

# **Towards Scalable Video Analytics using Compressed-domain Features at the Edge**



**Nguyen Van Dien**

Supervisor: Prof. Jaehyuk Choi

Prof. Eun-Seok Ryu

Department of Computer Science

Gachon University

A dissertation submitted to the department of Computer Science and the committee on graduate studies of Gachon University in partial fulfillment of the requirements for the degree of Doctor of Philosophy

November 2020



## **Acknowledgements**

First, I would like to express my gratitude and appreciation to my major advisor, Dr. Jaehyuk Choi's mentorship and support, I was able to get access to numerous opportunities, develop myself to be a better researcher, and I hope our cooperation will continue in the future. Thanks to Dr. Eun-Seok Ryu for letting me join the Graduate School to study and research at Gachon University. I also want to thank my committee members: Drs. ???, ???, ??? and Dr. ??? for supporting me complete the PhD program as well as all the valuable inputs to my dissertation. Additionally, the discussions with them helped me broaden my perspective on my research.

I am grateful to the support from my parents in VietNam. I thank all my friends at Gachon for making my life here a happy and precious experience: Le Thanh Tuan, Nguyen Ngoc Quang ... and many others. Finally, a very special thanks to my wife Tran Thi Thu, a woman who is excellent in all aspects, beautiful, smart, and generous. I am very grateful for her encouragement, her patience, and all her love and tenderness.



## **Abstract**

Intelligent video analytics systems have come to play an essential role in many fields, including public safety, transportation safety, and many other industry areas, as automated tools for data extracting and analyzing huge datasets such as multiple live video streams transmitted from a large number of cameras. A key characteristic of such systems is that it is critical to perform real-time analytics so as to provide timely actionable alerts on various tasks, activities and conditions. Due to the computation-intensive and bandwidth-intensive nature of these operations, however, video analytics servers may not fulfill the requirements when serving for a large number of cameras simultaneously. To handle these challenges, this dissertation present an edge-computing based system that minimizes transfer of video data from the surveillance camera feeds on a cloud video analytics server. Based on a novel approach of utilizing the information from the encoded bitstream, the edge can achieve low processing complexity of object tracking in surveillance videos and filters non-motion frames from the list of data which will be forwarded to the cloud server. To demonstrate the effectiveness of the approach, we have implemented a video surveillance prototype consisting of edge devices with low computational capacity and a GPU-enabled server. The evaluation results show that our method can efficiently catch the characteristics of the frame and is compatible with the edge-to-cloud platform in terms of accuracy and delay sensitivity. The average processing time of this method is approximately 39 ms/frame with high definition resolution video, which outperforms most of the state of art method. In addition to scenario implementation of the proposed system, the method helps the cloud server reduce 49% load

of GPU, 49% load of CPU, and 55% of network traffic while maintaining the accuracy of real-time 18 video analytics event detection.

# **Table of contents**

<b>List of figures</b>	<b>ix</b>
<b>List of tables</b>	<b>xi</b>
<b>Nomenclature</b>	<b>xiii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Background</b>	<b>5</b>
2.1 Surveillance Video Analytics Architectures and Challenges . . . . .	5
2.1.1 Edge Camera Based Implementation . . . . .	6
2.1.2 Video Management Server Based Implementation . . . . .	6
2.1.3 Video Codec and Video Coding Motion Vectors . . . . .	8
2.2 Compressed-Domain Based Moving Object Detection . . . . .	14
2.3 Pixels-Domain Based Moving Object Detection . . . . .	16
2.3.1 Hybrid model of background subtraction and object classification based moving object detection . . . . .	16
2.3.2 Deep Learning based Moving Object Detection . . . . .	19
<b>3 Methodology</b>	<b>23</b>
3.1 The Edge-to-cloud System Model for Surveillance Camera based Applications	24

3.2 The Light-weight Runtime Moving Object Detection in Video Compressed Domain . . . . .	25
3.2.1 Median Filter and Moving Object Detection . . . . .	26
3.2.2 The IoU based Moving Objects Tracking . . . . .	29
3.3 Performance Evaluated Model . . . . .	31
<b>4 Implementation And Performance Evaluation</b>	<b>33</b>
4.0.1 Scenario Setup . . . . .	34
4.1 The Light-weight Runtime Moving Object Detection in Video Compressed Domain . . . . .	36
4.2 Performance Evaluation Results . . . . .	39
4.2.1 Computing Resources Consumption . . . . .	39
<b>References</b>	<b>43</b>
<b>Appendix A How to install L<sup>A</sup>T<sub>E</sub>X</b>	<b>49</b>
<b>Appendix B Installing the CUED class file</b>	<b>55</b>

# List of figures

1.1	Overall of cloud based video analytics server . . . . .	2
2.1	Video Analytics Implementation, (a) Video management server based implementation, (b) Edge camera based implementation . . . . .	6
2.2	Video encoder block diagram. . . . .	9
2.3	Motion Vector in Video Codec. . . . .	11
2.4	Macroblock in Video Encoder. . . . .	11
2.5	Macroblock (4:2:0). . . . .	12
2.6	Reference motion vectors in video coding. . . . .	13
2.7	Example of motion vectors extraction.(a) Test video sequence from our recorded video, (b) Test video sequence from VIRAT. . . . .	15
2.8	Smoke detection process: (a) input image, (b) foreground subtraction, (c) blob detection, (d) smoke classification . . . . .	16
2.9	Flow chart of our video- based smoke detection algorithm. . . . .	17
2.10	CNN based Object Detection Framework Structure.(a) RCNN, (b) Faster-RCNN, (c)Fast-RCNN. . . . .	18
2.11	YOLO Strucer. . . . .	19
2.12	YOLOv3 Network Model. . . . .	21
3.1	Overview of the proposed edge-to-cloud system model. . . . .	23

3.2	Workflow of the proposed detection-based tracking approach in compressed domain. . . . .	25
3.3	Video coding motion vector extraction from consecutive frames with two video test sequences. . . . .	26
3.4	Motion vector of a moving object in different video test sequences: (a, b) our record video, (c, d) Test video in VIRAT . . . . .	27
3.5	Moving objects detection by the proposed method with two different video test sequences:(a,e) MV extraction; (b,f) Apply median filter; (c,g) Clustering MVs; (d,f) Blob detection. . . . .	29
3.6	Object matching method . . . . .	30
4.1	Overview of System Design . . . . .	34
4.2	Testbed: (a) Scenario Setup, (b) The implemented edge device. . . . .	35
4.3	Moving objects detection by the proposed method in different scenarios: (a, b, c, d) human walking; (e, f, g, h) human running; (i, j, k, l) test with a far distance of camera. . . . .	37
4.4	Moving objects tracking by the proposed method with different $\alpha$ thresholds: $\alpha = 0.25$ in (a, b, c), $\alpha = 0.4$ in (d, e, f) and $\alpha = 0.5$ in (d, e, f). . . . .	38
4.5	GPU Monitoring: (a) With the conventional method, (b) With the proposed method. . . . .	40
4.6	CPU Monitoring with both the conventional method and the proposed method. . . . .	40
4.7	Network download throughput monitoring with both the conventional method and the proposed method. . . . .	41

# List of tables

4.1	Hardware Specifications. . . . .	34
4.2	Video Test Sequence. . . . .	34
4.3	Testbed: Tested Video Grouth-truth motion time. . . . .	35
4.4	Average IoU of the moving object detection in compressed-domain in different scenarios. . . . .	37
4.5	Average per-frame running times for preprocessing and tracking procedures. Values are expressed in miliseconds (ms) and frame per second(FPS). . . . .	37
4.6	Average computing resources of both the conventional method and the proposed method. . . . .	39



# Nomenclature

## **Acronyms / Abbreviations**

AVC Advanced Video Coding

CCTV Closed Circuit Television

CNN Convolutional Neural Networks

CPU Central Processing Unit

CUDA Compute Unified Device Architecture

FPS Frames Per Second

GPU Graphics Processing Unit

HEVC High Efficiency Video Coding

IoU Intersection Over Union

MCP Motion-Compensated Prediction

MV Motion Vector

R-CNN Regions with CNN

RAM Random Access Memory

RTSP Real-Time Streaming Protocol

SSD Single Shot Multi-box Detector

VA Video Analytics

VMS Video Management Software

YOLO You Only Look Once

# Chapter 1

## Introduction

Today the world is witnessing an exponential increase in camera deployment [10] with cities and organizations steadily increasing the size and reach of their deployments. For example, cities now deploy tens thousands of cameras, each continually collecting and streaming rich video data [1], [2], [3]. According to IHS Markit's annual report [48], as of 2018, China has one camera for each 4.1 people in the country and the United State has a people-to-camera ratio of 4.6-to-1. The massive deployments of cameras are brought on mainly by the growth of the video surveillance industry due to increasing concerns about public safety and security. With such prevalent trend, intelligent video analytics systems have been playing an essential role, performing important task in various fields including surveillance, transportation, manufacturing, etc. These systems analyze video feeds to guide long-running tasks such as traffic monitoring, customer tracking, and surveillance. Key to the success of such applications has been recent advances in computer vision, particularly neural network based techniques for highly accurate object detection and recognition [15], [33], [40].

In a typical real-time video analytics pipeline as Figure 1.1 illustrates a traditional cloud-based video stream analytics system. A large number of camera transfer video data to a central cloud (datacenter) where video analytics is performed. However, this traditional approach makes it difficult to perform real-time analytics on live video streams from many

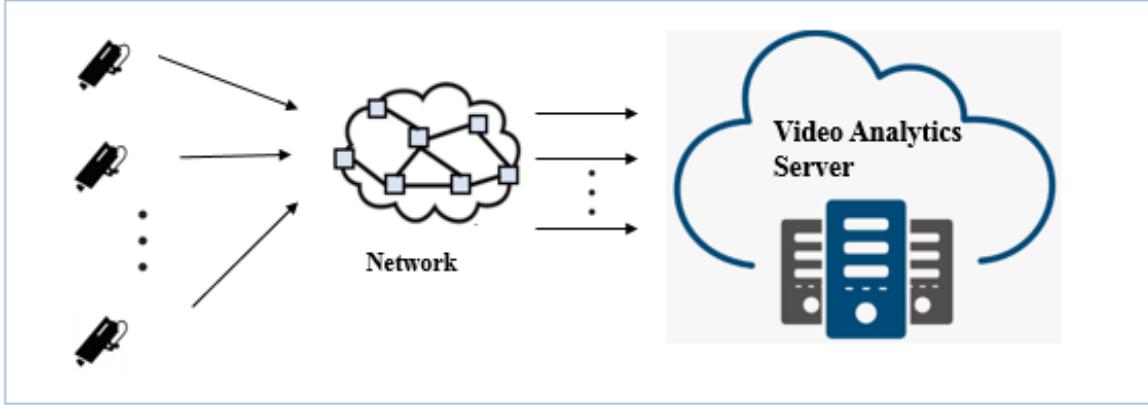


Fig. 1.1 Overall of cloud based video analytics server

cameras because the video analytics involves several computation-intensive tasks such as object detection, object tracking, object recognition and so on. Besides, streaming video from multiple camera to the cloud consumes a lot of network bandwidth over limited-bandwidth networks, which leads to high latency, causing significant challenge for real-time video analytics.

Significant work has been expended to improve the efficiency of video analytics pipelines [16], [18], [26], [29]. Across these systems, a prevailing (and natural) strategy is to improve efficiency by filtering out frames that do not contain relevant information for the query. Conceptually, filtering out a frame would affect a query result. In this study, to address addresses the question of how to overcome this network bottleneck and offload large volumes of data from a distributed camera deployment in real-time to a datacenter for further processing, we aim to develop a solution by harnessing edge computing.

Video analytics at the edge has multiple benefits such as decreasing the response time, saving network bandwidth, and minimizing the peak workload to the cloud. However, edge devices are typically much less powerful than the cloud, with limited computing resources such as a few GPUs (graphics processing units) and CPUs (central processing units), as well as smaller RAM (Random Access Memory) capacities [54]. In the field of public safety, the ability to simultaneously process multiple feeds and provide real-time video analytics is critical.

Therefore, this study seeks to answers how edge devices and the cloud can cooperate in an efficient manner to achieve real-time and scalable video analytics.

One of the features frequently observed in surveillance camera images is that captured images remain unchanged for a long time and video content is often quite redundant. Therefore, it is undesirable to conduct analytics on redundant video frames from cameras, leading to a waste of computing resources and power consumption in the cloud. Motivated by this, we propose a pre-process module that acts as a video filter function at the edge device to eliminate the redundant static images before feeding videos to the cloud for further analysis. The proposed method runs on an edge device and recognizes the motions (i.e., moving objects) in the consecutive video frames. Based on the motion recognition result, our module decides whether to pass the video frame to the cloud or filter it out. As a result, the proposed motion-based filtering module can reduce not only computational load of the cloud nodes but also network traffic to the cloud.

Motion recognition schemes can be classified into two categories: (i) video pixel-domain based and (ii) video compressed-domain based approaches. In pixel-domain based approaches [44], [34], [25], [59] video is completely decoded before using background modelling or vision based deep learning framework to detect moving objects at the pixel level. The performance of video pixel-domain processing in large systems may be challenged by the load of decoding multiple streams and the image pixel-based calculation. Thus, pixel-based approach requires higher computational complexity and can make it difficult to fulfill the real-time requirement of edge devices. Compressed-domain approaches [22],[60],[20],[9] rely on video coding artifacts of compressed bitstreams such as motion vector (MV), macroblock partitions, and quantization coefficients for recognizing motion. Compared to pixel-domain based algorithms, compressed-domain methods generally require less computational resources because analyzing input information is already possible in the bitstream.

