Litedge: Towards Light-weight Edge Computing for Efficient Wireless Surveillance System

Yutong Liu
isabelleliu@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Linghe Kong
linghe.kong@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Muhammad Hassan
hassankhan@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Long Cheng
lcheng2@clemson.edu
Clemson University
Clemson, South Carolina, USA

Guangtao Xue
x-gt@cs.sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Guihai Chen
gchen@cs.sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

ABSTRACT

Wireless surveillance systems are rapidly gaining popularity due to their easier deployability and improved performance. However, cameras inside are generating a large amount of data, which brings challenges to the transmission through resource-constrained wireless networks. Observing that most collected consecutive frames are redundant with few objects of interest (OoIs), the filtering of these frames can dramatically relieve the transmission pressure. Additionally, real-world environment may bring shielding or blind areas in videos, which notoriously affects the accuracy of frame analysis. The collaboration between cameras facing at different angles can compensate for such accuracy loss.

In this work, we present *Litedge*, a *light*-weight *edge* computing strategy to improve the QoS (*i.e.*, the latency and accuracy) of wireless surveillance systems. Two main modules are designed on edge cameras: (i) the light-weight video compression module for frame filtering, mainly realized by model compression and convolutional acceleration; and (ii) the collaborative validation module for error compensation between the master-slave camera pair. We also implement an enhanced surveillance system prototype from real-time monitoring and pre-processing on edge cameras to the backend data analysis on a server. Experiments based on real-world collected videos show the efficiency of *Litedge*. It achieves 82% transmission latency reduction with a maximal 0.119s additional processing delay, compared with the full video transmission. Remarkably, 91.28% of redundant frames are successfully filtered out, greatly reducing the transmission burden. *Litedge* outperforms state-of-the-art light-weight AI models and video compression methods by balancing the QoS balance ratio between accuracy and latency.

CCS CONCEPTS

• **Computer systems organization** → **Client-server architectures**.

KEYWORDS

QoS, wireless surveillance system, light-weight AI, edge computing, collaborative validation

ACM Reference Format:
Yutong Liu, Linghe Kong, Muhammad Hassan, Long Cheng, Guangtao Xue, and Guihai Chen. 2019. Litedge: Towards Light-weight Edge Computing for Efficient Wireless Surveillance System. In *IEEE/ACM International Symposium on Quality of Service (IWQoS '19), June 24–25, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3326285.3329066

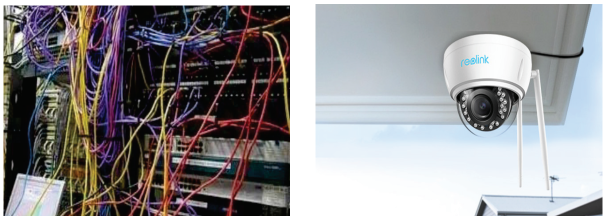
1 INTRODUCTION

Video surveillance systems nowadays are pervasively deployed in various daily scenarios: traffic monitoring, parking management, public security in campus, office buildings, or residential communities. As shown in Fig. 1, different from the wired camera infrastructure, which incurs heavy burdens on deployment and maintenance, smart surveillance systems composed by wireless cameras and cloud computing are gaining great popularity with their higher security levels and easier installations. A 2018 survey revealed that 430 wireless cameras deployed in Warren County Public Schools of Washington, D.C., has 24 hours' close watch on nine schools with 5000 students and 800 employees [1]. This monitoring system continuously protects their security in the campus. However, according to this survey, these 430 cameras produce 864 gigabytes video every day. With the growing size of monitoring areas and the number of cameras, the increasing quantity of video streams imposes great challenges on transmission through resource-constrained wireless networks. The quality of service (QoS) for wireless surveillance systems, including the latency and accuracy, will be affected accordingly.

Fortunately, mobile edge computing [2] provides a promising direction to deal with these challenges. It brings the

IWQoS '19, June 24–25, 2019, Phoenix, AZ, USA

ACM ISBN 978-1-4503-6778-3/19/06...$15.00
https://doi.org/10.1145/3326285.3329066



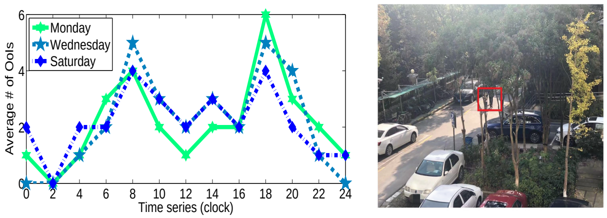
(a) Inflexible deployment of wired surveillance deployment.
(b) Easy deployment of wireless surveillance cameras.

Figure 1: Inflexible wired surveillance system VS. easily deployed wireless system.

network functions, contents and resources closer to end devices (e.g., edge cameras in surveillance systems) [3], which makes it possible for local-processing of collected videos before transmission. [4] and [5] suffer from either coarse filtering or non-negligible detection latency, resulting in poor QoS balance in terms of accuracy and latency. In addition to traditional video coding [6] or capturing rate deduction [7], a light-weight and accurate *content-aware local-processing* method on edge cameras is desirable. On the other hand, complicated physical environment brings shielding or monitoring blind areas, which may inevitably cause information loss in video analysis. The *collaborative validation* among multiple cameras can proactively compensate the error caused in the former frame filtering analysis. Despite researches on distributed validations [5, 8, 9] based on Multi-Camera Coordination and Control (MC^3) scheme [10], they are lacking in error compensation mechanism.

Our approach is motivated by two observations: First, according to 168 hours (one week) video analysis on one camera in a residential community, except the rush hours (7:00-8:00 and 17:00-19:00), there were fewer OoIs (*i.e.*, human, cars, and animals[1]) detected in videos shown in Fig. 2(a). It implies large redundancy in monitoring videos. Second, the massive buildings, plants, and facilities in residential communities may bring the shielding even blind monitoring areas, where only partial OoIs can be detected or totally covered. Two persons in the frame (labeled by the red box in Fig. 2(b)) are partially recognized, since they are covered by trees.

In this paper, we propose *Litedge*, a QoS-enhanced edge computing strategy for wireless surveillance systems. This strategy is mainly composed of two modules: light-weight video compression on edge cameras and collaborative validation among them. In the first module, we optimize a light-weight Single Shot MultiBox Detector (SSD) model to filter out redundant frames by OoI detection. Three optimizations are proposed to be adapted to wireless cameras with lower computational capabilities: (i) background pruning among consecutive frames from raw videos; (ii) model compression and convolutional acceleration; and (iii) output class deduction and merging. The second module is complementary to



(a) The average number of OoIs in different time durations within a day.
(b) The tree shielding on two monitored persons.

