Toward Intelligent Surveillance as an Edge Network Service (iSENSE) using Lightweight Detection and Tracking Algorithms

Seyed Yahya Nikouei, Yu Chen, Sejun Song, Baek-Young Choi, Timothy R. Faughnan

***Abstract*—Edge computing extends the realm of information technology beyond the boundaries defined by cloud computing. Performing computation near the sensors, edge computing is promising to address the challenges in many bandwidth-and delay-sensitive applications. Although recently many smart video surveillance approaches based on Machine Learning (ML) algorithms become available, it is still challenging to efficiently migrate those smart algorithms to edge. In this paper, we propose an intelligent Surveillance as an Edge Network Service (iSENSE), which explores the feasibility of moving ML to the edge by testing two popular human-object detection schemes. Besides, a lightweight Convolutional Neural Network (L-CNN) is introduced to improve computational execution by leveraging the depth-wise separable convolution. To enhance performance on edge, we propose a hybrid lightweight tracking algorithm, Kerman (Kernelized Kalman filter), which is a decision tree based hybrid Kernelized Correlation Filter algorithm designed for human-object tracking. We have implemented both Kerman and L-CNN algorithms on edge by using different types of single board computers. The proposed iSENSE system was validated using both real-world campus surveillance video and open image sets. The experimental results present that the proposed algorithms can track the human objects in real-time with a good accuracy with limited resource available in edge devices.**

***Keywords*—Edge Computing, Smart Surveillance, Lightweight Convolutional Neural Network (L-CNN), Human Detection-tracking, Lightweight Trackers.**

I. Introduction

To cope with the unprecedented scale and speed of urbanization [37], cities face the daunting task of accommodating the urban dynamics with smarter management strategies [55]. The recent concept of "Smart Cities" has attracted the attention of the urban planners and researchers to enhance the security and well-being of the residents. One of the essential smart community services is the intelligent object surveillance [6] for situational awareness (SAW), particularly for many dynamic data-driven tasks [3]. It enables a broad spectrum of promising applications including access control in areas of interest, human identity or behavior recognition, detection of anomalous behaviors and so on, [26]. North America alone in 2016 has more than 62 million public surveillance cameras watching the movement of the civilian for their safety and SAW [29]. Many of these smart surveillance applications require significant computing and storage resources handling massive contextual data created by video sensors. However, significant challenges exist to support big dynamic data collection and analytic in real time [35].

Traditional surveillance systems depend on human operators to manipulate the processing of captured video [7]. Being aware of the growing demand for human resources to interpret the data due to the ubiquitous deployment of networked static and mobile cameras [34], the human operator's role is minimized in the second generation of surveillance [8]. Various intelligent machine learning algorithms [44] and statistical analysis approaches [14] take in object detection and the responsibility of abnormal behavior detection at the cloud [48].

The cloud computing paradigm provides excellent capabilities for storing, processing, and analyzing big video data and is also scalable corresponding to the increasing number of surveillance cameras. In practice, however, there are significant hurdles for the remote cloud-based smart surveillance architecture for the collection of big dynamic data in real-time. According to the recent studies [9], the video data dominates the real-time traffic and creates a heavy workload on the communication networks. Although many network delivery protocols with higher efficiency have been suggested [54], the delay and dependence on communication networks still hinder running-time interactions. Edge-Fog-Cloud hierarchical computing architecture is considered as an answer to the shortcomings of cloud computing [5]. The edge computing technology migrates more computing tasks to the connected smart "things" (sensors and actuators) at the edge of the network [45].

While the edge layer pre-processes and extracts features from a video, the fog layer contextualizes and makes decisions based on the extracted features, and cloud computing layer performs historical data analysis and fine tuning. This platform consists of several pieces that exceed this paper's boundaries. Specifically, this paper focuses on the edge layer to improve pre-processing and feature extraction from the local video data. The detailed discussions on the container implementation and secure networking architecture of the platform can be find in related papers [39], [53].

S. Y. Nikouei and Y. Chen are with the Department of Electrical and Computer Engineering, Binghamton University, SUNY, Binghamton, NY 13902 USA, email: {snikoue1, ychen}@binghamton.edu.

S. Song and B. Y. Choi are with School of Computing and Engineering, University of Missouri-Kansas City, Kansas City, MO 64110 USA, email: {songsej, choiby}@umkc.edu.

T. R. Faughnan is with the New York State University Police, Binghamton University, SUNY, Binghamton, NY 13902 USA, email: tfaughn@binghamton.edu.

Manuscript received in August 2018.

The features extracted here are based on input needs of a fuzzy model, which is consulted with the police authorities and is based on their preferences. Although for many applications higher resolution and frame rate is required, in context of the human movement speed, the proposed platform achieves sub-second real-time performance. Also, new releases of hardware may improve the speed of the overall frame processing.

We develop and integrate two popular human-object detection schemes. We first develop a lightweight Convolutional Neural Network (L-CNN) to improve computational execution by leveraging the depth-wise separable convolution. L-CNN enables a real-time human as objects of interest (human-object) identification on a network edge using fewer resources, yet preserving the high accuracy of CNNs. Moreover, to enhance the system performance on edge, we propose a hybrid lightweight tracking algorithm, Kerman (Kernelized Kalman filter), which is a decision tree-based hybrid Kernelized Correlation Filter (KCF) algorithm tuned for human-object tracking. Kerman works along with L-CNN to further improve the speed and reliability of feature extraction for human abnormal behavior detection in iSENSE. For a research prototype system of iSENSE, we have implemented both Kerman and L-CNN pipeline on edge device by using a couple of different types of single board computers (SBC) including a Raspberry PI 3 model B and a Tinker board. For validation of our proposed, real-world campus surveillance video streams as well as open image datasets, Pascal Visual Object Classes (VOC) including VOC07 and VOC12 [13] are used both training and detection. The experimental results demonstrate that the proposed algorithms can track the humans as objects in real-time with decent accuracy at a resource consumption affordable by edge devices.

The rest of the paper is organized as follows. Section II provides background of the closely related work. Section III discusses human detection algorithms. Then, Section IV introduces different tracking algorithms. In Section V, The detection, and tracking algorithms are put together to create one feature extractor algorithm. Section VI is the comparative study on the performances of the L-CNN and other algorithms, along with the results of the tracking algorithm implemented on a Raspberry PI 3 model B and a Tinker board. Finally, Section VII concludes the paper with discussions of our on-going efforts.

II. Background and Related Work

A. Human-Object Detection

Traditional human detection needs good handcrafted features extraction method and a powerful classifier that work best together. Haar-like feature extraction is well suited for face and eye detection [10]. Haar models are light weighted and very fast, which are appreciated as a candidate for edge implementation. However, the human body will appear each time differently according to ambient lighting, size, and clothing which makes it harder for this type of simplistic pixel value-based models to achieve a high detection accuracy [19]. Grids of Histograms of Oriented Gradient (HOG) can produce reliable features for human detection [11]. While the Haar features fail to detect humans when the body angle toward the camera changes, HOG features continue to perform well in every scene. HOG features are given to a Support Vector Machine (SVM) classifier to create a human detection algorithm called HOG+SVM [32]. Scale Invariance Feature Transformation (SIFT) is another well-known algorithm for human detection through extracting distinctive invariant features from images, which provides features that can be used to perform reliable matching between different views of an object or scene [23].

Powerful machine learning algorithms are recognized as a solution to take full advantage of big data in many areas [30], [56]. GoogleNet [47] and ResNet [21] are two examples of well-known CNN architectures for image classification with high accuracy. They can take a picture as an input and conduct classification for up to one thousand different objects using machine-based features (filters). The network has as many filters as needed to create a feature map that can differentiate between the possible classes of objects [25]. Recent attempts have been made to generate faster deep learning networks that require less resource without losing accuracy. Two architectures standout: SqueezeNet, as its name imply takes less memory but achieves the same performance as AlexNet [28] and [49]; and, MobileNet which is another architecture created by Google to work on resource constraint devices [24]. MobileNet is not only memory efficient, but also runs faster because of a different convolutional architecture and results are comparable to GoogleNet in accuracy.

Apache MXNet is a deep learning platform that provides CNN layer architecture in an easy to use manner. In this work, the proposed L-CNN is trained using MXNet because of its implementation of Single Shot Multi-Detector (SSD) additional libraries and good documentation. The original implementation of the SSD is on Caffe which requires a plain text file as the layer architecture and proves to be difficult for editing. Other deep learning platforms do not have the SSD implementation or working with them is not as convenient as MXNet.

