2021 IEEE International Performance, Computing, and Communications Conference (IPCCC) | 978-1-6654-4331-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/IPCCC51483.2021.9679395

CODS: Cloud-assisted Object Detection for Streaming Videos on Edge Devices

Tengpeng Li*, Xiaoqian Zhang*, Nam Son Nguyen†, and Bo Sheng*
*University of Massachusetts Boston, Email:{tenpeng.li001,xiaoqian.zhang001,bo.sheng}@umb.edu
†University of Wisconsin Stout, Email: nguyens@uwstout.edu

***Abstract*—The advance of hardware has allowed edge devices to carry out varying applications, including computation-intensive machine learning tasks. This paper targets a real-time application of detecting objects on streaming videos with a tightened requirement on processing overhead. While the edge device still lacks the computation power to apply the object detection algorithm on every video frame, this paper integrates cloud-side servers in the solution as much prior work has attempted. However, our design is different from the conventional off-loading solutions that delegate the computation tasks to the cloud servers. We present a solution named CODS where the computation tasks are still conducted by the edge devices while the cloud server provides useful guidelines for choosing appropriate analysis algorithms. We implement our solution with the Jetson Nano device and evaluate it with videos from representative datasets. The results show that CODS is quite effective and superior to baseline alternatives.**

I. Introduction

Edge computing has become a prevalent framework in many applications. In this paper, we particularly consider object detection on streaming video frames which is a representative machine learning algorithm. We consider a system setting where a surveillance camera is connected to an edge device and a monitor. The captured video frames will be processed by the edge device to detect the objects in the frames. The recognized objects marked by rectangle regions will be rendered with the video frames together to display on the monitor. We consider real-time object detections that will help people identify and track the objects on the screen, especially suspicious or important objects.

In this paper, we present a solution for edge devices to support real-time object detection on streaming video frames. Our solution is also based on assistance from cloud-side servers. We name it CODS (Cloud-assisted Object Detection for Streaming Videos on Edge Devices). We consider that there are multiple options of algorithms that can be applied on the edge device and each of them yields varying performance in terms of processing delay and detection accuracy. In CODS, the cloud server will provide a guideline for the edge device to choose the appropriate algorithm at run time by analyzing the video frames in the past time window. In this way, the network delay is not a performance issue as the edge device is still processing the current frames, and the results of the past frames from the cloud server are not expected to be displayed.

In particular, this paper considers a simplified model where the edge device has two options for processing the video frames. One is to apply an object detection algorithm in the machine learning library, and the other is the tracking algorithm in computer vision. Generally, the tracking algorithm yields a small overhead, but the accuracy degrades when it is consecutively repeated. In CODS, the cloud server helps the edge device to determine the appropriate strategy. At run time, deriving the optimal strategy and the corresponding parameters is challenging for the edge device as it conducts the computation tasks at the same time. Our design goal is to leverage the computing resources on the server to provide useful guidance that is not strictly delay-sensitive.

II. Related Work

Object detection has made a great progress in recent work However, the heavy computation restrains their deployment on resource-inadequate devices, like mobile phones and regular IoT devices. Specifically tuned models like SSD [1], MobileNet [2], YOLO [3] are designed to consume fewer resources, but their accuracy is also harmed collaterally. In [4], the author shows that more accurate models require more time on inference in general.

To take the advantage of state-of-the-art machine learning models in real-time applications running on resource-limited devices, researchers offload the heavy computation process to the cloud server. The approaches can be dichotomized into two categories: (1) **Decreasing data packet size**. In [5], the authors decrease the data size by applying different encoding algorithms on different regions of the image. Grassi et al. [6] conducts image recognition locally and only uploads processed data for further processing on the cloud. Wang et al. [7] take the advantage of video encoders. Hu et al. [8] reduce the time cost by decreasing video resolution in real-time. (2) **Selectively sending data**. Satyanarayanan et al. [9], send every frame to the cloud server, the delay coming from the network and server processing makes the detection result hard to be updated in real-time. In contrast, recent systems send frames selectively to get a better user experience. The solutions in [10] and [5] only send the frame which is significantly different from the previous frame and locally track the recognized object. Other work in [7] and [11] introduces a content-aware frame selecting algorithm to filter the image with prior knowledge about the task.