In this study, we proposed a compressed-domain based moving objects detection that applies

a data clustering and an intersection of union (IoU) rate-based object tracking technique of computer vision. Using only MVs, which are provided by video encoders in a compressed bitstream, our approach is able to efficiently detect moving objects. Furthermore, an experimental evaluation of our method is provided to compare it with the state-of-the-art compressed-domain tracking methods in term of processing time, and demonstrate its functionalities to evaluate the efficiency of the proposed edge device with a cloud video analytics server in a real-world scenario. In summary, this study makes the following contributions.

- Introduction of a compressed-domain moving objects detection method that can be applied for numerous surveillance applications.
- Design an edge-to-cloud computing system for surveillance video analytics that applies the proposed method at edge devices to minimize the transfer of video data from surveillance camera feeds to the cloud.
- Implementation and evaluation of the proposed method with an edge-cloud system.  
The implementation is lightweight and easy to deploy at any edge devices.

# **Chapter 2**

## **Background**

Object detection in images and videos has received a lot of attention in the computer vision and video analytics communities in recent years. There are many different approaches to video moving object detection, both utilizing compressed and uncompressed data. In this chapter, we briefly introduce the state-of-the-art approaches for moving object detection on both compressed domain and pixel domain. In addition, the background information necessary to understanding VA architecture is described briefly.

### **2.1 Surveillance Video Analytics Architectures and Challenges**

Video surveillance systems are typically built using the following main components: surveillance cameras, VMS, storage and VA modules (optional). VA can be implemented in two main configurations, as discussed below.

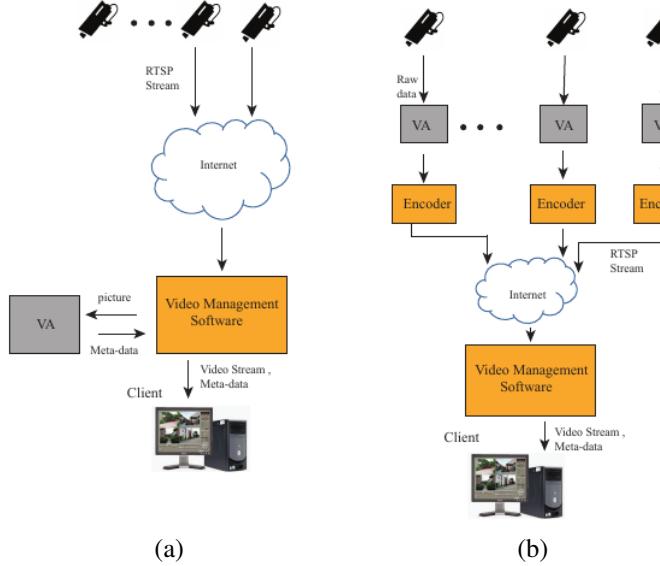


Fig. 2.1 Video Analytics Implementation, (a) Video management server based implementation, (b) Edge camera based implementation

### 2.1.1 Edge Camera Based Implementation

In this approach, the VA is implemented through a camera device or video encoder [17] such as that shown in Figure 2.1b, which must have sufficient processing power to run the VA functionality. Hence it is an expensive and challenging approach to wide-area video processing. On the surface, this approach seems otherwise ideal, however it does not perform satisfactorily in many cases as it imposes limitations on the overall surveillance system design and performance. Most camera device still lack sufficient processing power for highend VA requirements, and therefore, this approach has many drawbacks in real-world deployment.

### **2.1.2 Video Management Server Based Implementation**

In this approach, as shown in Figure 2.1a, VA is implemented through a dedicated server that pulls the video from the camera devices or from VMS, analyzes it, and issues analysis results. However, it has some challenges, which are listed below:

- The VA server requires the video to be transmitted and therefore causes an increase in network traffic. In detail, running video analytics by streaming all video to the cloud conflicts with the bandwidth constraints of some deployments, which preclude uploading all camera data. Each camera's uplink bandwidth is limited, both by the physical constraints of modern wide-area network infrastructure and the monetary cost of operating a widespread camera deployment. Specifically, we consider large-scale deployments where each camera receives a bandwidth allocation of a few hundred kilobits per second, or less. For comparison, a low-quality H.264-encoded 1080p (1920×1080 pixels) stream is approximately two Mb/s, an order of magnitude greater than our available uplink bandwidth. Yet, such low-quality data is often insufficient to perform accurate analysis: Modern 4K (3840×2160 pixels) cameras produce up to 30-40 Mb/s, two orders of magnitude beyond the uplink bandwidth, and this gap will only expand as 8K (7680 × 4320 pixels) cameras become more common. As a concrete example, we have an off-campus deployments where cameras are mounted next to traffic lights at an intersection. The local internet service provider charges \$400 per month for a single 35 Mb/s uplink, creating a strong economic incentive for us to share that bandwidth between as many cameras as possible (currently, eight 4K cameras share each uplink). This bandwidth gap, exacerbated by the requirement for high-quality data, necessitates an edge-based decision about which frames to send to the datacenter. An edge-based filter answers this challenge with semantic filtering that uploads only frames that are relevant to applications.
- The video quality being analyzed by the VA server is usually degraded because of compression and transmission effects, and therefore, the VA performance may be compromised.
- The VA server is limited by its processing power, which makes it infeasible for large scale surveillance installations which deploy hundreds (and increasingly thousands) of

cameras requiring a variety of VA functionalities. For example, a  $1920 \times 1080$  pixel stream at 30 frames per second is  $\approx 1.5$  Gb/s when decompressed. Accomplishing VA at scale requires abundant compute, memory, and storage resources, so existing systems often perform this processing in the cloud, using GPUs.

However, this approach is independent of video cameras, and is therefore applicable to most types of surveillance systems, and recent technological developments can reduce the effect of the above drawbacks. For example, the release of high definition video surveillance footage with resolutions up to 1080p will decrease the impact of the image quality degradation during codec processing and by releasing the new video coding standard as well as the transcoder [56], high efficiency video coding (HEVC)[55] which has achieved approximately twice the standard compression[45] will decrease traffic in the network and at the central server, where powerful devices will have sufficient processing power to handle hundreds of camera. Moreover, some edge-based filtering approaches that is designed to overcome these above issues [16][41][18]. These approaches use edge-compute resources collocates with the cameras to identify the video sequences that are most relevant to datacenter applications (“filtering”) and offloads only that data for further analysis (“forwarding”). In this way, it supports near-real-time processing running in datacenters while limiting the use of low-bandwidth wide-area network links. In this study, we present an edge-based filter, a system that offers the benefits of both edge computing and datacenter-centric approaches to wide-area video processing.

### 2.1.3 Video Codec and Video Coding Motion Vectors

#### Video Codec

The traditional video coding standards, such as MPEG-1, MPEG-2, H.264 and H.265, are based on block-based motion compensation, transform, quantisation and entropy coding. A video encoder as shown in Figure 2.2 consists of three main functional units: a temporal

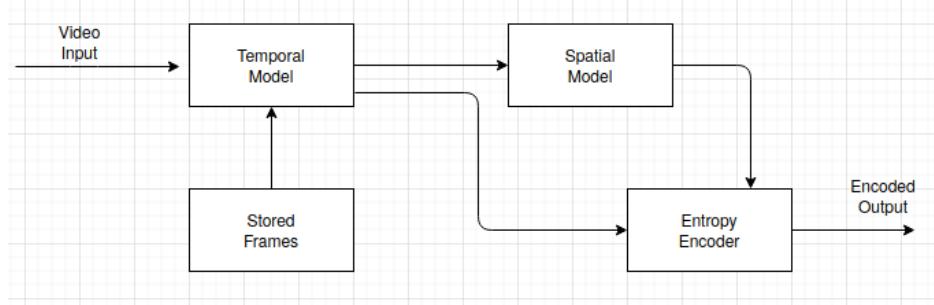


Fig. 2.2 Video encoder block diagram.

model, a spatial model and an entropy encoder. The input to the temporal model in an uncompressed video sequence. The temporal model attempts to reduce temporal redundancy by exploiting the similarities between neighbouring video frames, usually by constructing a prediction of the current video frame. In MPEG-4 Visual and H.264, the prediction is formed from one or more previous or future frames and is improved by compensating for differences between the frames (motion compressed prediction). The output of the temporal model is a residual frame (created by subtracting the prediction from the actual current frame) and a set of model parameters, typically a set of MVs describing how the motion was compressed. The residual frame forms the input to the spatial model which makes use of similarities between neighbouring samples in the residual frame to reduce spatial redundancy. In MPEG-4 Visual and H.264 this is achieved by applying a transform to the residual samples and quantizing the results. The transform converts the samples into another domain in which they are represented by transform coefficients. The coefficients are quantised to remove insignificant values, leaving a small number of significant coefficients that provide a more compact representation of the residual frame. The output of the spatial model is a set of quantised transform coefficients.

The parameters of the temporal model (typically MVs) and the spatial model (coefficients) are compressed by the entropy encoder. This removes statistical redundancy in the data (for example, representing commonly-occurring vectors and coefficients by short binary codes) and produces a compressed bitstream or file that may be transmitted and/or stored. A

compressed sequence consists of coded motion vector parameters, coded residual coefficients and header information.

The video decoder reconstructs a video frame from the compressed bitstream. The coefficients and motion vectors are decoded by an entropy decoder after which the spatial model is decoded to reconstruct a version of the residual frame. The decoder uses the motion vector parameters, together with one or more previously decoded frames, to create a prediction of the current frame and the frame itself is reconstructed by adding the residual frame to this prediction.

### **Video Coding Motion Vectors and Blocked-based Motion Estimation**

Changes between video frames may be caused by object motion (rigid object motion, for example a moving car, and deformable object motion, for example a moving arm), camera motion (panning, tilt, zoom, rotation) and lighting changes. With the exception of uncovered regions and lighting changes, these differences correspond to pixel movements between frames. It is possible to estimate the trajectory of each pixel between successive video frames, producing a field of pixel trajectories known as optical flows. In simple terms, optical flow gives the measure of movement of a pixel or a block in two consecutive frames. This measure of movement is given in the form of a vector where the magnitude of the vector signifies the amount of motion and the angle of the vector specifies the direction of the motion. This vector is called motion vector. Figure 2.3 shows the optical flow (or motion vector) field of an image. Concentrate on the objects in the image above and the motion vectors drawn on them. Analyzing the motion vectors, we can see that a bicycle is moving towards the left while two pedestrians are moving towards the right. The speed of their motion is categorized by the magnitude of the motion vectors and their direction by the angle of the motion vectors. And it would be necessary to send the optical flow vector for every pixel to the video decoder. In video coding standard, a practical and widely-used method of motion compensation for



Fig. 2.3 Motion Vector in Video Codec.

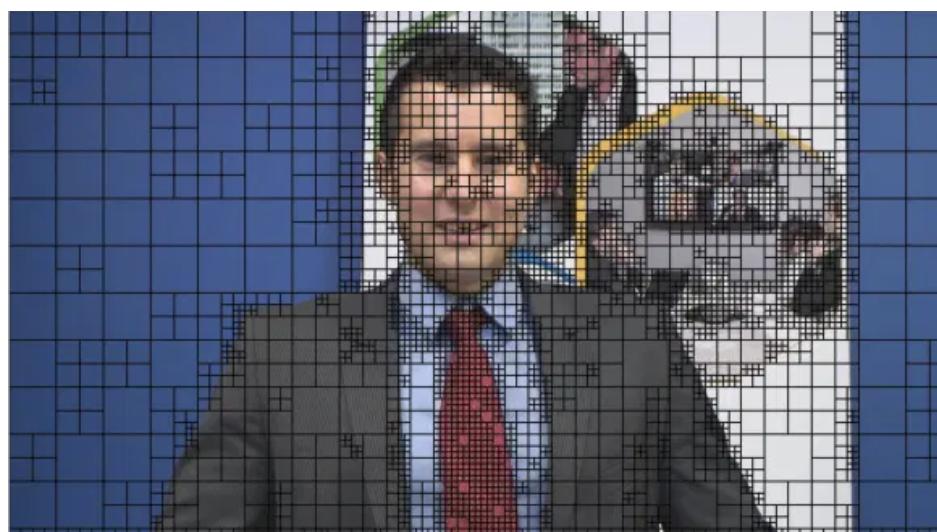


Fig. 2.4 Macroblock in Video Encoder.

