Facial Information Protection by Mosaicking Faces in Video Dissemination

Zhaoyuan Zhang
University of Michigan
Electrical Engineering and Computer Science
zhaoyuan@umich.edu

Yijie Yang
University of Michigan
Electrical Engineering and Computer Science
yyjyang@umich.edu

Ke Liu
University of Michigan
Electrical Engineering and Computer Science
kliubiyk@umich.edu

Mingxuan Qu
University of Michigan
Electrical Engineering and Computer Science
miqu@umich.edu

1. Introduction

In the past ten years, the popularization of photography equipment and wireless internet significantly reduces the cost of filming and disseminating videos. However, the frequent video dissemination brings various information security issues. Especially during the COVID-19 pandemic, the sudden growth of videotelephony inevitably increases the risk of information leakage. Nowadays, as more and more people consider information security their priority, it is necessary and meaningful to improve information protection technology in video dissemination.

Among various information security issues, the leakage of human facial information is one of the most urgent issues. As a unique human biological characteristic, facial information widely applies in identity authentication, mobile payment and business analysis. Meanwhile, video is the fundamental information carrier of human interactions and thus contains considerable human facial information. Therefore, protecting human facial information in video dissemination is a crucial component of information protection.

Currently, mosaicking human face in the video is a straightforward and effective approach to protect facial information as Figure 1 [7]. However, the available methods of mosaicking human faces are still far from satisfying level. On the search engine, the most common approach for mosaicking human face is provided by a paid application *Filomra*[3]. However, *Filomra* only provides a basic mosaicking function for a fixed rectangle area in video and performs poorly even with small facial movements.

In our project, we apply object tracking, Gaussian filtering and mosaicking effect to bring users better quality of facial information protection. Object tracking is a useful technique helping to identify and locate objects, and can be commonly used on human, vehicles, to name a few, and here we mainly need to track human face for locating accurately. Applying Gaussian filtering and mosaicking effect to the object being tracked can reduce the details, thus hide the information efficiently.

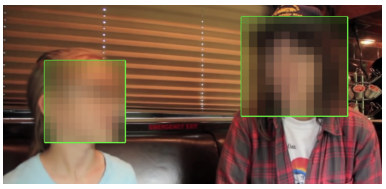

Figure 1. Example for mosaicking faces during an interview

Most of the functions can be achieved by using the OpenCV library. The object tracking tutorials provided by *pyimagesearch* [10] provide with some useful trackers from this library, where most of our work were built on. It is also available for tracking with webcam or well-recorded video simply by breaking the video frame by frame, then analyse the image sequences. In order to deal with emergency situation in streaming or live TV shows, we were inspired by another automatic face tracking algorithm[11]. Additionally, for evaluation and analysis purpose, we keep tracking the memory usage by using tracemalloc method[6], and apply *intersection over union* method to compare the output accuracy with ground truth table[9].

With all approaches explored above, we developed a multifuncional program to achieve the goal. This includes taking different trackers efficiency into consideration and pick the most suitable one. In case of the limit of auto detection, manual operation is allowed for people to make modification in time. Still, there should be more attempts to further perfect the model in a user-friendly way.

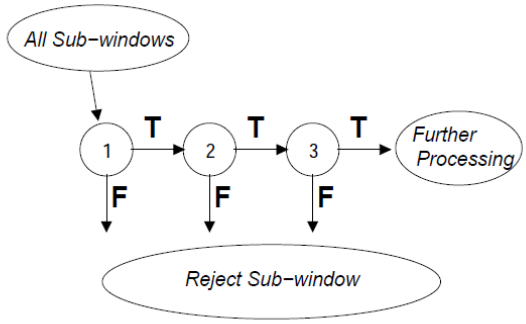

Figure 2. Classifier cascade in the CascadeClassifier class

Figure 3. Mosaicing effect on a man's face

2. Approach

2.1. Haar feature-based CascadeClassifier

For live TV shows or streaming services, it is necessary to ensure facial information protection to take place real time. Thereby, the program needs efficient classifiers to quickly track down faces and protect them. To implement the quick tracking, we adopted the CascadeClassifier class offered by OpenCV[5]. The CascadeClassifier class adopts the Haar Feature-based Cascade Classifier proposed by Paul Viola and Michael Jones. It is specialized for quick image processing and high detection rates[12]. The Haar Cascade Classifier adopts an intermediate representation of image to support efficient learning and applies the AdaBoost algorithm to quickly achieve high generalization performance[12]. It then forms a detection classifier cascade as Figure 2 [12].