978-1-6654-4331-9/21/$31.00 ©2021 IEEE

III. System Model and Problem Formulation

In this paper, we target a real-time object detection application that analyzes streaming videos. The real-time video frames are captured, analyzed by the edge device, and displayed with the objects recognized and marked on a screen.

A. Actions on Edge Devices

In our system model, we assume that the edge device can conduct two types of analysis on each video frame, object detection algorithm and tracking algorithm. Object detection refers to detecting instances of objects from a particular class in an image, such as finding dogs, humans in a given image. It requires a training phase to build models, and there have been object detection algorithms and models suitable for edge devices. The tracking algorithm, is a computer vision method that requires a completed object detection beforehand. After the detected objects are indicated by regions on a frame, the algorithm will process the successive frames to "track" the movement of those regions by comparing the image features. The result of either tracking or object detection would be rectangles surrounding the objects being detected or tracked. The edge devices will display the captured frame with those rectangles drawn on it. We assume the edge device can conduct either one of these two image analyses, but not both in parallel because of the computation limitation.

B. Representation of Analysis Result and Accuracy

Generally, tracking is much faster than object detection. To simplify the problem setting, we assume that the input video frames are captured at a constant rate where the interval between two consecutive frames is sufficient for tracking analysis. Therefore, if the edge device conducts a tracking analysis on the current frame, the results will be displayed on the next frame. During an object detection analysis, however, the edge device will continue to display the frames with the rectangles derived when the analysis starts (i.e., no update). We use N_{od} to represent the overhead of an object detection process in terms of the number of frames. The following Fig. 1 illustrates an example where $N_{od} = 4$ (an object detection process starts at frame 6 and ends at frame 10).

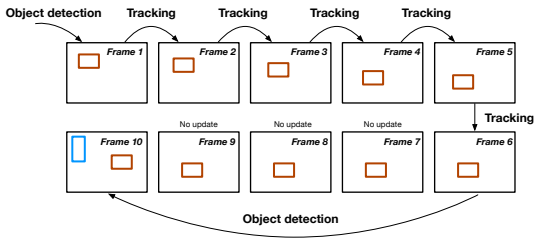

Fig. 1: An example of video processing at the edge device

Intuitively, object detection would yield the most accurate recognition results, and the following tracking processes could lead to errors. We define the following two types of results (the rectangles identifying objects) and their associated parameters:
Tracking result: We use this general term to represent the results obtained by object detection analysis or a tracking analysis following an object detection. The accuracy usually depends on the number of tracking processes after the last object detection. We call it the *tracking distance*. The result directly from an object detection analysis has a tracking distance of 0, then the distance value increments for the following continuous tracking analyses.
Stale result: This is the result the edge device uses during an object detection. It is derived from a tracking analysis but has been stale for a while. Two parameters are needed to describe its accuracy. First, we need the tracking distance value for the result because it is a tracking result. Second, we need to consider how long the result has not been updated. We call it the *stale distance* indicating the number of frames between the last tracking result and the current frame.

Therefore, when displaying a result R_t on a frame f_t, we use two parameters to describe its properties $R_t = \{TD_t, SD_t\}$, where TD_t indicates its tracking distance, and SD_t is the stale distance. Referring to Fig. 1, for frame 5 and frame 8, the displayed results are $R_5 = \{4, 0\}$ and $R_8 = \{5, 2\}$ respectively. With the input frame rate and the parameter N_{od}, the overheads of the tracking and object detection algorithms have been considered in our problem setting. Another critical performance metric is the detection accuracy. Let $A(R_t)$ denote the accuracy of R_t. Generally, $A(R_t)$ is a fractional value between 0 and 1, and the higher value indicates more accurate results.

IV. Design of CODS

In this section, we present our solution CODS. Essentially, there are two processes running on the edge device, analyzing the captured frames to identify objects, and displaying the results overlaid with the frames.

The displaying process is straightforward as introduced in Section III. The edge device will merge two layers: one is the current frame captured by the camera and the other is the recognized objects represented by a set of rectangles, and show them on a screen (illustrated in Fig. 2). If the edge device has just derived a result from tracking analysis on the previous frame, it will display the result with the current frame. If the edge device is conducting the object detection analysis, then the current frame will be displayed with the most recent result obtained before the ongoing object detection.