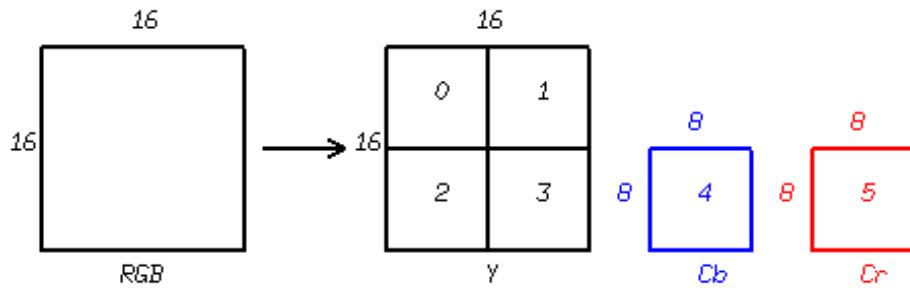


Fig. 2.5 Macroblock (4:2:0).

movement of rectangular sections or 'block' of the current frame. The current frame is sub-divided in  $M \times N$  blocks called macroblocks as shown in Figure 2.4. The following procedure is carried out for each block  $M \times N$  samples in the current frame:

- Search an area in the reference frame (past or future frame) to find a 'matching'  $M \times N$ -sample region. This process of finding the best match is known as motion estimation.
- The chosen candidate region becomes the predictor for the current  $M \times N$  block and is subtracted from the current block to form a residual  $M \times N$  block (motion compensation).
- The residual block is encoded and transmitted and the offset between the current block and the position of the candidate region (motion vector) is also transmitted.

In H264/AVC standard, the macroblock, corresponding to a  $16 \times 16$  pixel region of a frame is the basic unit for motion compensated prediction. In the RGB colour space, a colour image sample is represented by three numbers indicating the relative proportions of Red, Green and Blue. Computer display works naturally well in the RGB colour space. However, human eyes are more sensitive to luminance ( brightness ) than to colour. One can down sample an image by transforming the RGB to YCbCr representation and throw away some color components without causing much visual distortion. For example, in the 4:2:0 sampling format, Cb and Cr each has half the horizontal and vertical resolution of Y. Figure 2.5 shows a  $16 \times 16$  pixel macroblock represented in YCbCr 4:2:0 format. A macroblock of  $16 \times 16$  pixels

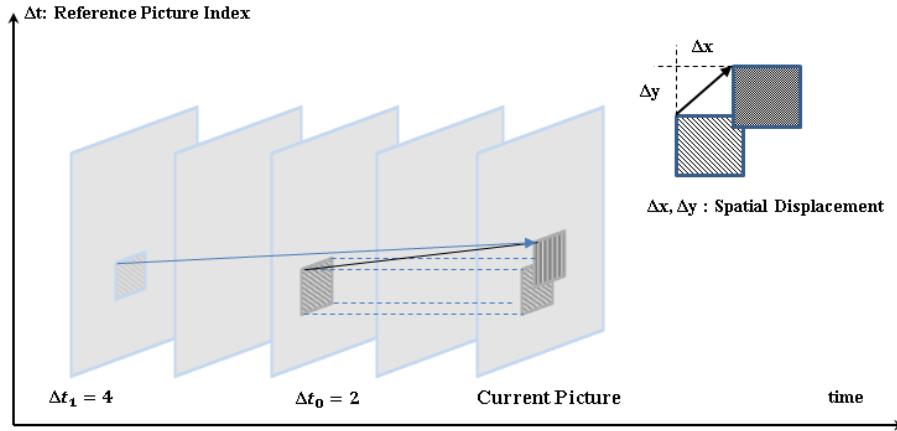


Fig. 2.6 Reference motion vectors in video coding.

can be divided into 8x8 sample blocks. In the RGB representation, a macroblock has a total of  $3 \times 4 = 12$  8x8 sample blocks ( four for each of the colors Red, Green and Blue ). In the YCbCr 4:2:0 format there are only six sample blocks, four for the Y component and 1 for each of Cb and Cr component.

In motion compensation, MV of a current block are correlated with the MVs of neighboring blocks in the current frame or in the earlier coded frames [35], [30]. While intra-picture prediction exploits correlations between spatially neighboring samples, inter-picture prediction makes use of temporal correlation between frames to derive a motion-compensated prediction (MCP) for a block of frame samples [14]. For intra-prediction, we assume that neighboring blocks possibly correspond to the same moving object with similar motion and the motion of the object is not likely to abruptly change over time. Consequently, using MV in neighboring blocks as predictor reduces the size of the signaled motion vector difference. For this block-based MCP, a video image is divided into rectangular blocks. Figure 2.6 shows the general concept of MCP based on a translational motion model. Using a translational motion model, the position of the block in a previously decoded picture is indicated by a motion vector ( $\Delta x, \Delta y$ ), where  $\Delta x$  specifies the horizontal and  $\Delta y$  the vertical displacement relative to the position of the current block. Note that the motion vectors  $\Delta x, \Delta y$  could be of

fractional sample accuracy to more accurately capture the movement of the underlying object. These MVs are coded by entropy coding and placed into a compressed bitstream for delivery to the video decoder, which uses MVs, along with one or more previously decoded pictures, to reconstruct the current image. On the other hand, H.264/AVC(Advanced Video Coding) in view of higher resolution, low bandwidth utilization and storage requirement becomes the popular video compression standard for surveillance video. In H.264/AVC, each video sequence is divided into groups of pictures (GOP), comprising at least one intra code I frame, uni-directionally predicted P frames and bi-directionally predicted B frames. Typically, the first frame in a GOP is intra coded I frame and follows by P, B frames. I frame uses raw data from camera, while other frames in a GOP utilize the predictive coding involved motion vectors displacement.

## 2.2 Compressed-Domain Based Moving Object Detection

As mentioned in last section, MVs are obtained for each motion block between the current and reference frames. By minimising the prediction residual, the MVs represent the temporal displacement between the two block in the process of motion compression. Therefore, MV information indeed follow the real motion of objects and can be used for tracking. Therefore, MV information indeed follow the real motion of objects and can be used for tracking.

In recent studies, compressed-domain based video analytics methods [12],[31] that rely on video coding artifact of compressed bitstreams, such as MVs, macroblock partition, and quantization coefficients, have been proposed. In [12], the authors applied a probabilistic technique of computer vision for image separation, known as Graph Cut [13], modeling with MVs rather than pixels and adapted to the additional temporal dimension of video signals. Using MVs and a spatio-temporal Markov random field (ST-MRF) model that naturally integrates the spatial and temporal aspects of the object's motion for tracking. In general,



Fig. 2.7 Example of motion vectors extraction.(a) Test video sequence from our recorded video, (b) Test video sequence from VIRAT.

these approaches do not rely on pixels and studies by only using the codec's MVs and block coding modes extracted bitstream through inexpensive partial decoding. In this manner, computing and storage requirements have been significantly reduced compared to "pixel-domain".

The primary limitation of this approach is that it may lead to a noisy MV field that does not necessarily correspond with actual object movement of the object in the scene, as shown in Figure 2.7. The noisy MVs fail to provide useful information such as those attributed to illumination changes and background movement. The amount of noise MVs is relatively reduced compared to the correctly estimated MVs because noisy vectors are continuous and similar MVs from real moving objects. Another challenge of this approach is the lack of information about the object's appearance such as color, edges and texture, because these features would require complete decoding of the compressed bitstream. In this study, our aim is to work in the "compressed domain" and uses only the MVs from the compressed bitstream to detect and track moving objects in video frames.

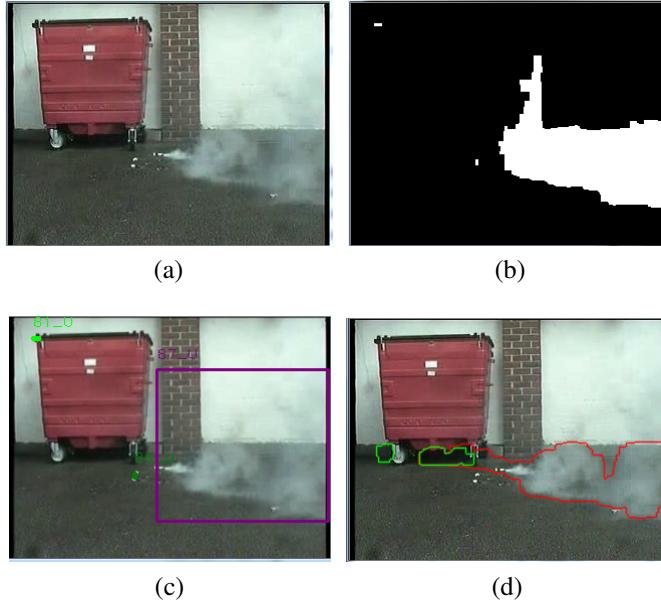


Fig. 2.8 Smoke detection process: (a) input image, (b) foreground subtraction, (c) blob detection, (d) smoke classification

## 2.3 Pixels-Domain Based Moving Object Detection

Due to more reliable features that can be extracted from the pixel data, the majority of moving object detections require full decoding of the video stream. In pixels domain, a video analytics server analyzes the red-green-blue image to determine the appearance objects and spatial events. Ubiquitous real-time video analytics remains an open and exciting research problem, and it is fueled by the recent advantages of hardware and deep learning [61],[19],[11],[58],[47],[46]. There are two primary methods that are considered for moving object detection in pixels domain.

### 2.3.1 Hybrid model of background subtraction and object classification based moving object detection

The background subtraction method [38][53] subtracts the current frame and background image to eliminate an image's background, and then detects moving targets based on gray value

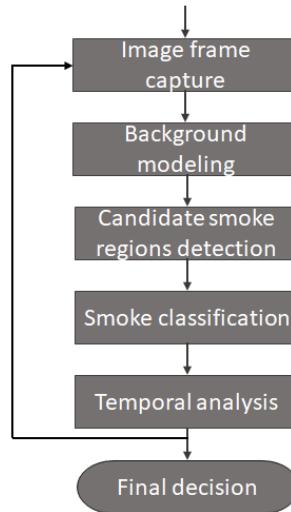


Fig. 2.9 Flow chart of our video- based smoke detection algorithm.

differences. This method is the most commonly used object detection technique. There are a number of methods that have been proposed for building the background model, e.g., [43] proposed a novel real-time motion detection algorithm that integrates the temporal differencing method, double background filtering method, optical flow method, and morphological processing methods to obtain excellent performance. Moreover, the authors of [53] discussed modeling each pixel as a grouping of Gaussians using an on-line approximation to renew the model. The Gaussian distributions of the adaptive mixture model were then assessed to determine which model can be presumed to be obtained from a background process. This method's advantage is its simple implementation, low computational resource requirements, robustness in the presence of environmental noise, and dynamic background. However, it has limitations with shadows, background changes, as well as object localization and classification. Furthermore, object detection based on the background model cannot detect the individual objects in a group or a block object. In general, the aim of background subtraction is to separate foreground images from background ones in the form of blobs, followed by an object classification process. The detected blobs then help classify each blob into subclasses, as shown in Figure 2.8, which shows the complete process of this approach is to perform smoke

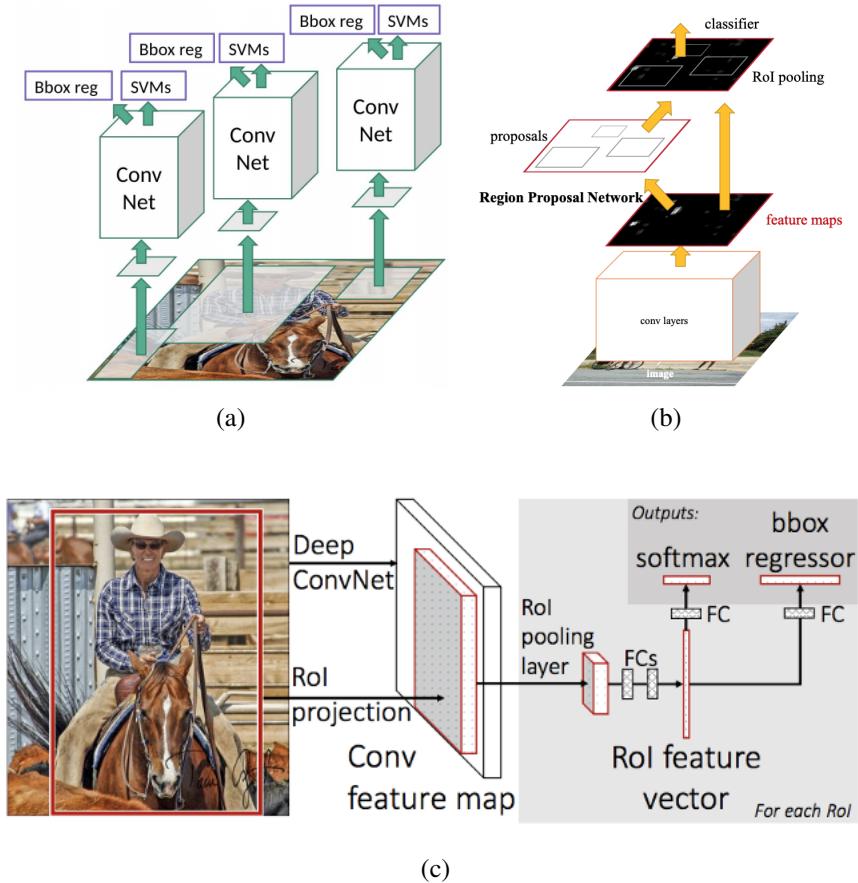


Fig. 2.10 CNN based Object Detection Framework Structure.(a) RCNN, (b) Faster-RCNN, (c)Fast-RCNN.

detection with the detail of flow-char is drawn in Figure 2.9. Fortunately, deep learning, a relatively new technique in machine learning, enables very accurate classification of images using convolutional neural networks (CNNs) [36][28][37]. The capacity of CNNs can be controlled by varying their depth and breadth; moreover, they make strong and mostly correct assumptions about the nature of images (i.e., stationarity of statistics and locality of pixel dependencies). For example, by combining CNN and transfer learning, the authors of [27] achieved an average accuracy of 70.1% using the CIFAR-10 [32] dataset.

