords=webcam&qid=1556816797&s=gateway&sr=8-5

[3]
https://www.instructables.com/id/How-to-Make-Raspberry-Pi-Webcam-Server-and-Stream-/

[4]
https://mosquitto.org/

[5]
https://github.com/roppert/mosquitto-python-example/blob/master/readme.md

[6]
https://www.raspberrypi.org/downloads/noobs/

[7]
https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html

[8]
https://www.superdatascience.com/blogs/opencv-face-recognition

[9]
https://aws.amazon.com/ec2/instance-types/