B. Object Tracking

The common challenges for trackers include partial or full object occlusions, scene illumination changes, and object shapes and motion [40]. The well-known region based tracking algorithm detects a human and extracts it from the background [50]. Because it only differentiates the background and foreground based on Gaussian modeling, the region based tracking is not suitable for surveillance application at hand. Another method is Feature Based Tracking, where a classifier looks for features that are well-describing the object of interest such as lines or point that separates the object from the background. The feature-based methods suffer from occlusion problem as they need at least some sub-features to remain visible and the accuracy of the classification drops with less visible features [51].

In 2017 a method called Need for Speed (NFS) was introduced as a dataset created with very high quality videos used for benchmarking and in its paper divides object tracking to deep trackers and correlation filter (CF) trackers [15]. Several

best performing algorithms are used in the benchmarks. The fastest algorithm is the Multi-Dimensional Network (MDNet) that has more than 50 FPS test result in the benchmark [36]. GoTURN [22] and SFC [2] are also deep trackers that are released in recent years that are GPU based and very fast. However, no matter the accuracy or performance on GPU, CNN based trackers and are not suitable by far for the edge with no GPU as shown in Section VI. In contrast, the CF trackers such as Multiple Instance Learning (MIL) [1] and Boosting algorithm [17] are slower. The MOSSE filter [4] is very fast but not accurate. The KCF [23] is based on MOSSE too but it achieved better accuracy with supports from the HOG features. More details of the KCF is explained later in Section IV-A. SAMF [12], DS-KCF [19] or LCT [33] are created to tackle the fixed bounding box size problem of KCF and make more robust algorithms. Because of the boundary issues in frequency domain learning [16], some researchers use boundary learning methods to reach good performances. KCF has a much higher speed on CPU than others with satisfying accuracy.

III. Human Detection

Although discussed comprehensively in literature, in this section a brief overview of the Harr-Cascaded and HOG+SVM algorithms are provided too. Their wide usage for human detection in surveillance makes them noticeable candidates for edge application and exploration gives insight about their weaknesses on the edge devices.

A. Harr-Cascade

Haar-like features (filters) consist of three general shapes of rectangular form and an area in white and areas in black. These filters are going to convolute over an input image and in each position, the sum of pixel values in black rectangles is subtracted from the sum of pixel values in white rectangles. All possible scenarios considered, even a 24×24 image will produce more than 160 thousand features since filters can have any combination of sizes, rotations, and positions. The learning process is computationally expensive and needs to take place at CPU clusters, which might not be available to many. However, once the training is finished, a feature set is ready and only the selected features are stored for future classification. Thus, the computational complexity of the overall algorithm is small.

During the training phase, the best performing features are selected. best feature selection is performed by the AdaBoost algorithm which stands for Adaptive Boosting and it is constructed from classifiers that are called "weak learners". This algorithm generates a weighted sum between results of weak learners, such as (Eq. 1), where $h(x)$ is considered as each weak learner for input x. During the learning process, each weak learner receives a weight (α_t) in summation for error calculation (Eq. 2), which is based on the lastly calculated boosted classifier. The goal is set to minimize error where i is every input for learning iteration t. α_t is moves to zero for $h(x)$ who perform poorly.

$$F_T(x) = \sum_t f_t(x), where f_t(x) = \alpha_t h(x) \quad (1)$$

$$E_e = \sum_i E[F_{t-1} + \alpha_t h(x_i)] \quad (2)$$

During execution, in areas where features give positive results rigid regression gives a more accurate coordination as the output. The algorithm although very fast in execution, suffers from a high false positive rate, mostly because of simple pixel value based features (please refer to Section VI).

B. SVM Classifier

HOG is an alternative of using pixel values as features because pixels are not reliable. In HOG feature extraction X and Y derivatives are calculated simply by subtracting the horizontal and vertical neighboring pixel values respectively corresponding to each pixel of interest. In particular, the X derivatives are fired by the vertical lines, and Y derivatives are fired by horizontal lines, which makes the overall features to be sensitive to lines and object edges. Changing the presentation format to amplitude and angle will result in unsigned gradient representation of each pixel. In practice, a filter can be used to convolute over the image and in each step, calculate the gradient for a given pixel. Because of the unsigned gradients, the angular values are between 0° to 180°. If nine bins of 20° each are considered, then the amplitude of the gradients can be represented in the respected bin. It is worth mentioning that if an angular value is closer to the border of two bins, then the corresponding amplitude is going to be divided into the two bins that the angle falls in between. If the input image has more than one channel such as RGB, then the channel with the highest amplitude is chosen, and also the respective angle is used for histogram representation.

In an attempt to capture all details with different distances from a camera lens, usually a pyramid of the image is generated where the image with initial resolution is considered first, and then some pixels in each row and column are discarded to create a lower resolution version, and the same HOG process generates another feature map. The steps iterate until it is not feasible anymore to conduct classification on the image.

The SVM classifies objects of interest at each stage, so multiple detection reports are possible and in different scenes fine-tuning of the HOG variables might be needed. Figure 1

Figure 1. False multiple detection for a single human object.

is an example of the human object detection that gives several bounding boxes because multiple feature maps which are provided to the SVM with different resolutions from the image pyramid. Assuming to use the general pre-tuned variables yields an extra step to take only one of the bounding boxes and discard the rest. Although the detection rate can be improved by fine-tuning the filter size and variables, in practice, it is non-trivial to reconfigure once the cameras have already been installed. This method also suffers from the huge computation requirements of different stages explained previously.

C. Lightweight CNN

Recently, CNNs has been widely applied as powerful tools for object classifications. However, it is non-trivial to fit the regular full-size CNNs into the network edge devices due to the very restrict constraints on resources. Even if the time consuming and computing intensive training can be outsourced to the cloud, edge devices still cannot afford the storage space for parameters and weight values of filters of these deep neural networks. Therefore, a lightweight designed CNN is expected in the edge environment.

In designing the L-CNN architecture Depthwise Separable Convolution [24], [46] is employed to reduce the computational cost of the CNN itself, without much sacrificing the accuracy of the whole network. Also, the network is specialized for human detection to reduce the unnecessary huge filter numbers in each layer. This yields to a network implementable at the edge.

By splitting each conventional convolution layer into two parts, computational complexity becomes more suitable for edge devices, which is done using depthwise separable convolution and pointwise separable convolution. More specifically, the plain convolution will take an input such as F, which has a dimensionality of $D_f \times D_f$ and of M channels, and maps it into G, which is N channels of $D_g \times D_g$ dimension. This is done by the filter K, which is a set of N filters, each of them is $D_k \times D_k$ and has M channels. (look at (Eq. 3)):

$$G_{k,l,n} = \sum_{i,j,m} K_{i,j,m,n} \cdot F_{k+i-1,l+j-1,m} \tag{3}$$

The depthwise separable convolution consists of two parts. First M channels of $D_k \times D_k \times 1$ filters will generate M outputs, which is a depthwise convolution layer. Results are passed to a pointwise convolution layer in which the filters are N channels of 1×1 filters. Similarly, with the input of F as before this layer will produce an output such as $\hat{G}$ in (Eq. 4):

$$\hat{G}_{k,l,n} = \sum_{i,j,m} \hat{K}_{i,j,m,n} \cdot F_{k+i-1,l+j-1,m} \tag{4}$$

$\hat{K}$ is a depthwise convolutional filter, which has a special dimension of $D_k \times D_k \times M$ and the m_{th} filter in $\hat{K}$will be applied on m_{th} channel of F. (Eq. 3).

The computational complexity of each convolutional layer is reduced by a factor calculated by Eq. 5 [24]. Thus, depthwise separable convolution makes a faster and more efficient network that is an ideal fit for edge devices.

Type / Window Stride	Layer parameters	
	Filter shape	*Input size*
Conv / s2	3×3×3×16	224×224×3
Conv dw / s1	3×3×16 dw	112×112×16
Conv / s1	1×1×16×16	112×112×16
Conv dw / s2	3×3×16 dw	112×112×16
Conv / s1	1×1×16×32	56×56×16
Conv dw / s1	3×3×32 dw	56×56×32
Conv / s1	1×1×32×32	56×56×32
Conv dw / s2	3×3×32 dw	56×56×32
Conv / s1	1×1×32×64	28×28×32
Conv dw / s1	3×3×64 dw	28×28×64
Conv / s1	1×1×64×64	28×28×64
Conv dw / s2	3×3×64 dw	28×28×64
Conv / s1	1×1×64×128	14×14×64
5×Conv dw / s1 5×Conv / s1	3×3×128 dw 1×1×128×128	14×14×128 14×14×128
Conv dw / s2	3×3×128 dw	14×14×128
Conv / s1	1×1×128×256	7×7×128
Conv dw / s2	3×3×256 dw	7×7×256
Conv / s1	1×1×256×256	7×7×256
Avg Pool / s1	Pool 7×7	7×7×256

Figure 2. L-CNN network layers specification.

$$\frac{CC_{Depth}}{CC_{Conventional}} = \frac{D_k \times D_k \times M \times D_f \times D_f}{D_k \times D_k \times M \times N} \frac{\frac{N \times M \times D_f \times D_f}{\times D_f \times D_f}}{} = \frac{1}{N} + \frac{1}{{D_k}^2} \tag{5}$$

Immediately after each convolutional step, there is a Batch Normalization layer and a ReLU layer for normalization and nonlinearity respectively.