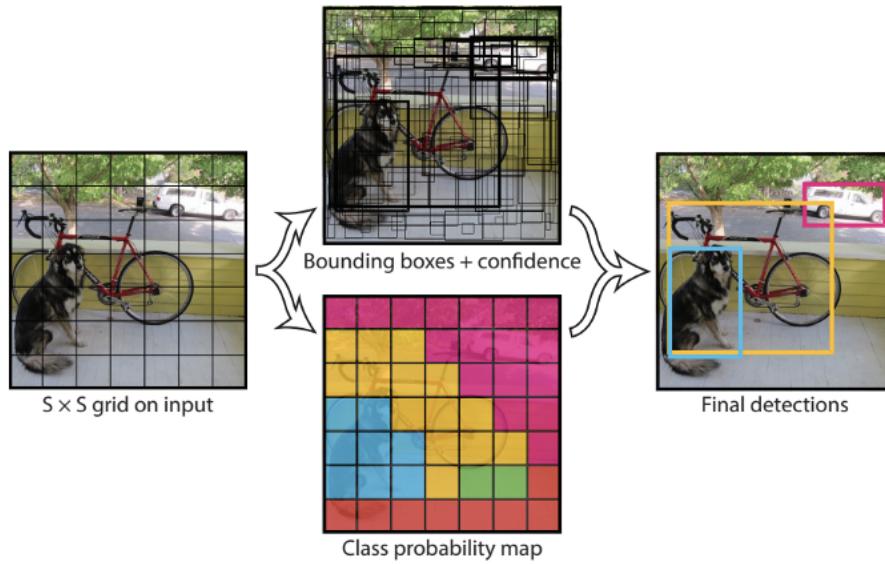


Fig. 2.11 YOLO Structer.

### 2.3.2 Deep Learning based Moving Object Detection

Hybrid background subtraction and deep learning classification enhance the system's accuracy; however, there are issues with detection of individual objects in a group or blocked objects and the background changes by the lighting condition and the environmental noise. Hence, the usage of deep learning is considered for robust object detection tasks. We have identified three primary object detection methods using deep learning:

- R-CNN, Fast R-CNN, Faster R-CNN
- You Only Look Once (YOLO)
- Single Shot Detectors (SSDs)

R-CNN [24] uses selective search [57] to extract several regions, called region proposals, from the image; it then attempts to classify a large number of regions. Each region proposal is placed into CNNs to extract features, which are fed to a support vector machine to classify the presence of object within the candidate region proposal as shown in Figure 2.10(a). The

limitation of this method is that it requires a large amount of time to train and deploy networks because it has to classify multiple region proposals per image. Fast R-CNN [23] solves the limitations of R-CNN by putting the entire image into the CNNs to generate a convolutional feature map and identify the region of proposals based on the map, thus reducing the number of classified region of proposals as drawn in Figure 2.10(c). Both R-CNN and Fast R-CNN use selective search to identify the region proposals. Selective search is a slow and tedious process that affects the network's performance. Faster R-CNN [51] was proposed to allow the network to learn the region proposals as shown in Figure 2.10(b); it uses a separate network to predict the region proposals rather than use a selection search algorithm on the feature maps output using the CNN layer. YOLO [49] is an object detection system targeted for real-time processing. Unlike R-CNN, Fast R-CNN and Faster R-CNN, which use regions to localize the object within the image, YOLO use a single neural network to predict the bounding boxes and the class probabilities for these boxes during training and test periods. Hence, YOLO only examines the input picture once to predict the presence and location of objects. YOLO divides the input image into an SxS grid, and multiple bounding boxes can exist within each grid. For each bounding box, the network outputs a class probability and offset value for the bounding box. The bounding boxes with class probabilities above a certain threshold value are then selected and used to locate the object within the image as shown in Figure 2.11. Specially, YOLOv3 network architecture has 24 convolutional layers, followed by two fully connected layers as shown in Figure 2.12. The initial convolutional layers of the network extract feature from the image, while the fully connected layers predict the output probabilities and coordinates. According to performance comparison in [50], YOLOv3 is faster (45 frames per second (FPS)) than the other object detection algorithms. Faster R-CNN is more accurate than YOLOv3 (a mean average precision (mAP) of 73.2, as compared to 63.4); however, YOLOv3 is considerably faster than Faster R-CNN (FPS of 45, as compared to 7). Therefore, SSDs [42] were released as a balance between these two

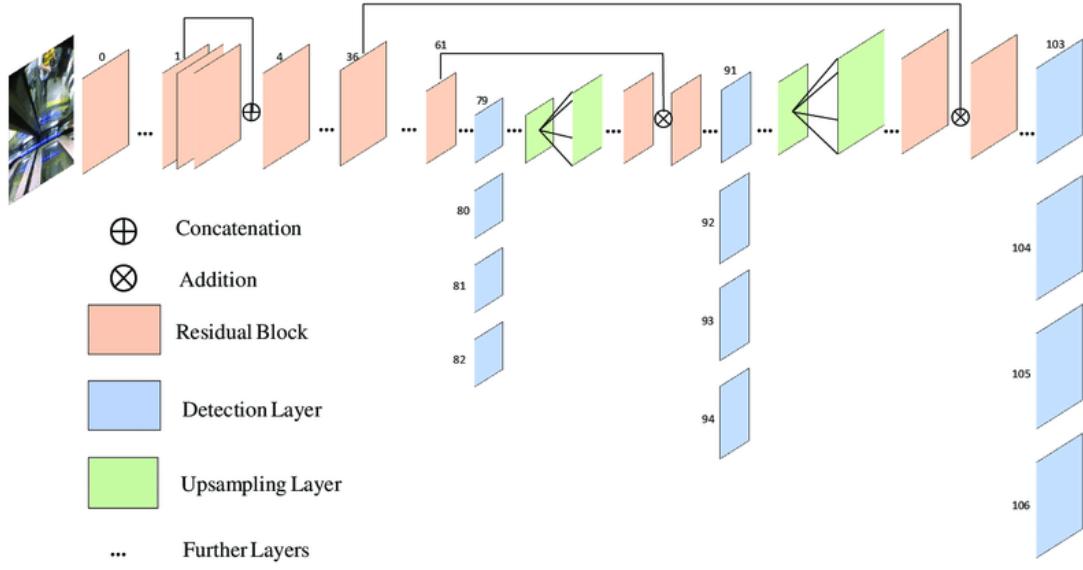


Fig. 2.12 YOLOv3 Network Model.

methods. Compared to YOLO, an SSD runs an input image through a convolutional network only once and computes a feature map. Then, a small  $3 \times 3$  sized convolutional kernels are run on this feature map to predict the bounding boxes and categorization probability. Moreover, SSD uses anchor boxes at various aspect ratios comparable to Faster-RCNN and learns the off-set to a certain extent compared to learning the box. The SSD is able to detect objects of multiple scales because every convolutional layer function at a diverse scale. Compared to the YOLOv3 method [50], the SSDs attain similar accuracy but run slower. In this study, YOLOv3 was applied to robust human detection at cloud servers for our implementation.



# Chapter 3

## Methodology

In this section, the proposed edge-to-cloud system for surveillance video analytics applications is presented detailly. And then, the method for moving objects detection in compressed-domain is analyzed. In addtion, the performance evaluated model for video analytics platform is introduced.

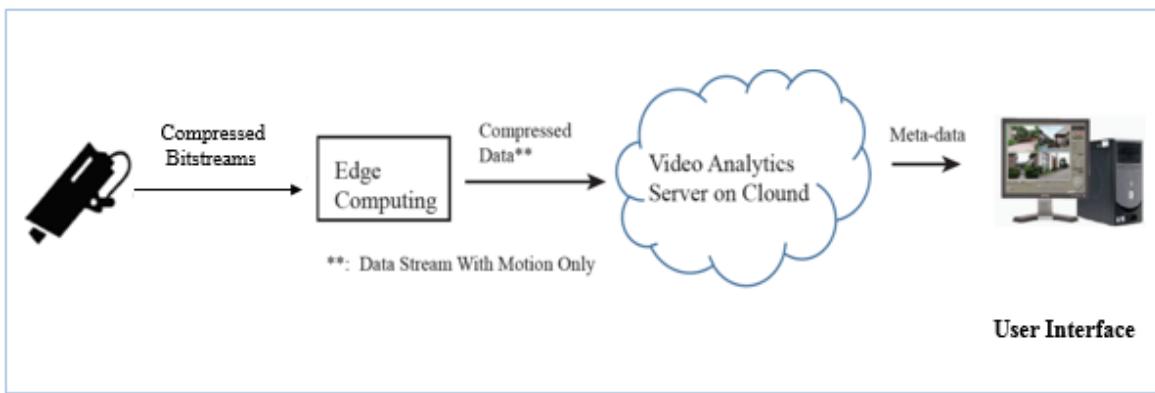


Fig. 3.1 Overview of the proposed edge-to-cloud system model.

### **3.1 The Edge-to-cloud System Model for Surveillance Camera based Applications**

There is a trend to forward computation from the network core to the edge where most of the data are generated. Edge computing has exhibited its potential in reducing the reaction time, minimizing bandwidth usage, and improving energy efficiency. Edge computing performs data processing at the “edge” of the network, close to the data source. For network cameras, audio and other sensors, there is a need to balance both cloud computing and edge computing domains to deliver refined, reliable, and usable data. For edge computing of delay-sensitive video tasks, a camera source node can offload its video task to nearby edge nodes via local wireless/optical networks and edge nodes are within the local communication range of the camera. The camera captures the video sequences and divides each of them into multiple video chunks, compresses these video chunks, and then delivers them to edge nodes. Next, edge nodes implement video processing functions on the received video chunks and upload the results to a cloud server for video analysis (such as object/event detection). A delay-sensitive video assignment is supposed to be processed within a limit and will fail if the deadline is passed. In this study, as shown in Figure 3.1, we consider an edge computing network, which comprises three primary components, namely, camera source node, edge node, and cloud server.

- Camera source node: the camera node periodically generates video tasks, divides each video task into a number of videos chunks, compresses video chunks at certain compression ratios, and then assigns compressed video chunks among all edge nodes as per scheduling policies.
- Edge node: the edge has computational ability and storage capacity and helps pre-process video chunks. Moreover, edge nodes can form cooperative groups based on

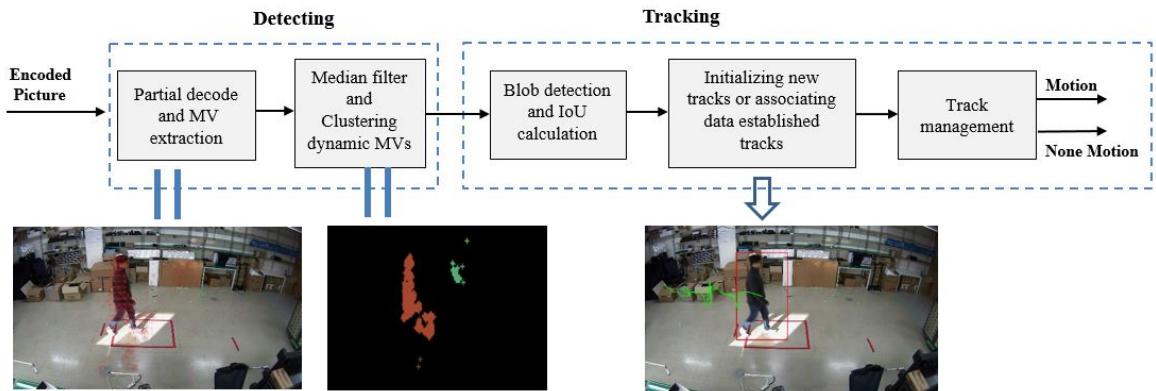


Fig. 3.2 Workflow of the proposed detection-based tracking approach in compressed domain.

specific group formation policy and receive compressed video chunks as per the video load assignment policy.

- Cloud Server: cloud server collects the preprocessing results from edge nodes, which has abundant computational abilities, and performs additional video analysis.

During a sparse edge node deployment, an edge node will only connect to one of the available cameras at a certain location; however, in a dense deployment, an edge node may have multiple choices on selecting multiple cameras. In this study, we focus on reducing processing load on cloud server by minimizing the video chunks that are fed from camera sources. Therefore, an edge device runs preprocessing tasks to filter uninteresting video chunks. The term “uninteresting video chunks” is defined via scenario specification: for surveillance scenarios, it is static scenes without any objects and gradual changes.

