Towards Scalable Video Analytics at the Edge

Nguyen Van Dien

Supervisor: Prof. Jaehyuk Choi

Prof. Eun-Seok Ryu

Department of IT Convergence Engineering

Gachon University

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

January 2021

Acknowledgements

First, I would like to express my gratitude and appreciation to my major advisor, Prof. 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 Prof. 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: Prof. Myung-Mook Han, Prof. Yong Ju Jung and Prof. Joohyung Lee 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 family in VietNam. I thank all my friends at Gachon and many others. Finally, a very special thanks to my wife, 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(VA) systems have play an important role in many areas, including public security, transportation safety, and many other industry fields, as automatically tools for data collecting and analyzing big datasets such as multiple live video streams transmitted from many cameras. An important characteristic of these systems is that it is critical and difficult to perform real-time analytics so as to provide timely event alerts on different tasks, activities and conditions. Due to the computation-limitation and bandwidth-limitation nature of these operations, however, video analytics servers may not meet the requirements when serving for a large number of cameras simultaneously. To overcome these challenges, this dissertation present an edge-computing based system that minimizes transfer of video data from the surveillance camera sources to a video analytics server on the cloud. Based on a novel solution 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 frames which will be forwarded to the cloud server. To demonstrate the effectiveness of the approach, we implemented a video surveillance system consisting of an edge device 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 48% load

of GPU, 48% load of CPU, and 51% of network traffic while still maintaining the accuracy of real-time video analytics event alerts.

Table of contents

List of figures

List of tables

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

Nowadays, the world is witnessing a very fast increment of camera deployment [**?**] as shown in Figure 1.1 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 [**?**], [**?**], [**?**]. According to IHS Markit's annual report [**?**], 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 come mainly from the growth of the video surveillance industry because of increasing the concerns about public security and safety. With this trend, the intelligent video analytics

(a) (b)

Fig. 1.1 Surveillance Camera Deployment and Application

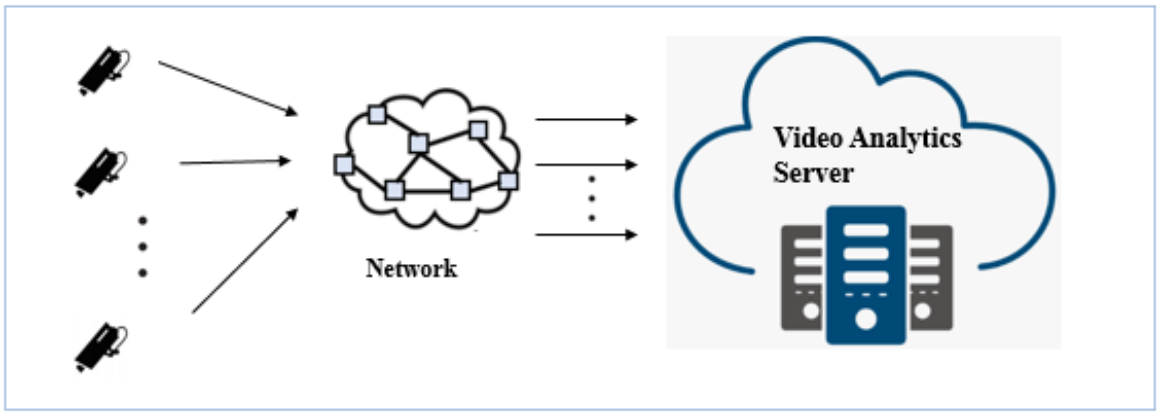


Fig. 1.2 Overall of cloud based video analytics server

systems have been playing an essential and important role, performing main tasks in various fields including surveillance area, transportation monitoring, manufacturing, etc. Video analytics software analyzes videos in order to detect events the system is programmed to look for – when something is moving in front of the camera.These systems analyze video sources to handle long-running tasks such as traffic monitoring, customer tracking, and surveillance. The important 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 [**?**], [**?**], [**?**].

In a typical real-time video analytics pipeline as Figure 1.2 illustrates a traditional cloud-based video stream analytics platform. Video analytics is performed in a a central cloud and a large number of camera transfer video data to there. However, this traditional solution makes it difficult to perform real-time video analytics on live video streams from many cameras because the VA involves several computation-consuming 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.

Many works has been conducted to improve the efficiency of video analytics platform [**?**], [**?**], [**?**], [**?**]. Accross these systems, a typical strategy is to improve efficiency by filtering

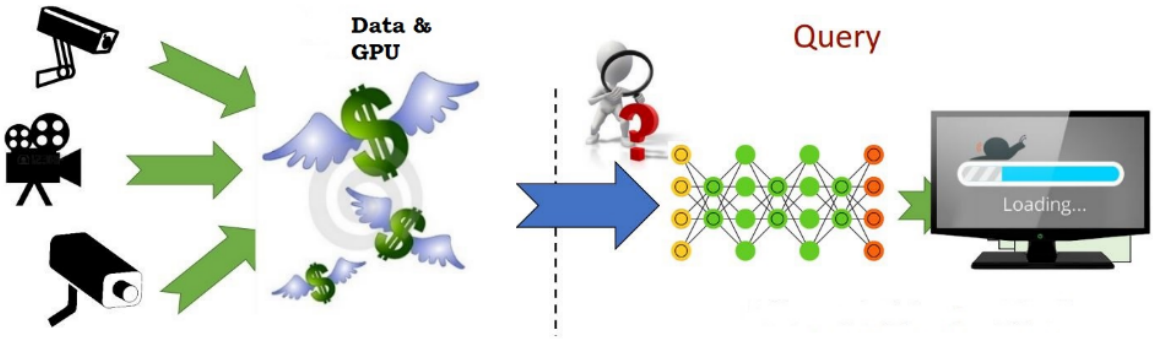


Fig. 1.3 Real-time Video Analytics Challenges

out frames that do not contain relevant information for the query using the additional edge devices. In this study, to address the question of how to address this network bottleneck and offload large volumes of data from a density 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. In previous works, the solutions are based on either:

- Binary classification model[**?**][**?**]: remove frames that do not contain any object of interest.
- Pixel-based background modeling[**?**]: eliminate frames which have not changed substantially.

Typically, exist systems use deep-learning framework to understand video frame context. Video frame is analyzed on pixel domain which require fully video decoding. However, edge devices are normally much less powerful than the cloud server, with limited computing resources such as a few CPUs (central processing units), GPUs (graphics processing units) and RAM (Random Access Memory) capacities [**?**]. In the field of public security, the ability to simultaneously process multiple camera sources and provide real-time video analytics is critical. Thus, our study seeks to overcome how edge devices and the cloud can cooperate in