The proposed L-CNN network architecture has 23 layers which considers depthwise and pointwise convolutions as separate layers and does not count the final Fully Connected, softmax, regression layers to give a bounding box around the detected object [38]. A simple fully connected neural network classifier takes the prior probabilities of each window of objects, maps the objects within the proposed window to the list of known objects, and adds a label for output bounding boxes at the end of the network. Figure 2 depicts the network filter specifications in each layer.

Downsizing is with the help of no striding in filters. The first convolutional layer of the L-CNN architecture is a conventional version, but the rest of the network uses depthwise along with pointwise convolutions. The L-CNN is focused on a human object detection such that the list of known objects

is minimized to one, which further simplifies the network and decreases the number of parameters to store.

Introduced in late 2016, the Single Shot Multi-Object Detector (SSD) method is faster than R-CNN [31] and has more accuracy than YOLO [42]. The name comes from the fact that in one feed forward through the network, results are generated and there is no need for extra steps taken such as in R-CNN. It is a unified framework for the detection of an object with a single network. For training purposes, SSD architecture needs more layer architectures than classifier CNN, and when implemented, it will receive the input image and output the coordination of each object detected in the image along with a label for the object. Instead of having the classical sliding window and checking in each window for an object to report, SSD at the beginning layer of convolutional filters will create a set of default bounding boxes over different aspect ratios, then scales them with each feature map through convolutional layers along the image itself. In the end, it will check for each object category presence, based on the prior probabilities of the objects in the bounding box and finally adjusts the bounding box to better fit the detection which means adding five layers and using outputs of two convolutional layers in SSD application in overall architecture. One of the downsides of SSD is that in smaller object detection the accuracy is low if prior probability extraction performed in one layer. In smart surveillance, this can lead to loss of generalization. Because the goal is to detect every human object regardless of distance to camera or angle towards it. However, if the output of different feature maps from different layers is used [43] detection rate can be increased.

For training, images have to be the same size as the input of the network. The network accepts colored (RGB) images with the size of 224×224 pixels. Thus, for training a blob of 16 images each having three-channel data created and for validation, blobs of the same size are used. 85% of the total VOC07 and VOC12 [13] set used for training and 15% for validation. Lightning Memory-Mapped Database (LMDB) files were produced, which store data in a format as a $\{$*key, value*$\}$ pair, which leads to a faster reading speed. Furthermore, before the training, the data is normalized by calculating the mean value of each RGB channel.

Training is done on a server machine with 28 CPU cores of Intel(R) Xenon(R) CPU at the base frequency of 2.4 GHz with physical memory of 256 GB. Training took 5.7 days for each run with 200 epoch on the whole dataset.

IV. Tracking

In this section, three commonly used trackers are introduced that lay the foundation of the Kerman.

A. Kernelized Coloration Filter

The KCF at its core uses the ridge regression instead of more complex classifiers such as Support Vector Machine (SVM). This feature helps KCF use less resources while having a reasonable accuracy [23]. Basically, the training phase aims at finding solutions to problem of Eq. (6):

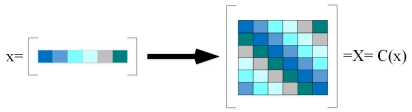

Figure 3. Converting a pixel vector x to a circulant matrix as X.

$$\min_{w} \sum_{i} (f(x_i) - y_i)^2 + \lambda \|w\|^2 \tag{6}$$

which is the square root error of the function $f(z) = w^t z$. Where λ is the regularization parameter and w is the vector of weights. Considering w as the linear combination of a function of samples yields:

$$w = \sum_{i} \alpha_i \varphi(x_i) \tag{7}$$

In Eq. (7), φ is the kernel function which updates the problem to finding the best α to minimize Eq. (6). Before jumping into high dimensions, let us take a step back and consider a simple form where the kernel function is not used and solution to w can be presented by Eq. (8), in which instead of a polynomial of kerneled inputs, it is equal to:

$$w = (X^H X + \lambda I)^{-1} X^H y \tag{8}$$

where y is the regression target and X^H is the Hermitian transpose of data matrix X that is generated with data samples x_i in each row.

Consider the positive sample as a vector called x, as depicted in Fig 3, the circulant translations of this vector are considered as negative samples [23]. The whole circulant matrix can be considered as the data samples that can be used in Eq. (8).

Different properties of the circulant matrices were exploited so far [18], with one of the most interesting properties being:

$$X = F diag(\hat{x}) F^H \tag{9}$$

In Eq. (9), F is a constant matrix which is known as the Discrete Fourier Transform (DFT) matrix. $\hat{x}$ is the DFT of x. Noted x here is the same as the vector used for circulant matrix generation. This yields taking the DFT of the vector and diagonalize it, with the DFT matrix gives the circulant matrix. Substituting $\hat{x}$ in the linear regression of Eq. (8), circulant properties can be used to simplify Eq. (8). The diagonal matrix can make all multiplications point wise.

$$X^H X = F diag(\hat{x}^\star . \hat{x}) F^H \tag{10}$$

where $\hat{x^\star}$ is the Hermitian transpose in the frequency domain, and Eq. (10) can grow into the linear regression in Eq. (8):

$$\hat{w} = \frac{\hat{x^\star} . \hat{y}}{\hat{x^\star} . \hat{x} + \lambda} \tag{11}$$

in Eq. (11) all operations are element-wise, even the division is between each element, which helps speed-up the operation. Thus, only DFT of the vector and the reverse DFT of w are needed.

Image processing based on a simple linear model often leads to an erroneous calculation, consequently, the object of interest is easily lost. Thus, a more robust but also more complex approach is adapted to map the linear data into higher dimensions. The same concept of circulant matrices, that make calculations faster, is going to be applied for non-linear regression too. One problem yet needs solution, which is: does the circulant properties remain after applying the kernel function. Based on Eq. (7), in order to find the solution to Eq. (6), a certain type of kernel function is needed on the right side of Eq. (12), which is called by function K here:

$$\varphi^T(x)\varphi(x') = K(x, x') \tag{12}$$

It can be shown that for a given circulant matrix, the corresponding kernel matrix is circulant too, if the kernel function has [23]:

$$K(x, x') = K(Mx, Mx') \tag{13}$$

where M is a permutation matrix, and most of the well-known kernel functions have this property and they can be leveraged to make faster online training and classification. Using this unique approach in online training, the KCF is more than $\times 5$ faster than tracking algorithms based on sophisticated machine learning methods [20]. The fast speed without GPU makes KCF an ideal candidate for applications at the edge of the network.

Meanwhile, the KCF has weaknesses to be addressed. It loses the object of interest if the object moves too fast, which is true no matter how well the algorithm is implemented. The reason is that the KCF algorithm trains itself in each frame, and if the object of interest moves out of the region of interest (bounding box around an object) within a couple of frames, it would train itself to focus on the background. Also, occlusions result in object loss with KCF, again because it trains itself with the foreground data and stops tracking the initial object, even when the object walks into a shadow this effect is seen as appearance changes suddenly. Moreover, human tracking is more complex in nature as there are many challenging scenarios in its movements in a crowded area. One solution is the depth information captured through a second camera [19]. However, nowadays it is common that there is only one single camera installed at each surveillance site which means the depth information cannot be extracted.