While positive results require passing every classifier, the first few simple classifiers remove large number of negative examples and thereby effectively reduce computation time. With the rapid face detection offered by the CascadeClassifier class, we implement the feature that users can quickly mosaic all detected faces by simply pressing 'e'.

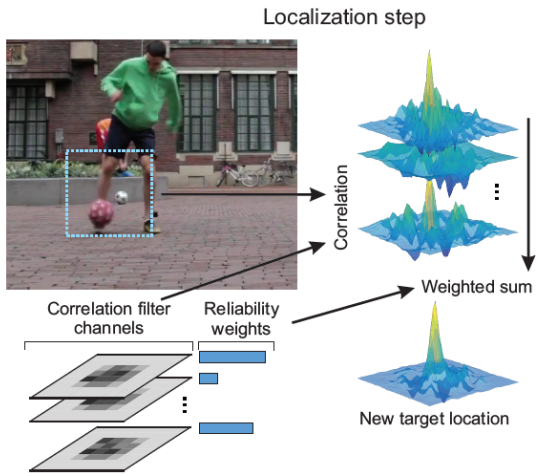

Figure 4. Localization step in the CSRT algorithm

2.2. Mosaicking effect

After quickly tracking down faces with the CascadeClassifier class, we implement the feature that users can quickly blur and add mosaic to all detected faces by simply pressing 'e'. To blur each of the faces encircled by a rectangle, we first applied a Gaussian blurring on the faces to reduce the facial details but remain edge information. After edges preservation, we divide the larger rectangle tracking box into smaller rectangle blocks, take the mean BRG values over each smaller block, and then draw a new rectangle on the same block with its mean BRG values. This process enables us to further remove the details in each of the smaller rectangle block and add mosaic to faces after combing all smaller rectangles as Figure 3[2]. In a word, the mosaicking effect after Gaussian blur removes details of facial information but still preserves the detectable edges of human face.

2.3. CSRT tracker

To continuously track and mosaic faces, we apply the OpenCV Tracking APIs to follow the face movements. While both Pillow and Opencv have image processing credibility and large community support, we pick Opencv simply due to personal interests as we have already applied Pillow in previous EECS 442 projects. After running test trials on our own camera and the Video Object Tracking dataset[8], we pick the CSRT(Discriminative Correlation Filter Tracker with Channel and Spatial Reliability) due to its high tracking accuracy and acceptable efficiency compared with other seven trackers[10]. Compared with CascadeClassifier, CSRT supports manual operation for selecting specific person's face to track and performs more steadily when face area is temporally incomplete.

According to Lukezic et al. [1], the CSRT algorithm

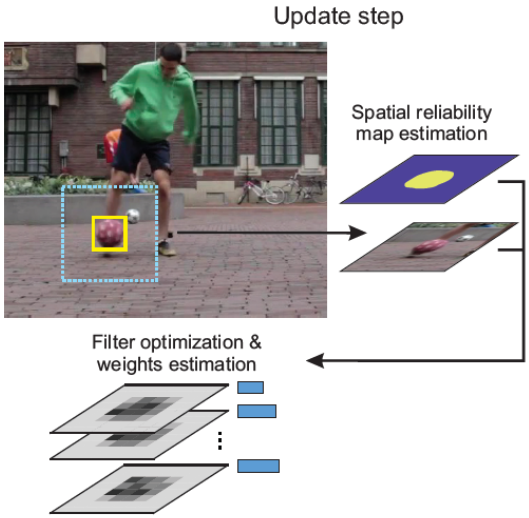

Figure 5. Update step in the CSRT algorithm

keeps a variable called Channel reliability **w**, which measures each channel's importance to locate an object. We represent the first phase of CSRT, called the 'localization step' in Figure 4[1]. It takes the image patch feature **f** from the object's previous position, calculate the correlation between filter **h** and **f** weighted by **w**, and take the place with maximum correlation as the next position. The second phrase of CSRT is called the "update step", which is represented by Figure 5[1]. It extracts and updates the object's foreground and background color model to build up a spatial reliability map **m** that identifies likely pixles in the training regions that belong to the object. It then takes **m** as input and updates the filter **h** by an efficient O(NLogN) algorithm. The Channel reliability **w** is recalculated and updated according to feature channel outputs.