Figure 2: Illustrations for two observations of monitoring redundancy and environment sheilding.

the accuracy loss in the first module. Once detecting unrecognized objects, which are partly shielded by the physical environment, the master camera requests on its neighboring slave camera for validation. Based on the fixed positions and distributions of cameras, we design a simplified projection depending on pin-hole imaging theory accompanied with AI matching to localize and validate the requested objects. After processing through these two modules, *Litedge* can successfully select fewer but representative frames in videos to reduce transmission redundancy, and thus improving QoS in wireless surveillance systems.

We implement a complete surveillance system prototype from edge cameras to the central server. Experiments based on this prototype and real-collected videos show the feasibility of *Litedge* on QoS enhancement. We mainly evaluate two QoS factors: latency and accuracy. As for the latency, *Litedge* achieves 82% transmission latency reduction with maximal 0.119s additional processing delay (0.049s for AI module and 0.07s for validation module), compared with the original surveillance system. And for the accuracy, it successfully filters out 91.28% redundant frames with a 5.69% compensation ratio by collaborative validation. Overall, its light-weight AI module acquires the highest QoS balance ratio (1746.9), compared with three state-of-the-art benchmarks: DeepMon [11] (27.98), DeepEye [12] (18.1), and MobileNet [13] (65.55). Further compared with the original video transmission, compression-only strategy, and the closely related method [5], *Litedge* also achieves the best QoS balance.

This paper makes the following contributions:

(1) **Light-weight video compression:** We optimize a light-weight SSD model to adapt to the limited computational capability of edge cameras in wireless surveillance systems. Its processing on videos has faster speed and requires fewer computational resources.
(2) **Collaborative validation on cameras:** We design a collaborative scheme between edge cameras for validating partially covered objects. It behaves as a compensation of the accuracy loss in the former compression stage, which is caused by physical environment shielding.
(3) **Surveillance system implementation:** We implement a complete surveillance prototype from video capturing, local-processing and validating on edge cameras, transmitting, and remote controlling in the server. Experiments based on real-life collected videos demonstrate the efficiency of our prototype.

[1] We select these three OoI categories based on observations from captured videos in residential communities, which are frequently appeared (contained in over 50% frames). And anomalous behaviors of these OoIs indicate potential safety risks in daily life.

Note that two modules designed in this paper can be easily extended to other application scenarios. The first light-weight AI optimization can be adapted to most sparse deep learning models occupied by rich convolutional layers, and further deployed on commodity IoT devices with constrained computational resources and capabilities. Moreover, the second collaborative validation can be utilized to other collaborative communication environment, such as robotic or manufactoria, to fulfill accuracy requirements.

2 RELATED WORK

In wireless surveillance systems, the interconnections between cameras and cloud via the wireless network create a typical edge computing architecture [14]. Recent advances have been made utilizing the local computation on edge devices and collaborative computation among cameras.

2.1 Local-processing on Edge Devices

Traditional solutions on video compression are video coding techniques standardized by MPEG [6] or simply reduce the sampling rate [7]. Although these solutions can keep low distortion rate, they are not adaptive and flexible enough for desirable redundancy deduction. Recently, content-aware video compression method attracts considerable attentions. Wu et al. [4] applied a foreground and background separation algorithm, where only regions of interest are extracted and processed. But this rough division leads to unsatisfied compression result. Zhang et al. [5] applied a vision algorithm on cameras for face detection. As this algorithm is not optimized according to the computational capability of cameras, it incurs non-negligible processing overhead.

AI techniques are widely used in video analysis. Limited by the computational capability of edge cameras, light-weight AI models are required for efficient computation. The cache-based convolutional optimization and tensor decomposition in DeepMon [11] can decrease the computational complexity of models. DeepEye [12] leverages the memory caching and SVD-based model compression to support multi-model running. MobileNet [13] presents a depth-wise separable convolution, combined by a single-filter derived convolution and 1*1 pointwise convolution. However, their targeting devices are supported by quite powerful CPU and GPU processors (*e.g.*, 2GHz), which is not applicable to commodity surveillance cameras. In this paper, our object is to optimize the deep learning model from both model construction and calculation aspects, to build a more efficient model structure.

2.2 Multi-camera Validation

Inevitable information missing (e.g., undetected or unclear objects) remains unsolved when using only one camera analysis. A scheme called MC^3 [10] addresses this issue. Collins et al. [15] firstly proposed a collaborative master-slave framework, where the master camera is used to track the trajectories of objects, and the slave camera performs specific recognition. This framework is utilized in different scenarios, including traffic anomalies [8], face detection [9], etc. Zhang et al. [5] leveraged object re-identification between cameras in a cluster based on a utilization matrix analysis. Nonetheless, they did not consider the error compensation after the projection, leading to 89 pixels' error between the projection and the target. In *Litedge*, we seamlessly connect the AI detection module with the validation module, where the AI detection results can help to match the projection with detected OoIs, and thus improving the validation accuracy.

