2019 International Conference on Deep Learning and Machine Learning in Emerging Applications (Deep-ML)

A Deep Learning Based Distributed Smart Surveillance Architecture using Edge and Cloud Computing

[1]Halil Can Kaşkavalcı
Dept. of Computer Engineering
Yeditepe University
Istanbul, Turkey
halil@kaskavalci.com

[2]Sezer Gören
Dept. of Computer Engineering
Yeditepe University
Istanbul, Turkey
sgoren@cse.yeditepe.edu.tr

***Abstract*—Smart surveillance is getting increasingly popular as technologies become easier to use and cheaper. Traditional surveillance records video footage to a storage device continuously. However, this generates enormous amount of data and reduces the life of the hard drive. Newer devices with Internet connection save footage to the Cloud. This feature comes with bandwidth requirements and extra Cloud costs. In this paper, we propose a deep learning based, distributed, and scalable surveillance architecture using Edge and Cloud computing. Our design reduces both the bandwidth and as well as the Cloud costs significantly by processing footage prior sending to the Cloud.**

***Index Terms*—Surveillance, Edge computing, Cloud computing, Internet of things, Deep learning, Smart gateway**

I. Introduction

Internet of Things (IoT) is a reality now. Many devices come with Internet connection nowadays and security cameras are no exception. IoT enabled surveillance promises ease of mind to people and is also followed closely by the law enforcement. These devices can record their footage directly to the Cloud without suffering loss in case of a damage. Smart indoor security cameras such as Nest IQ [1] offer facial recognition. If an unfamiliar face is captured software alerts owners.

However, those features do not come for free. Using Cloud for continuous storage is expensive. Nest charges $300 a year for 30 day footage history per camera and facial recognition features. This also requires reliable Internet connection with good upstream bandwidth. Most Internet Service Providers (ISP) offer reduces upstream speed to residential networks. This could be more expensive if network has bandwidth quotas.

In this article, we propose an architecture that utilizes Edge Computing to perform facial recognition and use Cloud and Internet connection only when necessary. We have designed our system so that it can scale, tolerate failures, and work with multiple devices and cameras. We containerized all of our system into Docker [2] images for easier and replicable deployment. Our aim is to reduce upstream bandwidth by processing video footage on the edge and use Cloud as a backup mechanism.

A deep learning approach is chosen to recognize faces. Davis King's dlib [3] has slightly modified model [4] that is based on ResNet-34 [5]. This modified version which is trained against 3 million images is used for face recognition on the edge. Amazon Web Services (AWS) is the choice of Cloud and Rekognition [6] is used for Cloud processing. Cluster of Raspberry Pi's are used as Edge Server.

This article is structured as follows. In Sect. II we give a brief overview of current literature. In Sect. III, we explain the technologies and libraries used in this work. In Sect. IV, we explain our proposed design. In Sect. V, we introduce our experimental setup. In Sect. VI, we present results of our experiments. Finally, in Sect. VII, we conclude our work.

II. Related work

Park et al. created an architecture with three layers: Device Layer, Edge Layer, and Server Layer [7]. Cameras, sensors and actuators belong to the Device Layer and connect to the nearest edge node. In Edge Layer, an edge server pre-processes the device data and feeds it to the Server Layer when it detects an intrusion or abnormal event. Server layer controls edge nodes and can request device data at any time. Raspberry Pi (RPI) is used as Edge Layer and it only performs face detection. When face is detected, it is sent to Server Layer. Authors used Dell T630 tower server as Server Layer and used Eigenfaces [8] for face recognition.

Popic, in his master thesis [9], compared Cloud based facial recognition system (AWS Rekognition) and facial recognition with IoT (Raspberry Pi). Advantages and disadvantages of both systems are documented. A Python library [10] based on dlib face recognition module is used on Raspberry Pi. He concluded that local processing on Raspberry Pi is immensely faster than processing on the Cloud, due to network overhead. Accuracy rate of both systems are comparable, although AWS Rekognition is slightly more accurate. On contrast to local approach, Cloud approach does not require maintenance and adding new faces to database is relatively easy. In the end, both systems are worthy of the task and should be used whenever necessary.

