Multi-Stage Dynamic Batching and On-Demand I-Vector Clustering for Cost-effective Video Surveillance

David Montero, Luis Unzueta, Jon Goenetxea, Nerea Aranjuelo, Estibaliz Loyo, Oihana Otaegui and Marcos Nieto

Vicomtech, Mikeletegi 57, 20009 Donostia-SanSebastian, Spain

Keywords: Face Recognition, Face Clustering, Video-Surveillance.

Abstract: In this paper, we present a cost-effective Video-Surveillance System (VSS) for face recognition and online clustering of unknown individuals at large scale. We aim to obtain Performance Indicators (PIs) for people flow monitoring in large infrastructures, without storing any biometric information. For this purpose, we focus on how to take advantage of a central GPU-enabled computing server, connected to a set of video-surveillance cameras, to automatically register new identities and update their descriptive data as they are re-identified. The proposed method comprises two main procedures executed in parallel. A Multi-Stage Dynamic Batching (MSDB) procedure efficiently extracts facial identity vectors (i-vectors) from captured images. At the same time, an On-Demand I-Vector Clustering (ODIVC) procedure clusters the i-vectors into identities. This clustering algorithm is designed to progressively adapt to the increasing data scale, with a lower decrease in its effectiveness compared to other alternatives. Experimental results show that ODIVC achieves state-of-the-art results in well-known large scale datasets and that our VSS can detect, recognize and cluster in real time faces coming from up to 40 cameras with a central off-the-shelf GPU-enabled computing server.

1 INTRODUCTION

In recent years, there has been a growing interest in obtaining a detailed operational experience of people flow monitoring in large infrastructures, by means of Performance Indicators (PIs), such as ”waiting times”, ”process throughput”, ”queue length overrun” and ”area occupancy” (Mayer et al., 2015). Current non-cooperative solutions are typically integrated in the terminal infrastructure or use data from existing processes, such as entrance pass scans or mobile devices and Wi-Fi signals, from which PIs can be derived. As stated in (Mayer et al., 2015), for this kind of applications, computer-vision-based facial recognition technology has been used mainly as a source to infer ”waiting times”, by comparing biometric information from entry and exit points.

Our motivation is to enhance and extend the applicability of computer-vision-based facial recognition technology to more cases than the estimation of ”waiting times” in a specific area of a large infrastructure. More specifically, our goal is to build a face recognition-based solution from which all the PIs can be derived for the whole infrastructure, trying to simplify as much as possible the required hardware setup, and with a better handling of data so that privacy issues can be avoided. This requires building an accurate and efficient face recognition system that can manage large-scale data, without storing the biometric information of the individuals. Deep Neural Network (DNN)-based large-scale clustering approaches are promising methodologies for this purpose (Wang et al., 2019; Shi et al., 2018; Otto et al., 2018).

Deploying computer vision algorithms to build a cost-effective Video-Surveillance System (VSS) is challenging. The latest trends rely on distributed computing infrastructures, based on cloud (Lim et al., 2018), fog (Nasir et al., 2018) and/or edge computing paradigms (Chen et al., 2019). These solutions are capable of processing multiple sensor data streams more efficiently at a bigger scale, compared to the traditional centralized infrastructures (Tsakanikas and Dagiuklas, 2017). The main open questions in the design of a VSS (distributed or centralized) are what equipment should be used and how to take advantage of the available computing capabilities. This work focuses on the second question when an off-the-shelf GPU computing server is considered. This solution directly addresses the centralized infrastructure case, but is likely extendable to distributed infrastructures.

Montero, D., Unzueta, L., Goenetxea, J., Aranjuelo, N., Loyo, E., Otaegui, O. and Nieto, M.
Multi-Stage Dynamic Batching and On-Demand I-Vector Clustering for Cost-effective Video Surveillance.
DOI: 10.5220/0010236204360443
In *Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2021) - Volume 5: VISAPP*, pages 436-443
ISBN: 978-989-758-488-6

These are the main contributions of our work:

- A cost-effective VSS for face recognition and online clustering of unknown individuals at large scale, without storing their biometric information, to obtain PIs for people flow monitoring.
- A Multi-Stage Dynamic Batching (MSDB) procedure to efficiently extract face attributes and i-vectors from captured images. This includes MB-MTCNN, a multi-batch version of a Multi-Task Cascaded Convolutional Network (MTCNN) (Zhang et al., 2016a), and an efficient dynamic batching strategy for the processing of dynamic lists of facial images.
- An On-Demand I-Vector Clustering (ODIVC) algorithm, designed to progressively adapt to the data scale, with a lower decrease on its effectiveness compared to state-of-the-art alternatives.

