SaFace: Towards Scenario-aware Face Recognition via Edge Computing System

Zhe Zhou [*1 2], Bingzhe Wu[*1], Zheng Liang[1], Guangyu Sun[1 2 †], Chenren Xu[1], Guojie Luo[1 2]
[1] *Peking University*
[2] *Advanced Institute of Information Technology, Peking University*

Abstract

Deep Convolutional Neural Networks (CNNs) have achieved remarkable progress in the field of face recognition (FR). However, developing a robust FR system in the real-world is still challenging due to vast variance of illumination, visual quality, and camera angles in different scenarios. These factors may result in significant accuracy drop, if the pre-trained model doesn't have perfect generalization ability. To mitigate this issue, we present a solution named SAFACE, which helps to improve FR accuracy through unsupervised online-learning in an edge computing system. Specifically, we propose a novel scenario-aware FR flow, then decouple the flow into different phases and map each of them to different levels of a three-layer edge computing system. For evaluation, we implement a prototype and demonstrate its advantages in both improving recognition accuracy and reducing processing latency.

1 Introduction

In recent years, deep learning algorithms have revolutionized the field of face recognition (FR), attributing to the emergence of powerful deep CNN architectures [3, 9, 12] delicate loss functions [3, 7, 8, 20], and large scale (public) face datasets [6, 11, 21]. However, deploying these FR models in the real-world is still challenging. Recent studies have shown that those FR algorithms trained with static datasets will suffer significant performance degradation when applied to real-world scenarios [4, 10]. This is caused by vast variance of face poses, illumination and visual quality of the captured pictures in different scenarios, as well as the limited generalization ability of FR models trained with static datasets [10]. A straightforward solution is to collect more training data from the target scenario and then fine-tune the FR models. Unfortunately, this method will introduce several issues: First, the procedure of collecting and labeling training data is both technically challenging and labor-intensive. Second, as data volume increases infinitely, it seems to be impossible to manage the data and build/train a model atop to best fit every scenario to deliver seamless service. These issues altogether suggest that such a solution cannot scale in practice.

To tackle these challenges, we present SAFACE, a practical face recognition system. It employs the idea of unsupervised online learning, which can gradually fine-tune the FR model and improve its accuracy in the targeted scenarios. It is realized by a novel computing flow to unify the inference (face detection and recognition) and online model training. SAFACE performs face detection, verification, and tracking simultaneously. By leveraging implicit information from continuous images in time series, SAFACE learns the discriminative features to retrain the FR model in an unsupervised manner with such online data.

From the deployment perspective, SAFACE borrows edge computing paradigm [15, 19, 23] as the substrate to natively solve the scalability issue. We decouple the scenario-aware FR flow into different phases and map each of them to different levels of a three-layer edge computing system. We further introduce the following optimization techniques to improve efficiency. First, we use a strategy only to fine-tune a portion of the FR model. This strategy can improve training efficiency, without sacrificing the accuracy and latency. Second, we propose a context-aware scheduling strategy based on the flow density of persons (faces). This strategy is capable of coordinating the FR inference and online training tasks so that both high inference throughput and high training efficiency are promised. For evaluation,we implement a prototype and demonstrate its advantages in both improving recognition accuracy and reducing processing latency.

2 Basic FR Flow

We start with the typical open-set FR problem. In a standard surveillance process, we first employ a face detection model **FD** to detect the **probes** (the faces that need to be recognized) in a frame. Then, these probes are aligned based on the facial landmarks obtained from a face aligning model **FA**. We

*Equal contribution †Corresponding author

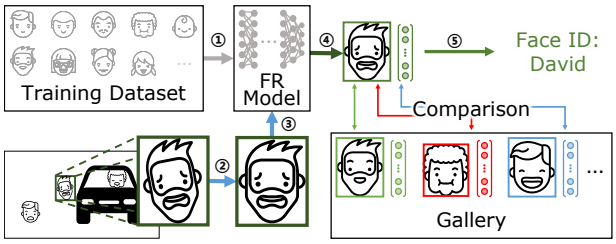


Figure 1: Illustration of a basic FR flow: step① FR model training, step ② face detection and alignment, step ③ feeding probes into FR model, step ④ extracting face representations, and step ⑤ comparing and determining the identity.