978-1-7281-2914-3/19/$31.00 ©2019 IEEE
DOI 10.1109/Deep-ML.2019.00009

Wazwaz et al. created a face detection and recognition system which utilizes a RPI and a cluster of servers [11]. A camera is connected to RPI physically and RPI streams live video from the camera. Boosted Cascade of Simple Features Algorithm (Haar Feature-based Cascade Classifiers [12]) is used to detect faces on RPI. Aim of their research is to optimize face detection and recognition speed. They run extended tests to fine tune algorithm variables to achieve maximum face detection rate. Detected faces are sent to cluster of servers for face recognition phase. They used one to three servers and compared the duration of processing per each frame. Local Binary Pattern (LBP) algorithm is used for face recognition. Each known face is processed and LBP values are generated before system initialization. Duration of recognition process per server number is compared but not the accuracy of the recognition.

We synthesized three works mentioned above. [11] gave us fine tuned parameters can be used to detect face from a surveillance camera in real life scenarios. [7] influenced our design the most. They used a workstation in their Server Layer. We chose to use a cluster of Single-board computers (SBC) for scalability and redundancy. Design of Park did not include Cloud. [9] compared image recognition locally and in the Cloud. We extended his work by combining both together. We used image recognition in the Cloud when it is not recognized locally. Since different algorithms are running in both platforms, we expect to increase face recognition rate. [7] aimed to reduce bandwidth to their Server Layer. We aim to reduce bandwidth to the Cloud.

III. Background

Surveillance systems are prone to damage when an intruder breaks in. In order to protect surveillance footage from tampering it must be persisted in different locations. Cloud is the simplest solution for data persistence. However, one must consider bandwidth limitations and associated Cloud costs. Cloud is cheap for short bursts of usage but continuous usage of cumulative storage will result in high bills. Edge computing solves this problem by processing video footage on the edge before using any upstream resources. Setting up an Edge Cloud reduces cloud dependency significantly.

Traditionally, computing nodes are connected to the devices physically such as a camera connected to computer by USB cable. However, device becomes inaccessible when connected node dies. Further, device is only accessible to the attached computer but not to others. We propose to use IoT devices in Device Layer, specifically IP cameras. IP camera is a surveillance camera with a web server that serve video footage over network. IP camera is crucial in our design because it physically detaches devices with their computing node. Physical locations of agents become irrelevant because all nodes and devices in the system are accessible over network. This distributed approach has several advantages.

First, it creates a fault tolerant system. Multiple computing nodes can read from the same device. If a node dies, others can carry on the task. In addition, one computing node can process multiple devices. This many-to-many design between devices and computing nodes increases system reliability.

Second, it creates a horizontal scalable system. Horizontal scaling (also known as scaling up) [13] means adding more computing nodes to a system. Heavy algorithms or busy places may require additional processing power. Instead of reserving powerful servers for certain devices, one can add more nodes to the same device.

Third, it protects local storage. If device and computing node are physically connected and intruder damages the node then local footage is lost. Since cameras are in visible locations, it is easy to remove or tamper evidence if storage directly connected to the camera. Detaching storage and camera adds another layer of protection to evidence.

Using deep learning in face recognition has been studied extensively in the past [5], [14], [15]. A model is created by training hundreds of millions face thumbnails of millions of individuals. Model maps each face thumbnail to a point in 128 dimensional space. This reduces the complexity of the facial properties into 128-D space. Recognition task is then reduced to distance calculation in 128 dimensional Euclidean space. Any nearest neighbor algorithm can be used to find closest point to perform recognition. After training, model can be used for getting 128-D representations of any face instantaneously. Accuracy of the model is determined by the loss function. Triplet loss used in [15] works as the following: select two matching face thumbnails and one non-matching face thumbnail and the loss aims to maximize distance with non-matching and minimize distance with matching thumbnails.