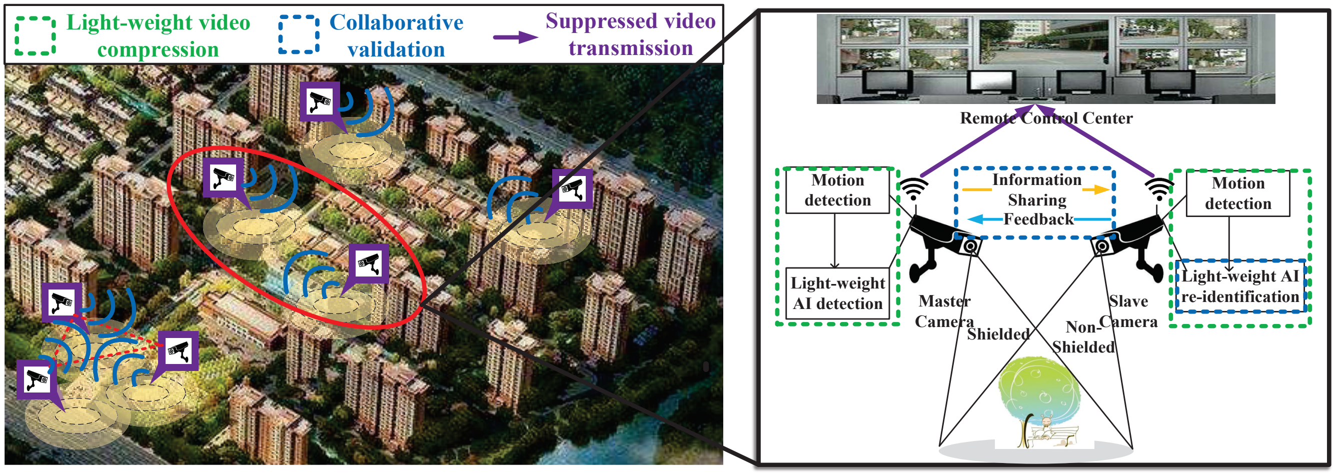
3 PROBLEM STATEMENT AND FRAMEWORK OF *LITEDGE*

The application scenarios of surveillance cameras include residential communities, offices, campus and so on. In this paper, we take the residential community as an example scenario, where the challenges faced by wireless surveillance system are more representative. As shown in the left figure of Fig. 3, surveillance cameras are deployed at important corners with fixed positions, focal lengths and monitoring directions. Neighboring cameras have overlapped monitoring areas but with different angles, and they can communicate with the embedded WiFi module in a short range (around 15 meters for commodity cameras) and the limited bandwidth. Each camera has two different monitoring modes: active mode and sleep mode. In the active mode, cameras capture videos in a high ratio (*e.g.*, 60 fps), but for the sleep mode is in a low ratio (*e.g.*, 15 fps) [16]. Each camera has its independent processing system with an embedded core processor without GPU, which has the lower computational capability on processing video streams, and difficult to handle heavy computational tasks (*e.g.*, running a deep learning model with above 10 layers). All these cameras have constant power supply, where power savings are out of our research scope.

The goal of our research is to increase the QoS of this wireless surveillance system, where two main factors in QoS are considered:

(1) **Low latency:** The low latency in this paper indicates both low processing latency on the edge and low transmission latency of videos from the edge to a cloud server. The processing latency is measured on a single frame, which is the elapsed time from AI triggering to feedback receiving. And the transmission latency is the total time of uploading the processed videos to a server.
(2) **High accuracy:** Two types of accuracies are measured: detection accuracy for the light-weight AI model running on the camera; and redundancy filtering accuracy whose ground truth is calculated by the original AI model running on collected videos at the server side.

Correspondingly, we design *Litedge*, a light-weight edge computing strategy for wireless surveillance systems shown in Fig. 3, which is composed of two main modules labeled by dotted boxes: light-weight video compression module and collaborative validation module. Considering that fewer OoIs typically show up other than rush hours in residential communities, each camera works in the sleep mode by default, and activates its AI module once the motion detected for



Surveillance cameras distribution diagram

Litedge on one master-slave pair

Figure 3: The framework of *Litedge*. [Left: Camera distribution in one community (Yellow area implies the monitoring coverage of the camera, and blue line implies their WiFi communications); Right: *Litedge* illustration from edge computing to transmission for one master-slave pair (marked in the red circle).]

electricity saving. **Light-weight video compression** aims to filter out redundant frames by OoI detection. According to our experimental measurements, three types of objects in frames are defined as: *recognized objects* with over 50% detection confidence; *unrecognized objects* with 0-50% detection confidence; and *none of interest* with 0% detection confidence. Correspondingly, the types of frames are: *selected frames* containing any recognized objects; *unrecognized frames* with only unrecognized objects inside; and *redundant frames* without any recognized or unrecognized objects appeared. Any recognized object founded leads to the direct upload of the corresponding frame. Whenever unrecognized objects detected in frames, the **collaborative validation** module will be triggered between master-slave camera pair.

As shown in the right figure of Fig. 3, a person under the tree is partially shielded, and thus recognized as an unrecognized object for the master camera. According to the position and direction of this person, the master camera will select a close neighbor with suitable angle as its slave camera. Although there is only one slave camera in one-time validation, any neighboring camera in the cluster within one-hop communication range has a chance to be selected. Necessary information associated with this object (*e.g.*, location and corresponding timestamp) will be shared from master to slave. Note that each slave node also has functionalities of master cameras, its AI detection results will be extracted for the matching stage to decrease the projection error. After matching, the decision about whether to upload the frame will be sent back to the master camera.

The object detection on cameras will keep running until no OoIs detected in consecutive frames, then cameras will be set back to the sleep mode. Finally, selected frames will be compressed by MPEG-4 standard and uploaded to the server, and shown in the screen of the remote control room.

4 DETAILED DESIGN OF *LITEDGE*

The detailed design of two main modules in *Litedge*: light-weight video compression module and collaborative validation module will be discussed in the following subsections.

4.1 Light-weight Video Compression

As one of the state-of-the-art OoI detection models, the SSD model [17] outperforms with higher accuracy and detection speed. However, it contains 11 convolutional layers and 8732 prior boxes for each output class, which will cause heavy computational burdens for cameras. To efficiently detect redundant frames on surveillance cameras, we optimize the SSD model to a light-weight structure on three aspects:

(1) **Feed data pruning:** The background pixels in frames of surveillance videos always keep still because of the fixed monitoring angles and ranges, which introduces redundancy for deep learning computation [11]. As our focus is on the dynamic foreground objects, we intend to prune the background before serving the frame to the model.
(2) **Model optimization:** To optimize the model structure, we first compress the SSD model by channel pruning [18] and then accelerate the convolutional computation which is proved to be computational heavier than other layers [11, 13, 19]. These optimizations are expected to preserve the detection accuracy within the acceptable error range.
(3) **Output class deduction:** Observing that three classes detected (*i.e.*, human, transportation, and animal) in residential communities dominates most (92% in our experiments) meaningful frames in surveillance videos, we deduct the number of output classes, where fewer checks are needed before giving detection results, and thus decreasing the AI processing time.

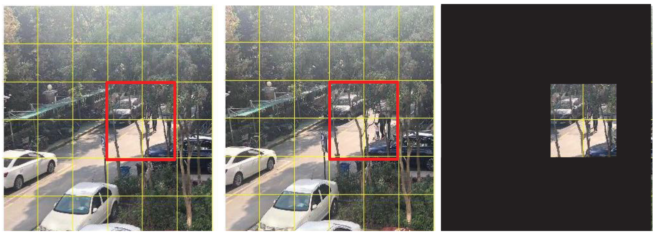


Figure 4: An example of frame pruning (the red box shows different blocks; the black area in the 3rd picture is the pruned pixels in the 1st picture).

