Edge-Based Video Surveillance with Embedded Devices

Discussion Paper

Hanna Kavalionak[1], Claudio Gennaro[1], Giuseppe Amato[1], Claudio Vairo[1], Costantino Perciante[2], Carlo Meghini[1], Fabrizio Falchi[1], and Fausto Rabitti[1]

[1] ISTI-CNR, via G. Moruzzi 1, 56124 Pisa, Italy name.surname@isti.cnr.it
[2] Fluidmesh Networks, via Carlo Farini 5, Milano, Italy
costantino.perciante@fluidmesh.com

Abstract. Video surveillance systems have become indispensable tools for the security and organization of public and private areas. In this work, we propose a novel distributed protocol for an edge-based face recognition system that takes advantage of the computational capabilities of the surveillance devices (i.e., cameras) to perform person recognition. The cameras fall back to a centralized server if their hardware capabilities are not enough to perform the recognition. We evaluate the proposed algorithm via extensive experiments on a freely available dataset. As a prototype of surveillance embedded devices, we have considered a Raspberry PI with the camera module. Using simulations, we show that our algorithm can reduce up to 50% of the load of the server with no negative impact on the quality of the surveillance service.

Keywords: Edge Computing · Distributed Architectures · Internet of Things · Video Surveillance · Embedded Devices.

1 Introduction

Video surveillance is of paramount importance in areas like law enforcement, military and even for a commercial environment. One of the straightforward approaches for video surveillance is to use the client-server model of communication: the surveillance devices stream the video directly to a main powerful server, where the data can be displayed to the human operators, who are responsible to analyze the video [8]. Human resources used in the field of video surveillance services are both costly and not reliable. Also, this approach presents several negative sides, like the creation of a bottleneck for the system security and reliability and the need to maintain a big and costly infrastructure of servers

dedicated only to the surveillance task. Hence, with the recent advances in smart technologies, automated video surveillance, where the video streams are analyzed automatically by network edge devices, gained a lot of interest [13, 11].

The idea of exploiting the computation capabilities and the topology of a distributed network of smart cameras to reduce the amount of messaging has been proposed in [14] and [15]. In both approaches, the communication is efficiently handled using a task-oriented node clustering that partition the network in different groups according to the pathway among cameras. The former work, however, is limited to face tracking, while the second one targets collaborative person tracking with a combination of hardware acceleration and middleware. These works focus on people tracking rather than recognition and use an efficient camera clustering protocol to dynamically form groups of cameras for in-network tracking of individual persons.

This work summarizes the contribution presented in [7]. The motivating scenario that we consider for this work is that of face recognition [10, 3, 1, 4]. The surveillance system should be able to recognize faces, and track their movements in a possibly large area (e.g. an airport, a city district, a campus, etc.), using various cameras appositely installed. Real-time video surveillance requires significant storage and processing resources. This work aims to analyze the issues and solutions related to resource allocation, for executing automated edge-based video surveillance, in a fully distributed environment [2, 12]. In particular, we suppose that the (smart) camera devices themselves can cooperate to execute the needed faces detection, recognition and visual analysis tasks. In this work, we study how the image recognition algorithms can be orchestrated across several devices so that bottlenecks are reduced and resource can be managed more effectively. This work extends and enhances our previously published research work [6], where we have proposed a distributed algorithm for load balancing between Smart Sensing Units for the video surveillance task. The adaptive algorithm distributes, at run time, the recognition tasks between the resources of surveillance devices and servers. The detection and recognition tasks are executed locally by surveillance devices, which exploit both the spatial and temporal topology of the moving people, to cache and reuse locally parts of the classification features. Only when devices are not able to execute the recognition task with the cache, a recognition request is sent to the server. We extend the previously proposed adaptive algorithm considering the overhead of real classification techniques.

The rest of the paper is structured as follows. Section 2 provides the definition of the system model and problem statement, whereas Section 3 presents the algorithm for the adaptive camera-assisted person recognition. Section 4 evaluates the distributed surveillance solution through simulations. Finally, Section 5 concludes the paper.