Containerization also known as operating-system level virtualization, is a method to separate applications on operating system layer in isolated user space instances called containers. In this approach, application is natively running in host operating system as sole application in the OS. Normally an application can access all OS resources including peripherals. However an application inside a container only sees what is available in this container. This method has almost zero cost bootstrap and execution time and lightweight containers. This is because operating system in container is usually stripped version linux, like Alpine distribution which is only 5 MB and container and host shares kernel. Any dependencies are added so container size is equal to application size and its direct dependencies. Currently most widely used containerization software is Docker and it is invested heavily by the industry.

Raspberry Pi is developed by Raspberry foundation in United Kingdom [16]. It is open source and designed to promote computer science education in schools and developing countries. Raspberry Pi currently dominates the market for Single Board Computers (SBCs). As of March 2018 Raspberry Foundation reported 19 million unit sales [17]. This makes Raspberry Pi the third best-selling General Purpose Computer ever after Apple Macintosh and Microsoft Windows Personal Computers (PCs). Currently there are three models of Raspberry Pi on the market. All devices contain HAT-compatible 40-pin header, Micro USB power, Mini HDMI port and Micro USB on-the-go (OTG) ports.

In this paper, we propose a system where a cluster of SBC nodes in several layers as Edge. We preferred minimized upstream bandwidth over continuous video footage. Faces are cropped from video footage and processed further on the Edge and the Cloud. Only unrecognized faces are uploaded to the Cloud. This reduces bandwidth requirements and effectively reduces bandwidth related costs. Cloud is used not only for storage but also face recognition on the unrecognized faces to improve face recognition success. We designed our system to be distributed so that single point of failure is not introduced.

IV. Proposed System Architecture

We created an architecture based on [7] with some modifications and enhancements. We renamed Edge Layer as Preprocessing Layer (PPL) and Server Layer as Processing Layer (PL). We added one more layer, Cloud Layer (CL). High level architecture is given in Fig. 1. PPL and PL are both located on the edge and perform face detection and face recognition, respectively. Cluster of System On Chip (SoC) devices are used for PPL and PL to build fault tolerance. CL is added to check for unrecognized faces as a last step and persist results.

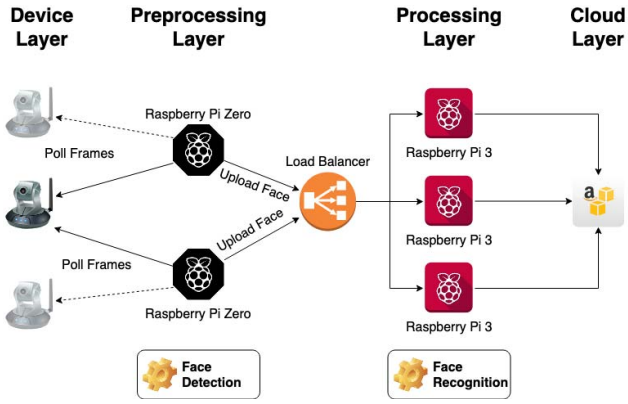

Fig. 1. System overview

A. Device layer

Device Layer (DL) contains an array of IP cameras. Web server in IP cameras listens for incoming requests and serves video footage to multiple clients simultaneously. RTP protocol [18] is used for this communication and OpenCV has native support to stream through RTP. In this work, we limited our DL to include IP cameras. However it can contain any type of IoT sensor such as smoke detector or alarm.

B. Preprocessing layer

Preprocessing layer (PPL) performs the same duties as Edge Layer in [7]; captures faces from raw footage from DL and feeds it to upper layer, PL. Our design includes cluster of nodes instead of one server. For this reason PPL is directly communicating with a load balancer to access an available PL node. Main duty of Preprocessing layer is to reduce bandwidth by processing sensor data before sending it to Processing layer. This also distributes computing power more evenly in the cluster. Processing Layer should only process what it receives from this layer rather than Device layer.