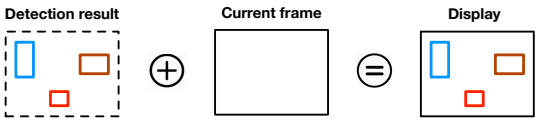

Fig. 2: Display on edge devices: Merge of a recognition result and a video frame

A. Statistics from Cloud Servers

In CODS, cloud servers play an important role of providing accuracy statistics to help the edge device make decisions. Specifically, the edge device sends the captured frames to a cloud server which applies the same object detection and tracking analysis as conducted by the edge device, but at a

much faster rate. The object detection algorithm is applied on every frame, and the result is considered as ground truth. Based on these ground truths, the cloud server further conducts the tracking algorithm and records its accuracy.

The following two pieces of information are derived and supplied to the edge device to help make the decision.

(1) Accuracy Matrix. The accuracy matrix is the most important statistical information from the cloud server. It is a two-dimensional matrix, one dimension for tracking distance and the other dimension for stale distance, and each element indicates the quantitative accuracy value. When the edge device displays the result $R_t = \{TD_t, SD_t\}$ on a frame f_t, we use $A(TD_t, SD_t)$ to represent the accuracy of the marked objects, where TD_t and SD_t indicate the tracking distance and stale distance respectively. Intuitively, $A(x, y)$ is a decreasing function on x and y. With this definition, the result from an object detection analysis has an accuracy of $A(0, 0)$, and the accuracy value of the first tracking after it is $A(1, 0)$. In particular, we consider an upper bound of the tracking times T_{max}, and then build a $T_{max}xN_{od}$ accuracy matrix.

Once the cloud server receives a new frame from the edge device, it will calculate an accuracy matrix A' based on this new frame and the historical frames. For example, when the server receives the t-th frame f_t, it can apply object detection on the frame and assign the resulting accuracy value to $A'(0, 0)$. It can also apply object detection on frame f_{t-1} that was received earlier and then use the result to conduct the tracking algorithm on f_t. The resulting accuracy will be assigned to $A'(1, 0)$. Similarly, the server can use different tracking distance and stale distance to test the accuracy of the analysis involving this new frame f_t, and derive a $T_{max}xN_{od}$ matrix A'. Finally, the server updates the main accuracy matrix A by merging the new information from A'. In CODS, we use exponentially weighted moving average (EWMA) for the update as follows,

$$A(i, j) = (1 - w) \cdot A(i, j) + w \cdot A'(i, j), \tag{1}$$

where $w \in [0, 1]$ represents the weight of the new information.

(2) Optimal Tracking Times. In CODS, the other important parameter the cloud server calculates is called optimal tracking times (OTT). OTT is supposed to be a guideline for the edge device to estimate how many times of tracking analysis should be conducted between two consecutive object detections.

OTT value is tightly related to and derived from the accuracy matrix $A(x, y)$. In fact, $A(x, y)$ varies for different objects, so is the OTT. The OTT value sent by the server is the average value of all the recognized objects. Here we first introduce the OTT's definition for a particular object. Let OTT_O denote the optimal tracking number between two object detection for object O. OTT_O is derived as

$$OTT_O = \arg\max_x \frac{\sum_{i\in[0,x]} A(i, 0) + \sum_{j\in[1,N_{od}]} A(x, j)}{x + N_{od}}. \tag{2}$$

Essentially, we consider a cycle of object detection and the following tracking analysis. We consider the end of an object detection as the starting point of the cycle which ends when the edge device finishes the next object detection, as illustrated in Fig. 3. Assume tracking is applied x times, then the sum of the accuracy of those x frames is $\sum_{i\in[0,x]} A(i, 0)$ because the tracking distance is incremental and the stale distance is always 0. When we conduct the next object detection, stale results will be used as shown in Fig. 3. There will be N_{od} frames using the same stale results, and their accuracy sum is $\sum_{j\in[1,N_{od}]} A(x, j)$, where only the stale distance is changing from 1 to N_{od}. Therefore, there are totally $x + N_{od}$ frames in the cycle, and we can represent the average accuracy as the function in Eq. 2. The cloud server can enumerate all possible values for x, and derive the optimal one as OTT_O.