B. Kalman Filter

Kalman Filter (KF) is one of the most popular methods for system control. In object tracking the object of interest is viewed as a system with the central point of the bounding box as its representation, using KF position in the next frame can be predicted. For a point, its position at each time slot t is calculable based on its acceleration and velocity:

$$x_{t+1} = \frac{at^2}{2} + vt + x_t \tag{14}$$

In a two dimensional image, Eq. (14) can be used for x and y separately. Substituting v with $\dot{x}$ and a with $\ddot{x}$, Eq. (14) can be rewritten in a matrix form to describe the object using the equations of state:

$$Q = \begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \dot{x}_{t+1} \\ \dot{y}_{t+1} \end{bmatrix} = A \begin{bmatrix} x_t \\ y_t \\ \dot{x}_t \\ \dot{y}_t \end{bmatrix} + Ba \tag{15}$$

The outputs of importance are the position of the object, in other words, x and y:

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \end{bmatrix} = C \begin{bmatrix} x_t \\ y_t \\ \dot{x}_t \\ \dot{y}_t \end{bmatrix} \tag{16}$$

Based on KF core, these equations predict the position of the object for the following frame based on the current information. Then Eq. (14) determines A, B and C as:

$$A = \begin{bmatrix} 1 & 0 & T & 0 \\ 0 & 1 & 0 & T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, B = \begin{bmatrix} \frac{T^2}{2} \\ \frac{T^2}{2} \\ T \\ T \end{bmatrix}$$
$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{17}$$

where T is the time interval between the two frames, which can be different depending on the speed of the processor and the camera settings. a is the acceleration which is considered as a constant and it is the input of the dynamic system for highest level of simplification. In each frame the position can be updated using the equation below:

$$\begin{aligned} P &= A \times P \times A' + E_x \\ K_{KF} &= P \times C'(C \times P \times c' + E_z)^{-1} \\ Q &= Q + K_{KF} \times (m - C \times Q) \\ P &= (I - K_{KF} \times C) \times P \end{aligned} \tag{18}$$

In Eq. (18), P is the covariance estimation matrix and acts as the feedback of the system, K_{KF} is the KF gain and m is the true current position of the object of interest. Also, two error terms are considered for the system noise and the noise in the measurement, which are represented using E_x and E_z respectively with Gaussian noises:

$$E_x = \begin{bmatrix} \frac{T^4}{4} & 0 & \frac{T^3}{2} & 0 \\ 0 & \frac{T^4}{4} & 0 & \frac{T^3}{2} \\ \frac{T^3}{2} & 0 & T^2 & 0 \\ 0 & \frac{T^3}{2} & 0 & T \end{bmatrix} \tag{19}$$

and

$$E_z = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix} \tag{20}$$

σ_x^2 is generally set to 1. However, the parameter assignment can change based on the scene for better standalone KF performance.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2916416, IEEE Transactions on Services Computing

KF needs to be fed with the actual position of the object of interest in each frame to create feedback and update its parameters. Hence, the true position of the object should be measured which yields that the human detection algorithm needs execution for each frame [41], so KF cannot be used standing alone for the scenario at hand because human detection algorithm is not fast enough on the edge.

C. Background Subtraction

In this paper, the background subtraction method is based on Gaussian Mixture-based Background/Foreground Segmentation Algorithm introduced in[57]. Based on a Bayesian model, the number of components that are used to model each pixel in the GMM is determined. An adaptive system enables a balance that triggers less computation for simple-modeled pixels and maintains accuracy for more complex-modeled pixels. A pixel belongs to the background if the value calculated by Eq. (21) is higher than 1:

$$\frac{p(BG|\vec{x}^{(t)})}{p(FG|\vec{x}^{(t)})} = \frac{p(\vec{x}^{(t)}|BG)p(BG)}{p(\vec{x}^{(t)}|FG)p(FG)} \tag{21}$$

or a pixel such as $\vec{x}^{(t)}$ can be classified as part of the background if:

$$p(\vec{x}^{(t)}|BG) > c_{thr}(= p(\vec{x}^{(t)}|FG)p(FG)/p(BG)) \tag{22}$$

Considering a uniform distribution for BackGround (BG) and ForeGround (FG), Eq. (22) is based on a Bayesian model where c_{thr} is the threshold for a pixel being classified as the background. Eq. (23) shows the the GMM model for each pixel:

$$\hat{p}(\vec{x}|X_t, BG + FG) = \sum_{m=1}^{M} \hat{\pi}_m N(\vec{x}; \vec{\hat{\mu}}_m, \hat{\sigma}_m^2 I) \tag{23}$$

where X is the sample set of pixels used for determination of background, which consists of M pixels. μ and σ are Gaussian parameters from each sample and will be updated as new frames are processed. Because pixels in X can be in either background or foreground in different frames, the probability is expressed as $\hat{p}(\vec{x}|X_t, BG + FG)$. Using a loop for each frame the parameters for the Gaussian model are updated much like the updates in the KF:

$$\begin{aligned} \hat{\pi}_m &= \hat{\pi}_m + \alpha(o_m^{(t)} - \hat{\pi}_m) \\ \vec{\hat{\mu}}_m &= \vec{\hat{\mu}}_m + o_m^{(t)}(\alpha/\hat{\pi}_m)\vec{\delta_m} \\ \hat{\sigma}_m^2 &= \hat{\sigma}_m^2 + o_m^{(t)}(\alpha/\hat{\pi}_m)(\vec{\delta_m}^T\vec{\delta_m} - \hat{\sigma}_m^2) \end{aligned} \tag{24}$$

In Eq. (24), $\vec{\hat{\mu}}_m$ is the mean estimation for each pixel in X. $\hat{\sigma}_m$ is the m_{th} pixel's variance that is used for Gaussian models. α is an exponentially decreasing value that indicates the effect of previous pixel values in the update for current variables, such that the older frames have less effect in the update. α may be simply set as $1/T$, so the effect decreases linearly. $\hat{\pi}_m$ is the estimated mixing weights for sample pixels.

Furthermore, $\vec{\sigma}_m = \vec{x}^{(t)} - \vec{\hat{\mu}}_m$ shows the difference of each pixel value with the mean of sample m. For a new sample, the ownership function is a step function ($o_m^{(t)}$) which is equal to one, for components with Mahalanobis distance ($\vec{\delta_m}^T\vec{\delta_m}/\hat{\sigma}_m^2$) less than a threshold from the component under update; and is zero for the rest. The ownership function shows what component means around a specific pixel of concern are used to model that pixel. It should be noted that the update of $\hat{\mu}_m$ will classify a new object as part of the background if the object does not move for a while.

If $-c$ represents the prior probability of the model for background or foreground classes (negative sign means that each class exists when previous information about them are available). Thus, first line of Eq. (24) can be rewritten as:

$$\hat{\pi}_m = \hat{\pi}_m + \alpha(o_m^{(t)} - \hat{\pi}_m) - \alpha c_T \tag{25}$$

in which c_T is the prior constant for time of sampling (T).

The background subtraction gives contour around areas of motion, and is not suitable when implemented alone. However, it is useful to recalibrate KCF and KF when associated with objects of interest.

Figure 4. KCF algorithm lag when the object of interest moves fast.

V. Putting All Together

The non-trivial lags between the boundary box and the actual position of the object that happen for different reasons will fail the tracking algorithm. Figure 4 shows example cases in which the object was lost when using the KCF algorithm only. In the upper left image the tracker lag results in re-detection of the human and label him as a new object. This comparison is based on the KCF because to the best of the authors' knowledge, it is the fastest among today's online tracking methods, which are not based on CNN and GPU.

Aiming for a tracker that performs best in an edge device and keeps the speed of the KCF, Algorithm 1 mixes aforementioned trackers to cover each one's weak points while keeping the execution speed high. This hybrid algorithm (Kerman) makes decisions based on a majority voting mechanism. It takes into account the outputs of KCF, KF, and BS for each object of interest in a queue, and decides on recalibration of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2916416, IEEE Transactions on Services Computing

Algorithm 1 Hybrid_Tracker

procedure $Decision\_Tree.objupdates$(frame_400, cont)
  $feature \leftarrow bbx\_area.HOG()$
  if flag **then** ▷ no errors in fog features and data matrix
    $KCF.training(features)$
  if cls_pxl $\leftarrow$ 1.5×bbx.area() **then**
    $bbx \leftarrow KCF.classify$ ▷ pixels in ×1.5 the area of given bbx (comes from previous frame) using the trained model
  else
    $bbx \leftarrow KCF.classify$ ▷ the area of the bbx
  $cnt\_KCF \leftarrow KCF.center(bbx)$
  if flag **then** ▷ KCF classification finishes
    $cnt\_KF.update(bbx)$
  else
    $cnt\_KF.update()$
  $KF\_grad \leftarrow gradient(cnt\_KCF, cnt\_KF)$
  for each cnt_cont **do** ▷ BS algorithm
    if cnt_cont in bbx **then**
      $bs\_grad \leftarrow gradient(cnt\_KCF, cnt\_cont)$
  if bs_grad - KF_grad $<$ trhd **and** bs_grad found **then**
    $flag \leftarrow TRUE$
    $bbx \leftarrow bbx(cnt\_KCF, cnt\_cont)$ ▷ the bounding box is changed to have center same as middle of KCF and BS centers
  else if 180 - (bs_grad - KF_grad) $<$ trhd **and** bs_grad found **then**
    $flag \leftarrow TRUE$
    $bbx \leftarrow bbx(cnt\_KF)$ ▷ Partial collision detected and KCF recalibrated
  else
    $flag \leftarrow FALSE$ ▷ bounding box is given by the KCF
  return bbx

procedure TRACKER(camera1.stream)
  $frame \leftarrow camera1.stream$ ▷ OpenCV library
  while frame not empty **do** ▷ camera on
    $frame\_400 \leftarrow frame.resize(400 \times 400)$
    if $(framecount > human\_chk\_thld)$ **then**
      $fnd\_obj \leftarrow feed(L-CNN(frame\_400))$ ▷ human detection CNN based feed-forward
      for each fnd_obj **do**
        for each trk_obj **do**
          $Checking\_for\_new\_Objects$
        $Decision\_Tree.trk\_obj\_Q \leftarrow new\_coord$ ▷ add new object to the queue
    $cont \leftarrow bg\_sub(frame\_400)$
    for each Decision_Tree.trk_obj_Q **do**
      $Desision\_Tree.objupdates(frame\_400, cont)$
    $show.frame()$

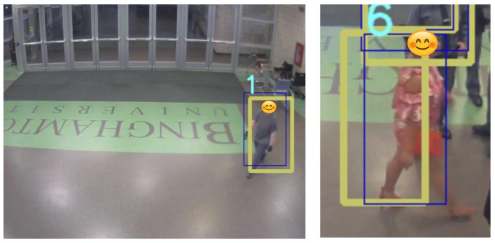

Figure 5. Kalman Filter will have a lag from KCF because KCF is used as real measurement.