C. Processing layer

Processing Layer (PL) accepts requests from Preprocessing layer. Requests contain detected faces and other metadata such as time of the capture, filename and camera location. Processing layer applies face recognition to received face against known people database. PL requires more computational power than PPL because face recognition is done in this layer. If a match is found, frame is ignored and result of the recognition is saved to the Cloud database. This saves bandwidth and also persist the result of the recognition. If no match is found, face is marked as unrecognized. Then, frame is uploaded to the Cloud. In both cases, we save our recognition result to a database to persist results in case of tampering.

D. Cloud layer

Cloud Layer (CL) is used for two reasons: persist recognition results and perform another face recognition. In order to persist recognition results and unknown faces, we chose a Cloud based database and object storage. Face recognition can be performed by custom Cloud deployments or using vendor's computer vision API offerings. Almost all Cloud vendors offer computer vision APIs. At this time or writing, only Google Cloud do not offer face recognition among AWS, Azure and IBM Cloud. Another way is to deploy a Tensorflow instance to the Cloud and perform further processing which may require more computational power.

V. Experiments

In this section, we present hardware and software choices made to materialize our proposed system. We installed the system in the living room of one of the authors and let it run for two months. Preprocessing layer gathered thousands of faces during this period. We used an additional step to categorize the faces to test the accuracy of our system.

Two Raspberry Pi Zero are used in PPL and three Raspberry Pi 3 Model B are used in PL. We used only one camera in our experiments. However, design of our system does not restrict number of devices. IP camera of our choice is a generic IP camera with 1280x960p resolution and ONVIF 2.0 support. One of the nodes in PL also worked as load balancer. A simple round robin load balancer is used.

TABLE I
Haar Cascade Algorithm Variables

Variable	Value
Scale Factor	1.1
Min. Neighbors	5
Min. Size	(20, 20)
Max. Size	(300, 300)

Python 2.7 is chosen as programming language. OpenCV 3.2 is used for face detection and image manipulation. Haar-like feature based cascade classifiers are used with

`frontalface_alt` configuration. Variables shown on Table I is used for Haar Cascade. These parameters are optimal for face detection from 4 meters based on research from [11].

Many deep learning models are available online from variety of researchers. Each module is optimized for certain recognition task. This paper focuses on face recognition and Labeled Faces in the Wild (LFW) [19] benchmark is well accepted criteria for face recognition accuracy. Sample model by dlib's creator Davis King [4] is used for face recognition module. The model is trained against 3 million images and has 99.38% accuracy on LFW benchmark. dlib's face recognition module with Python wrapper [10] is used for deep face recognition library.

Cloud Layer is presented in Fig. 2. AWS is chosen for CL. Reasons for this choice include generous all time free tier with additional 12 month free offerings and face recognition module, Rekognition. As of writing of this paper Google Cloud Platform do not offer face recognition module. Azure started offering Face API as of April 2017 which includes face recognition module. AWS Rekognition accepts one photo as reference and compares with all faces in the given image.

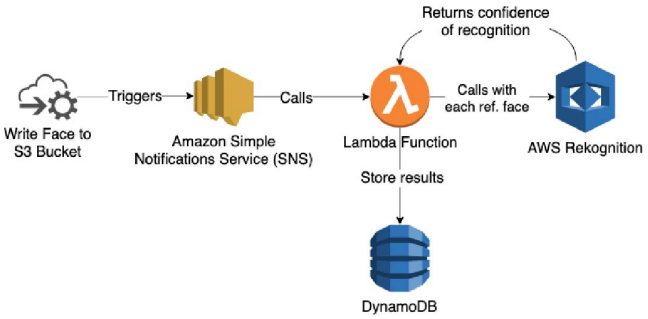

Fig. 2. Cloud Layer

Since cluster of devices are used in the system, bootstrapping a new device to create a replicable environment becomes a problem. Since Docker has native support for Raspberry Pi, we used Docker for containerization. Docker images used in this work are publicly available [20]. We compiled OpenCV 3.2 and dlib for ARM target and published the images online under `kaskavalci/opencv` and `kaskavalci/face_recognition` images respectively.