4.1.1 Feed Data Pruning. The idea of frame pruning is to erase the same pixels between two consecutive frames. To start the comparison, we first divide each $M \times M$ frame F_i into a grid with $m \times m$ size of blocks, which can be denoted as $b_{i1}, b_{i2}, ..., b_{ia}$. Here, a is the number of divided blocks and $a = \frac{M^2}{m^2}$. The similarity $s_{i(i+1)}$ is a result matrix calculated by the matrix deduction in paired blocks of two consecutive frames F_i and F_{i+1}. For instance, the similarity of the first block pair is $\mathrm{s}_{i(i+1)}[1] = |b_i[1] - b_{i+1}[1]|$. Same blocks where $s_{i(i+1)}[a] = 0$ will be pruned. As shown in Fig. 4, the first image and the second one are two consecutive frames, where only two persons in the center are moving while other background images are static. After comparing with the second frame, the static pixels in the first frame are pruned, which are marked as black in the third image. It is obvious that the pruned picture is far smaller than the original picture, which can successfully reduce the computational burdens in the following processing.

There are plenty of methods to find similar regions between two images, like SIFT [20], color-based algorithm [11], or Gaussian mixture model with color and deep channels [21]. But the tradeoff between limited computational speed and highly-required low latency cannot be realized when they are applied on surveillance cameras. The direct deduction between two frames is the simplest way for frame pruning. Even it is not the most accurate method, as a pre-processing method, its accuracy loss is tolerable. Important concerns like entering in and leaving from the monitoring areas and other abnormal situations such as vehicle collision, animal attacking, people fighting are always dynamic, and thus will be remained without pruning.

4.1.2 Model Optimization. The limited computational capabilities on commodity surveillance cameras cannot efficiently support the running of computationally expensive deep learning models (*e.g.*, the original SSD model with 11 convolutional layers). The deep learning model applied in this module should be light-weight to adapt to edge cameras. To this end, we design two optimized operations: model compression and convolutional acceleration.

Model compression. Depending on the sparsity of deep learning model, it can be compressed unstructured [22, 23] with higher precision but relying on specific libraries and platforms supports. So in *Litedge*, we utilize channel pruning [18], a structural compression directly on the SSD model. Channel pruning is suitable to compact single brunch model like the SSD model. During the layer-by-layer sequential pruning, the accumulated error should be minimized on each output feature maps. Two main steps are applied to the target model. First, the most representative channels will be selected, while redundant channels are filtered. Second, the selected channels will be reconstructed as the new feature map. For calculation convenience, we denote that a feature map in the SSD model has c channels; the size of convolutional filter W is $n \times c \times k_h \times k_w$; and the size of input volumes X is $N \times c \times k_h \times k_w$, deriving an $N \times n$ output matrix Y. N is the number of samples, and n is the number of output channels, together with $k_h \times k_w$ size of the kernel. The pruning process can be formulated as:

$$\underset{\beta,W}{\arg\min} \frac{1}{2N} \left\| Y^{'} - \sum_{i=1}^{c} \beta_i X_i W_i^T \right\|_F^2 \qquad (1)$$

$$\text{subject to } \|\beta\|_0 \le c^{'}.$$

Inside of the Frobenius Norm $\|\cdot\|_F$, β_i, X_i, W_i represent the status of channel i (*i.e.*, selected or not selected), new input which prunes channel i from X, and new filters which prunes channel i from W, respectively. $Y^{'}$ is specially designed for the whole model pruning, indicating all outputs from each layer. The constraint condition of this formula implies that the retained channels should be fewer than or equal to desired channels, which can be adjusted by the speed-up ratio in training. As mentioned before, the optimization of this formula requires two steps. The *channel selection* step can be calculated when W is fixed, while the *feature map reconstruction* step will be further processed when β is fixed. Note that the original implementation provided by He et al. [24] was specially designed for the GPU-only Caffe environment. We change its settings and test it on a CPU-only environment to simulate the surveillance camera settings, and acquire satisfactory results shown in Section 6.

Convolutional Acceleration. It is proved that the convolutional layer is the most time-consuming layer in deep learning computation [11]. In the original SSD model, 11 convolutional layers incurs considerable processing latency. Taking $M \times M$ and $m \times m$ as the size of inputs and kernels respectively, each kernel takes $O(M^2m^2)$ computational complexity to calculate dot products for a single forward and backward propagation. It is necessary to accelerate the convolutional calculation of the SSD model in a simple and fast manner.

It is common to use Fast Fourier transform (FFT) [25] for convolutional optimization, taking the idea that Hadamard product in the frequency domain is simpler than matrix cartesian product in the space domain. Nonetheless, we consider that its complexity $O(M^2log_2M)$ is still high considering the low latency requirement in wireless surveillance systems. For the further reduction on calculation complexity, we leverage overlap-and-add (OaA) technique [26] in our optimization, achieving the complexity of $O(M^2log_2m)$. For instance, a 128×128 picture with a 2×2 kernel has a complexity of $O(128^2 \times 4)$ for standard convolution, $O(128^2 \times 7)$ for

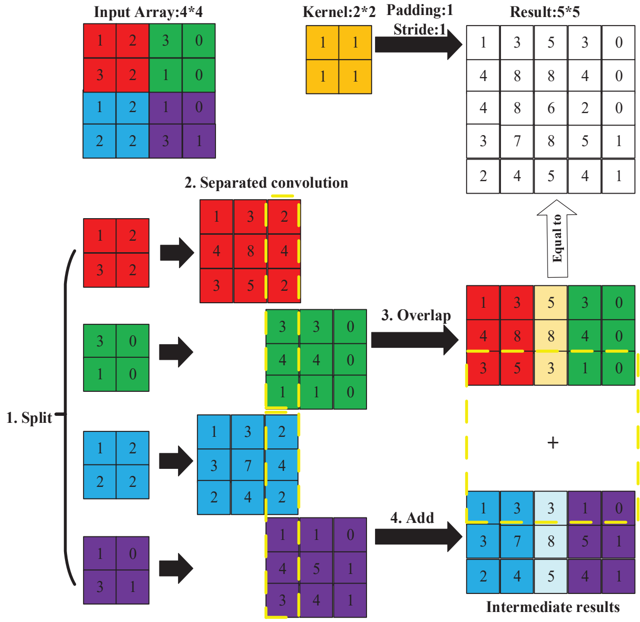


Figure 5: An example of OaA algorithm. [Composed of 4 steps: split, separated convolution, overlap, and add. The results of OaA is the same as the traditional convolution result shown in the first row.]