2 System model and problem statement

In our work, we model the geographical area, where the surveillance system is deployed, as divided into sectors. A set of *Smart Sensing Units* (SSUs) are

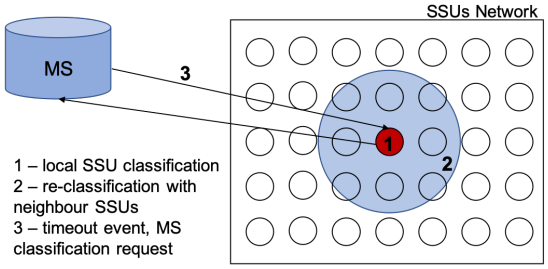


Fig. 1: System model

distributed among the sectors to execute the video surveillance task (Figure 1). Such units have various sensing, computing, storage, and communication capabilities. An SSU is a logical unit, it could be a smart camera having onboard sensing, computing, storage resources, or it could be composed of a camera connected to a computer. Moreover, we assume also the existence of a *Main Server* (MS) unit in the network. A main server is a stand-alone SSU that is characterized by (theoretically) infinite computing, storage, and communication capabilities. It also plays the role of the central server where the devices send their requests, in case of client-server modality.

In our work, we distinguish two possible scenarios: *basic* and *enhanced*. In the basic scenario, all SSUs are connected to the central MS and their functionality is limited to video observation and data streaming to the MS. In the enhanced scenario, on the other hand, all SSUs are connected in a structured peer-to-peer topology. SSUs can cooperate to execute complex tasks and build together a rich representation of the context where the infrastructure is deployed. We consider the peer-to-peer overlay connecting the SSUs to be fixed. Each SSU can contact directly the MS and its geographical neighbor SSUs. We assume the communication between SSUs is reliable, i.e., no message is ever lost or corrupted. In the enhanced scenario, each SSU has computational resources and local storage memory, where it keeps a *recognition library*, which is a collection of recognizers, each specialized to recognize a certain person. The MS's recognition library contains the recognizers for all the recognizable persons in the surveillance area, while the recognition library of an SSU is, basically, a local cache that keeps only a subset of all recognizable persons.

As an illustrative example, let us consider the following scenario, taking place in a geographical area (the surveilled area) of appropriate size, where a number of SSUs have been deployed for video surveillance purposes. Here we consider a person that is moving in the area. The system should visually recognize the person in order to evaluate the presence of this person in the area. One of the straightforward solutions for area video surveillance is to stream all the data from the SSUs to the MS for the following processing and storage. This approach can cause some additional system issues. At first, the interested organization needs to have a powerful server system devoted specifically to the surveillance needs.

Algorithm 1: Active thread algorithm executed by SSU

```
repeat
    recognizers.increaseAge();
    events ← getVisibleFaces();
    selection ← selectRecognizers(recognizers);
    foreach face in events do
        if alarm ←getStatus(face, selection) then
            send(FACEID, face, alarm, view);
            startAlarmActivity();
            return;
        if recognized ←getStatus(face, selection) then
            send(FACEID, face, recognized, view);
            return;
        if null ←getStatus(face, selection) then
            t_δ ←setRecognitionTimeout();
            reclBuffer.add(face, t_δ);
            send(ALARMREQUEST, face, null, view);
            return;
    wait ΔT;
until;
```

This server system needs to be powerful enough to continuously process the video streaming data that is coming from the SSUs. The second problem is connected with the delays for the person recognition, caused by possible server overloading and network bandwidth overhead in case of a high rate of requests for recognition.

In order to tackle these issues, our solution exploits a distributed architecture for face recognition, where the detection and recognition tasks are moved to SSUs when it is possible. We performed some simulations in order to move the most possible computation on the camera themselves and we used optimized strategies of distributed computation to solve the most challenging resource-demanding tasks.