Two identities with 363 face photos were used as known face database. The faces are obtained by placing individuals next to the camera and saving face cropping from the stream. We used the model to obtain 128-d representations of the faces and saved this data in Pickle form. Pickle is a serialization library for Python. We saved this pickled data to Simple Storage Service (S3) of AWS. Each node downloads this file to get latest version of the database. Since training data is encoded, no training is required on the nodes. If a new face is to be added to the database, nodes only need to download new encoding. Encoding file size is relatively small. In our experiments 363 photos resulted in 1.3 MB of face database, uncompressed. When compressed to zip, it resulted in 295 KB.

Source code for recording faces is available under tools directory as `face cropper.py` in PiGuard repository [21]. Pickle serialization is taken largely from PyImageSearch.com and also available as `encode_faces.py`.

Within two months, PPL detected 4932 faces from several individuals. During this period camera was able to capture real world data with all variety of angles of faces in different lighting conditions. However, due to imperfections of Haar Cascade algorithm, it resulted in some false positives. Some shapes such as books in library were detected as face continuously. In order to categorize real faces into known people to test our system and omit false positive tests we used another machine learning tool. Sole purpose of this step is to improve the dataset condition and test recognition rates of individuals.

Google Photos are used to improve the quality of our dataset. Google Photos is a photo storage service where users can upload their photos free of charge. Application offers some services such as face recognition which we used. After Photos service successfully tagged faces we exported people from the application. This step eliminated the false positives and recognized individuals in the dataset. We will use two members of the household, namely Aysenur and Halil. During this time frame there were other individuals present in the house in short periods. However analysis is only performed for household members. Details are presented in Table II. Others in this field indicates other faces and false positive face detections.

TABLE II
DATASET AFTER TWO MONTHS OF OPERATION

Variable	Value
Aysenur	1423
Halil	1291
Others	2218
Total	4932

Google Photos revealed 1423 face samples of household member Aysenur and 1291 of Halil, a total of 2714 images. Some face samples contain multiple faces in one frame if they are close to each other. In this case, these files are duplicated in both dataset. We identified 14 photos that are present in both individuals. Others include other individuals and faceless frames (false positives). With this information we can test the accuracy of our system.

VI. Results

Result of the recognition in both PL and CL in DynamoDB is exported to MySQL database for detailed analysis.

Let,

$$A = \{f \mid \text{recognized as Aysenur}\} \quad (1)$$

$$H = \{f \mid \text{recognized as Halil}\} \quad (2)$$

$$A \cap H = \{f \mid \text{recognized as Halil and Aysenur}\} \quad (3)$$

$$A \cup H = \{f \mid \text{recognized as either Halil or Aysenur}\} \quad (4)$$

Equation (1) and (2) defines the frames that are recognized as Aysenur and Halil, respectively. Equation (3) contains

frames where Halil and Aysenur recognized together and (4) defines all frames that has been recognized as either of them. 83 images are recognized as both people present in the frame by Processing Layer and 26 by Cloud Layer, compared to 14 photos that Google Photos identified. Processing Layer mistakenly recognized face as Halil and Aysenur. It should be noted that algorithm does not skip a face once it has been tagged as recognized. If the same face encoding is matched in multiple encodings we expect multiple successful matches. In other words, same face, if detected as two separate faces, can be recognized as two different people.

Table III shows number of recognized images of Halil (H) and Aysenur (A) on Processing and Cloud Layers. Overall 2526 frames are recognized by Processing Layer and 2522 by Cloud Layer.

Processing Layer successfully recognized 93.56% of our dataset whereas Cloud Layer recognized 93.41%. Our recognition platform performs slightly better than AWS Rekognition platform. In order to compare performance of PL and CL we uploaded all images to the Cloud Layer. However, our system design skips frame that has been recognized locally and only uploads unrecognized frames. Our system can reduce the bandwidth consumption as much as 93.56%.