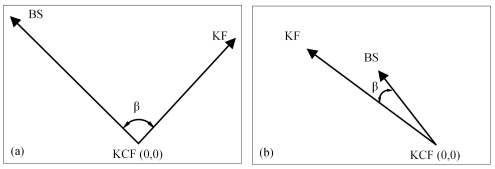

Figure 6. (a) Angle between BS and KF trackers' centers (β) that determines if the KCF should be recalibrated. (b) β too small yields to recalibration.

the bounding box or continuing with KCF's coordination. By default, the tracker is set to KCF's output, and center of the bounding box given by the KCF for each object is $(0, 0)$ for said object. Based on centers given by KF and BS, two vectors are calculated from the origin such as 6 (a).

If the BS and KF vectors show movement in the same direction as in Fig. 6(b) ($\beta < threshold$), KCF algorithm's coordination is recalibrated to a new coordinate which are between the BS and KCF algorithms' centers. This will resolve the problem of the KCF lagging behind if the object moves fast. Also, if the BS and KF show complete apposite directions, it is a sign of possible collision and the we use the KF coordination as the correct output. Note that using only one of these trackers with KCF will not give a robust decision because of their unreliable fluctuation. KF uses KCF as the true object poison and has fixed a, so it will follow KCF with a lag in case of sudden changes. This lag is obvious in Fig. 5, where the KF has a lag in comparison to the KCF algorithm when the object moves fast or changes its direction abruptly. Thus, if the object is moving fast at the instants of detection, both KF (having constant a) and BS which detects movement, follow the object and will not let KCF fall back by recalibrating it. Moreover, in case of occlusion, while KCF slows, KF will continue normal path momentary, also BS follows the portion of the body still visible (the window shrinks) and same effect will push the Kerman to detect object after occlusion. The effect is best seen with partial occlusions where KCF has low accuracy and KF can follow the object using previous information of speed and direction.

In the Kerman algorithm, a class $Decision\_Tree$ is introduced, objects from this class are created in a multi-thread manner in order to utilize the multi-core processor of the edge device. The number of threads depends on the number of cores the edge device has.

Figure 7 shows several examples of the different algorithm's centers.

As to be shown in Section VI even L-CNN as light as it is,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2916416, IEEE Transactions on Services Computing

Figure 7. Centers in each algorithm used for gradient calculation (red circle is KCF bounding box center, green is kF and blue is BS).

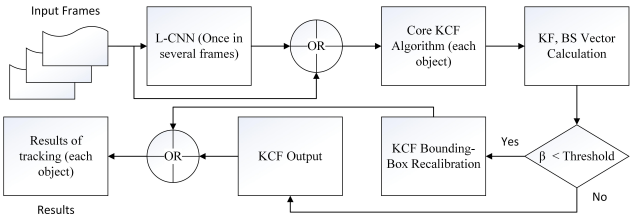

Figure 8. The overall algorithm work-flow.

reaches close to 2 FPS, which gives a very low resolution of extracted data for abnormality detection. Also, in order to have a history of each object's movement, detection is insufficient, because it gives the object's current coordinate. Thus, after detection of each new object, the L-CNN will add it to the tracking queue, so the tracker with faster performance follows him/her and deletes it after walking out of frame. As a result, the overall FPS of the algorithm improves. Noted that in case of undesirable object loss, the detection will add it to tracker queue again, though the interrupted tracking will further slow down the next steps in surveillance, *e.g.* anomalous behavior detection. The overall human detection and tracking algorithm work-flow is shown in Figure 8, where from the input frame the human object bounding boxes are given.

VI. Experimental Results

A. Experimental Setup

All of the above-discussed methods are implemented on two types of Single Board Computers (SBC) for test. One is a Raspberry PI 3 model B with 1 GB of RAM and ARMv7 1.2 GHz processor. The other device is a Tinker Board with 1.8 GHz ARM-based RK3288 SoC and 2 GB LPDDR3 dual-channel.

Single Board Computers (SBC), which run a full operating system and have sufficient peripherals (memory, CPU, power regulation) to start execution without the addition of hardware, are targeted industrial platforms such as vending machines. The Raspberry PI Foundation made the SBC accessible to almost anyone with low cost ($35) through delivering Raspberry Pi product family. However, with the low price the functionality of the SBC is not expected to be as high as some more expensive high-end chips. Given merits like commodity hardware, supporting high-level programming languages (e.g., Python) and running popular variants of Unix-based operating systems, The SBC is an ideal platform for Edge Computing.

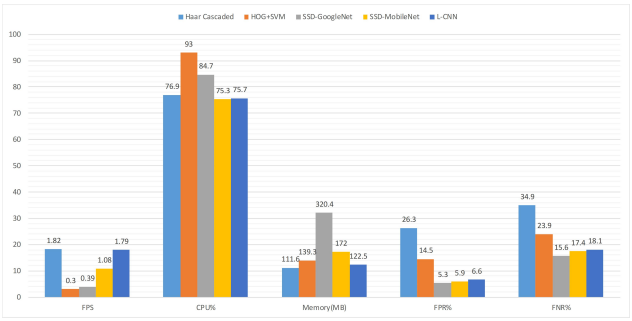

Figure 9. Performance in FPS, CPU, Memory Utility, Average False Positive Rate (FPR%) and Average False Negative Rate (FNR%).

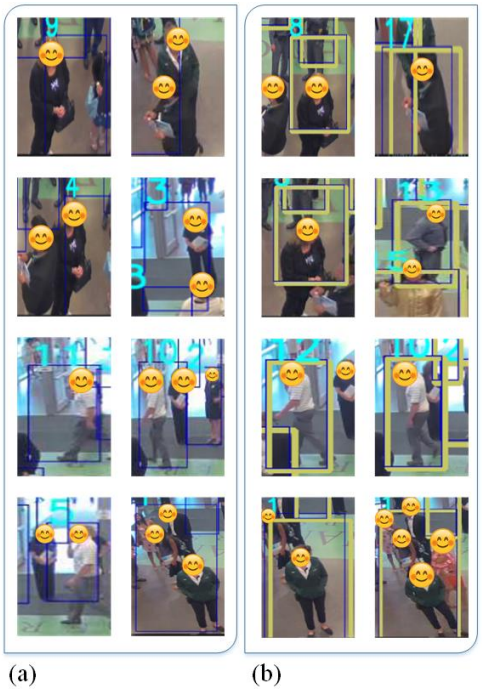

Figure 10. Performance comparison between the Kerman and KCF algorithms in case when objects move fast and occlusions exist. (a) KCF algorithm (b) KERMAN algorithm.

The CPU and memory utilization on edge are captured by a third party application named memory profiler. This software tracks python applications for their CPU and memory share in each second and saves the data which later can be plotted using python MATPLOTLIB library. FPS is the major parameter to evaluate the performance of human-object detection algorithms. Figure 9 shows the average FPS in 30 seconds of run time for each algorithm on a Raspberry PI. The proposed Kerman algorithm integrates three fast-tracking algorithms, KCF, KF, and BS, to achieve a higher accuracy than each individual one separately. Resulting differences are shown in Figure 10.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2916416, IEEE Transactions on Services Computing

Figure 11. Results in processing a sample surveillance video streams: (a) Haar Cascaded; (b) HOG+SVM; (c) SSD-GoogleNet; and (d) L-CNN.