The paper is organized as follows. Section 2 presents the related work. Section 3 describes the proposed VSS, followed by the computation improvements and optimizations in section 4. Section 5 provides experimental results that show the potential of our approach. Finally, section 6 concludes the paper.

2 RELATED WORK

The main challenges in our context are two: (A) how to cluster faces by identity, using an online and unsupervised approach, with the necessary accuracy and scalability; and (B) how to build a cost-effective VSS to deploy this technology.

2.1 Face Clustering

In recent years, face clustering in large-scale unconstrained scenarios has become a major challenge. The huge number of faces and the intra-class changes due to environmental variations (e.g., pose, illumination, occlusions, resolution, noise) lead to complex distributions of face representations. This makes it unsuitable to apply classic algorithms like K-Means (Lloyd, 1982), which tend to generate similar sized clusters, or spectral clustering (Shi and Malik, 2000).

State-of-the-art methodologies combine DNN-based face recognition models to extract i-vectors with clustering algorithms that can group them in distinguishable identities, despite the intra-class variability. In (Shi et al., 2018) the ConPaC algorithm is proposed, which is based on the estimation of an adjacency matrix using pairwise similarities between i-vectors. In (Wang et al., 2019) GCN is presented, where a DNN decides which pairs of nodes should be linked. In (Lin et al., 2018) an Agglomerative Hierarchical Clustering approach is adopted, considering the distance in the embedded space and the dissimilarity between groups of faces. In (Otto et al., 2018) an approximate rank-order clustering is presented, which predicts whether a node should be linked to its k Nearest Neighbors (kNN), and merges all linked pairs.

Nevertheless, these methodologies use offline algorithms. They process entirely the gathered data every time a new i-vector arrives with increasing computational cost. In addition, most of them suffer from scalability problems. For instance, the complexity of ConPaC can scale up to $O(TN^3)$, where N is the number of i-vectors and T the number of iterations. (Wang et al., 2019) and (Otto et al., 2018) reduce the complexity using kNN graphs to reduce the number of comparisons, but the cost is still too high for considering them for online applications.

To overcome these problems, we present On-Demand I-Vector Clustering algorithm (ODIVC). It is suitable for large-scale real-time applications, as it is devised to dynamically adapt to the inclusion of data samples without repeating the whole process.

2.2 Computing Infrastructures for Automated Video-Surveillance

New automated surveillance systems incorporate modern video sensors capable of capturing high definition footage and DNN-based processing pipelines in real-time. These systems frequently require large computational resources and large storage size. Both requirements could be resolved by integrating the VSS in a distributed computing infrastructure, relying on cloud, fog and/or edge computing paradigms. However, this introduces new challenges to be addressed (Tsakanikas and Dagiuklas, 2017).

A VSS relying on cloud computing (Lim et al., 2018) needs to take into account the latency and extra communication cost introduced between the sensors and the cloud infrastructure. Furthermore, it needs to deal with the latency introduced in IP networks, which is not only large but also fluctuating. Fog computing is a complementary technology to cloud computing, as it extends cloud capabilities to the edge of the network, comprising devices that have enough power to perform non-trivial computational tasks. A representative example of fog-based VSS is presented in (Nasir et al., 2018). Finally, edge computing refers to transferring computational and storage capacities from data centers to the video sensors, minimizing the network latency. Its application in a VSS requires the usage of special hardware and software close to the video sensors (Chen et al., 2019).

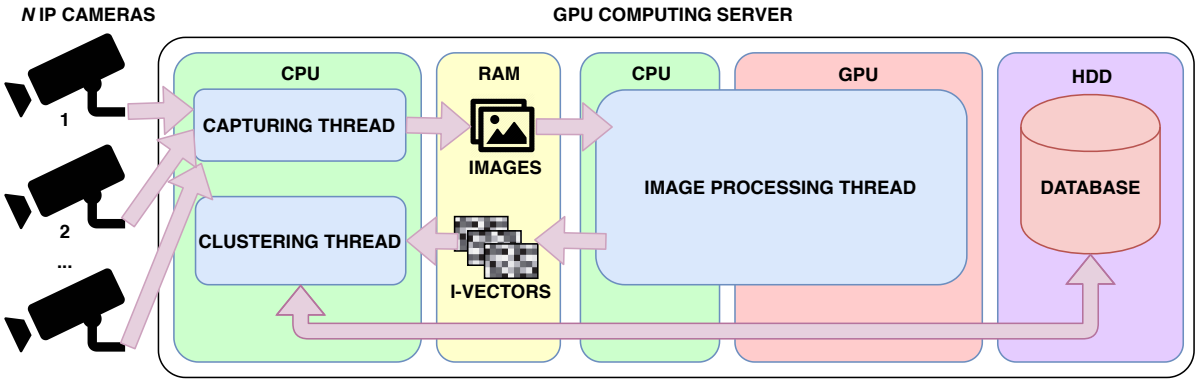

Figure 1: Overall processing architecture and data flow diagram of the proposed VSS.