TABLE III
NUMBER OF RECOGNIZED IMAGES IN PROCESSING AND CLOUD LAYERS

	Processing Layer	**Cloud Layer**	**Sample Size**
$\lvert H\rvert$	1182	1151	1291
$\lvert A\rvert$	1427	1397	1423
$\lvert A \cap H\rvert$	83	26	14
$\lvert A \cup H\rvert$	2526	2522	2714

We analyzed number of frames that has been recognized successfully by Cloud Layer and frames that have been missed by both systems. Let,

$$CL = \{f \mid \text{recognized by Cloud Layer}\} \quad (5)$$

$$PL = \{f \mid \text{recognized by Processing Layer}\} \quad (6)$$

$$NCL = CL \setminus PL \quad (7)$$

Equation (5) and (6) defines frames that are recognized by Cloud Layer and Processing Layer, respectively. Equation (7) contains frames that are recognized by CL but not PL. Fig. 3 shows Venn Diagram representation of recognition rates while highlighting NCL. Universal Set $\mathcal{E}$ contains frames that are not recognized by neither PL nor CL. Labels inside circles are cardinality of their corresponding intersections.

Our experiments show that 2526 frames are recognized locally by Processing Layer and 145 by Cloud Layer. This leaves 29 frames that could not be recognized by both systems. This is because we use different recognition algorithms in those layers. Faces that has been missed by PL can be recognized in CL. Both systems perform similarly compared to each other but they perform superior when they are combined.

Our architecture was able to recognize 93.56% of our dataset locally and 98.93% in overall pipeline ($N = 2714$).

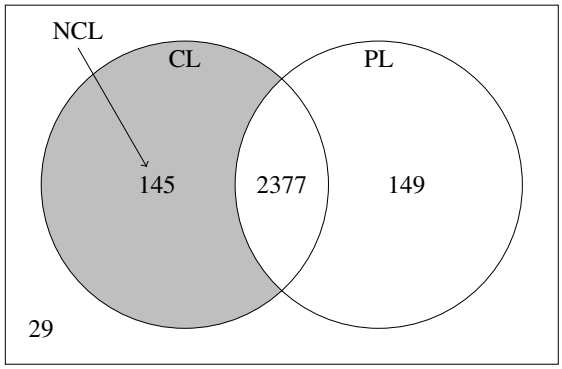

Fig. 3. Venn Diagram of recognition success rates

Using AWS Rekognition exclusively resulted in 93.41% success rate, insignificantly worse than dlib's performance. However, in our pipeline where only unrecognized photos are uploaded to the cloud, overall success rate becomes far superior. We realized that false positives can be quite high in [10] with their recommended distance threshold 0.6 (smaller is more strict). Reducing it to 0.5 eliminates all false positives in our dataset while reducing overall recognition success rate.

On average a face is recognized in $1.871s$ on Raspberry Pi and $1.642s$ in AWS Lambda. We achieved comparable results with AWS Rekognition in our local processing with 364 faces in face database. Fewer faces in database results in faster local computation. It should be noted that duration of the AWS Lambda does not take bootstrap time of AWS Lambda function and transmission time to the Cloud. Although locally recognition on Raspberry Pi is slightly slower than AWS Lambda, difference is not crucial. Network overhead is excluded from Cloud Layer duration in our statistics. Poor Internet connection can dwarf this result and can increase Cloud Layer duration significantly. Processing Layer is a safeguard against this with a minor delay in processing time. If high FPS is required by the application more powerful Edge Servers can reduce this delay significantly.

Our implementation permits adding more edge servers easily to the existing infrastructure. Each server is stateless and containerized. State is persisted on DynamoDB database on Processing and Cloud layers. In case of multiple cameras and high demanding applications, system can be scaled horizontally by adding more servers.

VII. CONCLUSION

In this paper, a hierarchical architecture is proposed for home surveillance using Raspberry Pi as Edge Server. We used Docker in Raspberry Pi successfully and setup a cluster of Raspberry Pi devices to create a highly available system locally. Our recognition algorithm utilized 364 faces of two individuals in its face database. We have used four Raspberry Pi 3 Model B in our Processing Layer while one also acted as a load balancer.

We have connected this local cluster to Cloud Layer and utilized AWS technologies. We used Lambda for serverless

execution, Rekognition for face recognition, S3 for object storage, SNS for notification system and DynamoDB as NoSQL database.