2.4. Usability design

Apart from the basic tracking algorithm, we also added several new features to enable blurring of multiple faces and increase usability. Apart from pressing 'e' to blur all the faces detected by the Haar Feature-based CascadeClassifier, users can also manually select a single or multiple areas to add blurring effects to. Blurring effects can also be removed by 1.clicking to select a blurred area and press 'd' on the keyboard 2. pressing 'r' to remove all current blurring effects. To improve understandability, we added user guides on the start screen and during the area selection process. For evaluation purpose, we also added runtime and memory usage tracking after blurring starts to take place.

2.5. Tracking failure handling

Although CascadeClassifier and CSRT perform satisfyingly in tracking human faces, they also fail in tracking faces when the facial information is incomplete in video frame. Therefore, in response to two common tracking failure cases, we implement corresponding handling methods for both CascadeClassifier and CSRT.

Case 1: Face covered by other objects
When users drink a cup of coffee, read messages on the phone or just put some objects in front of their faces, faces may be covered by other objects which can cause tracking failure. This failure case occurs most for CascadeClassifier due to the loss of Haar features. For avoiding facial information leakage in this case, we record the location of the last successful tracking box by CascadeClassifier and then keep tracking on the same location when tracking failures occur. As a result, when the objects in front of human faces are removed, CascadeClassifier can quickly continue track on faces and not cause information leakage.

Case 2: Face on the edge or out of frame
When users' faces temporarily move on the edge or out of frame, incomplete facial information can cause the tracking failure. This failure case occurs most for CSRT since it cannot recognize human faces. For resolving this issue, we detect the location of human faces through the movement of tracking boxes and then remain the mosaicking effect when faces are partially in frame or out of frame. The size of mosaicking effect will remain the same as the last time that human faces are fully in frame to avoid potential leakage of facial information.

3. Experiments

3.1. Data

The dataset we employ to evaluate the performance of tracking human faces is Video Object Tracking dataset in Kaggle[8] since it provides images for every video frame and corresponding ground truth values. For our experiment purpose, we only select video frames that contain facial information in front of a fixed or moving camera in our evaluation. Furthermore, our evaluation includes the edge cases for human face tracking, such as face covered by a book and face under extremely low illumination situation.

3.2. Metrics

Taken reference from the application of intersection over union(IoU) [9], we convert the accuracy of detection box to the fraction of coverage. The data from the ground truth table is showing well-designed bounding boxes for the object. What we want to compare with the those figure is the output from Cascade auto-tracking and several trackers in openCV, namely, CSRT tracker, KCF tracker, MOSSE tracker.

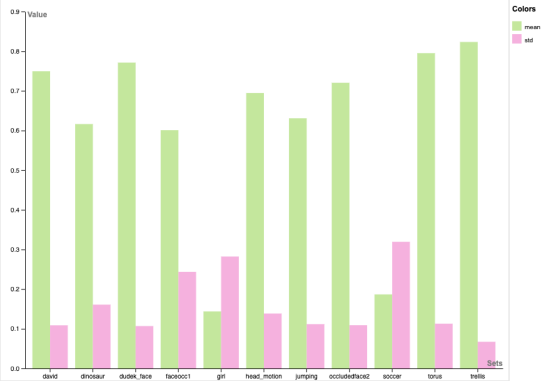

Figure 6. Performance of the CSRT Tracker

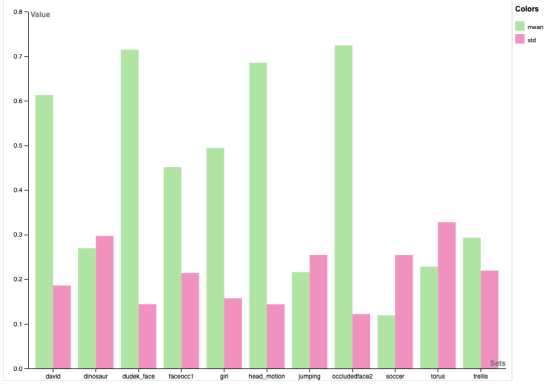

Figure 7. Performance of the KCF Tracker

3.3. Quantitative Result

After running our 3 trackers on all the 11 datasets, we got 33 sets of result in total and conclude that a valid tracker should have IOU mean of 70% above and std of about 10%. All following figures are ploted by RawGraphic [4]

The CSTR tracker has the best result among the three with its higher mean (Figure 8). The highest IOU mean, 82.3248%, and lowest std, 6.7233%, are also achieved by CSTR running on "trellis" Figure 6).