B. Detailed Performance Analysis

The performance of human-object detection is summarized comparatively in Figure 9. The fastest algorithm is the Haar Cascaded, the proposed L-CNN is the second and very close to the best. The figure also shows that Haar Cascaded is the best in terms of resource efficiency, and again the L-CNN is the second and very close. However, in terms of average false positive rate (FPR) our L-CNN achieved a very decent performance (6.6%) and False Negative Rate (FNR) of 18.1% that is much better than that of Harr Cascaded (26.3% and 34.9% respectively). In fact, the L-CNN's accuracy is comparable with SSD GoogleNet (5.3% and 15.6%), but the GoogleNet is a much higher resource demanding and an extremely low speed (0.39 FPS) architecture that makes it not suitable for the edge. In contrast, the average speed of L-CNN is 1.79 FPS and a peak performance of 2.06 FPS is obtained. This speed of 64% faster than MobileNet along less memory usage, makes L-CNN the best choice.

It is worth mentioning that GoogleNet does not use a huge memory portion in contrary to other reports because this is a reduced SSD based GoogleNet. As shown in Figure 9 and Table I, with fewer classes, less parameters (thus less memory) is needed to reach to the same accuracy. To compute these accuracy measures, real-life surveillance video is used along with the VOC12 test and so percentages reported here may be higher than general purpose usage reported in other literature.

Figures 11 (a) to (d) show the results of Haar Cascaded, HOG+SVM, GoogleNet and L-CNN in processing a sample surveillance video. To compare all the algorithms fairly, they all fed image with the same size.

The Haar Cascaded algorithm has a high rate of FPR as shown in Figure 11 (a) as an example. Meanwhile, the HOG+SVM algorithm does not make the same mistakes as illustrated in Figure 11 (b). However, the bounding box is not fixed around the human objects. This may lead to inaccurate tracking performances in later steps. Figures 11 (c) and 11 (d) verify the high accuracy achieved by the CNNs at edge.

Figure 12. L-CNN: A human object from variant angles and distances.

Figure 12 highlights the results of the L-CNN algorithm in processing video frames in which human-object is captured from variant angles and distances. These are challenging scenarios for detection algorithms to decide whether or not the objects are human beings. Not only the visible features vary when the angles and distances are different, but also sometimes the human body is only partially visible or in different gestures. For example, in the right-up subfigure, the legs of the worker standing in the middle, are overlapped and the second person has only head and part of the left arm captured. In the left-bottom subfigure, both two legs of the pedestrian are not visible. Many algorithms either cannot identify it is a human body or as a result of lowering the confidence threshold, high FPR is seen.

Table I compares different CNN architectures with the proposed SSD based L-CNN algorithm, including several well-known architectures such as VGG, GoogleNet, and the lightweight MobileNet. The result matches our intuition very well that many heavy algorithms are not good choices for an edge device as they require up to 20 times more memory space.

Architecture	Memory (MB)
VGG	2459.8
SSD-GoogleNet	320.4
SqueezeNet	145.3
MobileNet	172.2
SSD-L-CNN	139.5

Table I. MEMORY UTILITY OF CNNS.

The performance of the combined algorithm has been evaluated experimentally on the selected edge devices in terms of memory consumption, CPU utility, and the video processing speed (FPS).

Figure 10 shows the results of a real-life surveillance footage processing, which compares the tracking power of Kerman against the KCF algorithm. Looking more closely to this figure, part (a) shows results of instances from KCF based algorithm, where the object is lost or the bounding box around the object has a lag or contains a huge space. In contrast, Figure 10(b) shows the results of Kerman algorithm tracking on the same video stream with the same instances, but the bounding box is a better fit. Unfortunately, to the best of authors' knowledge,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2916416, IEEE Transactions on Services Computing

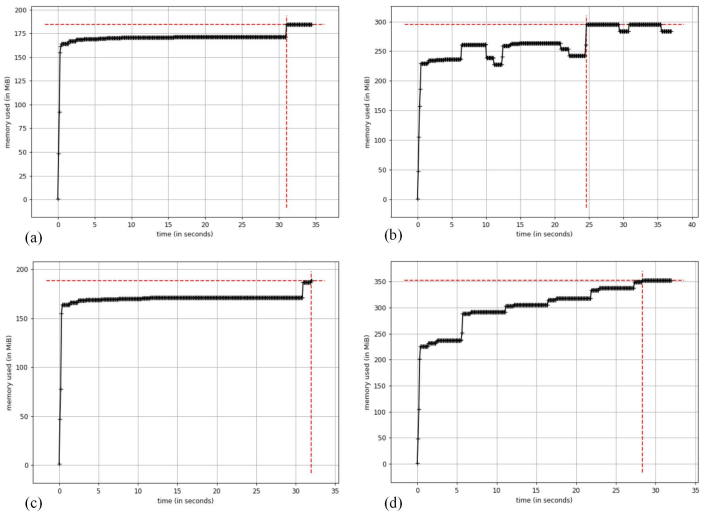

Figure 13. Memory needed in MB to run tracking and detection algorithms in 30 seconds run-time: (a) Kerman algorithm with 0-2 human objects in the frame (b) Kerman algorithm with 6-10 objects (c) KCF with 0-2 objects (d) KCF with 6-10 objects.

there are no data sets that can evaluate the whole process of human-object detection and tracking as a unified process explained here, that is why real surveillance video is used.

Figure 13 compares the memory consumption of the unified algorithm with Kerman for tracking against the memory used with the KCF tracking. The memory is read in 30 seconds of run time of two scenarios. In one scenario, there are only one or two human objects in the frame and they are positioned far away from each other. In the other scenario, the frame is more crowded with human objects and at most 10 pedestrians are in the frame at the same time. The memory utility is shown in Figure 13 includes both the tracking algorithms and L-CNN detection. The L-CNN detector is set to run two times per second and passing coordinates of possible new human-objects to the tracker which is sufficient based on human movement speed. In case there are no more than 10 humans in the frame the algorithm needs up to 350 MB of memory space, which is available even in a memory limited device like Raspberry PI with 1 GB of RAM. The experimental results also show that the memory consumption is not sensitive to the number of objects in the frame. The difference between having many objects and fewer objects is not significant, which verifies the Kerman algorithm is scalable in terms of memory utilization.

The CPU usage is also a critical metrics for the detection and tracking algorithms as a whole system designed for human surveillance automation. The percentage of CPU usage is read on Raspberry PI 3 and Tinker Board for 30 seconds of runtime and averaged. Same scenarios as used for memory consumption tests are applied to evaluate and assess the CPU usage, in the two scenarios. One scenario is with at most 2 human-objects in a frame and another is with 6-10 representing a crowded area. There is no surprise with the rise of CPU percentage when the number of the human-objects increases. At the same time, the tracking algorithm is not as heavy as the detection algorithm which allows for higher FPS.

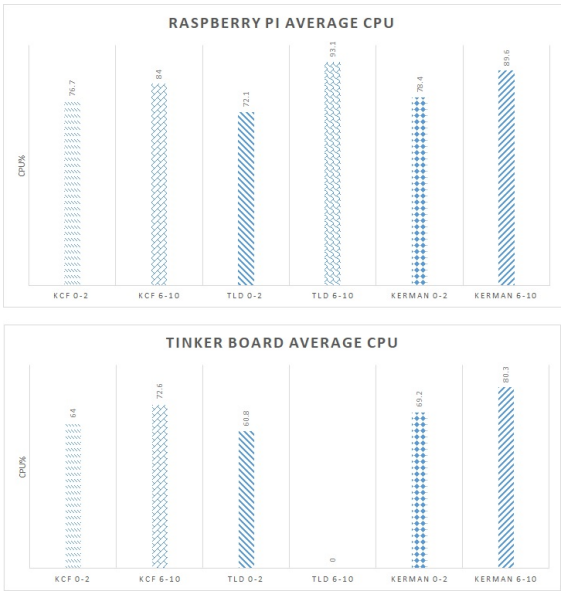

Figure 14. The average CPU usage. Raspberry PI 3 in upper part and Tinker Board in lower part (TLD was not successful when detecting more than 7 human objects on Tinker Board).

Other than the KCF algorithm, our Kerman is compared against several other good performing trackers such as Track-Learn-Detect (TLD) [52] that are implementable on edge. Figure 14 shows their CPU utilization. The TLD algorithm crashes when more than 7 human objects are present in the frame or in the tracking queue on Tinker Board. The problem may be in the power management of the Tinker Board, as it does not provide an enhanced operating system as Raspberry PI (Raspbian) does. It should be noted, however, that KCF performs best in continuous tracking even when sudden changes in the appearance of human-object take place [23]. Figure 14 also shows that the less powerful Raspberry PI device suffers from higher CPU usage. A good news is that there are new edge devices released with more capable hardware for edge computing as this field advances.