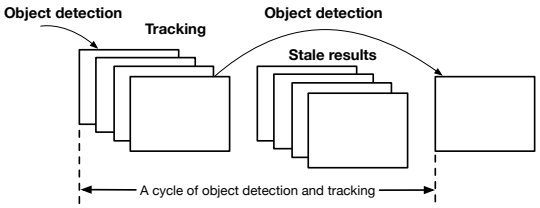

Fig. 3: Optimal Tracking Times

After calculating OTT_O for every recognized object O at the moment, the cloud server will send the average value as OTT to the edge device.

B. Optimal Processing Strategy for Edge Devices

In CODS, the major component is the algorithm running on the edge devices that determines the next image analysis approach. Considering the type of the analysis that has just been finished and the possible analysis for the next step, there are the following four possible cases:

Case 1: The edge device has just finished an object detection analysis and will continue to apply object detection.

Case 2: The edge device has just finished an object detection analysis and will continue to apply tracking.

Case 3: The edge device has just finished a tracking analysis and will continue to apply object detection.

Case 4: The edge device has just finished a tracking analysis and will continue to apply tracking.

The first two cases happen when the edge device has just finished an object detection analysis. In case 1, there is no tracking analysis. A cycle is represented by an object detection process during which the N_{od} frames will use the stale result. Therefore, the average accuracy is

$$AA_1 = \frac{\sum_{j\in[0,N_{od}-1]} A(0, j)}{N_{od}}. \tag{3}$$

In case 2, an object detection is followed by a sequence of tracking analysis. To estimate the average accuracy, we use OTT as the expected number of tracking operations. Therefore, a cycle in this case includes $OTT + N_{od}$ frames, and their average accuracy is

$$AA_2 = \frac{\sum_{i\in[0,OTT]} A(i, 0) + \sum_{j\in[1,N_{od}]} A(OTT, j)}{OTT + N_{od}}. \tag{4}$$

We consider that the case yielding a higher average accuracy value is the better choice.

Similarly, case 3 and case 4 are the options when the edge device has just finished a tracking analysis. We also calculate the average accuracy for a comparison. However, in stead of using a periodical cycle, we consider the phase from the current frame until the end of the next object detection analysis. In case 3, this phase does not include any tracking, and the average accuracy is simply calculated as

$$AA_3 = \frac{\sum_{j\in[0,N_{od}-1]} A(TD, j)}{N_{od}}, \tag{5}$$

where TD is the tracking distance of the current result. In case 4, the estimation is also based on the parameter OTT. The tracking is expected to continue for $max\{OTT-TD, 1\}$ times. And then an object detection will be conducted. Let $RT = \max\{OTT - TD, 1\}$ indicate the remaining tracking times. The average accuracy of our target phase will be

$$AA_4 = \frac{\sum_{i\in[1,RT]} A(TD+i, 0) + \sum_{j\in[1,N_{od}]} A(OTT, j)}{RT + N_{od}}). \tag{6}$$

The following Algorithm 1 illustrates the main steps of the edge device. Depending on the previous action $PreA$, the edge device will calculate and compare either AA_1 and AA_2 or AA_3 and AA_4. The option yielding a higher accuracy will be returned as the choice for analyzing the next frame.

Algorithm 1: Determine the next analysis type

Input: Accuracy matrix $A(x, y)$, the optimal tracking times OTT, and the type of the previous analysis $PreA$

Output: The next analysis type

```
if PreA is object detection then
    Calculate AA_1 (Eq. 3) and AA_2 (Eq. 4)
    if AA_1 > AA_2 then
        return object detection
    else
        return tracking
if PreA is tracking then
    Calculate AA_3 (Eq. 5) and AA_4 (Eq. 6)
    if AA_3 > AA_4 then
        return object detection
    else
        return tracking
```

In our implementation, when comparing two average accuracy (AA) values, the algorithm consider a weight value for each of them to tolerate the estimation errors and biases. The server also repeats the same algorithm and observes the correctness of the decision based on the ground truths. Then, the server will adaptively adjust the weight values and notify the edge device. We omit the details of this part to simplify the description in Algorithm 1. In addition, we only consider two options for the edge device in this paper. Our approach in Algorithm 1, however, can be easily extended to support more options, e.g., using different tracking algorithms and using different models for object detection.

V. Performance Evaluation

In this section, we evaluate and analyze the performance of CODS. The experiment results soundly prove the effectiveness of our algorithms.