## 3.2 The Light-weight Runtime Moving Object Detection in Video Compressed Domain

As mentioned in subsection ??, the aim of this method is to analyze only MVs at the edge device, recognize the real moving objects, and exclude the noised motions that are generated

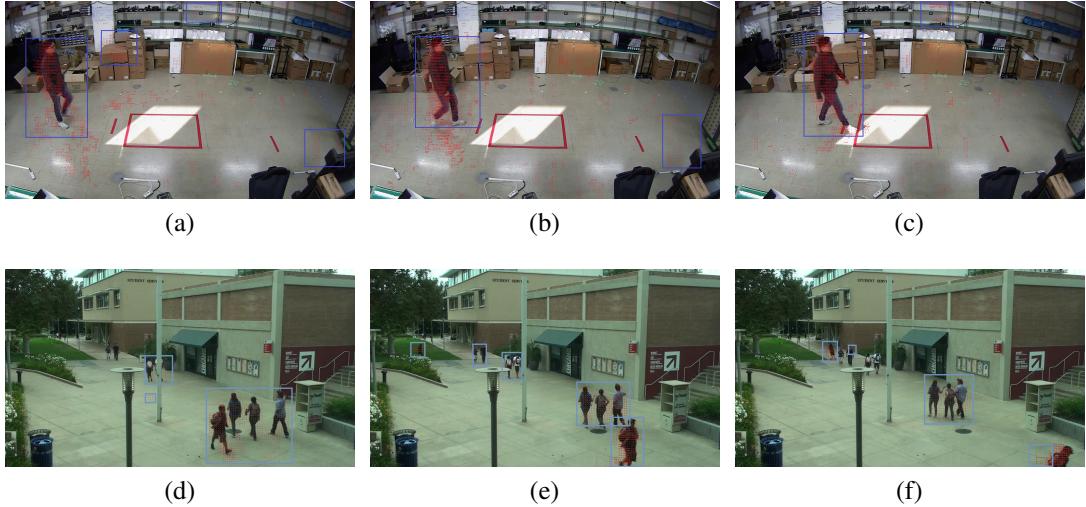


Fig. 3.3 Video coding motion vector extraction from consecutive frames with two video test sequences.

by environmental factors such as illumination changes and background movement, in the current video scene. For this purpose, we present a method that has a workflow shown in Figure 3.2 to analyze the video coding MVs. In order to extract MVs from compressed bitstream, the input video stream is partially decoded and the collected MVs will include both real object MVs and noise MVs. The example of extracted MVs are represented in Figure 3.3 in consecutive frames of two different video test sequences. To analyze these MVs, the following functionalities are involved.

### 3.2.1 Median Filter and Moving Object Detection

Compressed-domain MVs may not represent the actual of true motion due to the properties of motion estimation, biased towards efficient coding as shown in Figure 3.5(a,e). Therefore, it is desirable to eliminate motion vectors that can be categorised as unsuitable for moving objects detection. We assume that MVs whose magnitudes are very high or very low, compared to the other MVs related to the moving object, has to be replaced with the median value of neighbouring MVs. Therefore, the application of vector median filter aims to reduce

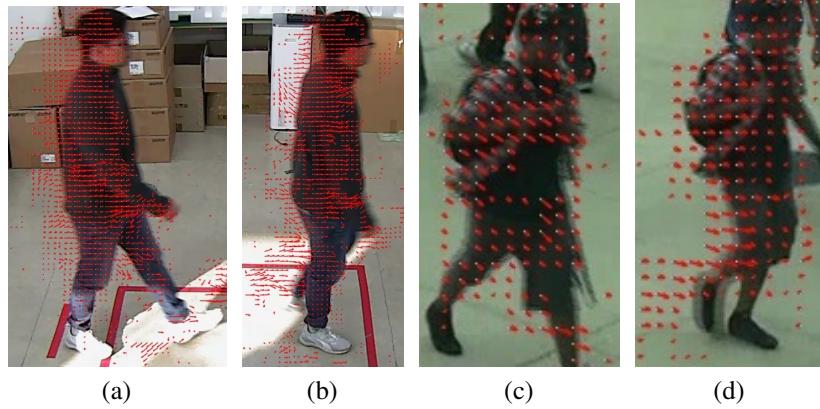


Fig. 3.4 Motion vector of a moving object in different video test sequences: (a, b) our record video, (c, d) Test video in VIRAT

isolated vector noises and smoothen the difference of MV between adjacent blocks. Note that the sliding window approach is used for median filtering. Because H.264/AVC allows for variable-sized block partitioning, we construct a uniformly sample MV field by mapping all MVs to 4x4 blocks. Theoretically, the motion vector average (normalized vector) among all elements  $N_{mv}$  in NxN ( $N=4$  in this study) window function is calculated by Equation 3.1:

$$N_{mv} = \frac{\sum_{i=1, j=1}^{N*N} MV_{ij}}{N * N} \quad (3.1)$$

Where:  $MV_{ij}$  is the MV elements in the  $N \times N$  window. Finally, the smoothed motion filter is presented, which is experimentally determined as shown in Figure 3.5(b,f) using the input MV, including isolated motion vectors noise, are smoothed in the areas that mostly correspond to the object boundaries. In order to detect the moving objects, a cluster detector is involved to determine the number of moving objects and their approximate positions in the scene. Then, a density-based cluster technique is applied to completely segment moving objects. The first step, MV whose magnitudes can be used to easily differentiate moving objects from the stationary background. However, it is difficult to directly and completely segment moving objects because not all MVs on the objects have the same motion state. While every part of a

rigid body maintains nearly the same motion state, different parts of a non-rigid body can move in various ways (see Figure 3.4). In order to cluster detected MVs into dynamic clusters, a range search algorithm based on the euclidean distance of the MV point is used, under the presumption that dense points represents the same object. It requires a parameter  $\varepsilon$  where  $\varepsilon$  is the spatial distance threshold between density reachable MV. The goal is to partition  $n$  MV points  $\chi \subset \mathbb{R}^d$  into  $k$  clusters using k-means filter. Each of the  $n$  data points will be assigned to a cluster with the nearest mean. The mean of each cluster is called its "center". For the k-means problem, we wish to choose  $k$  centers  $C$  so as to minimize the potential Equation 3.2:

$$\phi = \sum_{x \in \chi} \min_{c \in C} \|x - c\|^2 \quad (3.2)$$

From these centers, we can define a clustering by grouping data points according to which center each point is assigned to. With MV points, following steps are performed frame by frame:

- For each  $p_i \in \chi$ , find all the neighboring points within spatial distance  $\varepsilon$ .
- Cluster all the MV points that are density reachable or density connecting [21] and label them.
- Terminate the process after all the MV points are checked. The output is a set of clusters of dynamic points.

The distance threshold  $\varepsilon$  decides the range of density reachability [21]. After this process, the label list includes the moving objects and certain big MVs noises as shown in Figure 3.5(c,d). To eliminate these noises, we based the observation on the fact that noise motions because of lighting condition changes usually randomly occur without following a certain flow. Therefore, the tracking motion's trajectories length is then used to classify the detected motion into the real motion or noise group. If the object tracking trajectory length is larger than a certain threshold, it indicates that the objects move as a flow and the time is sufficiently

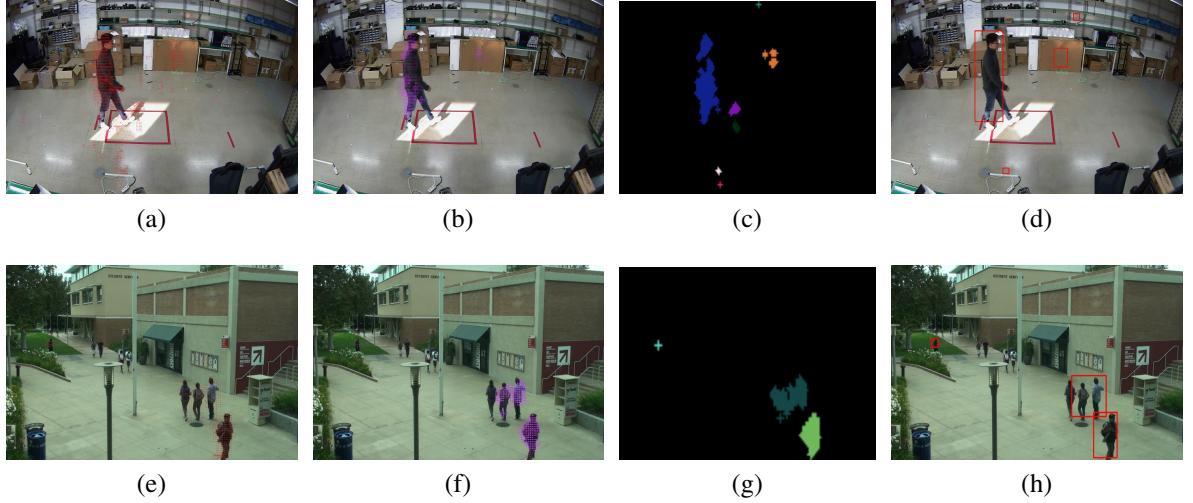


Fig. 3.5 Moving objects detection by the proposed method with two different video test sequences:(a,e) MV extraction; (b,f) Apply median filter; (c,g) Clustering MVs; (d,h) Blob detection.

long to consider it as a real motion. Moreover, I-Frame does not apply motion estimation process, therefore, MV analysis process will be skipped for I-Frames. In order to overcome the discontinuously object detection process, the object tracking algorithm will also be applied to derive the moving object's bounding box in I-frame based on the last states in P-frames. As mentioned in subsection ??, the IoU-based object tracking is applied to track motion because it is light-weight and widely tracking algorithm that calculates the overlap area between two bounding objects. Note that this tracking was implemented and evaluated in some previous studies [52], [39].

### 3.2.2 The IoU based Moving Objects Tracking

To apply the IoU-based object tracking, the detected clusters are normalized with a rectangle bounding box according to the cluster's size as shown in Figure 3.5(d,h). The correlated regions are connected into blobs. Each blob is represented by its top-left and bottom-right corners, i.e.,  $(x_1, y_1, x_2, y_2)$  may include one or many moving objects. Because the moving object's bounding box size and shape can be different comparatively frame by frame depend-

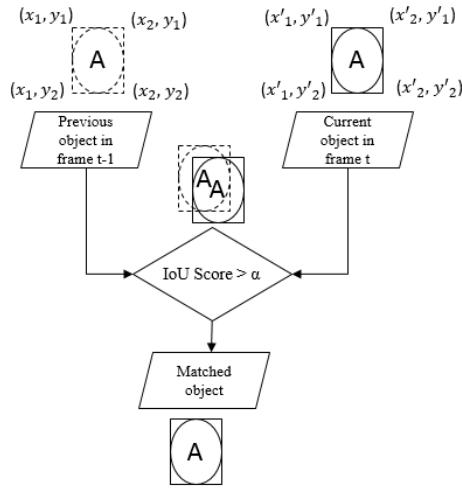


Fig. 3.6 Object matching method

ing on MV's intensity. Therefore, to track the moving object, an object matching algorithm based on the overlapped area of bounding boxes is applied. We assume that the existence of the real motion is continuous frame by frame. For each bounding box  $B_1$  in the previous frame, we identify a bounding box  $B_2$  at the current frame with the highest IoU rate. Note that IoU is attained by:

$$IoU(B_1 \cap B_2) = \frac{\|B_1 \cap B_2\|}{\|B_1 \cup B_2\|} = \frac{\|B_1 \cap B_2\|}{\|B_1\| + \|B_2\| - \|B_1 \cap B_2\|} \quad (3.3)$$

As its definition in the Equation 3.3, IoU is invariant to the scale, indicating that the similarity between two arbitrary shapes A and B is independent from the scale of their space. The IoU computation's pseudo-code is given in Algorithm 1. If two bounding boxes do not overlap, the IoU value will be 0 and if the IoU score is greater than a detection score threshold ( $\alpha$ ), two bounding boxes are considered in same account (Figure 3.6). The detection score threshold is determined through experiments and depend on the object velocity as well as the distance between objects and camera. Therefore, the method is run with multiple times with different detection score thresholds to tune the best value for each application scenarios.

---

**Data:** Corners of the two bounding boxes.

- First bounding box:  $A1(x_1, y_1), B1(x_2, y_1), C1(x_2, y_2), D1(x_1, y_2)$
- Second bounding box:  $A2(x'_1, y'_1), B2(x'_2, y'_1), C2(x'_2, y'_2), D2(x'_1, y'_2)$   
where  $x_1 \leq x_2, y_2 \leq y_1$  and  $x'_1 \leq x'_2, y'_2 \leq y'_1$ .