The dominant metrics is the throughput, the processing speed of these algorithms can achieve when implemented on the edge devices. Figure 15 presents the performances of L-CNN along with Kerman, KCF and TLD algorithms in terms of the average FPS. The values given in each second for the algorithm is based on the $FPS()$ function in the $imutils.video$ library. For 30 seconds of run time, these numbers are obtained using the same two operation scenarios, having at most two or 10 human objects in a frame on the two selected platforms. While the FPS of Kerman algorithm is lower than the KCF, the higher accuracy makes this trade-off tolerable. Normally, algorithms like TLD will use the entire CPU when conducting online training for the object they are tracking, which makes them slower than KCF algorithm, as shown in Figure 15. It should be noted that because other detectors simply do not have as good overall performance as L-CNN, the combination with trackers is not tested using each one.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2916416, IEEE Transactions on Services Computing

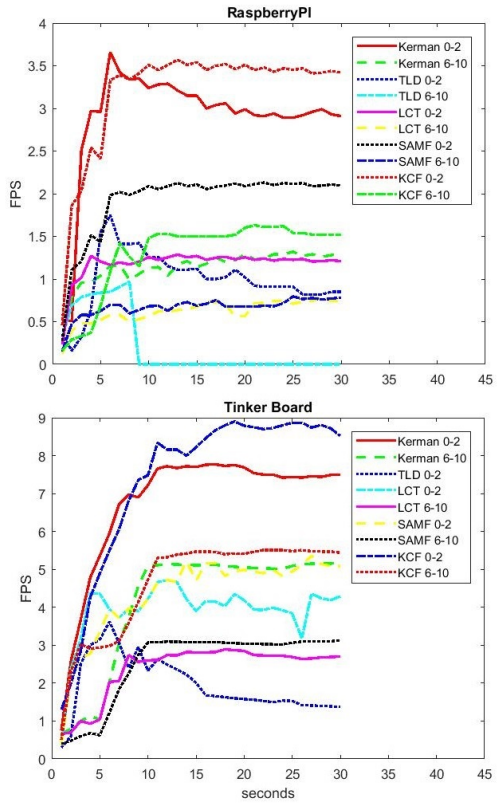

Figure 15. FPS for Kerman, KCF and TLD algorithms on two separate platforms.

Typically, today's smart mobile devices can run video playback at more than 60 FPS in some cases, for a 1080p stream. However, for surveillance cameras, the usual case ranges between 7.5 to 15 FPS. With the objective of human-oriented real-time (near real-time) surveillance, the real-time operation can be carried out with the worst case scenario of 5 FPS, which means every 200 ms the bounding box is moved. Normally the pedestrian speed is around 1.2 meters per second [27], and the minimum of 5 PFS can track such an object with ease. As the fastest speed of a person running is under 28 MPH and in average a person runs at 14 MPH or is less than six meters per second, even at worst case scenario, the object is still traceable with 1.2 meters per frame using the improved L-CNN detector and Kerman tracking algorithms.

The accuracy of the algorithms depends on many parameters such as the frame rate of the input video, background texture, object complexity, distance from the camera, and the illumination parameters of the scene. As trackers other than KCF and Kerman are very slow, implementing them on the edge gives a very low resolution of data extracted from the video, and so they are omitted from accuracy checks. Figure 16 compares the accuracy of the Kerman algorithm under different overlap thresholds with the KCF algorithm, as Kerman is based on KCF itself and tries to fix KCF deficiencies. Accuracy performance of the algorithms is discussed in literature comprehensively. The overlap threshold is the percentage of the

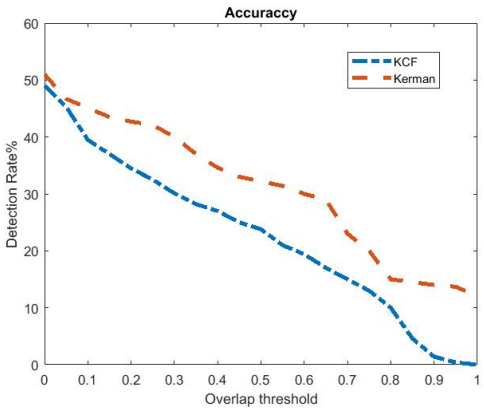

Figure 16. Accuracy comparison between Kerman and KCF algorithms.

Figure 17. Example of object loss and re-find.

object of interest that is blocked. As expected, the detection accuracy is dropped when more parts of the object get blocked. However, Kerman has a better detection rate Because of recalibration.

The last but not the least issue is the worst case scenario in which the Kerman algorithm loses the object under tracking. The detection algorithm runs two times per second, so the object can be found again quickly. The history of the object's movement may be lost in that unfortunate case. It is highly undesirable but in some cases may be unavoidable. Figure 17 shows a situation where the tracker fails to follow the object in the upper subplot and the L-CNN picks it up again and adds it back to the tracking queue and the bounding box around the object reappears.

VII. Conclusions

To make proactive urban surveillance and human behavior recognition and prediction as an edge network service, timely and accurate human-object detection at the edge is the essential first step. While there are many algorithms for human detection, they are not suitable for edge computing environment. In this paper, leveraging the Depthwise Separable Convolutional network, a lightweight CNN architecture is introduced for human object detection at the edge. This model was trained using VOC07 dataset and MXNet platform for neural networks, and later OpenCV libraries are used for implementation on the edge device.

Also in this paper, the Kerman algorithm is introduced that integrates three well-known lightweight tracking algorithms, KCF, KF and BS to enable the smart surveillance as an edge service. On the selected edge devices, the Kerman algorithm achieved decent performance in processing the real-world

security surveillance videos. The experimental results verified that the Kerman algorithm has solved the flaws associated with the KCF algorithm at a tolerable trade-off in processing time. It meets the design goals and is a feasible solution at the edge. Next step for us is to find reliable methods for abnormal behavior detection and prediction and implement them using the features extracted from the video for each object of interest.

REFERENCES

[1] B. Babenko, M.-H. Yang, and S. Belongie, "Visual tracking with online multiple instance learning," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 983–990.

[2] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr, "Fully-convolutional siamese networks for object tracking," in *European conference on computer vision*. Springer, 2016, pp. 850–865.

[3] E. Blasch, G. Seetharaman, S. Suddarth, K. Palaniappan, G. Chen, H. Ling, and A. Basharat, "Summary of methods in wide-area motion imagery (wami)," in *SPIE Defense+ Security*. International Society for Optics and Photonics, 2014, pp. 90 890C–90 890C.

[4] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, "Visual object tracking using adaptive correlation filters," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 2544–2550.

[5] H. Cao, M. Wachowicz, C. Renso, and E. Carlini, "An edge-fog-cloud platform for anticipatory learning process designed for internet of mobile things," *arXiv preprint arXiv:1711.09745*, 2017.

[6] A. Cenedese, A. Zanella, L. Vangelista, and M. Zorzi, "Padova smart city: An urban internet of things experimentation," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*. IEEE, 2014, pp. 1–6.

[7] F. F. Chamasemani and L. S. Affendey, "Systematic review and classification on video surveillance systems," *International Journal of Information Technology and Computer Science (IJITCS)*, vol. 5, no. 7, p. 87, 2013.

[8] N. Chen, Y. Chen, Y. You, H. Ling, P. Liang, and R. Zimmermann, "Dynamic urban surveillance video stream processing using fog computing," in *Multimedia Big Data (BigMM), 2016 IEEE Second International Conference on*. IEEE, 2016, pp. 105–112.

[9] Cisco, "Cisco visual networking index: Forecast and methodology, 20162021," *https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-indexvni/mobile-white-paper-c11-520862.html*, 2017.

[10] M. Cristani, R. Raghavendra, A. Del Bue, and V. Murino, "Human behavior analysis in video surveillance: A social signal processing perspective," *Neurocomputing*, vol. 100, pp. 86–97, 2013.

[11] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 886–893.

[12] M. Danelljan, F. Shahbaz Khan, M. Felsberg, and J. Van de Weijer, "Adaptive color attributes for real-time visual tracking," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Columbus, Ohio, USA, June 24-27, 2014*. IEEE Computer Society, 2014, pp. 1090–1097.

[13] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results," http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[14] T. Fuse and K. Kamiya, "Statistical anomaly detection in human dynamics monitoring using a hierarchical dirichlet process hidden markov model," *IEEE Transactions on Intelligent Transportation Systems*, 2017.

[15] H. K. Galoogahi, A. Fagg, C. Huang, D. Ramanan, and S. Lucey, "Need for speed: A benchmark for higher frame rate object tracking," in *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2017, pp. 1134–1143.

[16] H. K. Galoogahi, T. Sim, and S. Lucey, "Correlation filters with limited boundaries," in *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*. IEEE, 2015, pp. 4630–4638.