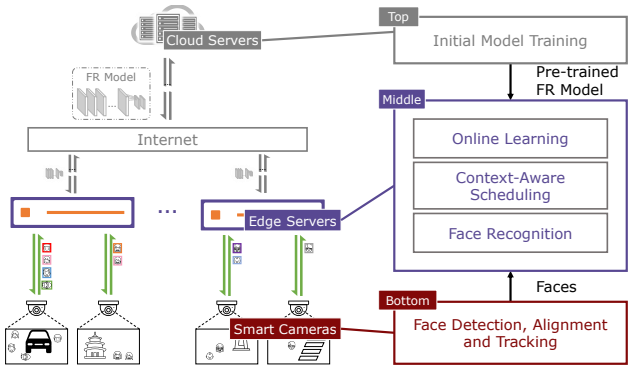


Figure 2: SAFACE overview.

denote $\mathbf{p}$ as the aligned version of a probe captured from the original frame. Formally, the aligned probes in $\mathbf{I}_t$ can be obtained by $\{\mathbf{p}_i\} = \mathbf{FA} \circ \mathbf{FD}(\mathbf{I}_t)$. After that, we can feed these aligned face images into the **FR** model to calculate their face representations, denoted as $\{\mathbf{x}_i\}$. At last, we compare the representations with those pre-registered in the gallery and determine their identities. The pipeline for open-set face recognition is also illustrated in Figure 1.

3 System Overview

The overall structure of the SAFACE is depicted in Figure 2. On the right side, the flow of SAFACE consists of three parts. The first one (top) is initializing and training a model using a public dataset. This is a one-time step, which is performed on a cloud server. Then, this pre-trained model is sent to edge servers at the starting point for FR inference. As addressed before, the performance may be degraded in different scenarios. The second part (middle) of SAFACE is responsible for FR inferences and online learning tasks, which are deployed on edge servers. The middle piece in this part is a context-aware scheduler. It is responsible for run-time adjusting computation resource utilization of on-line learning tasks. The goal is to avoid interfering the FR inference tasks. The input data for both FR inference and online learning tasks come from the last part (bottom), which is deployed on each smart camera. This part is responsible for face detection, alignment, and tracking tasks. It generates probes and extracts related timing information from live video streams and send them to edge servers.

The partition and mapping of SAFACE flow highly depends on the computation and communication characteristics of each sub-tasks. Face detection [24], alignment [17,24], and tracking are considered as light-weight tasks, which can be processed directly on smart cameras. It is also called *scenario-shared stage* of the flow. On the contrary, the middle part running on edge servers are called *scenario-aware stage*. The original FR model is adjusted into different versions by online learning, according to the specific environment of each camera. The reasons why we choose edge servers to deploy *scenario-aware stage*, rather than cloud servers are that, online learning algorithm involves significant extra data transmission between cameras and servers. This is not scalable as the number of surveillance cameras increases with a speed of 50 million per year [18]. In addition, the communication latency between surveillance cameras and the remote cloud server is relative high (typically range from 50 to 200 ms). It is not acceptable for some real-time FR scenarios. Using edge servers can efficiently solve these problems. Finally, since the pre-traning of FR model demands tremendous computing resources [7, 8] and only needs to be done once, we can naturally deploy it in cloud servers.

4 FR Flow of SaFace

In this section, we depict our scenario-aware FR flow. It can dynamically adjust a pre-trained FR model through unsupervised face representation learning. In order to achieve this, we divide the traditional flow into two stages, namely, *scenario-shared stage* and *scenario-aware stage*. Illustration of this scenario-aware FR flow is shown in Figure 3, which is introduced in detail in the rest of this section.

4.1 Scenario-shared Stage

The *scenario-shared stage* contains all essential steps in a traditional one. Besides, a face tracking step is introduced to leverage implicit information contained in live video streams. In fact, a probe with the same identity may appear in several consecutive frames of a stream. Thus, we use a face tracking algorithm to obtain the track associated with one specific identity. In this paper, we propose to track the faces by detection, as introduced in the prior study [2, 25]. All steps of this stage are shown in Figure 3.