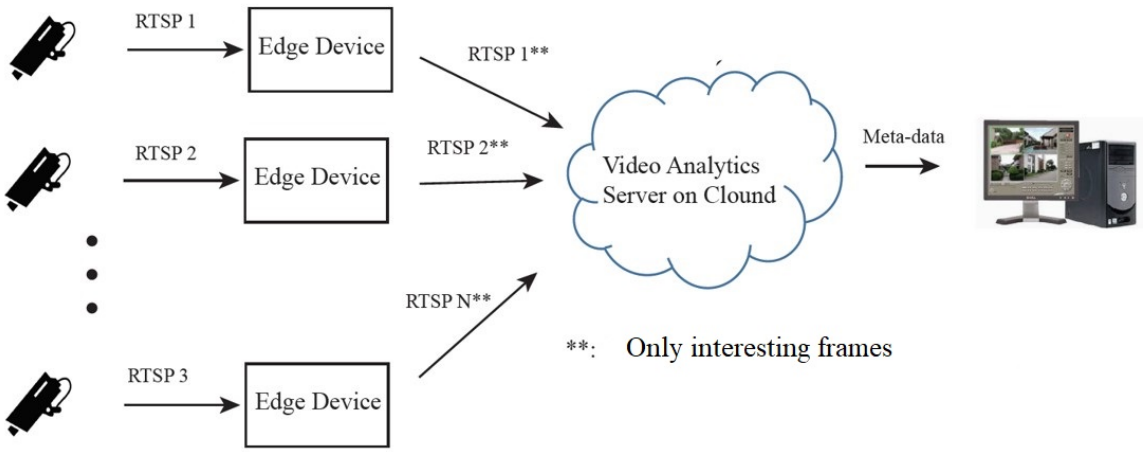


Fig. 1.4 The motivated system design

an efficient manner to achieve real-time and scalable video analytics. Our motivation is based on the observation that surveillance camera images are unchanged for a long time and video content is often unnecessary. So, it is undesirable to conduct analytics on redundant video frames from cameras, it will increase latency and system expense by a waste of network bandwidth, computing resources and power consumption in the cloud as described in Figure 1.3.

Motivated by this pupose, we present a pre-process module that acts as a video filter function at the edge device to remove out the redundant static images before forwarding video images to the cloud for further analysis as described in Figure 1.4. The proposed method is driven by the compressed-domain feature which doest not require GPU-based processing and fully video decoing. Therefore, it is light-weight method and cheap solution. The method performs on an edge device and recognizes the motions (i.e., moving objects) in the consecutive video frames. Depend on the motion recognition result, the edge device decides whether to pass the video frame to the cloud or filter it out. As a result, the proposed motion-based filtering edge device can reduce not only computational load of the cloud nodes but also network traffic to the cloud.

Motion recognition schemes can be devided into two categories: (i) video pixel-domain based

and (ii) video compressed-domain based approaches. In pixel-domain based approaches [**?**], [**?**], [**?**], [**?**] video is completely decoded and then applying background modelling or CNN based deep learning framework to detect moving objects at the pixel level. The performance of video pixel-domain processing in large systems may be very challenged by the computing load of decoding multiple streams and the image pixel-based calculation. Therefores, pixel-based solution requires higher computational complexity and can make it difficult to fulfill the real-time requirement of edge devices. Compressed-domain approaches [**?**],[**?**],[**?**],[**?**] 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.

- Utilizing Video Coding Motion Vectors (MV) analysis to filter out the static frames from video stream.
- Present a compressed-domain moving objects detection method that can be easily applied for numerous surveillance applications.

- Deploy a novel edge-to-cloud computing system for surveillance video analytics that applies the proposed method at edge devices to minimize the data transfer from surveillance camera feeds to the cloud.

- Implementation and evaluation of the proposed method on cheap edge device (Pi4) and achieve real-time processing time of 39 ms/frame with high definition resolution video.

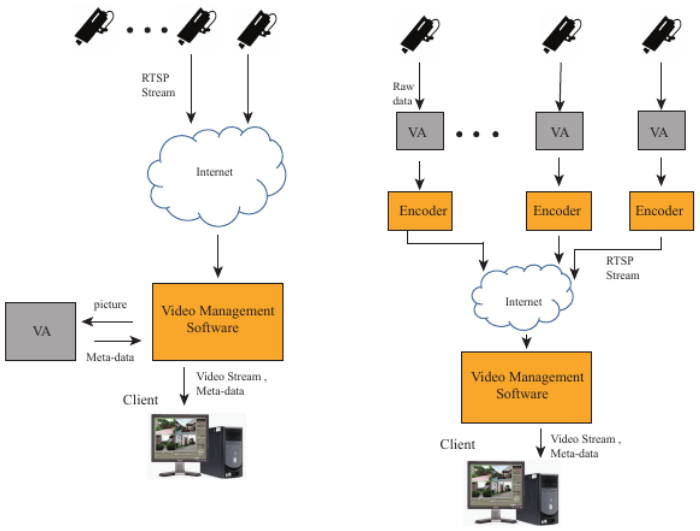
Chapter 2

Background

Object detection in images and videos has received a lot of attention of video analytics researchers in recent years. There are many different approaches to detect video moving object, both utilizing compressed and uncompressed video domain. In this chapter, we briefly introduce the current 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 normally built using the following main components: surveillance IP cameras, VMS, device storage and VA modules (optional). VA can be implemented in two main configurations, as discussed below.



(a) (b)

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

2.1.1 Edge Camera Based Deployment

In this approach, the VA is deployed through a camera device or video encoder [**?**] such as that shown in Figure 2.1b, which must have enough the processing power to run the VA functionality. Hence it is an expensive and challenging approach to wide-area video processing. This approach seems ideal, however it does not perform satisfactorily in many cases because of the limitations on the overall surveillance system design and performance. Most surveillance 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 congestion. In detail, running video analytics by forwarding all video to the cloud overloads with the bandwidth constraints of some deployments, which do uploading all camera data. Each camera's uplink bandwidth is limited, both by the physical constraints of network components such as: cable, switch, router.. 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 2 Mbits/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 to 40 Mbits/s. An edge-based filter answers this challenge with semantic filtering that uploads only frames that are relevant to applications.

- The video data quality analyzed by the VA server is usually degraded because of lossly compression and transmission effects, and therefore.

- 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 Full HD (1080P) stream at 30 FPS is $\approx$ 1.4 Gb/s when decompressed. Accomplishing VA at scale requires abundant compute, memory resources, so existing systems often perform this processing in the cloud, using GPUs.

However, this approach is independent with surveillance camera sources, 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 [**?**], high efficiency video coding (HEVC)[**?**] which has achieved

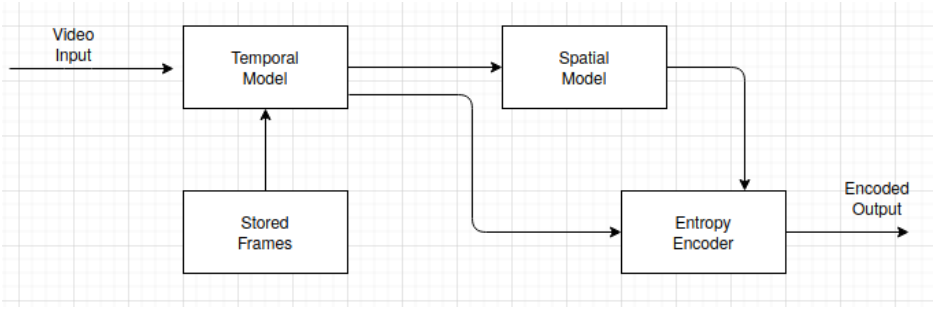


Fig. 2.2 Video encoder block diagram.

approximately twice the standard compression[**?**] 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 [**?**][**?**][**?**]. These approaches use edge-compute resources allocate with the cameras to identify the video sequences that are most relevant to datacenter applications and offloads only that data for further analysis. In this way, it supports to limit 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 previous 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 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 of future frames and is improved by compensating for differences between the frames (motion compressed prediction). The output of the temporal model is a residual frames (created by subtracting operation the prediction from the actual current frame) and a set of model parameters, typicially a set of MVs describing how the motion was compressed.