[17] H. Grabner, M. Grabner, and H. Bischof, "Real-time tracking via on-line boosting." in *Bmvc*, vol. 1, no. 5, 2006, p. 6.

[18] R. M. Gray *et al.*, "Toeplitz and circulant matrices: A review," *Foundations and Trends® in Communications and Information Theory*, vol. 2, no. 3, pp. 155–239, 2006.

[19] S. Hannuna, M. Camplani, J. Hall, M. Mirmehdi, D. Damen, T. Burghardt, A. Paiement, and L. Tao, "Ds-kcf: a real-time tracker for rgb-d data," *Journal of Real-Time Image Processing*, pp. 1–20, 2016.

[20] S. Hare, S. Golodetz, A. Saffari, V. Vineet, M.-M. Cheng, S. L. Hicks, and P. H. Torr, "Struck: Structured output tracking with kernels," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 10, pp. 2096–2109, 2016.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[22] D. Held, S. Thrun, and S. Savarese, "Learning to track at 100 fps with deep regression networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 749–765.

[23] 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*, vol. 37, no. 3, pp. 583–596, 2015.

[24] 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.

[25] W. Hu, Y. Huang, L. Wei, F. Zhang, and H. Li, "Deep convolutional neural networks for hyperspectral image classification," *Journal of Sensors*, vol. 2015, 2015.

[26] W. Hu, T. Tan, L. Wang, and S. Maybank, "A survey on visual surveillance of object motion and behaviors," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 34, no. 3, pp. 334–352, 2004.

[27] M. Hyland, H. Hunter, J. Liu, E. Veety, and D. Vashaee, "Wearable thermoelectric generators for human body heat harvesting," *Applied Energy*, vol. 182, pp. 518–524, 2016.

[28] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[29] N. Jenkins, "North american security camera installed base to reach 62 million in 2016," in *https://technology.ihs.com/583114/north-american-security-camera-installed-base-to-reach-62-million-in-2016*, 2016.

[30] K. Kolomvatsos and C. Anagnostopoulos, "Reinforcement learning for predictive analytics in smart cities," in *Informatics*, vol. 4, no. 3. Multidisciplinary Digital Publishing Institute, 2017, p. 16.

[31] 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*. Springer, 2016, pp. 21–37.

[32] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[33] C. Ma, X. Yang, C. Zhang, and M.-H. Yang, "Long-term correlation tracking," in *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*. IEEE, 2015, pp. 5388–5396.

[34] J. Ma, Y. Dai, and K. Hirota, "A survey of video-based crowd anomaly detection in dense scenes," *Journal of Advanced Computational Intelligence and Intelligent Informatics*, vol. 21, no. 2, pp. 235–246, 2017.

[35] O. Mendoza-Schrock, J. A. Patrick, and E. P. Blasch, "Video image registration evaluation for a layered sensing environment," in *Aerospace & Electronics Conference (NAECON), Proceedings of the IEEE 2009 National*. IEEE, 2009, pp. 223–230.

[36] H. Nam and B. Han, "Learning multi-domain convolutional neural networks for visual tracking," in *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*. IEEE, 2016, pp. 4293–4302.

[37] U. Nations, "World urbanization prospects: The 2014 revision, highlights. department of economic and social affairs," *Population Division, United Nations*, 2014.

[38] S. Y. Nikouei, Y. Chen, S. Song, R. Xu, B.-Y. Choi, and T. R. Faughnan, "Real-time human detection as an edge service enabled by a lightweight cnn," *arXiv preprint arXiv:1805.00330*, 2018.

[39] S. Y. Nikouei, R. Xu, Y. Chen, A. Aved, and E. Blasch, "Decentralized smart surveillance through microservices platform," *arXiv preprint arXiv:1903.04563*, 2019.

[40] S. Ojha and S. Sakhare, "Image processing techniques for object tracking in video surveillance-a survey," in *Pervasive Computing (ICPC), 2015 International Conference on*. IEEE, 2015, pp. 1–6.

[41] H. A. Patel and D. G. Thakore, "Moving object tracking using kalman filter," *International Journal of Computer Science and Mobile Computing*, vol. 2, no. 4, pp. 326–332, 2013.

[42] 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*, 2016, pp. 779–788.

[43] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017.

[44] M. Ribeiro, A. E. Lazzaretti, and H. S. Lopes, "A study of deep convolutional auto-encoders for anomaly detection in videos," *Pattern Recognition Letters*, 2017.

[45] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[46] L. Sifre, "Rigid-motion scattering for image classification, 2014," Ph.D. dissertation, Ph. D. thesis, PSU.

[47] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[48] X. Wang, "Intelligent multi-camera video surveillance: A review," *Pattern recognition letters*, vol. 34, no. 1, pp. 3–19, 2013.

[49] A. Wong, M. J. Shafiee, F. Li, and B. Chwyl, "Tiny ssd: A tiny single-shot detection deep convolutional neural network for real-time embedded object detection," *arXiv preprint arXiv:1802.06488*, 2018.

[50] C. R. Wren, A. Azarbayejani, T. Darrell, and A. P. Pentland, "Pfinder: Real-time tracking of the human body," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 19, no. 7, pp. 780–785, 1997.

[51] Y. Wu, J. Lim, and M.-H. Yang, "Object tracking benchmark," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 9, pp. 1834–1848, 2015.

[52] Z. Xin, Q. Qiumeng, Y. Yongqiang, and W. Congqing, "Improved tld visual target tracking algorithm," *Journal of Image and Graphics*, vol. 18, no. 9, pp. 1115–1123, 2013.

[53] R. Xu, S. Y. Nikouei, Y. Chen, E. Blasch, and A. Aved, "Blendmas: A blockchain-enabled decentralized microservices architecture for smart public safety," *arXiv preprint arXiv:1902.10567*, 2019.

[54] P. Zhao, W. Yu, X. Yang, D. Meng, and L. Wang, "Buffer data-driven adaptation of mobile video streaming over heterogeneous wireless networks," *IEEE Internet of Things Journal*, 2017.

[55] Y. Zheng, L. Capra, O. Wolfson, and H. Yang, "Urban computing: concepts, methodologies, and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 5, no. 3, p. 38, 2014.

[56] L. Zhou, S. Pan, J. Wang, and A. V. Vasilakos, "Machine learning on big data: Opportunities and challenges," *Neurocomputing*, vol. 237, pp. 350–361, 2017.

[57] Z. Zivkovic and F. Van Der Heijden, "Efficient adaptive density estimation per image pixel for the task of background subtraction," *Pattern recognition letters*, vol. 27, no. 7, pp. 773–780, 2006.

Mr. Sayed Yahya Nikouei earned his BS from the Electrical & Electronics Department at Shahrekord University, Iran in 2014. He received his MS in 2016 from University of Isfahan, Iran also in electrical & electronics specializing in random hierarchy algorithms to optimize Inverter performance. Currently he is pursuing his doctoral degree at Binghamton University and his research focuses on Machine Learning and pattern recognition methods and Edge-Fog-Cloud Computing and the applications in Smart Cities such as the smart surveillance system.

Dr. Yu Chen is an Associate Professor of Electrical and Computer Engineering at the Binghamton University - SUNY. He received the Ph.D. in Electrical Engineering from the University of Southern California (USC) in 2006. His research interest lies in Trust, Security and Privacy in Edge-Fog-Cloud Computing, Internet of Things (IoTs), and their applications in smart and connected environments. His publications include over 150 papers. He has served as reviewer for NSF panels and for journals, and on the TPC of prestigious conferences.

Dr. Sejun Song is an Associate Professor of Department of Computer Science Electrical Engineering at the University of Missouri-Kansas City. He received the Ph.D. in Computer Science and Engineering from the University of Minnesota, Twin Cities, in 2001. His research interest lies in Software defined networks, cloud computing and data center networks; Network systems security and fault-tolerance management; Embedded real-time systems and controller area networks; Mobile operating systems and mobile cloud computing. His publications include over 100 papers in scholarly journals, conference proceedings, and books.

Dr. Baek-Young Choi is an Associate Professor of Department of Computer Science Electrical Engineering at the University of Missouri-Kansas City. She received the Ph.D. in Computer Science and Engineering from the University of Minnesota, Twin Cities, in 2003. Her research interest lies in Cloud computing and software-defined networks; Network algorithms and protocols; Data storage and management systems; and Network traffic/performance/security: measurement, analysis and modeling.

Mr. Timothy R. Faughnan is the Chief of Police of the State University of New York Police Department at Binghamton University. He received the Bachelor of Science from Ithaca College in 1981. Responsible for command and overall leadership of a fully empowered, 24 hour per day police department, Mr. Faughnan bring his 36 years' experience in law enforcement to enable the team a perspective on all aspects of policing the community. He provides campus surveillance data and law enforcement consulting to the iSENSE project.