Given a probe $\mathbf{p}$ (associated with a face box) in the t-th frame $\mathbf{I}_t$,we firstly extract a set of face boxes in the next frame $\mathbf{I}_{t+1}$. The overlap between these boxes and $\mathbf{p}$ should be larger than a given threshold. Then, the face box with the maximum overlap is selected into the track. The tracking is terminated when there is no matched face in the next frame. Specifically, to reduce the noises in the face tracking caused by false alarm, we set a minimum length (*10 in our design*)

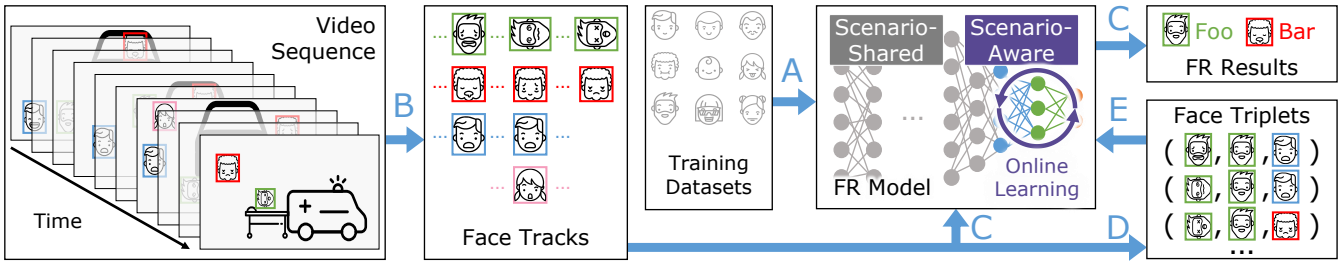


Figure 3: Detailed flow of SAFACE: (A) model pre-training, (B) face detection& tracking, (C) FR inference, (D) triplet generation, and (E) online learning.

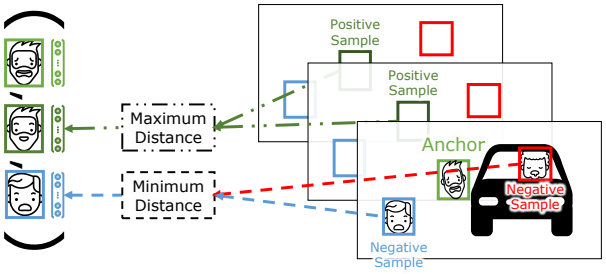


Figure 4: Example of triplets generation.

for each track. We ignore the tracks that are shorter than this threshold. These tracks are used for online training in next stage. Note that in some very crowded scenes, the risk of tracking wrong faces may increase. Adopting more advanced tracking methods [1, 14] will mitigate this risk. In our test cases, we find that simply using tracking-by-detection is good enough.

4.2 Scenario-aware Stage

In *scenario-aware stage*, we use the probe tracks from the last stage to incrementally improve the FR model with online learning. Motivated by prior works [3, 25], we adopt triplet loss to fine-tune the FR model. To introduce the way to leverage triplet loss for online learning, we first describe the triplets generation procedure with the face tracks.

4.2.1 Triplet Generation

A triplet consists of three probes, namely, an *anchor probe*, a *negative probe*, and a *positive probe*. For each track, we first select a probe as the anchor probe. Then, we go through all other probes in the same frame containing the anchor probe. Obviously, the identities of these probes are different from that of the anchor probe. Thus, they are named as negative samples in this work. The feature distance between the anchor probe and negative samples is calculated. The *negative probe* is the negative sample that has the minimum distance from the anchor probe. Similarly, we go through all other probes in the same track, which are supposed to be positive samples. The positive sample having the maximum distance with the anchor probe is selected as the *positive probe* in the triplet. An example is illustrated in Figure 4.

Algorithm 1: Face triplets generation

Input:
  Face tracks T
  Frame number $= N_v$
  Face triplets $P = \{\}$
for $t \leftarrow 0$ *to* $N_v - 1$ **do**
  for $\mathbf{p}_a$ *in* T_t **do**
    $\mathbf{p}_p = \arg\max_{\mathbf{p}_k}(\{dist(\mathbf{FR}(\mathbf{p}_a), \mathbf{FR}(\mathbf{p}_k)) : \mathbf{p}_k \in T(\mathbf{p}_a) \wedge \mathbf{p}_k \neq \mathbf{p}_a\})$
    $\mathbf{p}_n = \arg\min_{\mathbf{p}_k}(\{dist(\mathbf{FR}(\mathbf{p}_a), \mathbf{FR}(\mathbf{p}_k)) : \mathbf{p}_k \in \mathbf{I}_t \wedge \mathbf{p}_k \neq \mathbf{p}_n\})$
    P.push($\langle \mathbf{p}_a, \mathbf{p}_p, \mathbf{p_n} \rangle$)
  end
end
return P