**Caculation:** IoU value

- The area of first bounding box :  $Area_1 = (x_2 - x_1) \times (y_1 - y_2)$
  - The area of second bounding box :  $Area_2 = (x'_2 - x'_1) \times (y'_1 - y'_2)$
  - The area of overlap:  
$$Area_{overlap} = (max(x_2, x'_2) - min(x_1, x'_1)) \times (max(y_2, y'_2) - min(y_1, y'_1))$$
  - $$IoU = \frac{Area_{overlap}}{Area_1 + Area_2 - Area_{overlap}}$$
- 

**Algorithm 1:** IoU for two bounding boxes.

### 3.3 Performance Evaluated Model

Computing Resources of VA server is affected by many explicit factors. For example: whether VA function is running, the complexity of VA function, video resolution, which kind of deep learning model used for VA function, how many cameras is serving. However, the primary factor and biggest effect is number of serving cameras and whether VA function is running. Because if the VA server is in idle status, then other factors will be implicit. Let assume that we have a video test with N consecutive frames with K frames had the real motion ( $K \leq N$ ). For each frame, VA server cost S and T unit of average computing resource for processing and skip frame case respectively. For conventional method of video analytics server, where  $T = S$  because all frames are processed , the computing resource average is:

$$Comp_c = S \quad (3.4)$$

With our proposed method, the computing resource average during N frames is calculated as the following equation:

$$Comp_p = \frac{(K * S + (N - K) * T)}{N} \quad (3.5)$$

The performance ratio of two method is:

$$\frac{Comp_p}{Comp_c} = \frac{(K * S + (N - K) * T)}{N * S} = \frac{K}{N} + (1 - \frac{K}{N}) * \frac{T}{S} \quad (3.6)$$

if an object motion always appear in the video ( $K \approx N$ ), then:

$$\frac{Comp_p}{Comp_c} \approx 1. \quad (3.7)$$

In case of GPU computing resource, when VA server skip a frame then  $T = 0$  and:

$$\frac{Comp_p}{Comp_c} = \frac{K}{N} \quad (3.8)$$

If computing resource is CPU utilization and network throughput , T become very small. For example, T is used only for listening new data or connection with networking resource, then:

$$\frac{Comp_p}{Comp_c} \approx \frac{K}{N} \quad (3.9)$$

# **Chapter 4**

## **Implementation And Performance**

### **Evaluation**

In this section, the implementation and performance evaluation of the edge-to-cloud system with proposed method is presented. Furthermore, the information of video datasets and the scenario setup are provided. In this implementation, VA server executes the application of intrusion detection. Although the evaluated platform integrates specific application, it is a general design and can be extended for other application with few modifications. The workflow of the evaluated platform is represented in Figure 4.1. It has two main components:

- Edge node implementation: The streaming data from camera sources are parsed and the proposed method is applied to detect moving objects in the current frame. If the encoded frame includes the motion, it will be forwarded to a cloud node using its own real-time streaming protocol (RTSP) server. To avoid decoding inaccuracies at the cloud node, all frames from starting time to ending time of the motion are continuously delivered in a connection session. Each session will start with an intra-coded frame.

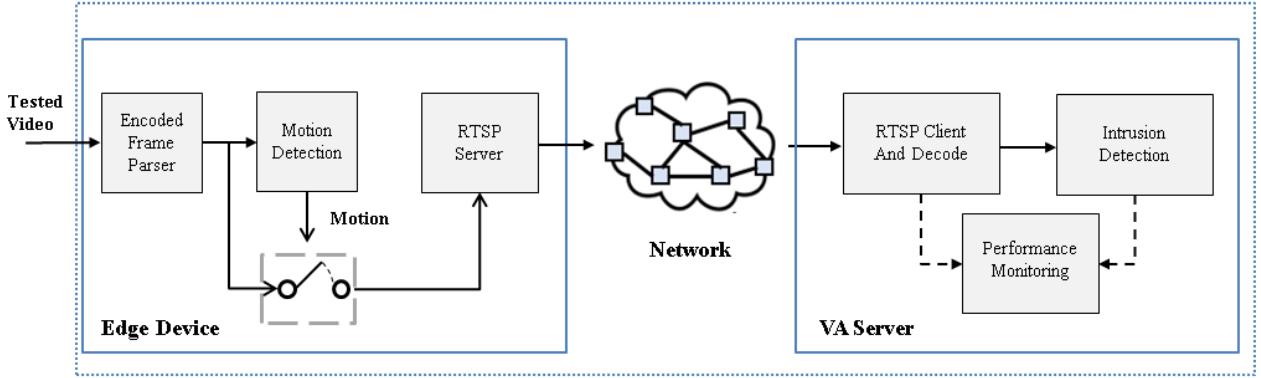


Fig. 4.1 Overview of System Design

Table 4.1 Hardware Specifications.

Specifications	Edge Device	Video Analytics Server
Device Name	Raspberry Pi 4	NVIDIA Jetson Xavier
Operating System	Ubuntu 18.04, 64 bits	Ubuntu 18.04, 64 bits
GPU	Not Supported	NVIDIA Maxwell architecture with NVIDIA CUDA
CPU	Quad-core ARM Cortex-A72	Quad-core ARM Cortex-A57 MPCore processor
RAM	4 GB	8 GB

- Cloud node implementation: Receiving the forwarded encoded frame with the motion from the edge node over the network and then decoding and placing the output images into the intrusion detection module, which uses YOLO to detect humans.

#### 4.0.1 Scenario Setup

- Testbed: We built a testbed comprising a single edge device node and a single video analytics server that runs as a cloud node, as shown in Figure 4.2. The edge device

Table 4.2 Video Test Sequence.

Tested Video Information	
Resolution	1920x1080
Length	6 minutes
Codec	H264
Group Of Picture (GOP)	30
Frame Rate	25

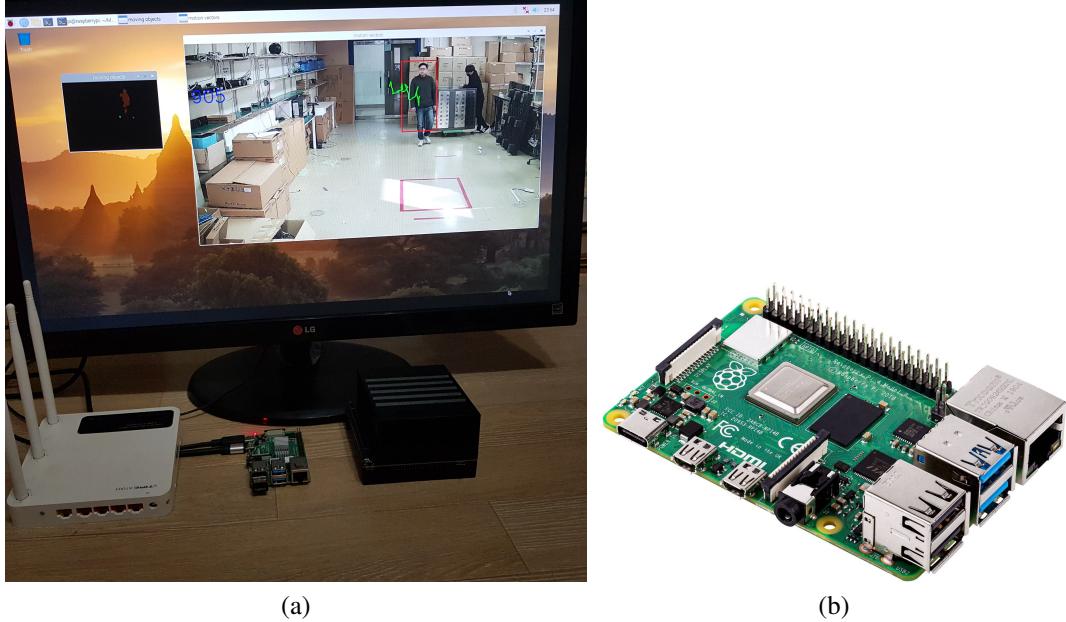


Fig. 4.2 Testbed: (a) Scenario Setup, (b) The implemented edge device.

Table 4.3 Testbed: Tested Video Grouth-truth motion time.

<b>Time</b>	<b>Duration (Seconds)</b>
00:00:50 00:01:10	20
00:01:25 00:01:45	20
00:02:12 00:01:10	5
00:02:39 00:02:45	6
00:03:50 00:04:48	58
00:05:00 00:05:35	35
00:05:40 00:06:00	20
Total	164

node involves the moving objects detection and runs on a low-computation device named *Raspberry Pi 4* and video analytics server is executed on *Nvidia- Jetson Xavier* because of a GPU that is supported to run YOLO. The hardware specifications of edge node and video analytics server are then listed in Table 4.1. Note that the two devices are directly connected to a router using a wired cable.

- Video Test Sequence: experiments have been conducted on the two video datasets. The VIRAT video dataset [4] was collected in natural scenes showing people performing normal actions for video surveillance domains. The second dataset is previously recorded from the our surveillance camera and uploaded [5]. The details of our video test sequence and the ground-truth motion time are listed in Tables 4.3 and 4.4.

We evaluate the performance of the moving object detection method and the proposed edge-to-cloud system separately.

## 4.1 The Light-weight Runtime Moving Object Detection in Video Compressed Domain

To evaluate the quality of the proposed method for the moving object detection, we calculated the IoU score metric of the detected object's bounding box with those of the ground-truth bounding box. Because the accuracy of the peoposed method is depend on the MV's density, we run the proposed method multiple times with different scenario application such as: different camera distances, different moving object speeds. We observe that except the I-Frame which does not apply motion estimation, the moving objects including sometimes big MV noises always be detected in other frames. Example of the moving object detection results are shown in Figure 4.3.

The green bounding boxes shown in the Figure are ground-truth bounding boxes, while the red bounding boxes are detected objects using the proposed method.The average IoU

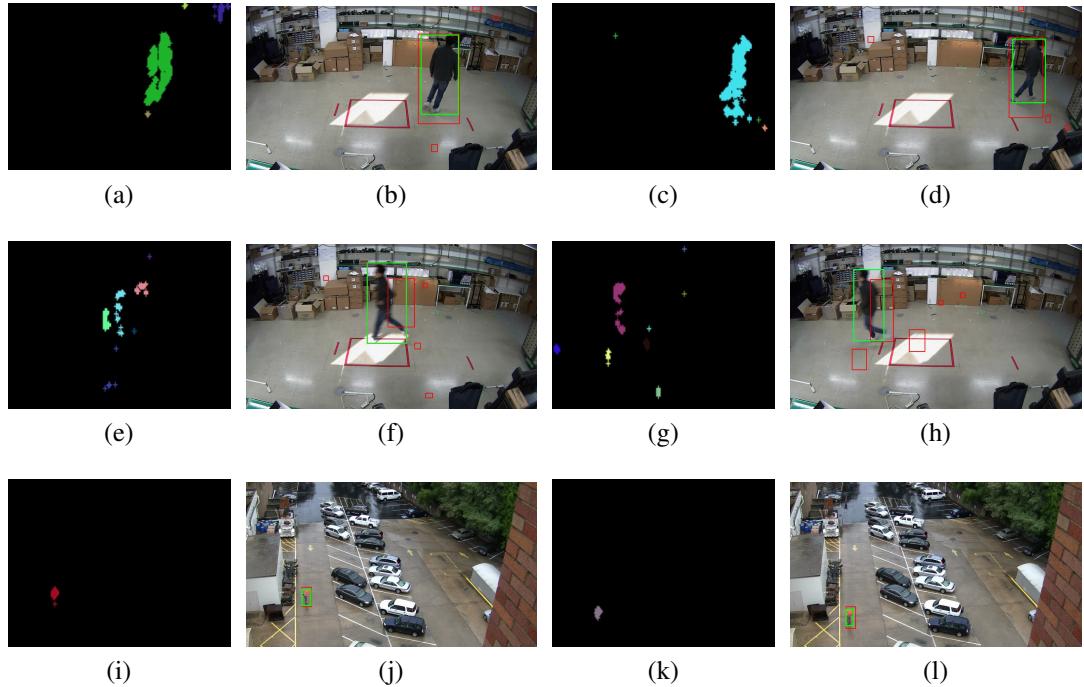


Fig. 4.3 Moving objects detection by the proposed method in different scenarios: (a, b, c, d) human walking; (e, f, g, h) human running; (i, j, k, l) test with a far distance of camera.

Table 4.4 Average IoU of the moving object detection in compressed-domain in different scenarios.

Video Test Sequence	Scenario		IoU Average
	Camera Position	Moving Speed	
Our recorded test video	Near	Normal	0.75
Our recorded test video	Near	Fast	0.26
Video Test from VIRAT	Far	Normal	0.6

Table 4.5 Average per-frame running times for preprocessing and tracking procedures. Values are expressed in miliseconds (ms) and frame per second(FPS).