The residual frame formes 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 archieved 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 coefficiencents 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. The 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 sequency consists of coded motion vector parameters, coded residual coefficients and hear 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.

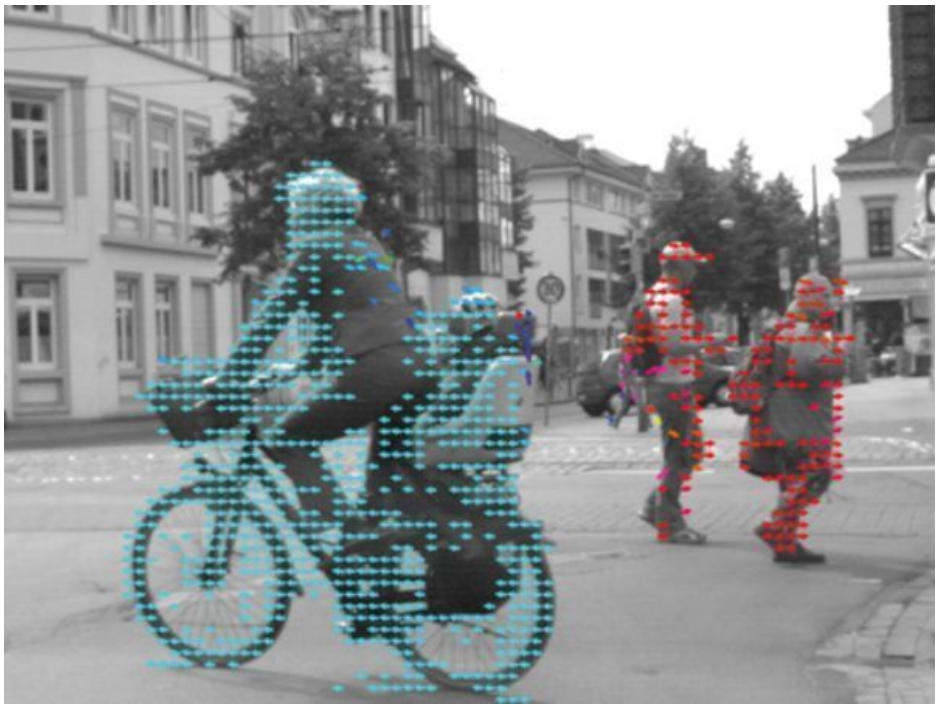


Fig. 2.3 Motion Vector in Video Codec.

Video Coding Motion Vectors and Blocked-based Motion Estimation

Changes between video frames may caused by object motion (regrid 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 coresspond 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 frame. 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

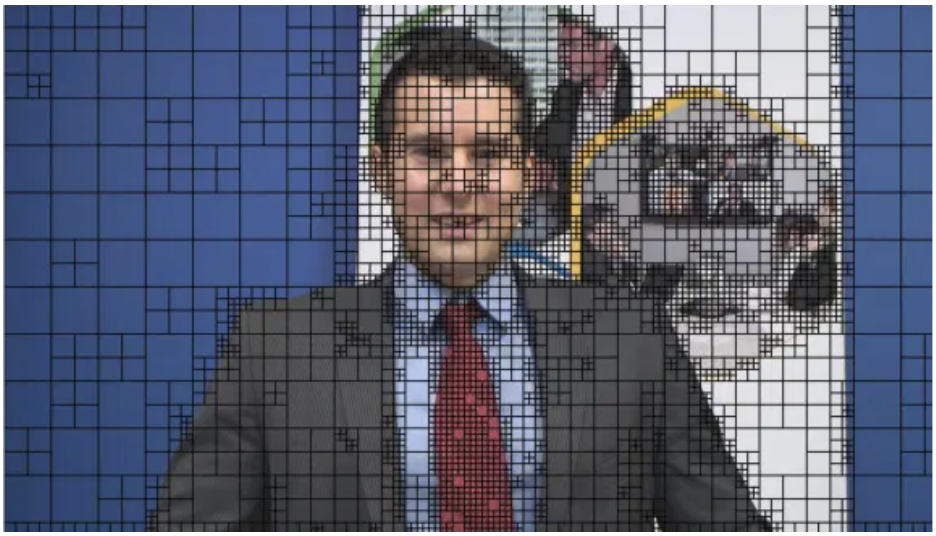


Fig. 2.4 Macroblock in Video Encoder.

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 angel 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 compensate for movement of rectangular sections or 'block' of the current frame. The current frame is sub-devided in MxN blocks called macroblocks as shown in Figure 2.4. The following procedure is carried out for each block MxN samples in the current frame:

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

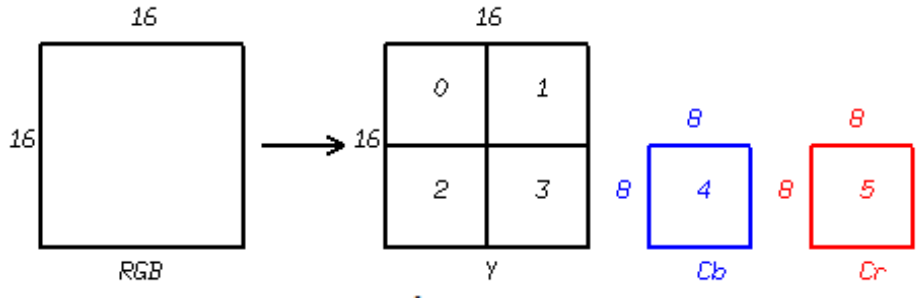


Fig. 2.5 Macroblock (4:2:0).

In H264/AVC standard, the macroblock, coressponding to a 16x16 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 16x16 pixel macroblock represented in YCbCr 4:2:0 format. A macroblock of 16x16 pixels can be divided into 8x8 sample blocks. In the RGB representation, a macroblock has a total of 3x4 = 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 [**?**], [**?**]. 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 [**?**]. 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

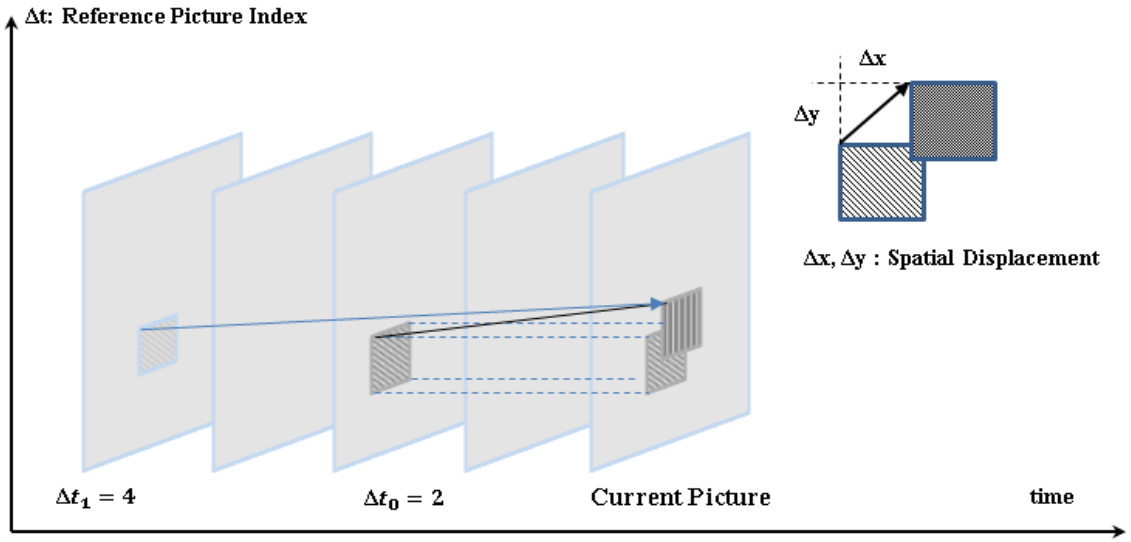


Fig. 2.6 Reference motion vectors in video coding.