As stated in (Tsakanikas and Dagiuklas, 2017), a promising approach for a VSS could be a distributed architecture, where certain characteristics of each approach are utilized to maximize its efficiency. In any case, effectively deploying DNN-based technology in such kind of equipment to build a cost-effective VSS, is a challenge to be addressed.

3 PROPOSED GPU COMPUTING SERVER-BASED VSS

3.1 Capturing Thread

The capturing thread is in charge of three tasks: camera connection handling, stream decoding and image acquisition. The effective camera management involves not only creating the connections when the system startups, but also periodically checking whether they are still working and reconnecting in case of a lost connection. Image streaming uses a compression method to reduce the network overload (e.g., H.264, H.265, etc) (Jankar and Shah, 2017), so a decoding phase is needed before feeding the images into the processing thread. This can cause a delay in the results. To reduce this effect, the capturing thread decodes each image stream separately. Thus, the current image is ready when the image processing thread needs to get the next frame. The computational cost of the capturing thread is low compared with the rest and it only uses a small amount of the CPU for frame decoding, mask applying, and connection checking.

3.2 Image Processing Thread

The image processing thread applies the DNN-based face analysis algorithms to the captured images, in order to get the required i-vectors for the re-identification. It consumes most of the computation

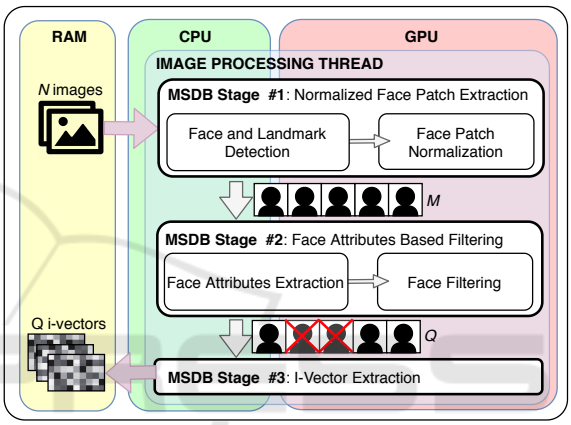

Figure 2: MSDB procedure in the image processing thread.

time of the entire VSS (90-95%). Thus, to improve the global performance, the system exploits the GPU resources for the DNN inferences, using the CPU for minor tasks like processing flow control, image cropping and managing patch lists (Figure 2).

This procedure has three sequential stages: 1) normalized face patch extraction, 2) facial attribute-based filtering, and 3) i-vector extraction.

The first stage detects all the faces present in the input image list and their facial landmarks, which are used for the face patch normalization following the method described in (Wang et al., 2018). For an efficient detection of the facial regions and landmarks in the N images from the capture process we propose a multi-batch version of MTCNN (MB-MTCNN), described in section 4.1, with a batch size equal to the number of images (i.e. N). This stage generates a list of M face regions with a set of five facial landmarks.

The VSS captures images from uncontrolled environments, so the detected faces may have different orientations, lighting conditions, partial occlusions, etc., which can negatively influence in the i-vector extraction, reducing the accuracy of the re-identification

process (Tan and Triggs, 2010). It is well known that the impact of these factors can be reduced by preprocessing the face patches before extracting the i-vector, as stated in (Chaudhari and Kale, 2010).

Thus, the second stage checks if the facial patch is suitable for re-identification using a set of automatically detected attributes and previously defined filtering considerations. In our context, we use head orientation and a set of angle thresholds for the filtering process, but other descriptive attributes could also be considered (e.g. age, gender, ethnic group, etc.). These attributes are estimated with Multi-Task Cascaded Convolutional Networks (TCDCN) (Zhang et al., 2016b). All the patches that do not match the established rules are removed.