FFT, and $O(128^2 \times 1)$ for OaA,. That is, the OaA achieves a 1/7 less computational complexity than FFT and a 1/128 less than the standard method. As shown in our evaluation results in Section 6, this method accelerates the running of the SSD model on cameras, with less latency than other three light-weight models (*i.e.*, DeepMon, DeepEye, MobileNet) in detection.

A simple example of 2D gray image processing by OaA in *Litedge* is shown in Fig. 5. It can be easily extended to color images. As we have broken each frame into $\frac{\mathrm{M}^2}{m^2}$ blocks in frame pruning stage, we set the kernel size same as the block size $m \times m$. Take a 4×4 image as an example. We divide the image into 4 blocks with 2×2 size, which is the same with the kernel size in convolution. As shown in Fig. 5, they are represented by four different colors. The left upper image is the input array and its kernel, while the right upper image is their convolutional result calculated by the standard convolutional process. The processing flow of OaA algorithm is illustrated under the first line, including splitting, separated convoluting, overlapping and adding stages. In the splitting stage, four blocks are split from the original input array to four sub-arrays. Each sub-array will be separately convoluted by the same kernel, padding rate and stride with the standard convolution, and getting four convolutional results. These four results will be further added together pair by pair with the overlap rate of $m - 1$ (1 in this example), leading to two intermediate results. These two intermediate results will again be added by rows with the same overlap rate $m - 1$ to output the final result of OaA. This final result is the same as the standard one given in the first row. Note that the separated convolutional stage is computed in the frequency domain for large scale pictures. The separation in OaA can reduce the convolutional scale, while the overlap and add operations are simpler than dot product while keeping the consistency of final results.

The possible extra cost for OaA is block breaking. As input images have already been divided in the former frame pruning operations, only the size of the kernel is needed to be set the same as block size with negligible cost.

4.1.3 Output Class Deduction. In the SSD model, 8732 prior box results are calculated for each detected class. Taking the MS COCO dataset [27] as an example, 90 classes in this dataset require a large number of calculations. From our observations, some objects have no need to be detected even it is dynamic, including plants (*e.g.*, trees, grass, flowers) and buildings. While for other objects like doors, mailboxes, and exercise facilities, etc., we consider that their movements are involved in human activities. So as long as we detect humans in frames, these facilities will also be included in selected frames, with no need for special detection. At last, we select seven categories in MS COCO dataset to be selected, including human, dog, cat, car, bicycle, bike, and bus (especially for campus scenario), which are frequently appeared in captured videos and have potential risks in our daily life. Furthermore, these several selected categories have similar features: dog and cat are all animals with four feet and small shapes; car, bicycle, bike, and bus are all transportation with wheels. So we combine these similar categories to three kinds in total. That is, human, transportation, and animals should be detected in frames. The corresponding image sets and annotation XML file are summarized according to our 3 specified categories.

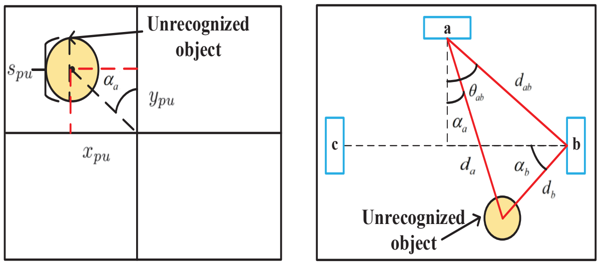
4.2 Collaborative Validation

The complicated physical environment may bring potential error in the former object detection, due to the following two typical cases:

(1) **Shielding:** Buildings, plants or facilities are natural barriers for monitoring, especially for modern communities with dense population. Because of the fixed angles of cameras, the video captured in a single camera cannot clearly show every object, suffering from such shielding.

(2) **Blind area:** The sparse surveillance distribution may not cover the whole area, leaving some places undetected. Limited by the fixed positions and monitoring ranges of cameras, it is possible that partial object captured in frames, decreasing the accuracy of object detection in frame analysis.

To guarantee the accuracy requirement in wireless surveillance systems, we design a collaborative validation scheme among cameras to compensate these undetectable cases. Each camera node has its application-specific tasks like object detection and functionalities like validation. Such validation depends on the share of their knowledge on target objects within neighboring nodes and finally gives the feedback for frame selection. For the accuracy requirement in QoS, the



(a) Angle and distance in the frame.

(b) Relative position in physical 2-D map.

Figure 6: Collaborative validation calculation.

object detection results on frames will be stored for further error compensation in validation. Concretely, the design of this collaborative validation module is described below.

When there are unrecognized objects detected, the master camera will send a validation request to its slave peer. The request message includes the physical location of requested objects and corresponding timestamps. Once the slave camera receives a request, it will project the relative locations of these objects in its view and locate them in corresponding recorded frames. Depending on the pin-hole imaging theory, we design a *linear image projection algorithm*. First, several objects whose detection confidence is lower than the threshold T ($T = 50\%$ in our experiments) have been marked as unrecognized objects u by the former AI module. As illustrated in Fig. 6, its estimated type t_u, centroid location (x_{pu}, y_{pu}), size (maximum occupied pixels lengthways) s_{pu} in the frame, and the corresponding timestamp t will be fed to this collaborative validation module as inputs. And the angle α_a is measured between the object and its axle line. We assume that the relative position, deployed height and the focal of each camera are pre-known.

According to the pin-hole projection formula [28], we calculate the distance d_a from the unrecognized object to the master camera a by:

$$\mathrm{d}_a = \frac{h_u \times f}{s_{pu} + h_a}, \tag{2}$$

where h_u is the practical height of the unrecognized object u, f is the focal of camera (same for each camera), and h_a is the deployed height of the master camera a. According to the common sense, we set the average heights for three detected classes as: 170 cm for human, 20 cm for animals, 150 cm for vehicles. Based on the estimated type t_u, the value of h_u can be set correspondingly.