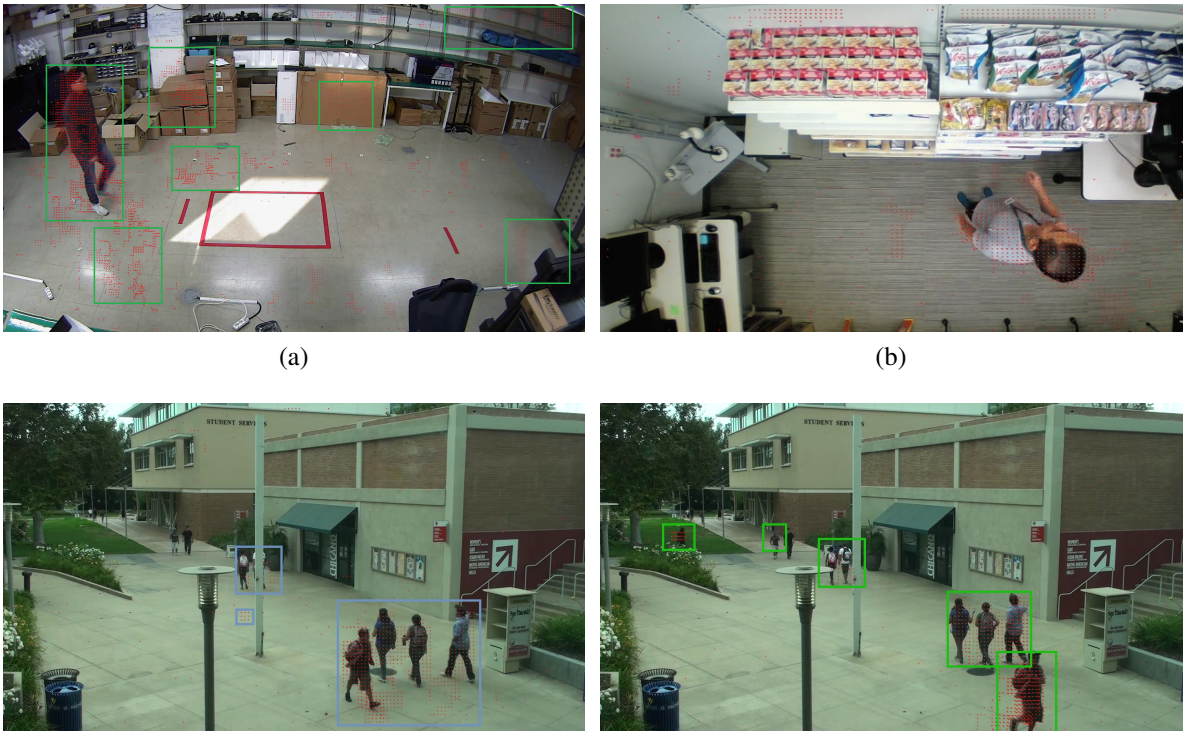
in neighboring blocks as predictor reduces the size of the signaled motion vector difference. For this block-based MCP, a video image is divided into rectangular blocks. Figure 2.6 shows the general concept of MCP based on a translational motion model. Using a translational motion model, the position of the block in a previously decoded picture is indicated by a motion vector $(\Delta x, \Delta y)$, where Δx specifies the horizontal and Δy the vertical displacement relative to the position of the current block. Note that the motion vectors Δx, Δy 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 devided 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 ultilize the predictive coding involved motion

vectors displacement.

2.2 Compressed-Domain Based Moving Object Detection Using Motion Vector

As mentioned in last section, MVs are obtained for each motion block between the current and referenece 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 [**?**],[**?**] that is based on video coding features such as MVs, macroblock partition, and quantization coefficients, have been proposed. In [**?**], the authors applied a probabilistic technique of computer vision for image separation, known as Graph Cut [**?**], 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, 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



(a) (b)

(c) (d)

Fig. 2.7 The example of MV extraction.(a) Test video sequence from recorded camera, (b, c, d) Test video sequence from VIRAT dataset.

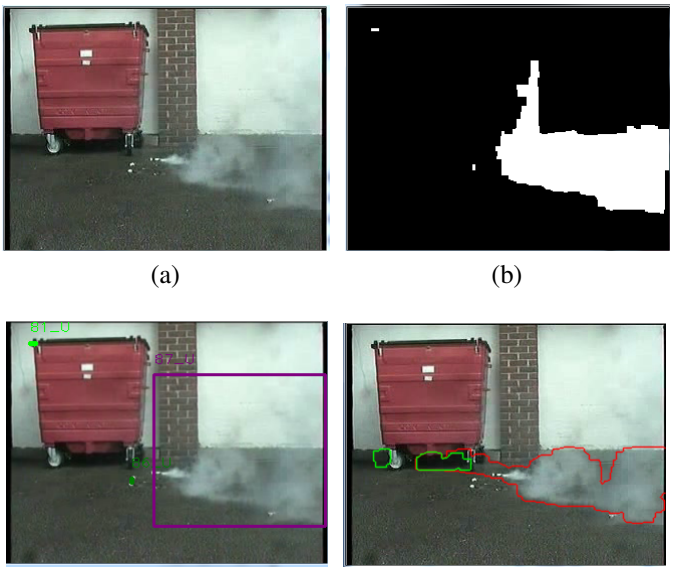
reduced compared to the correctly estimated MVs because noisy vectors are continuous and similar MVs from real moving objects. Another challenge of this approach is the lack of information about the object's appearance such as color, edges and texture, because these features would require complete decoding 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.

2.3 Pixels-Domain Based Moving Object Detection using Deep Learning

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 [**?**],[**?**],[**?**],[**?**],[**?**],[**?**]. There are two primary methods that are considered for moving object detection in pixels domain.