Finally, the last stage extracts the i-vectors from the list of filtered face patches given by the filtering stage. For our experiments we use a DNN model based on ResNet100 architecture (He et al., 2016) with ArcFace loss (Deng et al., 2019). We use this architecture, despite its complexity, because we need to generate the i-vectors as robust as possible in order to get highly-reliable PIs. The DNN inferences applied to the dynamic lists of cropped facial images are made following efficient dynamic batching procedures, as explained in section 4.2.

3.3 Clustering Thread

This thread analyzes the incoming i-vectors to automatically register new people and update their descriptive data with new samples as they are re-identified, in an unsupervised way. Algorithm 1 shows our proposed ODIVC procedure for this task.

The identities database is represented with four lists: **C**, containing the representative i-vector (centroid) of each registered identity; **SC**, containing the sum of the candidates i-vectors of each registered identity; **uid**, with the unique identification numbers for those identities; and **nm**, with the number of detections matched with each identity. Thus, the database is expressed as $<\mathbf{C}, \mathbf{SC}, \mathbf{uid}, \mathbf{nm}>$. The new identity candidates do not get a unique identification number until they have reached a certain number of matched detections. This is done to filter erroneous new identity candidates, created with unsuitable faces passed through the filters of the detection stage.

When the VSS is launched, the identities database is empty. Therefore, the first incoming i-vector will be used as the representative i-vector of the first identity candidate in the database. From then on, every new incoming i-vector is compared with the centroid of every registered identity. The comparison is done by computing the cosine similarity, as the employed face recognition model was trained to work with this metric (Deng et al., 2019), but other similarity measures may be considered. To speed up the computation of the cosine similarity, every incoming i-vector and centroids are normalized, so that only the dot product between them needs to be computed. Besides, the calculation of the dot product is parallelized. Once the cosine similarity is extracted, the algorithm searches for the best candidate above a predefined minimum threshold ($thresh_{low}$). If the selected identity has more than nm_{min} members, then the centroid is considered robust enough for using $thresh_{low}$ and the i-vector is matched to that identity. Otherwise, the similarity measure must be higher than $thresh_{high}$. This searching process is repeated until there is a match or until the best similarity measure is lower than $thresh_{low}$. If there is a match between the incoming i-vector and a registered identity, the new i-vector is used to update

Algorithm 1: On-Demand I-Vector Clustering.

Input: new i-vector **u**, database of identities $<\mathbf{C}, \mathbf{SC}, \mathbf{uid}, \mathbf{nm}>$ (initially empty), low similarity threshold $thresh_{low}$ [-1,1], high similarity threshold $thresh_{high}$ [-1,1], minimum cluster members for low threshold nm_{min}
Output: Updated DB $<\mathbf{C}, \mathbf{SC}, \mathbf{uid}, \mathbf{nm}>$

$\mathbf{u}_{norm} = ||\mathbf{u}||_2$
$\mathbf{sim} = \mathbf{u}_{norm} \cdot \mathbf{C}$
$match$ = -1, sim_{best} = -1
if $sizeof(\boldsymbol{R}) > 0$ **then**
  $idx_{best} = maxIdx(\boldsymbol{sim})$
  $sim_{best} = \mathbf{sim}[idx_{best}]$
end
while $sim_{best} > thresh_{low}$ **do**
  if $\boldsymbol{nm}[idx_{best}] > nm_{min}$ *or* $sim_{best} > thresh_{high}$ **then**
    $match = idx_{best}$
    break
  end
  $\mathbf{sim}[idx_{best}]$ = -1
  $idx_{best} = maxIdx(\boldsymbol{sim})$
  $sim_{best} = \mathbf{sim}[i_{best}]$
end
if $match > -1$ **then**
  $\mathbf{nm}[match] = \mathbf{nm}[match] + 1$
  $\mathbf{SC}[match] = \mathbf{SC}[match] + \mathbf{u}_{norm}$
  $\mathbf{C}[match] = ||\mathbf{SC}[match]||_2$
else
  uid_{new} = generateNewUID()
  Append $\mathbf{u}_{norm}$ to **C** & **SC**, uid_{new} to **uid** and 1 to **nm**
end
return $<\mathbf{C}, \mathbf{SC}, \mathbf{uid}, \mathbf{nm}>$

the sum and the normalized centroid of the identity, as shown in the algorithm. Otherwise, it becomes directly the representative centroid of a new identity. The time complexity of the algorithm is $O(MC)$, where M is the i-vector dimensionality and C is the number of active identities.

Alternative registration and updating strategies might be considered, for example, a set of i-vectors per identity represented by their median for the comparisons (which theoretically is more robust to outliers than the running average). However, we register each individual with only a single i-vector and update it with the running average strategy. This ensures the minimal amount of information is stored, which is critical to handle large scales. In addition, according to our experiments, it is less susceptible to noise, i.e. erroneous faces that have reached the clustering stage.