For the angle calculation, we simplify the problem that the angle between the centroid of an unrecognized object and the axle line in the frame is the same as the angle between object and the normal direction of camera in physical space. In Fig. 6(a), the angle α_a can be easily calculated by the location of centroid (x_{pu}, y_{pu}) in frame f. To imply the east or west of the angle, it will be represented by the positive or negative value for differentiating. Illustrated in Fig. 6(b), we select the camera b rather than c. Based on the relative angle θ_{ab}, the relative distance d_{ab} between the camera a and b, the calculated distance d_a and the angle α_a, the distance d_b for the slave camera b can be calculated as:

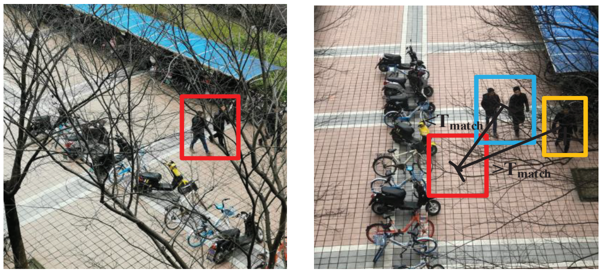
$$d_b = \sqrt{(d_a \sin(\theta_{ab} - \alpha_a))^2 + (d_{ab} - d_a \cos(\theta_{ab} - \alpha_a))^2} \tag{3}$$

And the angle α_b for slave camera b can be calculated as:

$$\alpha_b = \arccos \frac{d_{ab} - d_a \cos(\theta_{ab} - \alpha_a)}{d_b} \tag{4}$$

Inversely, such physical distances and angles can be transferred to the size s_{pb} and the angle α_b in the frame f_{bt} of camera b at time t according to (2). A $s_{pb} \times s_{pb}$ box will be drawn as the localization of unrecognized objects in the camera b.

However, as the projection from the 3-D physical world to 2-D pictures inevitably lose one-dimension parameters, this simplified algorithm will bring potential accuracy loss. Correspondingly, we design a *matching mechanism* for error compensation. The slave node will extract its OoI detection regions on the frame f_{bt} and further compared with the localized region. If there is a shortest distance between two centroids of regions which is no more than threshold T_{match}, we consider these two regions as the same region, where we will match the calculated location with this detected location. The threshold T_{match} is experimentally set as the average value in multiple measurements. Illustrated in Fig. 7, the covered human labeled in red box of the left picture is localized as the same red box in the second picture by calculation. The two OoI detection regions are labeled in blue and yellow boxes. After calculating the distances between the localized box and detection boxes, it is obvious that the red box is closer to the blue box, and the distance is shorter than the threshold T_{match}. So we will match the red box with the blue box.



(a) Unrecognized object founded in red box.

(b) Red box will be redirected to blue box, rather than yellow one.

Figure 7: Localization compensation strategy.

After the matching stage, three kinds of results will be given in the validation module, which are sent as feedbacks to the master camera:

(1) **No match:** The slave camera sends the feedback "drop" to the master camera, implying that this request is invalid. The master camera will not select this frame for transmission.
(2) **Matched, and OoI found:** Only the slave camera uploads the frame. It also sends the feedback "finish" to the master camera.

(3) **Matched, but also unrecognized:** The slave camera uploads the frame and sends the feedback "upload". The master camera also uploads the corresponding frame, leaving them to the server for further checking.

5 IMPLEMENTATION

As shown in Fig. 8, we implement a prototype for the surveillance system on two sides: the edge camera side and the server side. In the camera side, we use Raspberry Pi 3b for experiments (labeled in the blue box), embedded with ARM Cortex-A53 and no GPU. Supported by 802.11n WiFi module and external camera module v2, it performs as the surveillance camera with video capturing and wireless communication functions. To simulate cameras deployed in different angles and capturing ratios, we select the camera module on Raspberry Pi and a commodity external camera with USB connection (labeled in yellow boxes). The evaluation results on camera processing can be displayed on the right monitor connected with Raspberry Pi (labeled as edge side). In the server side (left red box), the video analysis is running on a desktop computer, with 64-bit Ubuntu 14.04 LTS version, 8 core 3.6GHz Intel Core i7 CPU and Kabylake GT2 770 GPU.

We utilize the motioneyeos API provided in [29] for the motion detection function in each camera. It can increase the capturing frame rate for cameras once any motion is detected. The frame rate in the sleep mode and the active mode are 15Hz and 32Hz, respectively. And we set the capturing resolution as the typical 640*480 dpi.

The optimized SSD model is trained locally with the filtered MS COCO dataset [27], including 28462 samples for people, 10190 samples for transportation, and 3759 samples for animals in total. The 10% of total samples will be divided as testing samples while the 90% samples combine the training set. It is running on Raspberry Pi for real-time video analysis. On the server side, the original SSD model is trained using large MS COCO dataset as a baseline in performance evaluations. According to the continuity of frames, consecutive frames in a short duration may have similar object detection results. So the SSD model processes frames sequentially by taking the next input after finishing the process of the current frame. Measured from the experiments, the sampling rate for the SSD model is 12.5Hz. For another parameter setting, according to our experimental results shown in Fig. 9(a), 8*8 size of blocks yield better detection accuracy and lower latency, therefore, we set it as the size of divided blocks.

For the collaborative validation module, the master and slave cameras will first share their location information, including the relative distances and angles. We implement this module by writing approximately 200 lines of Python codes on master and slave nodes, including neighboring selection, information sharing, projection calculation, and matching. The camera that firstly gives the feedback to the master camera will be selected as the slave camera, which is considered as the nearest node within the communication range. When the validation is triggered, the calculated angles, the physical locations of unrecognized objects, and the corresponding timestamps will be packed and shared to the slave camera in multiple times until getting "received" feedback.

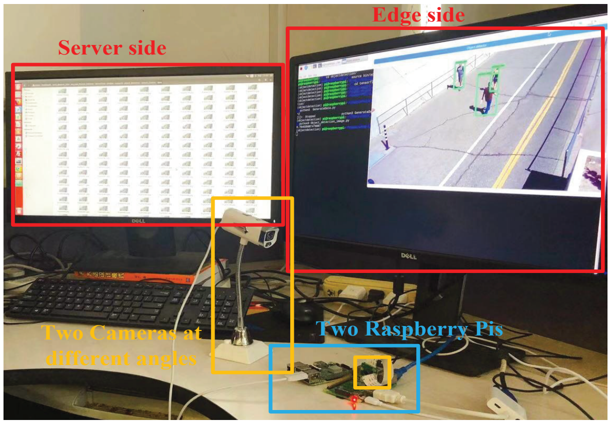
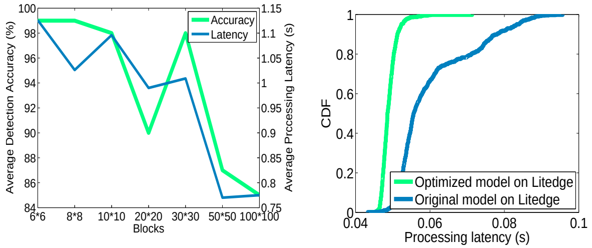


Figure 8: Prototype illustration.