3 Algorithm

In this section, we describe our enhanced algorithm for distributing the recognition processes on the SSUs. Each camera in the system keeps the list of its neighbors SSUs, which we name *view*. In order to improve the efficiency of the recognition process each camera locally keeps a recognition library that contains a subset of all recognizers for *faces*. Using these local recognizers, each camera in the network tries to recognize the face of the detected person locally and with the help of neighbor SSUs, without involving the MS, thus reducing its computational and bandwidth consumption. The size of the library is limited to $recognizersLimit$ according to the available storage space of an SSU.

Three threads are executed by each of the SSUs: (1) *active thread* (Algorithm 1) is responsible for the active area monitoring, (2) *time-out thread* corresponds to the timeout event in the face recognition by neighbors and (3) *passive thread* (Algorithm 2) processes the incoming messages received by an entity.

Algorithm 2: Passive thread algorithm executed by SSU

```
on event msg(type, face, status, sender) receive do
    if type == ALARMREQUEST then
        result ← getStatus(face, selection);
        send(ALARMREPLY, face, result, sender);
    if type == ALARMREPLY then
        if sender == MS then
            if status == alarm then
                startAlarmActivity();
            integrate(recognizers, face);
            send(FACEID, face, status, view);
        else
            recBuffer.getFace(face).counter++;
            if recBuffer.contains(face) and status == null then
                if recBuffer.getFace(face).counter ≥ view.size then
                    send(ALARMREQUEST, face, null, MS);
                    recBuffer.remove(face);
            else
                if status == alarm then
                    startAlarmActivity();
                integrate(face, recognizers);
                recBuffer.remove(face);
                send(FACEID, face, status, view);
    if type == FACEID then
        integrate(face, recognizers);
```

Every ΔT each SSU executes the active thread Algorithm 1. At first, it increases the "age" of all known recognizers in the library *recognizers*. Then, it processes the current surveillance area image and extracts the facial features for the detected persons. Finally, it selects the subset *selection* of *recognizers* with the lowest *age* parameter from the local collection.

For each $face$ in *events* an SSU executes the recognition algorithm. In case $face$ corresponds to the entry in *recognizers* with *alarm* status, the SSU (1) sends a notification message type FACEID with a $face$ features and the *alarm* tag to the neighbor SSUs; then (2) starts predefined alarm activity, for example, video recording and live streaming to the operator displays.

In case a $face$ is *recognized*, a notification FACEID is distributed between the neighbor SSUs. If there is no correspondence in *selection*, the camera establishes a recognition waiting time and adds the face features into the reclassification buffer $reclBuffer$. It also sends the recognition request ALARMREQUEST to the neighbors' SSUs. When the recognition waiting time has passed, if the object is still in the $reclBuffer$, then the camera sends the classification request ALARMREQUEST to the MS and removes the features from the$reclBuffer$.

When an SSU receives a message, it is processed according to the Algorithm 2. We consider three types of messages: ALARMREQUEST, ALARMREPLY and FACEID. The ALARMREQUEST message contains the request for a $face$ classification. The recipient checks the status of the received $face$ in the local subset of a *recognizers* list and sends an ALARMREPLY containing the results of the recognition *result* to the *sender*. The FACEID message is used for the face fea-

tures broadcasting to the neighbor SSUs. Hence, whenever a camera receives this message it integrates the received face features to the local $recognizers$ library.

The ALARMREPLY message corresponds to the reply to a recognition. If it comes from MS entity, the recipient integrates the recognized face and its status to the local $recognizers$ and sends the face notification message FACEID to its neighbors $view$. Moreover, in case the received status is $alarm$ the camera initiates some predefined alarm activity. Each SSU keeps track of the replies of all the neighbor SSUs for each $face$ in the $reclBuffer$. If no neighbor SSU is able to recognize the current $face$, then the camera sends the recognition request ALARMREQUEST to the MS and removes the $face$ from the reclassification buffer $reclBuffer$. If, on the other hand, the received status is not $null$ it means the $face$ has been recognized by the sender and the camera integrates it to the local $recognizers$, removes the face from the reclassification buffer and sends the face notification message FACEID to its neighbors SSUs. In case the received status is $alarm$, the camera also initiates some predefined alarm activity.