The RAM memory is the fastest option to store these data but it is not safe to system failures (e.g. power supply failures). To avoid data loss, the system also stores the registration information in a persistent database, to use it as an information cache if the system fails or needs to be restarted. Additional spatiotemporal constraints related to characteristics of the infrastructure (e.g. one-way zone-to-zone doors, or one-way exit doors that assure that an individual has left the infrastructure at least for a certain time, etc.), allow reducing the number of comparisons and, in consequence, potential erroneous matches.

4 MSDB OPTIMIZATIONS

4.1 Multi-Batch MTCNN

To improve the computation performance of the face and landmark detections from full-images in the first stage of MSDB, we adopt and modify the MTCNN method (Zhang et al., 2016a), resulting in the proposed MB-MTCNN approach, which includes multi-batch processing and a series of parallelization strategies on each stage of MTCNN.

We chose MTCNN as the detection network because of some remarkable characteristics. First, it performs a multi-scale search, generating candidate face regions at several image scales. This allows us to parallelize the generation of candidates by image scale, and also allows configuring the scales to optimize the inference time. Furthermore, it provides face regions and landmarks in a single forward pass, sharing features between both kinds of detections and saving inference time. Finally, it needs a small size in memory, allowing us to use more limited hardware resources.

MTCNN has three stages. The first one processes the input image at different scales, and generates facial region candidates using a convolutional neural network (CNN) called *Proposal Network* (P-Net). The second stage refines the candidate regions using a CNN called *Refinement Network* (R-Net). The third stage refines the facial regions generated by R-Net, and detects five landmarks inside each of them using a CNN called *Output-Network* (O-Net).

In MB-MTCNN, the key factor to accelerate this process is the inclusion of parallel while loops, built upon flexible and expressive control-flow primitive operators (TensorFlow, 2017), executed in *execution frames* of the GPU that can be nested, allowing further optimizations. Thus, in the first parallel while loop of MB-MTCNN, the images are scaled and processed in batch using P-Net to obtain the region candidates. With a nested parallel while loop, the candidates of each scale and image are postprocessed following the original implementation; scaled to their real size, refined and filtered with a non-maximum suppression (NMS). Finally, the candidates from all the scales are grouped by image in batch and NMS is applied again to merge candidates of different scales using a parallel while loop. In the subsequent stages, the candidates are filtered and refined using R-Net and O-Net networks. In these stages, the batches of candidates from each image are processed in parallel. This is more efficient than processing all the candidates in one batch, as it avoids reallocating the candidates for preprocessing and postprocessing.

4.2 Efficient Dynamic Batching for Facial Images Processing

The number of cropped facial images that reach stages 2 and 3 of MSDB is unknown and can vary in every iteration. Different dynamic batching methods can be considered, which depend on the used GPU architecture and DNN deployment tool. We focus on NVIDIA GPUs and Google's TensorFlow and NVIDIA's TensorRT frameworks, due to their suitability for the inference of DNNs (Yadwadkar et al., 2019).

The naive dynamic batching approach consists in putting the whole batch of candidates directly into the network, varying the network's input size in every iteration. In TensorFlow constantly varying the batch size produces important time overheads, as the network needs to be re-adapted for the new size. TensorRT, optimizes the network for a given batch size, allowing handling smaller sizes with a drop in performance. Nevertheless, in both cases there are time overheads proportional to the network complexity.

Those time overheads due to network re-adaptation can be avoided using parallel loops, built upon flexible and expressive control-flow primitive operators (Agarwal, 2019; Radul et al., 2019). For instance, one can set the input size of the network to a certain value, so the original input batch is divided into mini-batches of that size, and process these mini-batches with TensorFlow's parallel while loop (TensorFlow, 2017). The mini-batch size must be set manually taking into account the expected minimum, maximum, and average batch sizes. Nevertheless, if there are big variations in the number of candidates, the number of mini-batches may be too high for a single parallelization, causing a drop in performance.

For this reason, our proposal for stages 2 and 3 of MSDB is to load and infer multiple instances of the network, optimized for different batch sizes. Thus, for each inference, we divide the input batch size in the minimum number of mini-batches that fit the different network sizes. In order to require less GPU memory to load and infer each network, we recommend applying optimizations to the trained network such as layer fusion, kernel auto-tuning, dynamic tensor memory and multi-stream execution.