A. Settings and Workload

Our system is evaluated with three videos from the well-known MOT2015 dataset [12], which contains video sequences in various environments filmed with both static and moving cameras. The characteristics of these three videos are listed in Table I. We calculate the average number of objects in each frame, the average value of the object width and height in pixels, and the average value of the moving distance between the positions of the same object in consecutive frames.

Property	Video 1	Video 2	Video 3
Number of Frames	795	837	71
Frame Width	768	640	640
Frame Height	576	480	480
Avg # of Objects	5.85	8.07	5.06
Avg Object Width	29.30	88.76	76.99
Avg Object Height	83.32	246.95	198.75
Movement	4.18	6.98	4.66

TABLE I: Characteristics of the test videos: the units are pixels except "Number of Frames"

In addition, we have implemented our solution and tested it on Jetson Nano to measure the time cost of each component. In our experiments, the choice of tracking algorithm, by default, is Channel and Spatial Reliability Tracking (CSRT) [13] implemented in OpenCV [14]. And the object detection algorithm is based on model MobileNetV2 [15]. According to our experiment results, object detection takes $\sim$ 300ms to complete one inference on average and a tracking with CSRT needs $\sim$ 100ms to process one frame. Considering the measured processing overheads of object detection and tracking, we assume 3 frames are needed for each object detection, tracking is able to complete in real-time, i.e., $N_{od} = 3$.

During the object detection process, the edge device will not analyze the upcoming frames. The stale results obtained before the inference is triggered will be used as the detection results for those frames. In all the tests, we take the first 50 frames to train our system and start object detection and tracking after that. The weight parameter w in Eq. 1 is set to 0.1.

B. Accuracy Metric

In this paper, accuracy is a critical performance metric. In our experiments, we use average precision (AP) to measure the accuracy performance.

To calculate the AP, we need to get the precision and recall rate first, which can be calculated as follows

$$\text{Precision} = \frac{TP}{TP+FP} = \frac{TP}{\text{all detections}}, \tag{7}$$

$$\text{Recall} = \frac{TP}{TP+FN} = \frac{TP}{\text{all ground truths}}. \tag{8}$$

Then the AP is obtained as

$$AP = \sum_{n=0} (r_{n+1} - r_n)\, \rho_{interp}\,(r_{n+1}), \tag{9}$$

with $\rho_{interp}\,(r_{n+1}) = \max_{\tilde{r}:\tilde{r}\geq r_{n+1}} \rho(\tilde{r})$, where $\rho(\tilde{r})$ is the measured precision at recall $\tilde{r}$.

C. Evaluation of Accuracy

To evaluate CODS's performance, we compare CODS with another two alternative approaches: tracking a fixed number of frames and continuous object detection without tracking.

In the first approach, each object detection is followed by a fixed number of tracking. In the second approach, the system keeps conducting object detection on the video. After obtaining the average precision on each frame, we calculate their mean value as mAP and draw the final result in Fig. 4.

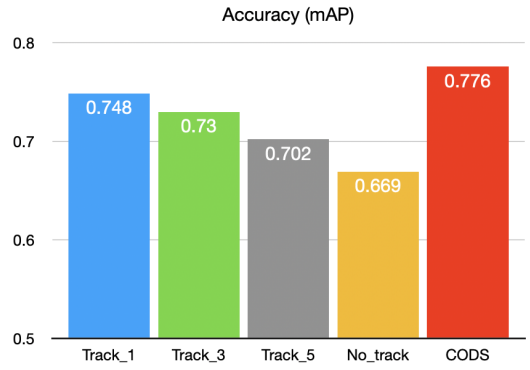

Fig. 4: Comparsion of accuracy (mAPs)

In the plot, the y-axis stands for mAP values and the x-axis marks the configuration used to get that data. *Track_num* represents the alternative of using a fixed number of tracking, where *num* stands for how many frames are tracked after each object detection. The *No_track* category keeps running object detection throughout the whole experiment without any tracking. Apparently, CODS obtains the highest mAP value overall as 0.776 while the *No_track* category gets the lowest mAP as 0.669. As CODS adaptively adjust the number of tracking, it is also superior to the approaches with a fixed number of tracking.