All three trackers can successfully detect objects under different light conditions, but they can hardly follow it when the object is moving, for example, in the "jumping" and "soccer" dataset, KCF and MOSSE only got about 20% and 10% as mean respectively. Meanwhile, CSTR succeeds on "jumping" but still failed on "soccer".

3.4. Qualitative Result

Those having 30% std (which is triple of the normal) implies that the tracker did detect correctly at the very beginning (high IOU), but got lost after a while (low IOU). For example, when monitoring KCF running on "dinosaur", "jumping" and "torus", we saw it starting to track other objects after our main object did a fast-moving(Figure 7). CSTR does a good job even the object is in rapid motion ("jump"). And note that there's one exception occured on the "girl" dataset because CSTR detected the wrong "box" when the gril in video turned her head(Figure 9). Although CSTR did track the "wrong" object closely, our frame ended up being far away from the ground truth one and resulted in a low IOU score.

4. Implementation

During our project's planning process, we refer to the single-object tracking algorithm proposed by *pyimagesearch*[10]. We took inspiration from *pyimagesearch*'s codes on pressing 's' to select a single tracking object. To satisfy our needs for multiple object blurring, we modified

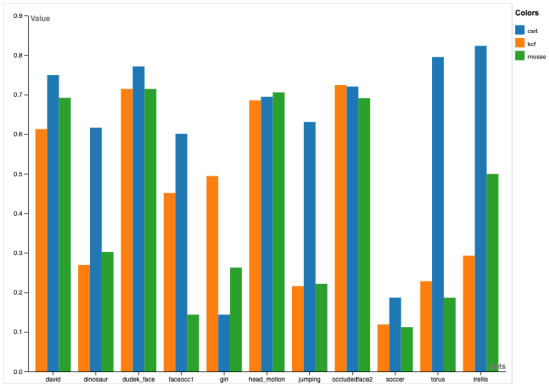

Figure 8. Compare trackers by mean IOU

Figure 9. Miss tracking case of CSRT in video "girl"

the code into multiple object tracking and implemented the blurring algorithm. We also add the features including 1. select multiple objects by pressing 'm' 2. cancel a selection by clicking the area and press 'd' 3. remove all blurring by pressing 'r'. Another previous work we refer to is the automatic face tracking algorithm proposed by *Shantnu Tiwari* [11]. It is where we learned about the existence of Haar Feature-based CascadeClassifier. During testing we found the CascadeClassifer may lose tracking after an object has partially left the screen. We fixed the drawback and added the feature where pressing 'e' will add blurring to all faces detected by the CascadeClassifer to start face protection in the first place.

References

[1] L. C. Z. J. M. Alan Lukezic, Tom'as Voj'ir and M. Kristan. Discriminative correlation filter tracker with channel and spatial reliability. *International Journal of Computer Vision*, 2018. 2, 3

[2] C. Briggs. A man's face. https://en.wikipedia.org/wiki/Face#/media/File:Sabaa_Nissan_Militiaman.jpg. 2

[3] L. Brown. How to blur face in video. https://filmora.wondershare.com/video-editing-tips/blur-face.html. 1

[4] C. DensityDesign and Inmagik. Rawgraphs 2.0 beta. https://app.rawgraphs.io. 4

[5] Doxygen. Cascade classifier. https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html. 2

[6] R. Keith-Magee. Monitoring memory usage of a running python program. https://medium.com/survata-engineering-blog/monitoring-memory-usage-of-a-running-python-program-49f027e3d1ba. 1

[7] KidsInterviewBands. Kids interview bands - courtney barnett. https://www.youtube.com/watch?v=4XI-UgXCsUc. 1

[8] K. S. Mader. Video object tracking. https://www.kaggle.com/kmader/videoobjecttracking. 2, 3

[9] A. Rosebrock. Intersection over union(iou) for object detection. https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/. 1, 3

[10] A. Rosebrock. Opencv object tracking. https://www.pyimagesearch.com/2018/07/30/opencv-object-tracking. 1, 2, 4

[11] S. Tiwari. Face recognition with python, in under 25 lines of code. https://realpython.com/face-recognition-with-python. 1, 4

[12] P. Viola and M. J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, 2004. 2