5 EXPERIMENTAL RESULTS

In this section, we present the results of two groups of experiments conducted to evaluate the proposed clustering algorithm (ODIVC) and VSS. The first group aims to measure the performance of ODIVC in terms of accuracy and processing time, comparing it with other state-of-the-art offline clustering methods. The second group focuses on testing the performance and scalability of the proposed VSS and the impact of the optimizations presented in Section 4.

The server used for the experiments has the following specifications: 1 processor Intel XeonTM E5-1650v4, 1 GPU NVIDIA Quadro P6000 and 2 RAM modules, each one of 6GB DDR4 2400MHz.

5.1 Face Clustering Performance

For the first experiment, we select the IJB-B dataset (Whitelam et al., 2017), a well-known dataset of unconstrained in-the-wild face images. This dataset includes a clustering protocol consisting of seven subtasks that vary in the number of identities and the number of faces. We select the last subtask, as it is the most challenging one, with the highest number of identities (1,845) and faces (68,195). The performance is measured using BCubed F-Measure metric, following the recomendations in (Amigó et al., 2009).

Table 1: Comparison with baseline methods in terms of BCubed F-Measure and processing time using IJB-B-1845. Superscript* denotes results reported from original papers, otherwise it uses the i-vectors from (Wang et al., 2019).

Method	F-Meas	Time
ARO (Otto et al., 2018)	0.755	00:01:13
PAHC* (Lin et al., 2018)	0.61	00:03:56
ConPaC* (Shi et al., 2018)	0.634	02:53:58
DDC (Lin et al., 2018)	0.800	00:05:32
GCN (Wang et al., 2019)	**0.814**	00:06:03
ODIVC (ours)	0.778	**00:00:28**

Table 2: Comparison with baseline methods in terms of BCubed F-Measure and processing time using IJB-C dataset. All methods use the same i-vectors extracted from our VSS and the same hardware.

Method	F-Meas	Time
ARO (Otto et al., 2018)	0.768	00:09:39
GCN (Wang et al., 2019)	0.890	00:10:32
ODIVC (ours)	**0.931**	**00:00:52**

Since we want to demonstrate that ODIVC is independent of the recognition model used and a fair comparison with other methods, the same vectors used in (Wang et al., 2019) are selected for this experiment. These vectors have 512 dimensions. We adjust the parameters of ODIVC empirically: $thresh_{low}$ = 0.38, $thresh_{high}$ = 0.4 and nm_{min} = 4.

The results of the experiment are presented in Table 1. It can be observed that our method achieves the third position in terms of F-Measure, but outperforms the rest of the methods considering the processing time. For instance, it runs 12 times faster than GCN-A under the same hardware conditions.

In the second experiment we test the performance of ODIVC using a different face recognition model and a different dataset. We select the face recognition model used by our VSS, described in Section 3.2. We use IJB-C (Maze et al., 2018), another well-known dataset of unconstrained face images, with 3531 identities and 140623 faces. We use MB-MTCNN for the face and landmarks detection. In order to obtain better quality feature vectors, we filter faces with less than 45 pixels per side and we normalize the patches as described in Section 3.2. After filtering, 120661 vectors belonging to 3529 identities are extracted.

We compare our algorithm with the two best state-of-the-art methods: GCN (Wang et al., 2019), which achieved the highest accuracy in the first experiment, and ARO (Otto et al., 2018), which achieved the best trade off between accuracy and speed (without considering ours). For both methods, we adjust the parameters in order to achieve the best performance. Furthermore, for GCN, we retrain the network follow-

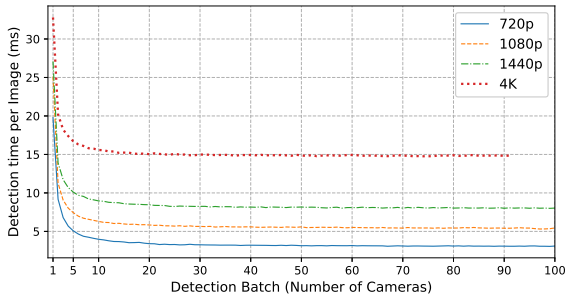

Figure 3: Detection time per image for different resolutions and batch sizes in MB-MTCNN (in stage 1 of MSDB), compared to MTCNN (batch=1).