(a) The corresponding accuracy and latency for different blocks.

(b) The processing latencies for two models on edge cameras.

Figure 9: Evaluations on block division and the processing latencies on edge cameras.

6 EVALUATION

In this section, the QoS performance of *Litedge* will be evaluated on real-life collected videos. Section 6.1 evaluates its latency performances, in terms of both processing and transmission latencies. Section 6.2 further evaluates its accuracy performances, including detection accuracy, collaborative validation compensation, and the final filtering accuracy. The overall QoS evaluation is introduced in Section 6.3. First, *Litedge* is compared with three state-of-the-art light-weight AI methods to evaluate its model optimization performance. Next, *Litedge* is compared with the original video transmission, the compression-only strategy, and the closely related method with both video compression and camera collaboration proposed by Zhang et al. [5]. Its overall QoS tradeoff between accuracy and latency is also measured in this comparison.

6.1 Latency Evaluation

In this experiment, we collect two video traces from two different angles monitoring the same area in a certain community. These two video traces are synchronously captured for a duration of half hour. With two different capturing ratios on sleep and active modes, we acquire around 30,000

Table 1: Different compression performance for captured videos.

Method	Original	MPEG-4	Litedge
Video size	325M	133M	96.82M
Transmission Latency	676s	296.64s	121.62s
Reduction Ratio	-	56.1%	82%

frames after cropping the beginning and the ending of this video to keep its quality. We aim to answer the following three questions in latency evaluations:

(1) How much processing latency is reduced by the optimized SSD model?
(2) How much processing latency is incurred by collaborative validation?
(3) How much transmission latency is reduced by *Litedge*?

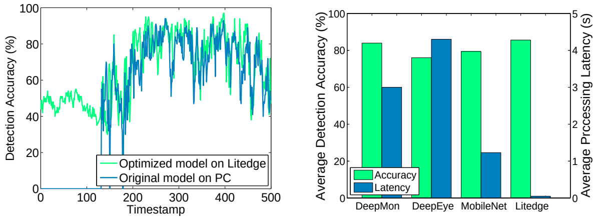
To answer the first question, we feed these 30,000 frames to both the optimized and original SSD models for comparison. To measure the processing delay incurred by collaborative validation, unrecognized frames acquired from the optimized SSD model will be input into the collaborative validation module. To evaluate the overall latency reduction level, original 480p video, 360p video compressed by MPEG-4 algorithm, and 480p video compressed by *Litedge* (frame filtering with MPEG-4 compression) will all be transmitted to the server, with the 100K/s uploading speed.

Fig. 9(b) shows the CDF of processing latency for the optimized SSD model on each frame, compared with the original model running on cameras. We observe that the average latency is 0.049s for the optimized model, while it is 0.06s for the original model, leading to around 18.3% deduction. Our optimized operations in Section 4.1 accelerates the computation of the original SSD model and also fits the model into light-weight devices with limited computational capability.

The average latency for the collaborative validation module is 0.07s, leading to 0.119s additional processing delay in total. Moreover, 27.2% frames are deducted compared with the whole-frame transmission. Table. 1 shows different transmission latency for the original video, the MPEG-4 processed video, and the *Litedge* processed video. Clearly shown in the fourth row of this table, *Litedge* achieves the highest transmission latency reduction ratio (82%) among these performances.

6.2 Accuracy Evaluation

To measure the filtering accuracy for the optimized model, we first run the optimized model on cameras, with running the original model on PC as the ground truth. As the results are quite stable after over 500 attempts, we only show accuracy results for the former 500 frames in Fig. 10.



(a) The detection accuracies for the optimized model and the original model.

(b) The accuracy and latency comparisons with other light-weight AI methods.

Figure 10: Evaluations on detection accuracies and the QoS performance of the light-weight AI module.

We further define the compensation level to evaluate the performance of collaborative validation module. We denote the redundant frames founded by light-weight AI as f_{AI}; the newly detected redundant frames in validation module as f_{val}; and the number of real redundant frames defined by original model running on PC as f_{real}. Then, the final accuracy for *Litedge* $Acc_{Litedge}$ can be represented by this compensation ratio R_{com}:

$$Acc_{Litedge} = \frac{f_{AI}}{f_{real}} + R_{com} = \frac{f_{AI} + f_{val}}{f_{real}} \tag{5}$$

As shown in Fig. 10(a), the optimized model running on our prototype has quite similar performance with the original model running on PC. Note that in the first 200 frames, there is no OoI shown in the video, leading to 0% detection accuracy for the original model. However, the less training set derives the relatively lower performance of the optimized model, which will mistakenly acquire around 40% detection accuracy. As we define the confidence for the redundant frame is over 50%, such error will not affect the final frame selection results. After eliminating the first 200 results, the redundant frame detection accuracy in the optimized model is 85.6%, with the ground truth (93%) on the original SSD model. Interestingly observed in this result, the detection accuracy on the optimized model sometimes even higher than the original model. We think it is because of the deduction of output classes, which increases the concentration on OoIs in the optimized model. The compensation ratio is further calculated as $R_{com} = 5.64\%$. It indicates 91.28% redundant frames are successfully filtered out, where only 17 redundant frames are mistakenly selected.

6.3 Overall QoS Performance Evaluation

The balance between latency and accuracy can show the overall QoS performance of *Litedge*. First, we evaluate the balance by comparing the light-weight AI module with three popular light-weight AI structures: DeepMon, DeepEye, and MobileNet. As shown in Fig. 10(b), *Litedge* achieves a better balance between detection accuracy and latency. Considering that more accuracy and less latency are better, we simply divide the accuracy with latency to represent the *balance ratio*, which is 27.98 for DeepMon, 18.1 for DeepEye, 65.55 for MobileNet, and 1746.9 for *Litedge*. It implies the feasibility of our light-weight AI design, which can fulfill the QoS requirements in wireless surveillance systems.

Moreover, we evaluate the balance by comparing the overall QoS performance of *Litedge* with the original video transmission, the compression-only strategy, and the closely related method [5]. Their comparisons are summarized in Table. 2.

Table 2: The summarization and comparisons on the overall QoS performance.