2.3.1 The combination of background subtraction and object classification based moving object detection

The background subtraction method [**?**][**?**] subtracts the current frame and background image to eliminate an image's background, and then detects moving targets based on gray value differences. This method is the most commonly used object detection technique. There are a number of methods that have been for building the background model, e.g., [**?**] proposed a novel real-time motion detection algorithm that integrates the temporal differencing method, double background filtering method, optical flow method, and morphological processing



(a) (b)

(c) (d)

Fig. 2.8 Smoke detection pipeline: (a) input frame, (b) the foreground subtraction, (c) the blob detection, (d) the smoke candidates classification

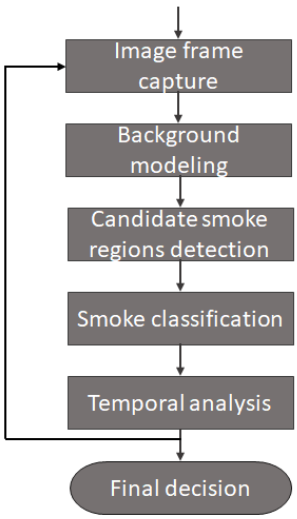


Fig. 2.9 The pipeline of video- based smoke detection algorithm.

methods to obtain excellent performance. Moreover, the authors of [**?**] 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 detection with the detail of flow-char is drawn in Figure 2.9. In this example, smoke classification was done by trained convolutional neural networks (CNN) model which was described in [**?**] for image classification. This a relatively new technique in machine learning, enables very accurate classification of images using CNNs. [**?**][**?**][**?**]. Figure 2.10 shows the original architecture of the trained model, first five layer are convolutional and some of them are followed by max pooling layer. The next are three fully-connected layer, the last fully-connected layer computes class score of trained class labels.

The capacity of CNNs can be changed 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 [**?**] achieved an average accuracy of 70.1% using the CIFAR-10 [**?**] dataset.

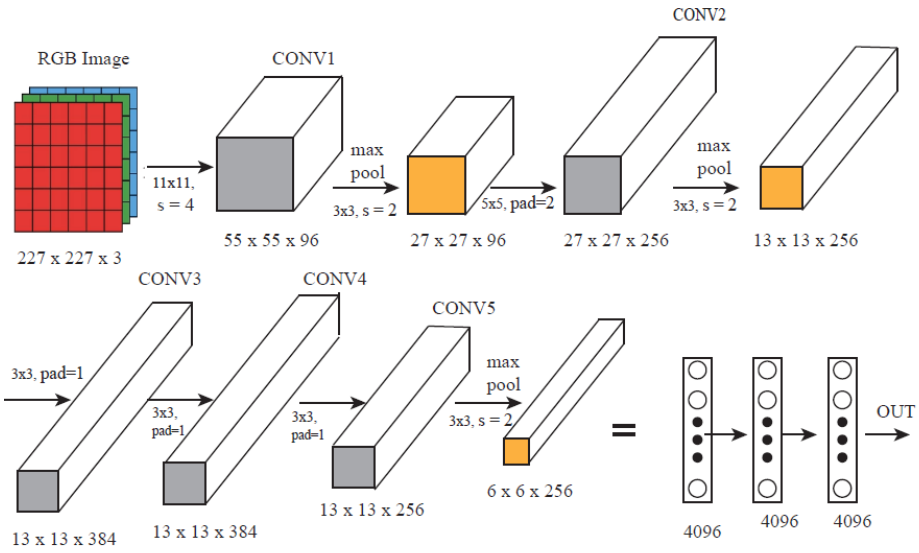


Fig. 2.10 Architecture of the Alexnet CNN model.

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

R-CNN [**?**] uses selective search [**?**] 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 2.11(a). The limitation of this method is that it requires a large amount of time to train and deploy networks

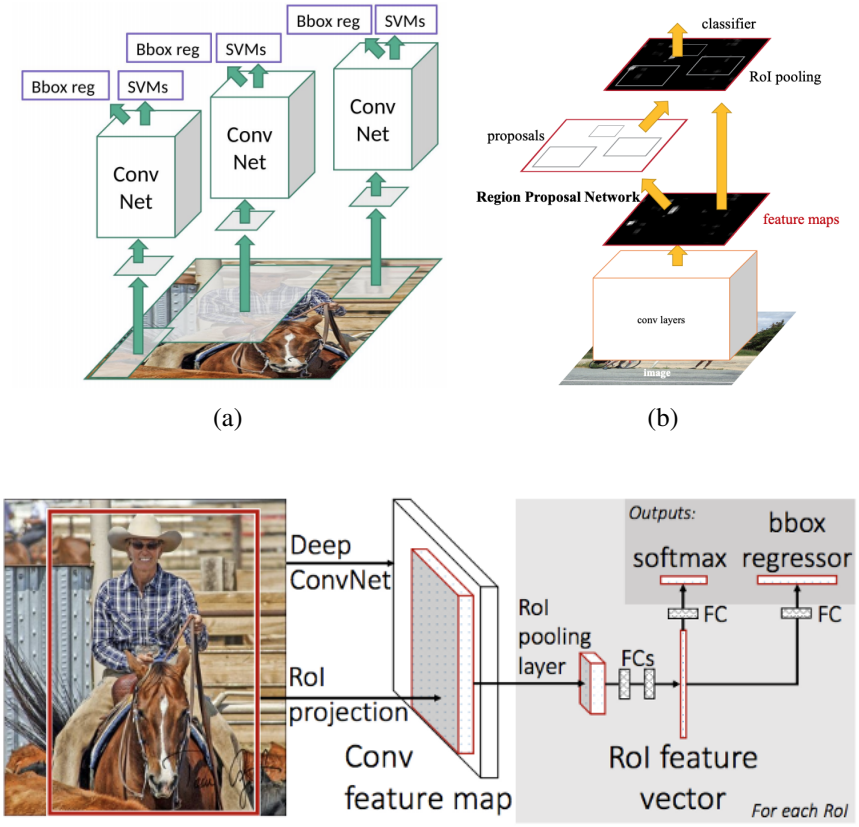


Fig. 2.11 CNN based Object Detection Framework Structure.(a) RCNN, (b) Faster-RCNN, (c)Fast-RCNN.

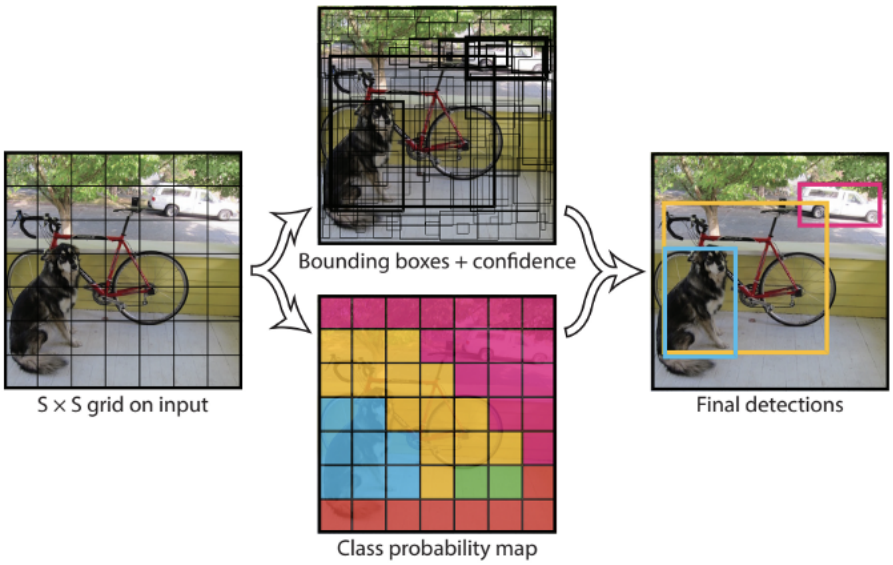


Fig. 2.12 YOLO Structer.

because it has to classify multiple region proposals per image. Fast R-CNN [**?**] 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 2.11(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 [**?**] was proposed to allow the network to learn the region proposals as shown in Figure 2.11(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 [**?**] 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 2.12. Specially, YOLOv3 network architecture has 24 convolutional layers, followed by two fully connected layers as shown in Figure 2.13. 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 [**?**], 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 [**?**] were released as a balance between these two methods. Compared to YOLO, an SSD runs an input image through a convolutional network only once and computes a feature map. Then, a small 3×3 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 [**?**], the SSDs attain similar accuracy but run slower. In this study, YOLOv3 was applied to robust human detection at cloud servers for our implementation.