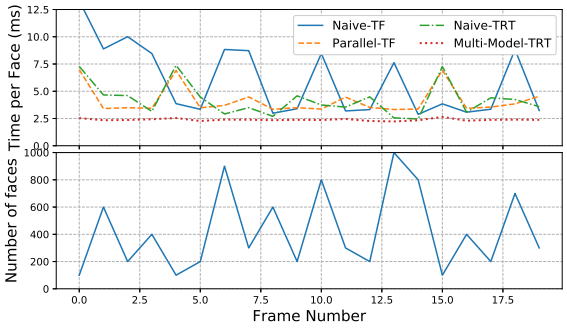

Figure 4: Average time per face of the dynamic batching procedure for stages 2 and 3 of MSDB, compared to alternative state-of-the-art approaches.

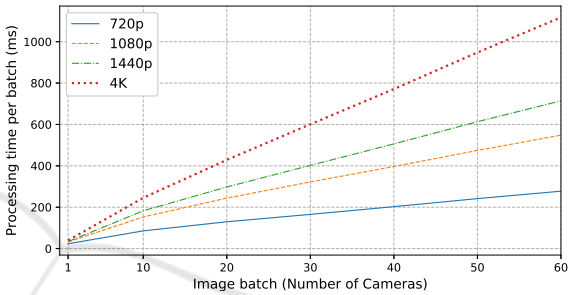

Figure 5: Average times per batch (for the image processing thread) with the considered setup for different resolutions.

ing recommendations in (Wang et al., 2019). Finally, we tune the parameters of ODIVC: $thresh_{low}$ = 0.37, $thresh_{high}$ = 0.4 and nm_{min} = 4.

The results of this experiment, presented in Table 2, show that our method outperform the others in terms of F-Measure and processing time. Under the same hardware conditions ODIVC runs more than 12 times faster than GCN and more than 11 times faster than ARO. These time ratios show that our method is more scalable than the others. In this experiment ODIVC achieves a better F-Measure than GCN. We believe this is because we have reduced the number of outliers by filtering the smallest faces and using by a better face recognition network. Therefore, ODIVC is more sensible to outliers, but it is more accurate when using robust face representations.

5.2 Video Surveillance System Performance

We start evaluating the performance of MB-MTCNN compared to the original implementation. Figure 3 shows the average time of the detection stage per image for different resolutions and batch sizes using our approach (MTCNN corresponds to batch=1). The results show a great reduction in the processing time per image (more than 5 times for 720p). This reduction increases with the batch size but reaches a saturation point due to hardware limitations.

We also test the performance of the proposed dynamic batching procedure compared to the alternatives mentioned in Section 4.2. We measure the recognition time per face when the VSS is processing a sequence of images containing variable numbers of faces. To better visualize time variations, we augment the number of faces by a factor of 10, so they may vary from 10 to approximately 1000. The results are shown in Figure 4, where TF stands for TensorFlow and TRT for TensorRT. The mini-batch size selected for the Parallel-Loop-TF approach is 20 and those used for Multi-Instance-TRT are 400, 200, 100, and 20. It can be observed that the Multi-Instance-TRT outperforms the other approaches, not only in the average but also in the maximum peak times.

Finally, to test the potential scalability of our VSS, we run it to process images captured from a scaling number of videos, at different resolutions and with a detection batch size set with the same value as the number of videos. Then, we measure the average times per image batch in the image processing thread, the main bottleneck of the system. The results are shown in Figure 5. In our context, it is enough to deliver the PIs with near-real time performance. Hence, if we consider acceptable that the processing thread responds every 200 ms, this setup could theoretically be scaled up to 40 720p cameras. The results reveal that the proposed VSS allows designing cost-effective GPU-server-based solutions for our purpose.

6 CONCLUSIONS

In this work, we have presented a cost-effective VSS for face recognition and online clustering of unknown individuals at large scale, without storing their biometric information, in order to obtain PIs for people flow monitoring in large infrastructures. The

VSS is composed of three main computing threads executed asynchronously, using CPU and/or GPU capabilities and sharing data sequentially. Experimental results with challenging scenarios reveal the high effectiveness and scalability of the proposed approach. Furthermore, we have presented an online and unsupervised clustering approach (ODIVC), which achieves state-of-the-art results in well-known large-scale datasets, with a reduced computational cost compared to the alternatives.

Future work will focus on extending our VSS to distributed computing infrastructures, with heterogeneous hardware as nodes of the VSS, including GPU computing servers that process and share data for video-surveillance purposes.

REFERENCES

Agarwal, A. (2019). Static automatic batching in TensorFlow. In *36th ICML*, volume 97, pages 92–101. PMLR.

Amigó, E., Gonzalo, J., Artiles, J., and Verdejo, M. (2009). Amigó e, gonzalo j, artiles j et ala comparison of extrinsic clustering evaluation metrics based on formal constraints. inform retriev 12:461-486. *Information Retrieval*, 12:461–486.