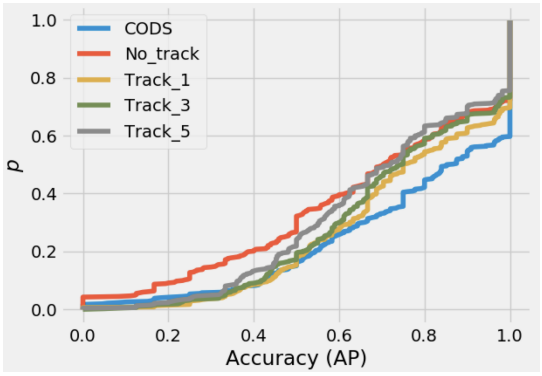

Fig. 5: Cumulative density function (CDF) of accuracy (APs)

To get more insights, we draw the Cumulative Density Function (CDF) of average precisions using different configurations in Fig. 5. The x value in the figure stands for the average precision, the y value stands for the cumulative probability. The figure further confirms that continuous object detection performs the worst among all, as half of the APs in *No_track* category are less than 0.8. As a comparison, more than 40% of APs in CODS are 1.0 in the experiment.

The performance of tracking for a fixed number of frames, though seems to be close to CODS overall in Fig. 4, their performance deteriorates when the experiment video does not fit their patterns. We draw the performance of each configuration for each video respectively, the result is displayed in Fig. 6. In video 1, the mAP of *Track_5* is as high as 0.76 but the accuracy decreases to 0.68 in video 2 when the video pattern changes. As a comparison, CODS achieves the best result in both video 1 and video 2. Video 3 only contains 71 frames, though CODS does not have enough time to adjust its strategy to the optimal, it still achieves an accuracy as high as 0.87.

D. Sensitivity Analysis

In this subsection, we further explore how some important system parameters influence the CODS's performance.

Object Detection Lag. The object detection lag is an important factor being considered in the system design. To comprehensively examine our system, we evaluate CODS using different detection lags. The result is shown in Fig. 7.

The x value stands for the detection lag, which is calculated as the number of frames required to complete one object detection, and the y value stands for the mean average precision. Even though the performance of all configurations is getting worse as the detection lag increases, CODS maintains the best performance. One noticeable discovery is that CODS is capable of choosing the optimal solution even in the case that the system can finish real-time object detection in the ideal case, i.e., the detection lag is 1. This uncovers the great potential of our algorithms in various scenarios.

Tracking Algorithm. The tracking algorithm is crucial to our system, and there exist different options. To evaluate the importance of the tracking algorithm in our system, we test CODS with the other two tracking algorithms in the OpenCV library, Kernelized Correlation Filters (KCF) [16] and Minimum Output Sum of Squared Error (MOSSE) [17]. KCF and MOSSE trackers take less time and resources to run. But they suffer from low accuracy. According to our experiment, both of them can be updated with an FPS larger than 60. To take the advantage of these two tracking algorithms, we assume that they can be applied on every frame while CSRT has to be paused during the object detection.

In Fig. 8, we use the performance of the *No_track* as the baseline and show the accuracy improvement with different trackers. The bars with different colors represent the different tracking algorithms and the axis x marks the settings. The shortcomings of configurations that track for a fixed number of frames are exposed when the tracking algorithms are not effective. According to the experiment, *Track_3* and *Track_5*