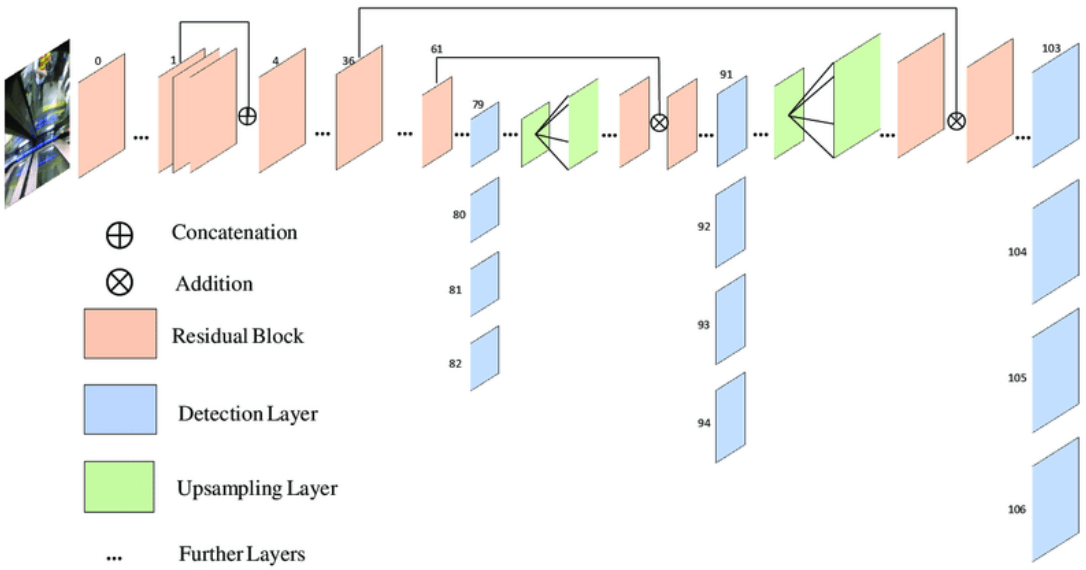


Fig. 2.13 YOLOv3 Network Model.

Chapter 3

Methodology

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

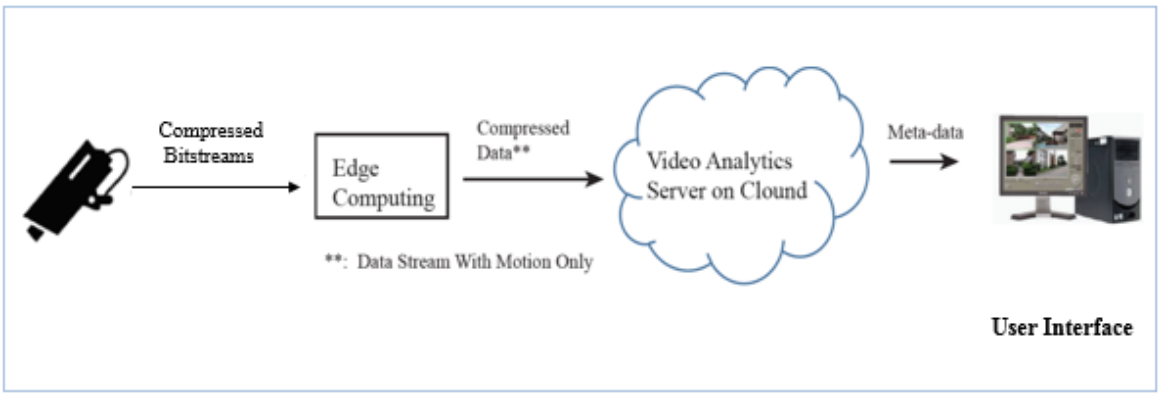


Fig. 3.1 Overview of our proposed edge-to-cloud system.

3.1 The Edge-to-clound Model for Surveillance 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

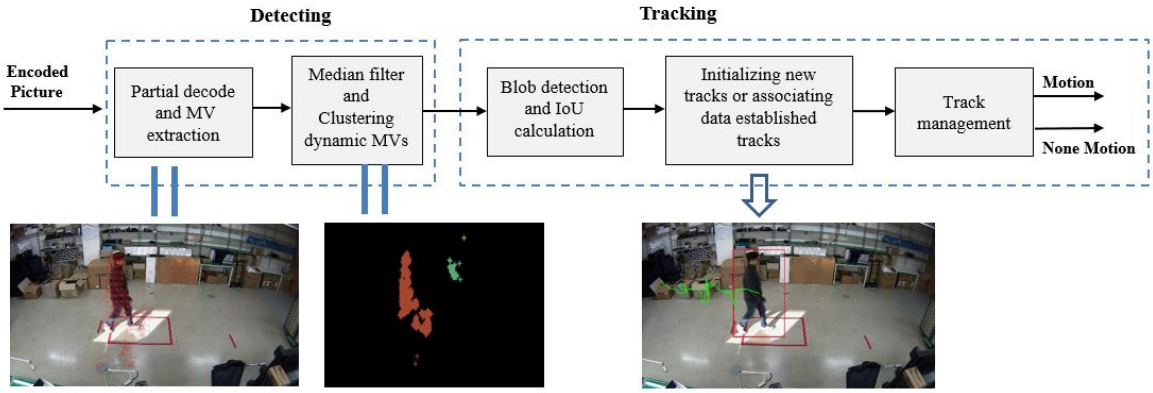


Fig. 3.2 Workflow of the proposed method of moving object detection in compressed domain.

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

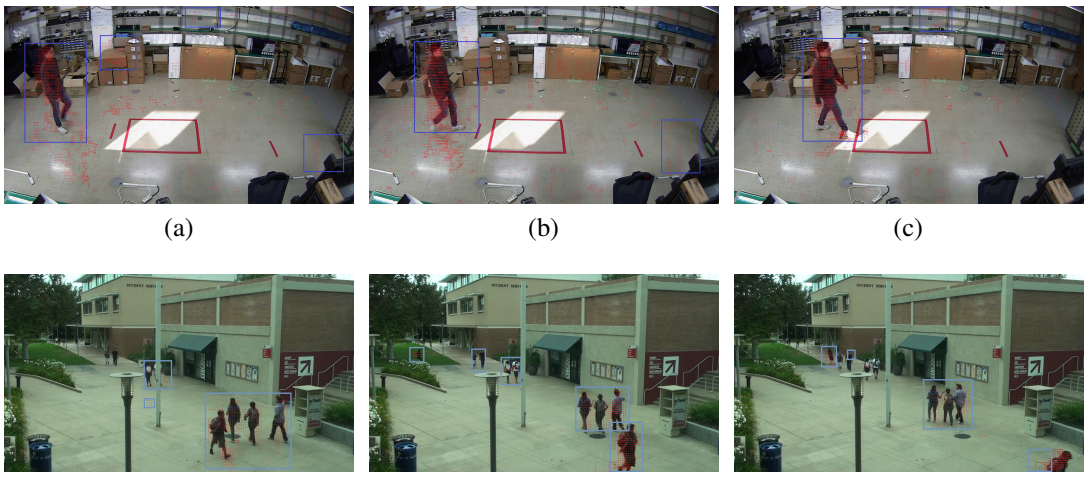
- Cloud Server: cloud server collects the preprocessing results from edge nodes, which has abundant computational abilities, and performs additional video analysis.