Chaudhari, S. T. and Kale, A. (2010). Face normalization: Enhancing face recognition. In *2010 3rd ICETET*, pages 520–525.

Chen, J., Li, K., Deng, Q., Li, K., and Yu, P. S. (2019). Distributed deep learning model for intelligent video surveillance systems with edge computing. *IEEE Trans. on Industrial Informatics*, pages 1–1.

Deng, J., Guo, J., Xue, N., and Zafeiriou, S. (2019). Arcface: Additive angular margin loss for deep face recognition. In *2019 IEEE CVPR*, pages 4685–4694.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE CVPR*, pages 770–778.

Jankar, J. R. and Shah, S. K. (2017). Computational analysis of hybrid high efficiency video encoders. In *ICISS*, pages 250–255.

Lim, K.-S., Lee, S.-H., Han, J. W., and Kim, G. W. (2018). Design considerations for an intelligent video surveillance system using cloud computing. In *PDCAT*, pages 84–89.

Lin, W., Chen, J., Castillo, C. D., and Chellappa, R. (2018). Deep density clustering of unconstrained faces. In *2018 IEEE/CVF CVPR*, pages 8128–8137.

Lloyd, S. P. (1982). Least squares quantization in pcm. *IEEE Trans. Inf. Theory*, 28:129–136.

Mayer, C. A., Felkel, R., and Peterson, K. (2015). Best practice on automated passenger flow measurement solutions. In *Journal of Airport Management*, volume 9, pages 144–153.

Maze, B., Adams, J., Duncan, J. A., Kalka, N., Miller, T., Otto, C., Jain, A. K., Niggel, W. T., Anderson, J., Cheney, J., and Grother, P. (2018). Iarpa janus benchmark - c: Face dataset and protocol. In *2018 ICB*, pages 158–165.

Nasir, M., Muhammad, K., Lloret, J., Kumar, A., and Sajjad, M. (2018). Fog computing enabled cost-effective distributed summarization of surveillance videos for smart cities. *Journal of Parallel and Distributed Computing*, 126.

Otto, C., Wang, D., and Jain, A. K. (2018). Clustering millions of faces by identity. *IEEE TPAMI*, 40(2):289–303.

Radul, A., Patton, B., Maclaurin, D., Hoffman, M. D., and Saurous, R. A. (2019). Automatically batching control-intensive programs for modern accelerators. *ArXiv*, abs/1910.11141.

Shi, J. and Malik, J. (2000). Normalized cuts and image segmentation. *IEEE TPAMI*, 22:888–905.

Shi, Y., Otto, C., and Jain, A. K. (2018). Face clustering: Representation and pairwise constraints. *IEEE Transactions on Information Forensics and Security*, 13(7):1626–1640.

Tan, X. and Triggs, B. (2010). Enhanced local texture feature sets for face recognition under difficult lighting conditions. *IEEE Transactions on Image Processing*, 19(6):1635–1650.

TensorFlow, A. (2017). Implementation of control flow in tensorflow. *TensorFlow Whitepaper*.

Tsakanikas, V. and Dagiuklas, T. (2017). Video surveillance systems-current status and future trends. *Computers & Electrical Engineering*, 70.

Wang, H., Wang, Y., Zhou, Z., Ji, X., Gong, D., Zhou, J., Li, Z., and Liu, W. (2018). Cosface: Large margin cosine loss for deep face recognition. In *2018 IEEE/CVF CVPR*, pages 5265–5274.

Wang, Z., Zheng, L., Li, Y., and Wang, S. (2019). Linkage based face clustering via graph convolution network. *2019 IEEE/CVF CVPR*, pages 1117–1125.

Whitelam, C., Taborsky, E., Blanton, A., Maze, B., Adams, J., Miller, T., Kalka, N., Jain, A. K., Duncan, J. A., Allen, K., Cheney, J., and Grother, P. (2017). Iarpa janus benchmark-b face dataset. In *2017 IEEE CVPRW*, pages 592–600.

Yadwadkar, N. J., Romero, F., Li, Q., and Kozyrakis, C. (2019). A case for managed and model-less inference serving. In *HotOS '19*, pages 184–191.

Zhang, K., Zhang, Z., Li, Z., and Qiao, Y. (2016a). Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10):1499–1503.

Zhang, Z., Luo, P., Loy, C., and Tang, X. (2016b). Learning deep representation for face alignment with auxiliary attributes. *IEEE TPAMI*, 38(5):918–930.