Note that, if there is only one probe (i.e., anchor probe) in a frame, we cannot find the corresponding negative probe for this anchor probe. Thus, we skip this anchor probe. For each track, we need to extract all triplets. The corresponding process is listed in Algorithm 1. Each triplet is represented as $< \mathbf{p}_a, \mathbf{p}_p, \mathbf{p}_n >$. These generated triplets are used for providing the supervised signal for the online learning, which is introduced as follows.

4.2.2 Online Training

We adopt the triplet-loss function [8] to learn from the generated triplets. In the context of training deep CNN model, the common practice is to train on all the weight parameters, but in our case, it suffices to adjust a part of the parameters of the FR model. To be specific, we divide the **FR** CNN model into two separated parts: scenario-shared extractor and scenario-aware learner, as shown in Figure 3. During the online learning phase, the parameters in the scenario-shared extractor are fixed once the initialized FR model is deployed. All scenarios share the parameters in this part. On the other hand, to learn dedicated face representations from a specific scenario, the SGD is performed on the parameters in the scenario-aware learner.

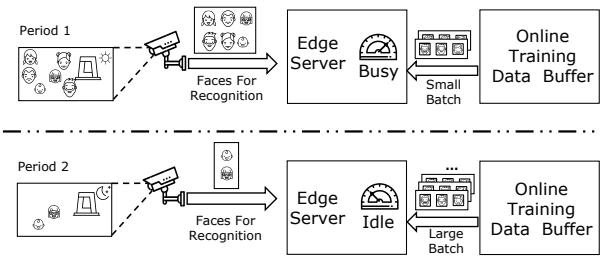


Figure 5: Context-aware Scheduling

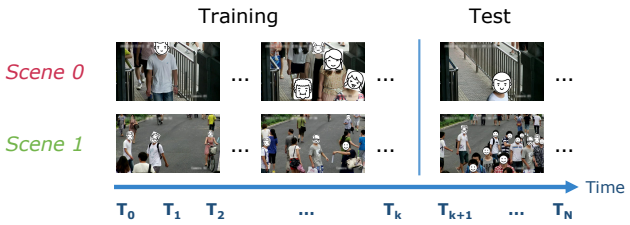


Figure 6: Dataset visualization. (The faces are intentionally covered in consideration of privacy)

4.3 Context-aware Scheduling

When deploying SAFACE on an edge computing system, both FR model inferences and online training tasks are assigned to edge servers. Each edge server is responsible for multiple cameras, as shown in Figure 2. Obviously, the original FR tasks should be given a higher priority to guarantee real-time responses. The online learning tasks, on the contrary, are performed incrementally without any real-time requirement. Thus, the online learning tasks should not interfere with the FR tasks and only leverage the idle resources on edge servers. Thus, we propose a context-aware scheduling method for online training tasks.

In practice, the number of video frames are sampled in a fixed rate (e.g. 25fps) at each surveillance camera. This rate is denoted as R_C. Assume that the maximum number of cameras that can be connected to an edge server is denoted as N_C. We assume that the maximum number of probes contained in a frame is limited as $N_{P_{max}}$. Thus, the total number of probes that can be processed on an edge server in a time interval $\Delta t = 1/R_C$ is defined as its computation capability, denoted as N_E, which should satisfy the following requirement,

$$N_E \geq N_C \times N_{P_{max}} \tag{1}$$

Since the incoming rates of probes vary a lot in a real-work application, the pivot is to select a proper number (batch size) of triplets for training in a time interval $\Delta t = 1/R_C$. It should adjust according to the run-time computation utilization occupied by FR. An example is illustrated in Figure 5. In the daytime, the incoming rate is high so that a small batch size is used for online training. During the night, a larger batch size is used.