Frame Size	ST-MRF[16]	Graph Cuts[15]	Proposed Method
1280x720	64 ms (16 FPS)	62 ms (17 FPS)	<b>39 ms (26 FPS)</b>
1920x1080	N/A	N/A	<b>69 ms (14 FPS)</b>

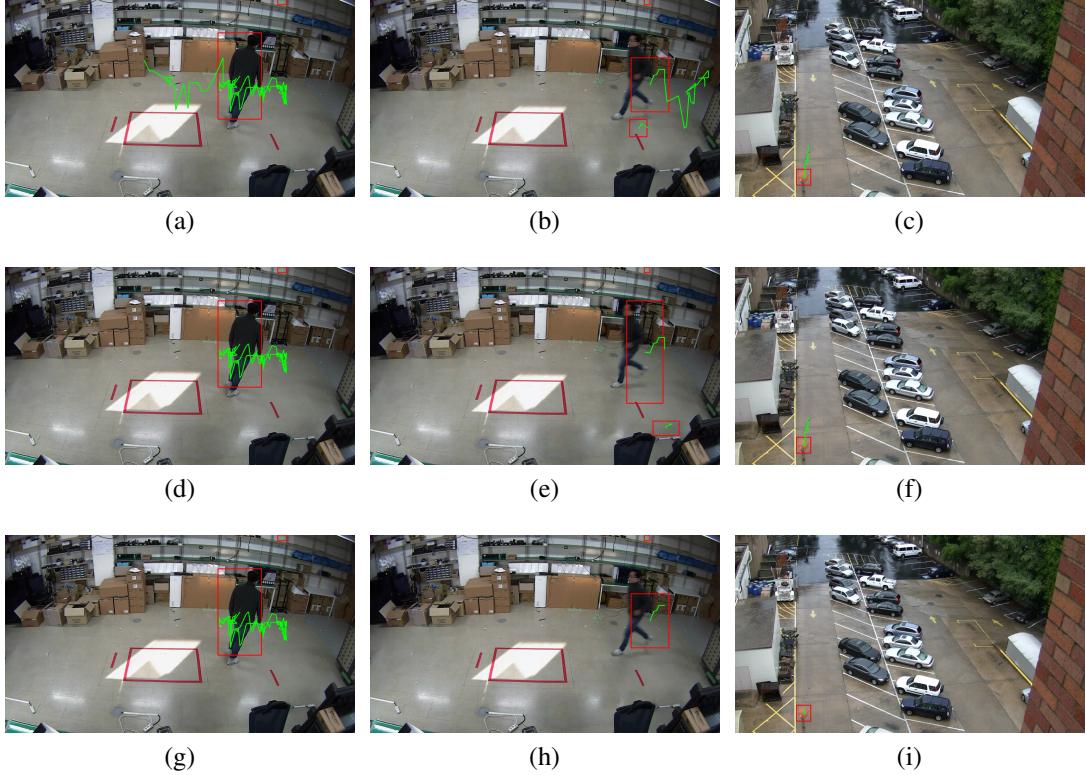


Fig. 4.4 Moving objects tracking by the proposed method with different  $\alpha$  thresholds:  $\alpha = 0.25$  in (a, b, c),  $\alpha = 0.4$  in (d, e, f) and  $\alpha = 0.5$  in (d, e, f).

scores are shown in detail for each scenario in Table 4.4. The results show that when a camera is placed at a close distance and the human does not move fast (i.e., human walking), the proposed method detected well and achieved good average IoU score of 0.75. However, when speed of human is fast in case of running or the camera is in far distance, the MV's density is decreased, the detector return lower average IoU scores. For object tracking evaluation, the scenarios are run with multiple times with different detection score threshold  $\alpha$  of 0.25, 0.4 and 0.5 with the results are shown in Figure 4.4. In this experiment, each  $\alpha$  threshold was applied for three same scenarios of human walking in Figure 4.4 (a, d, g), human running in Figure 4.4 (b, e, h) and the far camera distance in Figure 4.4 (c, f, i). We see that with each test scenario, with lower  $\alpha$  threshold value, the tracking object capability is better with longer the object trajectories. However, it will increase the number of false alarm detection if the

Table 4.6 Average computing resources of both the conventional method and the proposed method.

<b>Computing Resources</b>	<b>Conventional Method</b>	<b>Proposed Method</b>	<b>Performance Ratio</b>
GPU Utilization (%)	65.61	33.73	0.51
CPU Utilization (%)	50.24	25.9	0.51
Download Throughput (Kbps)	4028	1808.2	0.45

noise MVs appear frequently in some specified areas.

To optimize the running time speed as well as the performance, the edge node and cloud node all are implemented in C++ using the multithread architecture. We observe that the proposed method does not indicate additional computational difficulty and achieve the approved evaluated results in terms of processing time when compared with previous studies [12] [31]. The average consuming time is measured with different video resolutions and shown in Table 4.5. Compare to other studies, the average processing time for high definition resolution video is approximately 39ms/frame, which outperforms most of the state of art method. This indicates that the proposed algorithm almost handles the data in real-time.

## 4.2 Performance Evaluation Results

In this experiment, the proposed edge-to-cloud platform will be implemented and evaluated. Our recorded video test sequence is selected because of including different moving object speeds. After experiments using different  $\alpha$  in different scenarios, we found that the optimal  $\alpha$  is 0.5 for this scenario. The entire demonstration was recorded and uploaded [6]. The evaluated performances of the demonstration are presented in detail as follows.

### 4.2.1 Computing Resources Consumption

During demonstration of intrusion detection application, the clouds node's computing resources, including CPU, GPU utilization, and network download throughput, was monitored

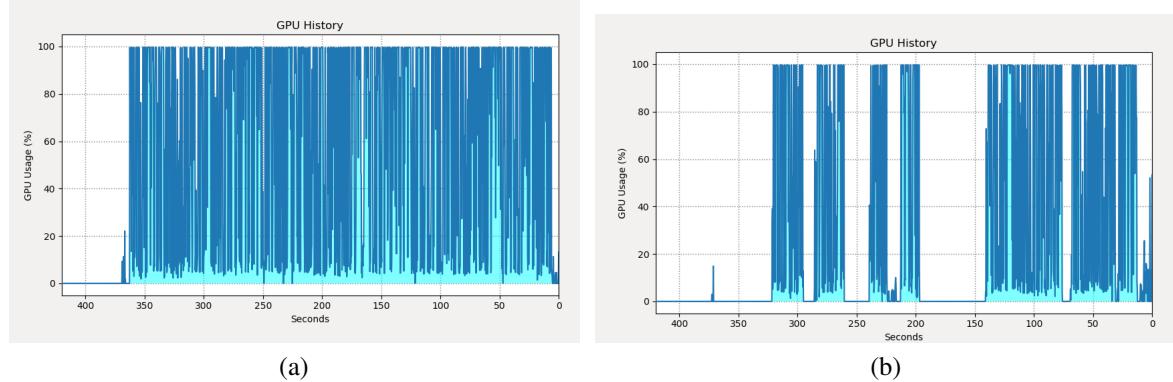


Fig. 4.5 GPU Monitoring: (a) With the conventional method, (b) With the proposed method.

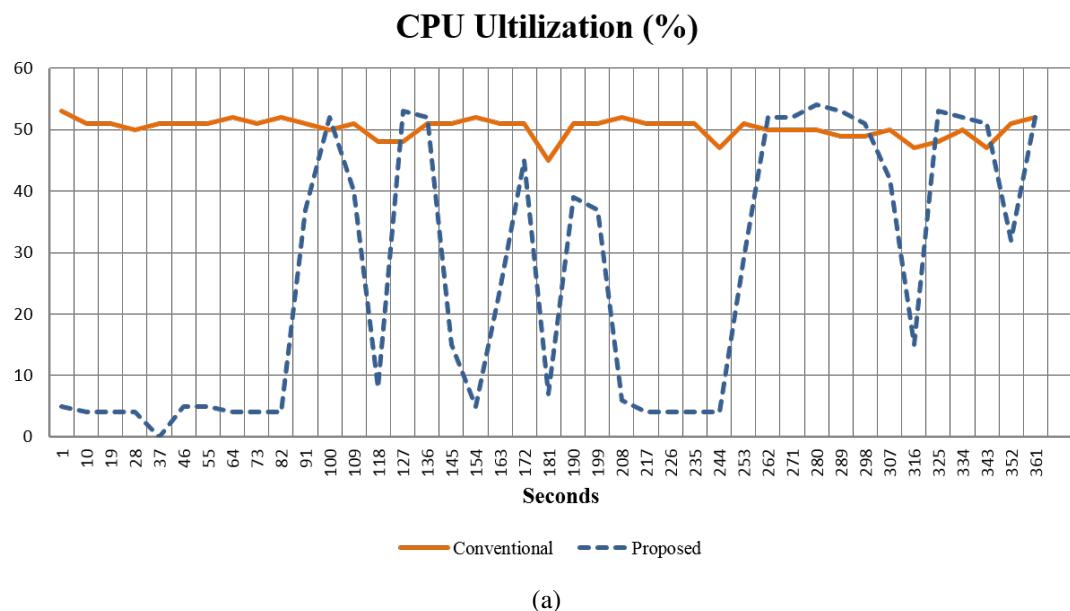
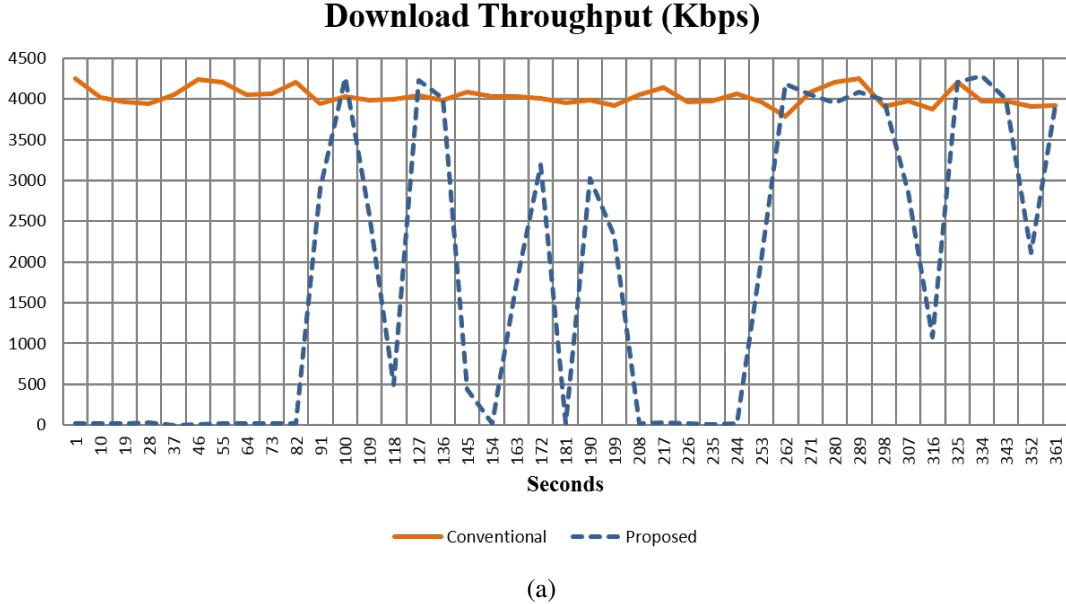


Fig. 4.6 CPU Monitoring with both the conventional method and the proposed method.



(a)

Fig. 4.7 Network download throughput monitoring with both the conventional method and the proposed method.

and recorded and shown in Figure 4.1. For comparison, the performance results are compared to that of the conventional method, which did not use edge node as shown in Figure 1.1. The results indicate that both methods achieve the same accuracy in terms of alert notification when humans entered the restricted area. The demonstration's record of both these methods are uploaded to [7], [8]. In terms of consumption of computational resources, GPU, CPU utilizations, and download throughput of both methods are measured and presented in Figure 4.5, 4.6 and 4.7 respectively. The figures clearly show the advantage of using the proposed method. While the conventional method of processing all video frames captured from camera leads to consumption of computational resources, our method only processes when there is motion, and therefore computational resources are dynamically allocated and economical. Another observation is that our proposed method is extremely effective for detecting motion of the scene compared to the ground-truth motion time in Table 4.3. In detail, the time for consuming and releasing computing resources of VA server was matched with the time for appearance and disappearance of motion in ground-truth table. Since the conventional

method processes all video frames, we assume that the its computing resource average is S. According to formulation 3.6, the performance ratio of both methods in this video will be as follows:

$$\frac{Comp_p}{Comp_c} = \frac{(K * S + (N - K) * T)}{N * S} = \frac{K}{N} + (1 - \frac{K}{N}) * \frac{T}{S} = \frac{140}{360} + (1 - \frac{140}{360}) * \frac{T}{S} = 0.4 + 0.6 * \frac{T}{S}$$
(4.1)

Compared with the performance ratio, which is calculated in real scenario test in Table 4.6, our performance theory model and real measurement are matching and reasonable.

# References

- [1] (2020 (accessed September 3, 2020)a). *Available Online*. <https://www.nytimes.com/2018/05/26/us/chicago-police-surveillance.html>.
- [2] (2020 (accessed September 3, 2020)b). *Available Online*. <https://www.telegraph.co.uk/technology/10172298/One-surveillance-camera-for-every-11-people-in-Britain-says-CCTV-survey.html>.
- [3] (2020 (accessed September 3, 2020)c). *Available Online*. <https://www.wsj.com/articles/BL-CJB-22562>.
- [4] (2020 (accessed September 3, 2020)). *Available Online*. <https://viratdata.org>.
- [5] (2020 (accessed September 3, 2020)). *Available Online*. <https://youtu.be/v24ldT1AGRw>.
- [6] (2020 (accessed September 3, 2020)). *Available Online*. <https://github.com/diennv/MotionVectorAnalysis>.
- [7] (2020 (accessed September 3, 2020)). *Available Online*. [https://www.youtube.com/watch?v=Cz\\_zxr\\_ElTU](https://www.youtube.com/watch?v=Cz_zxr_ElTU).
- [8] (2020 (accessed September 3, 2020)). *Available Online*. <https://www.youtube.com/watch?v=-fRc36HAduI&feature=youtu.b>.
- [9] Achanta, R., Kankanhalli, M., and Mulhem, P. (2002). Compressed domain object tracking for automatic indexing of objects in mpeg home video. In *Proceedings. IEEE International Conference on Multimedia and Expo*, volume 2, pages 61–64. IEEE.
- [10] Ananthanarayanan, G., Bahl, V., Cox, L., Crown, A., Nogbahi, S., and Shu, Y. (2019). Demo: Video analytics-killer app for edge computing. In *ACM MobiSys*.
- [11] Babaee, M., Dinh, D. T., and Rigoll, G. (2018). A deep convolutional neural network for video sequence background subtraction. *Pattern Recognition*, 76:635–649.
- [12] Bombardelli, F., Gül, S., Becker, D., Schmidt, M., and Hellge, C. (2018). Efficient object tracking in compressed video streams with graph cuts. In *2018 IEEE 20th International Workshop on Multimedia Signal Processing (MMSP)*, pages 1–6. IEEE.
- [13] Boykov, Y., Veksler, O., and Zabih, R. (2001). Fast approximate energy minimization via graph cuts. *IEEE Transactions on pattern analysis and machine intelligence*, 23(11):1222–1239.