Method	Original Transmission (Ground Truth)		Compression Only (Control experiment)		Litedge (Our design)		Zhang et al. [5] (Related method)	
Performances	Accuracy	Latency	Accuracy	Latency	Accuracy	Latency	Accuracy	Latency
Results	93%	676s	85.6%	247.62s	91.28%	262.62s	90%	329.88s
Balance ratio	0.138		0.346		**0.348**		0.273	

The balance ratio in the fourth row again demonstrates the feasibility of *Litedge* on QoS enhancement in wireless surveillance systems.

7 CONCLUSION

To enhance the QoS performance of wireless surveillance systems, we presented *Litedge*, a light-weight edge computing strategy to accurately filter out redundant frames in transmission. To better fit this strategy into edge cameras limited by its computational capability, we optimize the deep learning model used for object detection with model compression and convolutional acceleration. A collaborative scheme is further designed between edge cameras to avoid possible analysis loss. Experiments show its better QoS provisioning, including low latency, resonable accuracy, and a better balance between them.

To further increase *Litedge*'s utility, we highlight several interesting future directions of our work. Operations such as eliminating the low illumination and bad weather effects can be introduced to increase the processing accuracy. Video compression algorithms can be designed to decrease its distortion rate. Moreover, we plan to extend monitoring scenarios and camera types of *Litedge* in our future work.

ACKNOWLEDGMENTS

This work was supported in part by the National Key R&D Program of China 2018YFB1004703, in part by China NSF grant 61672349, 61672353, and in part by the Joint Key Project of NSFC (U1736207), NSFC (61572324).

REFERENCES

[1] D. Tynan, Digital surveillance systems help keep k12 students, staff safe from harm, https://edtechmagazine.com/k12/article, accessed April, 2018.

[2] G. Jia, G. Han, H. Rao, L. Shu, Edge computing-based intelligent manhole cover management system for smart cities, IEEE Internet of Things Journal 5 (3) (2018) 1648–1656.

[3] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, W. Wang, A survey on mobile edge networks: Convergence of computing, caching and communications, IEEE Access 5 (2017) 6757–6779.

[4] D. Wu, C. Song, H. Luo, Y. Ye, H. Wang, Video surveillance over wireless sensor and actuator networks using active cameras, IEEE Transactions on Automatic Control 56 (10) (2011) 2467–2472.

[5] T. Zhang, A. Chowdhery, P. Bahl, K. Jamieson, S. Banerjee, The design and implementation of a wireless video surveillance system, in: ACM/IEEE Mobicom, Paris, France, 2015, pp. 426–438.

[6] J. L. Mitchell, W. B. Pennebaker, C. E. Fogg, D. J. Legall, Mpeg video compression standard 99 (5) (1996) 1666–1675.

[7] Y. Zhang, H. Wang, D. Zhao, Up-sampling dependent frame rate reduction for low bit-rate video coding, in: DCC, Snowbird, Utah, USA, 2011, p. 489.

[8] R. Khoshabeh, T. Gandhi, M. M. Trivedi, Multi-camera based traffic flow characterization and classification, in: ITSC, Seattle, WA, USA, 2007, pp. 259–264.

[9] C. Micheloni, E. Salvador, F. Bigaran, G. L. Foresti, An integrated surveillance system for outdoor security, in: IEEE AVSS, Como, Italy, 2005, pp. 480–485.

[10] P. Natarajan, P. K. Atrey, M. S. Kankanhalli, Multi-camera coordination and control in surveillance systems: A survey, TOMCCAP 11 (4) (2015) 57:1–57:30.

[11] H. N. Loc, Y. Lee, R. K. Balan, Deepmon: Mobile gpu-based deep learning framework for continuous vision applications, in: MobiSys, Niagara Falls, NY, USA, 2017, pp. 82–95.

[12] A. Mathur, N. D. Lane, S. Bhattacharya, A. Boran, C. Forlivesi, F. Kawsar, Deepeye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware, in: MobiSys, Niagara Falls, NY, USA, 2017, pp. 68–81.

[13] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications, CoRR abs/1704.04861.

[14] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, K. Chan, When edge meets learning: Adaptive control for resource-constrained distributed machine learning, in: IEEE INFOCOM, Honolulu, HI, USA, 2018, pp. 63–71.

[15] R. T. Collins, A. J. Lipton, H. Fujiyoshi, T. Kanade, Algorithms for cooperative multisensor surveillance, Proceedings of the IEEE 89 (10) (2001) 1456–1477.

[16] T. Rault, A. Bouabdallah, Y. Challal, Energy efficiency in wireless sensor networks: A top-down survey, Computer Networks 67 (2014) 104–122.

[17] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, A. C. Berg, SSD: single shot multibox detector, in: ECCV, Amsterdam, The Netherlands, 2016, pp. 21–37.

[18] Y. He, X. Zhang, J. Sun, Channel pruning for accelerating very deep neural networks, in: IEEE ICCV, Venice, 2017, pp. 1398–1406.

[19] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, A. Krishnamurthy, MCDNN: an approximation-based execution framework for deep stream processing under resource constraints, in: MobiSys, Singapore, 2016, pp. 123–136.

[20] J. Morel, G. Yu, ASIFT: A new framework for fully affine invariant image comparison, SIAM J. Imaging Sciences 2 (2) (2009) 438–469.

[21] Y. Song, S. Noh, J. Yu, C. Park, B. Lee, Background subtraction based on gaussian mixture models using color and depth information, in: ICCAIS, Gwangju, South Korea, 2014, pp. 132–135.

[22] S. Han, H. Mao, W. J. Dally, Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding, CoRR abs/1510.00149.

[23] X. Liu, J. Pool, S. Han, W. J. Dally, Efficient sparse-winograd convolutional neural networks, CoRR abs/1802.06367.

[24] X. Z. Yihui He, J. Sun, Channel pruning github code, https://github.com/yihui-he/channel-pruning.

[25] M. Mathieu, M. Henaff, Y. LeCun, Fast training of convolutional networks through ffts, CoRR abs/1312.5851.

[26] T. Highlander, A. Rodriguez, Very efficient training of convolutional neural networks using fast fourier transform and overlap-and-add, in: BMVC, Swansea, UK, 2015, pp. 160.1–160.9.

[27] T. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, C. L. Zitnick, Microsoft COCO: common objects in context, in: ECCV, Zurich, Switzerland, 2014, pp. 740–755.

[28] wikipedia, Pin-hole projection formula, https://en.wikipedia.org/wiki/Pinhole_camera_model.

[29] ccrisan, Motioneyeos, https://github.com/ccrisan/motioneyeos.