The maximum batch size that can be processed in a time interval can be obtained in advance. This is denoted as B_{max}. Then, the run-time batch size B_t can be calculated with the following equation,

$$B^t = \max(0, B_{max} \times (1 - \alpha \frac{\sum_{i=1}^{N_C} N_{P_i}}{N_E})) \tag{2}$$

The term $\frac{\sum_{i=1}^{N_C} N_{P_i}}{N_E}$ represents the computation utilization by current FR model processing, where $\sum_{i=1}^{N_C} N_{P_i}$ calculate the total number of probes from all cameras in this time interval. Parameter α is a pre-defined coefficient to adjust so-called effective computation utilization. It accounts for the overhead caused by task switching, data movement, etc. This is a scenario dependent parameter, which should be adjusted in deployment.

5 Prototype

For evaluation, we implement a prototype. We adopt several models of InsightFace [3] to serve as the baseline FR models. To simulate the edge computing system, We adopt Hisilicon Hi3516CV500 IP Camera to serve as the camera node, whose computation power is sufficient to conduct real-time face detection and tracking. The edge node is emulated with a desktop PC equipped with an Intel i7-6700k CPU and Nvidia GTX1080 GPU. For communication between the edge server and camera nodes, we use a TP-Link WDR5620 router through a 100Mbps Ethernet cable. The round trip time between camera nodes and the edge server is less than 20ms. The cloud side is a powerful GPU server with 4×GTX 1080TI GPU.

We build the training and testing datasets based on a private dataset used in [16], which is obtained by the real surveillance cameras located in two different scenarios. As shown in Figure 6, Scene0 and Scene1 have different camera angles, illumination and face resolution. We split each video sequence into two time-independent parts, one for training and another for test. The ratio of training and test frames is 5 to 1. Note that the test data is manually labeled, while the training data has no labels, and our system learns with it through unsupervised manner.

6 Evaluation Results

In this section we present some preliminary results that demonstrate the efficiency of our designed FR system, including the accuracy improvement, speedup of partial fine-tuning and the throughput improvement of context-aware scheduling.

6.1 Accuracy Improvement

As introduced before, for each scenario there is a training dataset and a testing dataset based on the experiment settings. For unsupervised online learning, we fine-tune the initialized

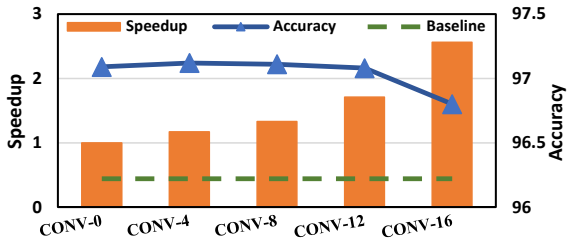


Figure 7: Speed-accuracy trade-off (#Scenario1)

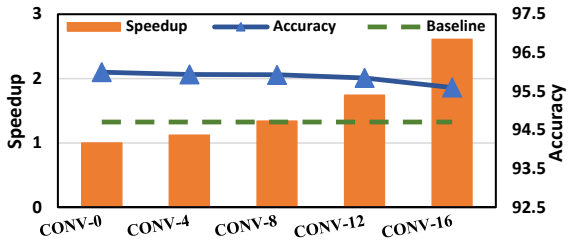


Figure 8: Speed-accuracy trade-off (#Scenario2)

model via the proposed flow using video streams from the training dataset of the specific scenario. The verification accuracy on the testing dataset is used to measure the performance of the FR model quantitatively. We perform experiments on three different FR models with different CNN models. The overall results are shown in Table 1.

As the results show, there is a considerable performance gap between the research benchmark and the benchmark built on real surveillance data. For example, ResNet50 achieves a superior accuracy of 99.80% on LFW [3]. There is an obvious drop when applying this model to our surveillance images. Specifically, ResNet50 achieves an accuracy of 96.74% on *Scenario*1 and 95.62% on *Scenario*2, respectively. With the help of the proposed flow, ResNet50 fine-tuned using online learning (denoted as After) has obviously improved the accuracy. It indicates that the fine-tuned model has a better generalization in the real scenario than the original FR model.