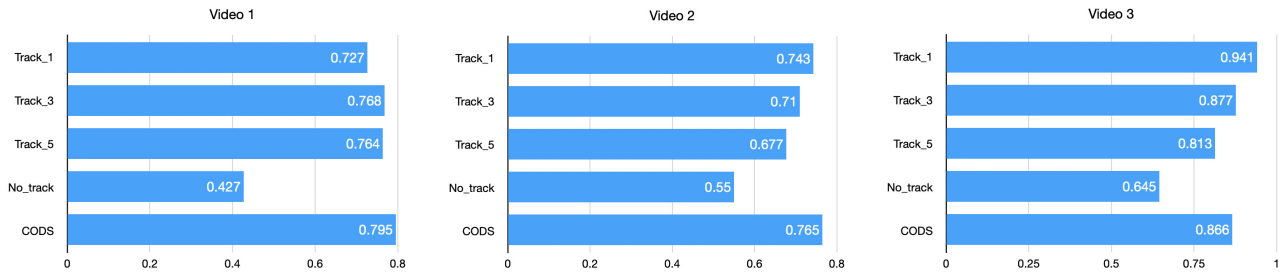

Fig. 6: Accuracy (mAPs) in different videos

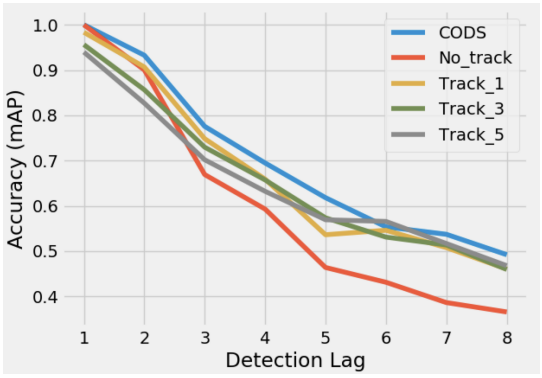

Fig. 7: mAP for different object detection lags

obtain an even worse accuracy when MOSSE algorithm is applied. As a comparison, CODS can adaptively adjust its strategy and always achieve optimal performance.

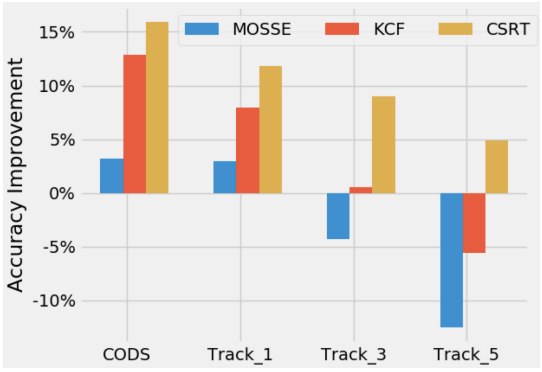

Fig. 8: mAP for different trackers

VI. Conclusion

In this paper, we present a cloud-assisted framework for edge devices to recognize the objects in streaming videos in real-time. In our system, we consider there exist different options for edge device to choose for analyzing the frame each with different detection accuracy and processing overhead. The cloud server in our system provides a guideline to help the edge device make the decision. We have implemented our system on Jetson Nano, and comprehensively evaluated the performance with a well-accepted video workload. The experimental results demonstrate the benefit of adaptive adjustment of the analysis algorithm and show significant performance improvement comparing to the alternative baselines.

References

[1] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*.

[2] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[3] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*.

[4] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors," in *Proceedings of the IEEE conference on computer vision and pattern recognition*.

[5] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *MobiCom*. ACM, 2019.

[6] G. Grassi, K. Jamieson, P. Bahl, and G. Pau, "Parkmaster: An in-vehicle, edge-based video analytics service for detecting open parking spaces in urban environments," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*.

[7] J. Wang, Z. Feng, Z. Chen, S. George, M. Bala, P. Pillai, S.-W. Yang, and M. Satyanarayanan, "Bandwidth-efficient live video analytics for drones via edge computing," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*.

[8] J. Hu, A. Shearer, S. Rajagopalan, and R. LiKamWa, "Banner: An image sensor reconfiguration framework for seamless resolution-based tradeoffs," in *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*.

[9] M. Satyanarayanan, V. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Computing*, 2009.

[10] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, "Glimpse: Continuous, real-time object recognition on mobile devices," in *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*.

[11] X. Wang, A. Chowdhery, and M. Chiang, "Skyeyes: adaptive video streaming from uavs," in *Proceedings of the 3rd Workshop on Hot Topics in Wireless*.

[12] L. Leal-Taixé, A. Milan, I. Reid, S. Roth, and K. Schindler, "MOTChallenge 2015: Towards a benchmark for multi-target tracking," *arXiv:1504.01942 [cs]*. [Online]. Available: http://arxiv.org/abs/1504.01942

[13] A. Lukezic, T. Vojir, L. Cehovin Zajc, J. Matas, and M. Kristan, "Discriminative correlation filter with channel and spatial reliability," in *Proceedings of the IEEE conference on computer vision and pattern recognition*.

[14] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[15] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*.

[16] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-speed tracking with kernelized correlation filters," *IEEE transactions on pattern analysis and machine intelligence*.

[17] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, "Visual object tracking using adaptive correlation filters," in *2010 IEEE computer society conference on computer vision and pattern recognition*.