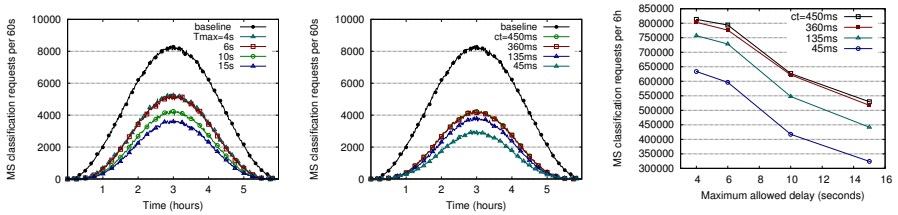
4 Evaluation of the area surveillance algorithm

In this section, we present a simulation experiment in order to evaluate the load imposed on the MS and the performance of the distributed surveillance system.

In our simulation, we assume that the SSUs are uniformly distributed in the monitored area. Persons in the "surveillance area" move according to the mobility model described in [5] for the movements of avatars in a virtual environment. In this model, the area has a number of *hotspots* equals to n_{hs}. The hotspots are the most attractive places in the area where it is expected to find more persons there than in other places. Each person that enters the surveillance area follows the following behavior: the person chooses a random number of hotspots to visit and moves towards them, in sequence. After all the hotspots have been visited, the person leaves the surveillance area. The population of the area follows a bell-shaped curve very similar to the normal distribution in which the population starts from 0 and reaches a maximum of 800 persons. In our simulations, we considered the 10% of persons in the area to be $unknown$ (i.e. associated with alarm tag).

We executed simulation of 6 hours using the *peersim* simulator [9] and the mobility model described above. We evaluate two algorithms: (1) when all the data is transmitted to the MS for the analysis (our baseline) and (2) the adaptive algorithm described in Section 3.

In the baseline model, all SSUs stream the video directly to the MS, where the MS extracts the face features and recognizes the persons. The baseline curve in Figure 2 shows the number of recognition the MS is required to process in a 60sec time interval. Other curves on the figure show the evaluation of the recognition requests to MS when the enhanced algorithm is applied and under different system parameters, such as T_{max} and ct, where T_{max} is the maximum allowed face recognition delay and ct is the time needed to compare the extracted features with one class in the recognition library. The results show that the



(a) $ct = 450ms$; workload A (b) $T_{max} = 10s$; workload A (c) aggregated requests to SC; 6 h experiment; workload B

Fig. 2: MS recognition requests per 60sec for different system parameters.

enhanced algorithm can reduce up to 50% of the recognition activity of the MS unit in peak hours.

Each class of the cache is a simulated sample of the face features that characterize a person. Therefore, the time needed by an SSUs to perform a single face recognition task is given by the size of the cache selection times ct. A higher number of the face features in the class increases the accuracy of the recognition, but at the same time increases also the ct.

As we can see in Figure 2a, in case of a lower T_{max} the system does not have enough time for the whole algorithm recognition cycle and it is forced to rely on the MS recognition more often. Instead, a higher T_{max} allows an SSU to perform the recognition using the cache and to receive the replies from the neighbor SSUs. Nevertheless, even in case of low T_{max}, the enhanced algorithm significantly reduces the recognition load of MS compared to the baseline solution.

Figure 2b shows the influence of ct time interval on the MS recognition load. Even in case of high accuracy of local recognition and relatively low T_{max}, the load of MS by recognition requests is significantly lower than the baseline solution. Moreover, lower ct values further decreases the load imposed on the MS by the recognition. Figure 2c shows the impact of the system parameters on the algorithm effectiveness. Lower values of ct and larger of T_{max} significantly reduce the MS recognition load. One of the straightforward ways to minimize ct time is to reduce the accuracy of the local recognition algorithm by decreasing the number of features in the sample. In case fast person recognition is not a requirement, the increment of T_{max} can also reduce the load of the MS.