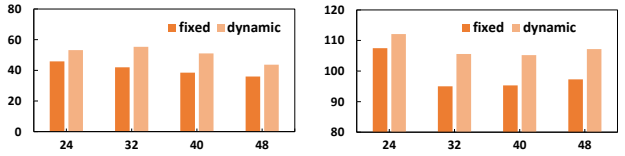
6.2 Partial Fine-tuning

Motivated by previous practice in transfer learning [5, 22], we introduce a strategy in Section 4, which is to fine-tune a part of parameters while freezing others in the phase of online learning. Here, we quantitatively show how the strategy affects the performance of the online learning. To this end, we conduct experiments to fine-tune different parts of the *Sphere-20* network and show how the performance changes.

Figure 7 and Figure 8 show the speed-accuracy trade-off in the two scenarios. We can easily find that in *Sphere-20* model, even if we freeze he first 12 layers in the CNN model and only fine-tune the last 8 layers, the FR accuracy drops a little compared to fine-tuning the whole model. However it achieves about 1.8x speed up. If we further freeze more layers, say 16 layers, the accuracy drops a lot in both two scenarios.

Table 1: Face verification accuracy (%).

Model	*Scenario*1		*Scenario*2	
	Before	After	Before	After
MobileNet	95.70	**96.12**	92.69	**93.51**
Sphere20	96.22	**97.13**	94.71	**96.20**
ResNet50	96.74	**97.33**	95.62	**96.43**



(a) Scenario1 (b) Scenario2

Figure 9: The comparison of context-aware scheduling (denoted as dynamic) and the strategy that uses fixed batch size (denoted as fixed). The x-axis is the fixed batch size, while the y-axis represents the throughput (triplets/min)

6.3 Context-aware Scheduling

As mentioned in Section 4, we introduce a context-aware scheduling strategy (denoted as `dynamic`) to coordinate the online learning and inference tasks. In this part, we validate the effectiveness of this strategy on both benchmarks compared with a fixed strategy, which uses a fixed training batch size (denoted as `fixed`). For the `fixed` strategy, we test different batch sizes. Note that using different batch size leads to different training speed of the `fixed` strategy. Therefore, for fair comparison, we adjust *alpha* in Equation 2 to ensure `dynamic` strategy has a similar training speed to `fixed` strategy. The results are shown in Figure 9.

As shown in results, our `dynamic` strategy achieves consistent improvements (measured as throughput) on different batch sizes, also in different scenarios. The throughput in our context is defined as the number of probes that the system can process in one second. For example, given a batch size of 40 for `fixed` strategy in *scenario1*, `dynamic` achieves a throughput of 51.03 probes per second while `fixed` achieves the throughput of 38.52 probes per second. Our strategy shows a 32.4% relative improvements. All these results indicate that our `dynamic` strategy can achieve better inference throughput while at a high training efficiency compared with the `fixed` strategy.

7 Conclusion

In this paper we introduce SAFACE. It exploits implicit timing series information from live video to perform un-supervised online training. It can improve performance of the FR model according to the real environment of a scenario. In addition, SAFACE deploys the whole algorithm flow on an edge computing system rather than a centric cloud system. With further assistance of a run-time task scheduler, efficient execution of both FR inference and online training tasks can be promised.

8 Discussion

• **Generality of SAFACE.** In this work, we focus on face recognition problems. However, we believe that SAFACE can be generalized to support many other tasks such as person re-identification tasks [13, 26]. Since they share similar dataflow with CNN based face recognition. More experiments need to be done to confirm this conjecture.
• **Better Offloading Strategy.** SaFace currently adopts an offloading strategy which allocates the face detection/alignment to the IoT device and offloads other tasks into edge nodes. This method reduces the average face detection and recognition latency from about 120ms to less than 60ms in our prototype. Theoretically, we can build better analytic model to achieve optimal latency based on the current states of the edge and its managing IoT node. We mark this as future work.
• **Different Training Modes.** For online-learning, there are two different modes. 1) Always-on mode: SAFACE collects triplets all the time and performs fine-tuning immediately after a batch of training data is prepared. 2) Periodical training mode: SAFACE periodically fine-tune the FR model. It collects triplets, stores it, and then fine-tunes the model every one or two hours. The differences of the two modes with respect to performance/accuracy remain to be discussed.
• **Evaluate in More Realistic Scenarios.** For fast evaluation, we use two video sequences to evaluate SAFACE. It is better to deploy SAFACE in a more realistic scenarios to evaluate its effectiveness, stability and scalability, etc. More works remain to be done to make SAFACE a more practical system for actual use.