- [14] Bross, B., Helle, P., Lakshman, H., and Ugur, K. (2014). Inter-picture prediction in hevc. In *High Efficiency Video Coding (HEVC)*, pages 113–140. Springer.
- [15] Cai, Z., Saberian, M., and Vasconcelos, N. (2015). Learning complexity-aware cascades for deep pedestrian detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3361–3369.
- [16] Canel, C., Kim, T., Zhou, G., Li, C., Lim, H., Andersen, D. G., Kaminsky, M., and Dulloor, S. R. (2019). Scaling video analytics on constrained edge nodes. *arXiv preprint arXiv:1905.13536*.
- [17] Chen, N., Chen, Y., Ye, X., Ling, H., Song, S., and Huang, C.-T. (2017a). Smart city surveillance in fog computing. In *Advances in Mobile Cloud Computing and Big Data in the 5G Era*, pages 203–226. Springer.
- [18] Chen, T. Y.-H., Ravindranath, L., Deng, S., Bahl, P., and Balakrishnan, H. (2015). Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 155–168.
- [19] Chen, Y., Wang, J., Zhu, B., Tang, M., and Lu, H. (2017b). Pixel-wise deep sequence learning for moving object detection. *IEEE Transactions on Circuits and Systems for Video Technology*.
- [20] Dong, L., Zoghlami, I., and Schwartz, S. C. (2006). Object tracking in compressed video with confidence measures. In *2006 IEEE International Conference on Multimedia and Expo*, pages 753–756. IEEE.
- [21] Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231.
- [22] Favalli, L., Mecocci, A., and Moschetti, F. (2000). Object tracking for retrieval applications in mpeg-2. *IEEE Transactions on Circuits and Systems for Video Technology*, 10(3):427–432.
- [23] Girshick, R. (2015). Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448.
- [24] Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587.
- [25] Gujrathi, P., Priya, R. A., and Malathi, P. (2014). Detecting moving object using background subtraction algorithm in fpga. In *2014 Fourth International Conference on Advances in Computing and Communications*, pages 117–120. IEEE.
- [26] Hsieh, K., Ananthanarayanan, G., Bodik, P., Venkataraman, S., Bahl, P., Philipose, M., Gibbons, P. B., and Mutlu, O. (2018). Focus: Querying large video datasets with low latency and low cost. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 269–286.

- [27] Hussain, M., Bird, J. J., and Faria, D. R. (2018). A study on cnn transfer learning for image classification. In *UK Workshop on Computational Intelligence*, pages 191–202. Springer.
- [28] Jarrett, K., Kavukcuoglu, K., LeCun, Y., et al. (2009). What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision (ICCV)*, pages 2146–2153. IEEE.
- [29] Jiang, J., Ananthanarayanan, G., Bodik, P., Sen, S., and Stoica, I. (2018). Chameleon: scalable adaptation of video analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 253–266.
- [30] Jiang, X., Song, T., Katayama, T., and Leu, J.-S. (2019). Spatial correlation-based motion-vector prediction for video-coding efficiency improvement. *Symmetry*, 11(2):129.
- [31] Khatoonabadi, S. H. and Bajic, I. V. (2012). Video object tracking in the compressed domain using spatio-temporal markov random fields. *IEEE transactions on image processing*, 22(1):300–313.
- [32] Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images. Technical report, Citeseer.
- [33] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90.
- [34] Kumar, S. and Yadav, J. S. (2016). Segmentation of moving objects using background subtraction method in complex environments. *Radioengineering*, 25(2):399–408.
- [35] Laroche, G., Jung, J., and Pesquet-Popescu, B. (2008). Rd optimized coding for motion vector predictor selection. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(9):1247–1257.
- [36] LeCun, Y., Kavukcuoglu, K., and Farabet, C. (2010). Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 253–256. IEEE.
- [37] Lee, H., Grosse, R., Ranganath, R., and Ng, A. Y. (2009). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th annual international conference on machine learning*, pages 609–616. ACM.
- [38] Lee, J. and Park, M. (2012). An adaptive background subtraction method based on kernel density estimation. *Sensors*, 12(9):12279–12300.
- [39] Li, C., Xing, Q., and Ma, Z. (2020a). Hksiamfc: Visual-tracking framework using prior information provided by staple and kalman filter. *Sensors*, 20(7):2137.
- [40] Li, H., Lin, Z., Shen, X., Brandt, J., and Hua, G. (2015). A convolutional neural network cascade for face detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5325–5334.

- [41] Li, Y., Padmanabhan, A., Zhao, P., Wang, Y., Xu, G. H., and Netravali, R. (2020b). Reducto: On-camera filtering for resource-efficient real-time video analytics. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 359–376.
- [42] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016). Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer.
- [43] Lu, N., Wang, J., Wu, Q., and Yang, L. (2008). An improved motion detection method for real-time surveillance. *IAENG International Journal of Computer Science*, 35(1).
- [44] Lu, X., Izumi, T., Takahashi, T., and Wang, L. (2014). Moving vehicle detection based on fuzzy background subtraction. In *2014 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 529–532. IEEE.
- [45] Ohm, J.-R., Sullivan, G. J., Schwarz, H., Tan, T. K., and Wiegand, T. (2012). Comparison of the coding efficiency of video coding standards—including high efficiency video coding (hevc). *IEEE Transactions on circuits and systems for video technology*, 22(12):1669–1684.
- [46] Ou, X., Yan, P., Zhang, Y., Tu, B., Zhang, G., Wu, J., and Li, W. (2019). Moving object detection method via resnet-18 with encoder-decoder structure in complex scenes. *IEEE Access*, 7:108152–108160.
- [47] Patil, P. W. and Murala, S. (2018). Msfgnet: A novel compact end-to-end deep network for moving object detection. *IEEE Transactions on Intelligent Transportation Systems*, 20(11):4066–4077.
- [48] Philippou, O. (2020 (accessed September 3, 2020)). *Video Surveillance Installed Base Report – 2019*. <https://technology.informa.com/607069/video-surveillance-installed-base-report-2019>.
- [49] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788.
- [50] Redmon, J. and Farhadi, A. (2018). Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.
- [51] Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99.
- [52] Sheu, R.-K., Pardeshi, M., Chen, L.-C., and Yuan, S.-M. (2019). Stam-ccf: Suspicious tracking across multiple camera based on correlation filters. *Sensors*, 19(13):3016.
- [53] Stauffer, C. and Grimson, W. E. L. (1999). Adaptive background mixture models for real-time tracking. In *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*, volume 2, pages 246–252. IEEE.

- [54] Stone, T., Stone, N., Jain, P., Jiang, Y., Kim, K.-H., and Nelakuditi, S. (2019). Towards scalable video analytics at the edge. In *2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pages 1–9. IEEE.
- [55] Sullivan, G. J., Ohm, J.-R., Han, W.-J., and Wiegand, T. (2012). Overview of the high efficiency video coding (hevc) standard. *IEEE Transactions on circuits and systems for video technology*, 22(12):1649–1668.
- [56] Thanh Le, T., Jeong, J., and Ryu, E.-S. (2019). Efficient transcoding and encryption for live 360 cctv system. *Applied Sciences*, 9(4):760.
- [57] Uijlings, J. R., Van De Sande, K. E., Gevers, T., and Smeulders, A. W. (2013). Selective search for object recognition. *International journal of computer vision*, 104(2):154–171.
- [58] Wang, Y., Luo, Z., and Jodoin, P.-M. (2017). Interactive deep learning method for segmenting moving objects. *Pattern Recognition Letters*, 96:66–75.
- [59] Wang, Z., Sun, X., Diao, W., Zhang, Y., Yan, M., and Lan, L. (2019). Ground moving target indication based on optical flow in single-channel sar. *IEEE Geoscience and Remote Sensing Letters*, 16(7):1051–1055.
- [60] Yoneyama, A., Nakajima, Y., Yanagihara, H., and Sugano, M. (1999). Moving object detection and identification from mpeg coded data. In *Proceedings 1999 International Conference on Image Processing (Cat. 99CH36348)*, volume 2, pages 934–938. IEEE.
- [61] Zeng, D. and Zhu, M. (2018). Background subtraction using multiscale fully convolutional network. *IEEE Access*, 6:16010–16021.



# **Appendix A**

## **How to install L<sup>A</sup>T<sub>E</sub>X**

### **Windows OS**

#### **TeXLive package - full version**

1. Download the TeXLive ISO (2.2GB) from  
<https://www.tug.org/texlive/>
2. Download WinCDEmu (if you don't have a virtual drive) from  
<http://wincdemu.sysprogs.org/download/>
3. To install Windows CD Emulator follow the instructions at  
<http://wincdemu.sysprogs.org/tutorials/install/>
4. Right click the iso and mount it using the WinCDEmu as shown in  
<http://wincdemu.sysprogs.org/tutorials/mount/>
5. Open your virtual drive and run setup.pl

or

## **Basic MikTeX - T<sub>E</sub>X distribution**

1. Download Basic-MiK<sub>T</sub>E<sub>X</sub>(32bit or 64bit) from  
<http://miktex.org/download>
2. Run the installer
3. To add a new package go to Start » All Programs » MikTeX » Maintenance (Admin)  
and choose Package Manager
4. Select or search for packages to install

## **TexStudio - T<sub>E</sub>X editor**

1. Download TexStudio from  
<http://texstudio.sourceforge.net/#downloads>
2. Run the installer

## **Mac OS X**

### **MacTeX - T<sub>E</sub>X distribution**

1. Download the file from  
<https://www.tug.org/mactex/>
2. Extract and double click to run the installer. It does the entire configuration, sit back  
and relax.

### **TexStudio - T<sub>E</sub>X editor**

1. Download TexStudio from  
<http://texstudio.sourceforge.net/#downloads>

2. Extract and Start

## Unix/Linux

### TeXLive - TeX distribution

#### Getting the distribution:

1. TexLive can be downloaded from

<http://www.tug.org/texlive/acquire-netinstall.html>.

2. TexLive is provided by most operating system you can use (rpm, apt-get or yum) to get TexLive distributions

#### Installation

1. Mount the ISO file in the mnt directory

```
mount -t iso9660 -o ro,loop,noauto /your/texlive####.iso /mnt
```

2. Install wget on your OS (use rpm, apt-get or yum install)

3. Run the installer script install-tl.

```
cd /your/download/directory  
./install-tl
```

4. Enter command ‘i’ for installation

5. Post-Installation configuration:

<http://www.tug.org/texlive/doc/texlive-en/texlive-en.html#x1-320003.4.1>

6. Set the path for the directory of TexLive binaries in your .bashrc file

**For 32bit OS**

For Bourne-compatible shells such as bash, and using Intel x86 GNU/Linux and a default directory setup as an example, the file to edit might be

```
edit $~/.bashrc file and add following lines  
PATH=/usr/local/texlive/2011/bin/i386-linux:$PATH;  
export PATH  
  
MANPATH=/usr/local/texlive/2011/texmf/doc/man:$MANPATH;  
export MANPATH  
  
INFOPATH=/usr/local/texlive/2011/texmf/doc/info:$INFOPATH;  
export INFOPATH
```

**For 64bit OS**

```
edit $~/.bashrc file and add following lines  
PATH=/usr/local/texlive/2011/bin/x86_64-linux:$PATH;  
export PATH  
  
MANPATH=/usr/local/texlive/2011/texmf/doc/man:$MANPATH;  
export MANPATH  
  
INFOPATH=/usr/local/texlive/2011/texmf/doc/info:$INFOPATH;  
export INFOPATH
```

**Fedora/RedHat/CentOS:**

```
sudo yum install texlive  
sudo yum install psutils
```

**SUSE:**

```
sudo zypper install texlive
```

**Debian/Ubuntu:**

```
sudo apt-get install texlive texlive-latex-extra
```

```
sudo apt-get install psutils
```



# Appendix B

## Installing the CUED class file

$\text{\LaTeX}$ .cls files can be accessed system-wide when they are placed in the  $\langle\text{texmf}\rangle/\text{tex}/\text{latex}$  directory, where  $\langle\text{texmf}\rangle$  is the root directory of the user's  $\text{\TeX}$  installation. On systems that have a local texmf tree ( $\langle\text{texmflocal}\rangle$ ), which may be named "texmf-local" or "localtexmf", it may be advisable to install packages in  $\langle\text{texmflocal}\rangle$ , rather than  $\langle\text{texmf}\rangle$  as the contents of the former, unlike that of the latter, are preserved after the  $\text{\LaTeX}$  system is reinstalled and/or upgraded.

It is recommended that the user create a subdirectory  $\langle\text{texmf}\rangle/\text{tex}/\text{latex}/\text{CUED}$  for all CUED related  $\text{\LaTeX}$  class and package files. On some  $\text{\LaTeX}$  systems, the directory look-up tables will need to be refreshed after making additions or deletions to the system files. For  $\text{\TeX}{}_{\text{Live}}$  systems this is accomplished via executing "texhash" as root. MIK $\text{\TeX}$  users can run "initexmf -u" to accomplish the same thing.

Users not willing or able to install the files system-wide can install them in their personal directories, but will then have to provide the path (full or relative) in addition to the filename when referring to them in  $\text{\LaTeX}$ .