5 Conclusion

In this paper, we propose a distributed edge-based protocol for area video surveillance based on image classification. In order to minimize the classification load on the main server, recognition tasks take place on the SSUs when possible. To perform person recognition, an SSU uses the local resources together with the resources of the neighbor SSUs. The surveillance devices fall back to the main server when the classification cannot be done in the desired time interval. In this extended abstract, we gave an overview of how our system works, we described

the proposed algorithms and we presented simulations that show that by partially placing the recognition tasks on the local resources of surveillance devices can reduce the load on the main server up to 50%.

References

1. Amato, G., Carrara, F., Falchi, F., Gennaro, C., Vairo, C.: Facial-based intrusion detection system with deep learning in embedded devices. In: International Conference on Sensors, Signal and Image Processing (SSIP). pp. 64–68. ACM (2018)
2. Amato, G., Chessa, S., Gennaro, C., Vairo, C.: Efficient detection of composite events in wireless sensor networks: design and evaluation. In: 2011 IEEE Symposium on Computers and Communications (ISCC). pp. 821–823. IEEE (2011)
3. Amato, G., Falchi, F., Gennaro, C., Vairo, C.: A comparison of face verification with facial landmarks and deep features. In: 10th International Conference on Advances in Multimedia (MMEDIA). pp. 1–6 (2018)
4. Amato, G., Gennaro, C., Massoli, F.V., Passalis, N., Tefas, A., Trvillini, A., Vairo, C.: Face verification and recognition for digital forensics and information security. In: IEEE 7th International Symposium on Digital Forensic and Security (ISDFS). pp. 1–6. IEEE (2019)
5. Kavalionak, H., Carlini, E., Ricci, L., Montresor, A., Coppola, M.: Integrating peer-to-peer and cloud computing for massively multiuser online games. Peer-to-Peer Networking and Applications (PPNA) pp. 1–19 (2013)
6. Kavalionak, H., Gennaro, C., Amato, G., Meghini, C.: Dice: A distributed protocol for camera-aided video surveillance. In: Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), 2015 IEEE International Conference on. pp. 477–484. IEEE (2015)
7. Kavalionak, H., Gennaro, C., Amato, G., Vairo, C., Perciante, C., Meghini, C., Falchi, F.: Distributed video surveillance using smart cameras. Journal of Grid Computing **17**(1), 59–77 (2019)
8. Mishra, R., Kumar, P., Chaudhury, S., Indu, S.: Monitoring a large surveillance space through distributed face matching. In: Computer Vision, Pattern Recognition, Image Processing and Graphics (NCVPRIPG), 2013 Fourth National Conference on. pp. 1–5. IEEE (2013)
9. Montresor, A., Jelasity, M.: PeerSim: A scalable p2p simulator. In: Proc. of P2P'09. pp. 99–100. IEEE (2009)
10. Parkhi, O.M., Vedaldi, A., Zisserman, A.: Deep face recognition (2015)
11. Song, M., Tao, D., Maybank, S.J.: Sparse camera network for visual surveillance–a comprehensive survey. arXiv preprint arXiv:1302.0446 (2013)
12. Vairo, C., Amato, G., Chessa, S., Valleri, P.: Modeling detection and tracking of complex events in wireless sensor networks. In: 2010 IEEE International Conference on Systems, Man and Cybernetics. pp. 235–242. IEEE (2010)
13. Wang, X.: Intelligent multi-camera video surveillance: A review. Pattern recognition letters **34**(1), 3–19 (2013)
14. Yoder, J., Medeiros, H., Park, J., Kak, A.C.: Cluster-based distributed face tracking in camera networks. Image Processing, IEEE Transactions on **19**(10), 2551–2563 (2010)
15. Zarezadeh, A., Bobda, C., Yonga, F., Mefenza, M.: Efficient network clustering for traffic reduction in embedded smart camera networks. Journal of Real-Time Image Processing pp. 1–14 (2015)