References

[1] Luca Bertinetto, Jack Valmadre, Joao F Henriques, Andrea Vedaldi, and Philip HS Torr. Fully-convolutional siamese networks for object tracking. In *European conference on computer vision*, pages 850–865. Springer, 2016.

[2] Samyak Datta, Gaurav Sharma, and CV Jawahar. Unsupervised learning of face representations. In *Automatic Face & Gesture Recognition (FG 2018), 2018 13th IEEE International Conference on*, pages 135–142. IEEE, 2018.

[3] Jiankang Deng, Jia Guo, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. *arXiv preprint arXiv:1801.07698*, 2018.

[4] Changxing Ding and Dacheng Tao. Trunk-branch ensemble convolutional neural networks for video-based face recognition. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[5] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, JMLR Workshop and Conference Proceedings, 2014.

[6] Cao et al. Vggface2: A dataset for recognising faces across pose and age. In *Automatic Face & Gesture Recognition*, 2018.

[7] Liu et al. Sphereface: Deep hypersphere embedding for face recognition. In *CVPR*, 2017.

[8] Schroff et al. Facenet: A unified embedding for face recognition and clustering. In *CVPR*, 2015.

[9] Taigman et al. Deepface: Closing the gap to human-level performance in face verification. In *CVPR*, 2014.

[10] Wang et al. Face recognition in real-world surveillance videos with deep learning method. In *ICIVC*, 2017.

[11] Yandong Guo, Lei Zhang, Yuxiao Hu, Xiaodong He, and Jianfeng Gao. MS-Celeb-1M: A dataset and benchmark for large scale face recognition. In *European Conference on Computer Vision*. Springer, 2016.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[13] Alexander Hermans, Lucas Beyer, and Bastian Leibe. In defense of the triplet loss for person re-identification. *arXiv preprint arXiv:1703.07737*, 2017.

[14] Matej Kristan, Ales Leonardis, Jiri Matas, Michael Felsberg, Roman Pflugfelder, Luka Cehovin Zajc, Tomas Vojir, Gustav Hager, Alan Lukezic, Abdelrahman Eldesokey, et al. The visual object tracking vot2017 challenge results. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 1949–1972, 2017.

[15] He Li, Kaoru Ota, and Mianxiong Dong. Learning iot in edge: deep learning for the internet of things with edge computing. *IEEE Network*, 32(1):96–101, 2018.

[16] Lu Pang and Wang et al. Cross-domain adversarial feature learning for sketch re-identification. In *2018 ACM Multimedia Conference on Multimedia Conference*, 2018.

[17] Shaoqing Ren, Xudong Cao, Yichen Wei, and Jian Sun. Face alignment at 3000 FPS via regressing local binary features. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus,*

OH, USA, June 23-28, 2014, pages 1685–1692. IEEE Computer Society, 2014.

[18] SDM. The worldwide installed surveillance cameras. https://www.sdmmag.com/articles/92407-rise-of-surveillance-camera-installed-base-slows.

[19] Weisong Shi and Cao et al. Edge computing: Vision and challenges. 2016.

[20] Yi Sun, Yuheng Chen, Xiaogang Wang, and Xiaoou Tang. Deep learning face representation by joint identification-verification. In *Advances in neural information processing systems*, pages 1988–1996, 2014.

[21] Dong Yi, Zhen Lei, Shengcai Liao, and Stan Z. Li. Learning face representation from scratch. *CoRR*, abs/1411.7923, 2014.

[22] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3320–3328, 2014.

[23] Wei Yu, Fan Liang, Xiaofei He, William Grant Hatcher, Chao Lu, Jie Lin, and Xinyu Yang. A survey on the edge computing for the internet of things. *IEEE access*, 6:6900–6919, 2017.

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

[25] Shun Zhang, Jia-Bin Huang, Jongwoo Lim, Yihong Gong, Jinjun Wang, Narendra Ahuja, and Ming-Hsuan Yang. Tracking persons-of-interest via unsupervised representation adaptation. *arXiv preprint arXiv:1710.02139*, 2017.

[26] Liang Zheng, Yi Yang, and Alexander G Hauptmann. Person re-identification: Past, present and future. *arXiv preprint arXiv:1610.02984*, 2016.