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 Proposed Method of Moving Object Detection in Video Compressed Domain

MV represents the changes between video frames may caused by:

- Object motion (car, human activity)



(a) (b) (c)

(d) (e) (f)

Fig. 3.3 Video coding MV extraction from consecutive frames with two different video test sequences.

- Camera motion (panning, tilt, zoom, rotation).
- Lighting condition changes

The MV group generated by lighting change, usually randomly occur without following a certain flow and have various sizes. The MV group generated by human motion (walking, running), move in certain direction and It's size is similar with object size in video frame. This observation is good factor to classify MVs into real-motion and noise motion group. 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 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.

(a) (b) (c) (d)

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

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 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:

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

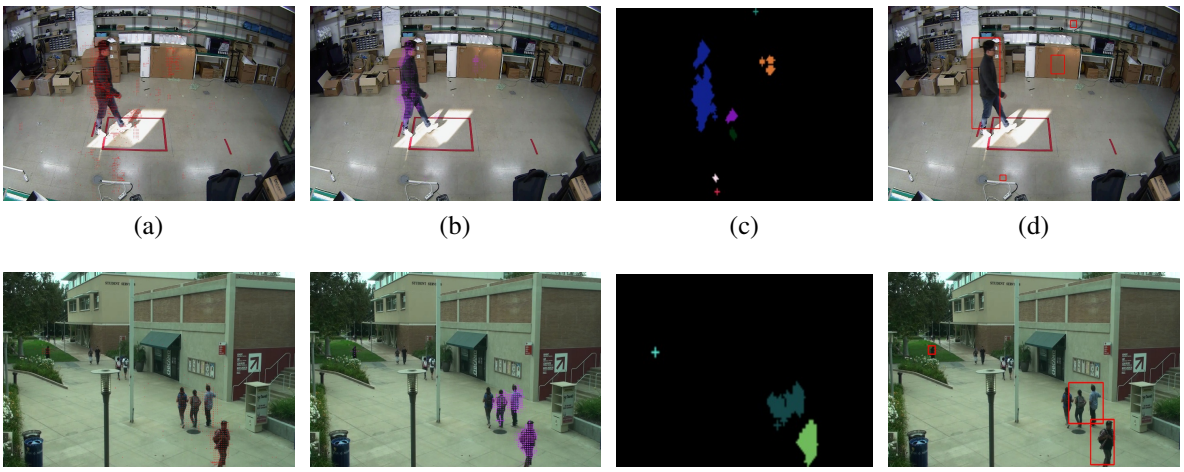
Where: MV_{ij} is the MV elements in the N×N window. Finally, the smoothened motion filter is presented, which is experimentally determined as shown in Figure 3.5(b,f) using the input MV,

including isolated motion vectors noise, are smoothened 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 ε where ε 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 $\mathbb{C}$ so as to minimize the potential Equation 3.2:

$$\phi = \sum_{x \in \chi} \min_{c \in \mathbb{C}} \|x - c\|^2 \tag{3.2}$$

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 ε.
- Cluster all the MV points that are density reachable or density connecting [**?**] and label them.
- Terminate the process after all the MV points are checked. The output is a set of clusters of dynamic points.



(a) (b) (c) (d)

(e) (f) (g) (h)

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,f) Blob detection.

The distance threshold ε decides the range of density reachability [**?**]. After clustering, the label list includes all candidate objects with various size, represented by different colors. Typically, the size of candidate object is followed by object size in the video frame. In small object dataset, objects are small when they have mean relative overlap (the overlap area between bounding box area and the image is) from 0.08% to 0.58%. Thus, we used a cluster threshold size to filter out the small clustered to the candidate lists as shown in Figure 3.6.

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 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

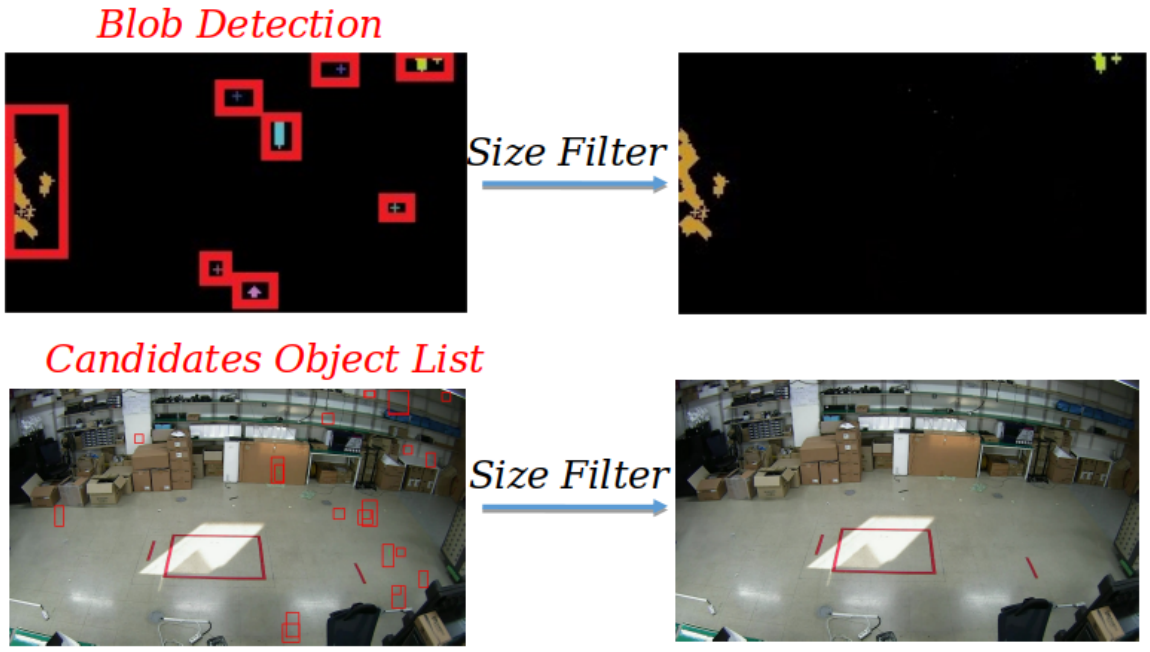


Fig. 3.6 Motion size based Filtering

tracking algotithm will also be applied to derive the moving object's bounding box in I-frame based on the last states in P-frames. Because the real object moves with a certain flow, the tracking motion's trajectories length is then used to classify the candidate motions into the real motion or noise group, 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 [**?**], [**?**].