Primary contribution of this paper is to connect Edge Cloud to remote Cloud for face recognition. We demonstrated an end to end system with basic IP cameras, SoC devices and Cloud resources. We have used a highly accurate deep learning model for face recognition. We observed a face recognition accuracy of 98.93% in our overall pipeline.

We published our source code under Mozilla Public License on Github [21]. Docker images used in this work are publicly available under Docker Hub [20]. Configuration parameters are defined as environmental variables and documented under the Github project. We believe this work can be duplicated by other researchers with minimal effort.

REFERENCES

[1] Nest. (2019, Jan) Nest Aware – continuous video recording for Nest cams. [Online]. Available: https://nest.com/cameras/nest-aware

[2] Docker. (2018, Jun) Build, ship, and run any app, anywhere. [Online]. Available: https://www.docker.com/

[3] D. E. King, "Dlib-ml: A machine learning toolkit," *Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009.

[4] D. King. (2017, Jul) dlib c++ library: High quality face recognition with deep metric learning. [Online]. Available: http://blog.dlib.net/2017/02/high-quality-face-recognition-with-deep.html

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[6] Amazon. (2019, Jan) Amazon Rekognition video and image – AWS. [Online]. Available: https://aws.amazon.com/rekognition

[7] H. D. Park, O.-G. Min, and Y.-J. Lee, "Scalable architecture for an automated surveillance system using edge computing," *The Journal of Supercomputing*, vol. 73, no. 3, pp. 926–939, 2017.

[8] P. Menezes, J. C. Barreto, and J. Dias, "Face tracking based on haar-like features and eigenfaces," *IFAC Proceedings Volumes*, vol. 37, no. 8, pp. 304–309, 2004.

[9] U. Popic, "Two approaches for face recognition with IoT technologies," Master's thesis, Politecnico di Milano, Piazza Leonardo da Vinci, 32 20133 Milano, Italy, April 2018, http://hdl.handle.net/10589/139068.

[10] A. Geitgey. (2018, Sep) ageitgey/face_recognition: The world's simplest facial recognition api for Python and the command line. [Online]. Available: https://github.com/ageitgey/face_recognition

[11] A. A. Wazwaz, A. O. Herbawi, M. J. Teeti, and S. Y. Hmeed, "Raspberry pi and computers-based face detection and recognition system," in *2018 4th International Conference on Computer and Technology Applications (ICCTA)*. IEEE, 2018, pp. 171–174.

[12] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, Dec 2001, pp. I–I.

[13] M. Michael, J. E. Moreira, D. Shiloach, and R. W. Wisniewski, "Scale-up x scale-out: A case study using nutch/lucene," in *2007 IEEE International Parallel and Distributed Processing Symposium*, March 2007, pp. 1–8.

[14] O. M. Parkhi, A. Vedaldi, A. Zisserman *et al.*, "Deep face recognition." in *BMVC*, vol. 1, no. 3, 2015, p. 6.

[15] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.

[16] R. Mullins. (2017, Nov) Department of Computer Science and Technology: Raspberry Pi. [Online]. Available: https://www.cl.cam.ac.uk/projects/raspberrypi

[17] E. Upton. (2018, March) Raspberry pi 3 model b+ on sale now at $35 - raspberry pi. [Online]. Available: https://www.raspberrypi.org/blog/raspberry-pi-3-model-bplus-sale-now-35

[18] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," Internet Requests for Comments, RFC Editor, RFC 3550, July 2003. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3550.txt

[19] G. B. Huang, M. Mattar, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database forstudying face recognition in unconstrained environments," in *Workshop on faces in'Real-Life'Images: detection, alignment, and recognition*, 2008.

[20] H. C. Kaskavalci. (2018, Oct) kaskavalci - Docker Hub. [Online]. Available: https://hub.docker.com/r/kaskavalci

[21] ——. (2019, Jan) kaskavalci/PiGuard. [Online]. Available: https://github.com/kaskavalci/PiGuard