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-

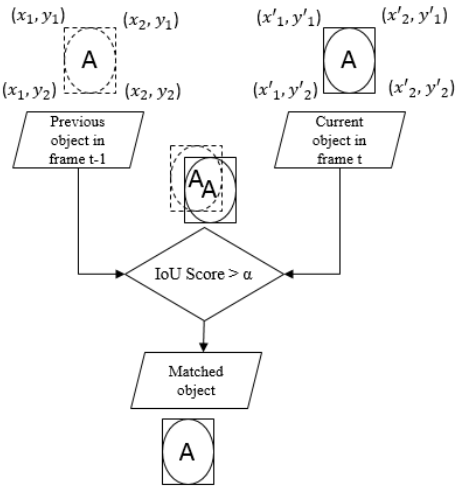


Fig. 3.7 Object matching method

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:

$$IoU(B_1 \bigcap B_2) = \frac{\|B_1 \bigcap B_2\|}{\|B_1 \bigcup B_2\|} = \frac{\|B_1 \bigcap B_2\|}{\||B_1| + |B_2| - B_1 \bigcap B_2\|} \tag{3.3}$$

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 (α), two bounding boxes are considered in same account (Figure 3.7). 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 : $Area_1 = (x_2 - x_1) \times (y_1 - x_2)$
- The area of second bounding box : $Area_2 = (x'_2 - x'_1) \times (y'_1 - x'_2)$
- 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<= 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:

$$Comp_c = S \tag{3.4}$$

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

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

The performance ratio of two method is:

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

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

$$\frac{Comp_p}{Comp_c} \approx 1. \tag{3.7}$$

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

$$\frac{Comp_p}{Comp_c} = \frac{K}{N} \tag{3.8}$$

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:

$$\frac{Comp_p}{Comp_c} \approx \frac{K}{N} \tag{3.9}$$

Chapter 4

Implementation And Performance Evaluation

In this section, the implementation and performance evaluation of the edge-to-clound 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.

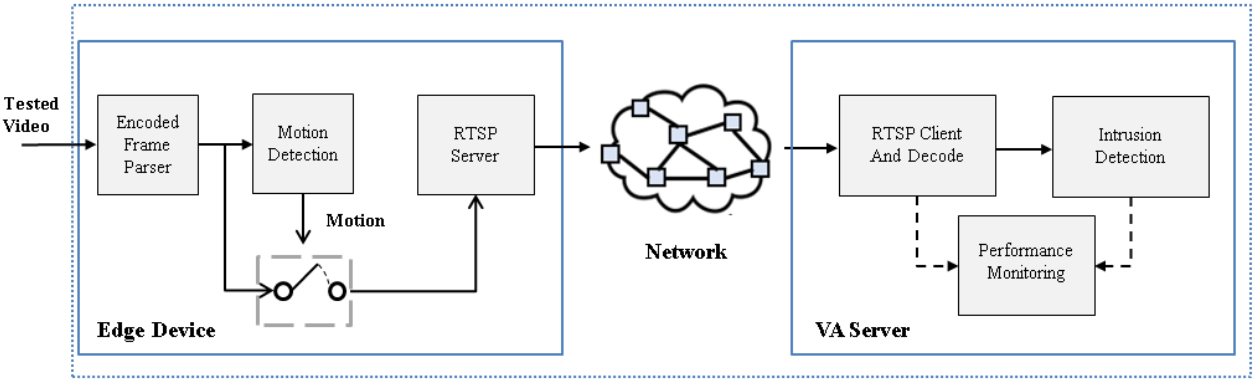


Fig. 4.1 Overview of System Design

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 c
CPU	Quad-core ARM Cortext-A72	Quad-core ARM Cortext-A57 MPCore processor
RAM	4 GB	8 GB

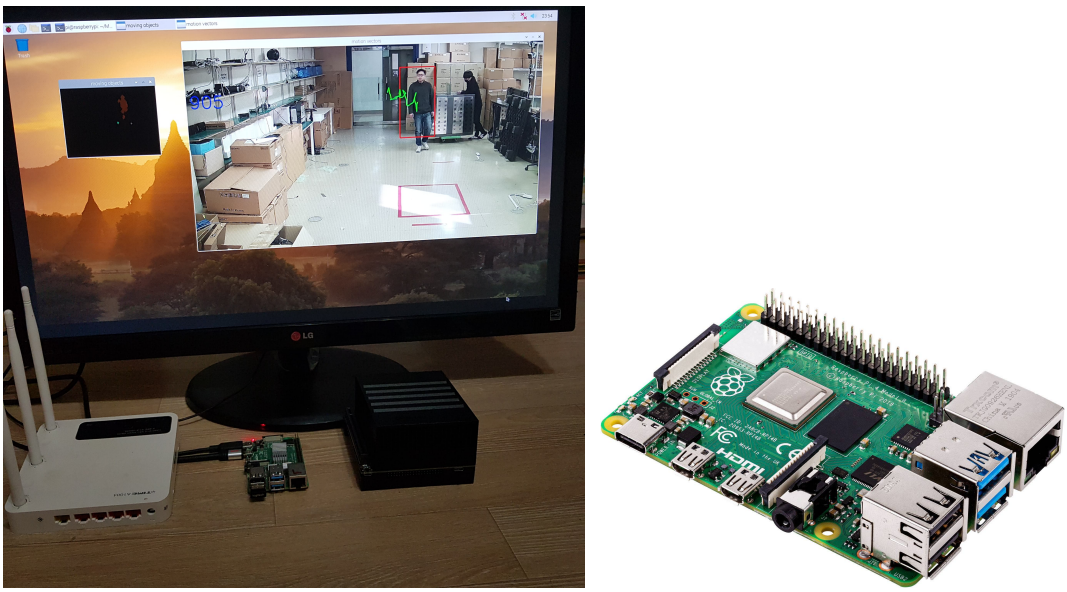
- Cloud node implementation: Receiving the forwarded encoded frame with the motion from the edge node over the network and then decoding and placing the output images into the intrusion detection module, which uses YOLO to detect humans.

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 Squence.

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



(a) (b)

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

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

Time	Duration (Seconds)
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

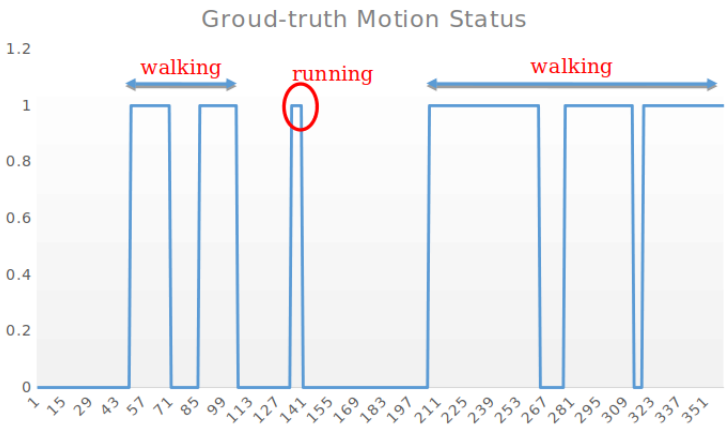


Fig. 4.3 The ground-truth motion time of our video test sequence.

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 [**?**] 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 [**?**]. The details of our video test sequence and the ground-truth motion time are shown in Figure 4.3 and listed in Tables 4.3,4.4.

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 mettric 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 4.4.

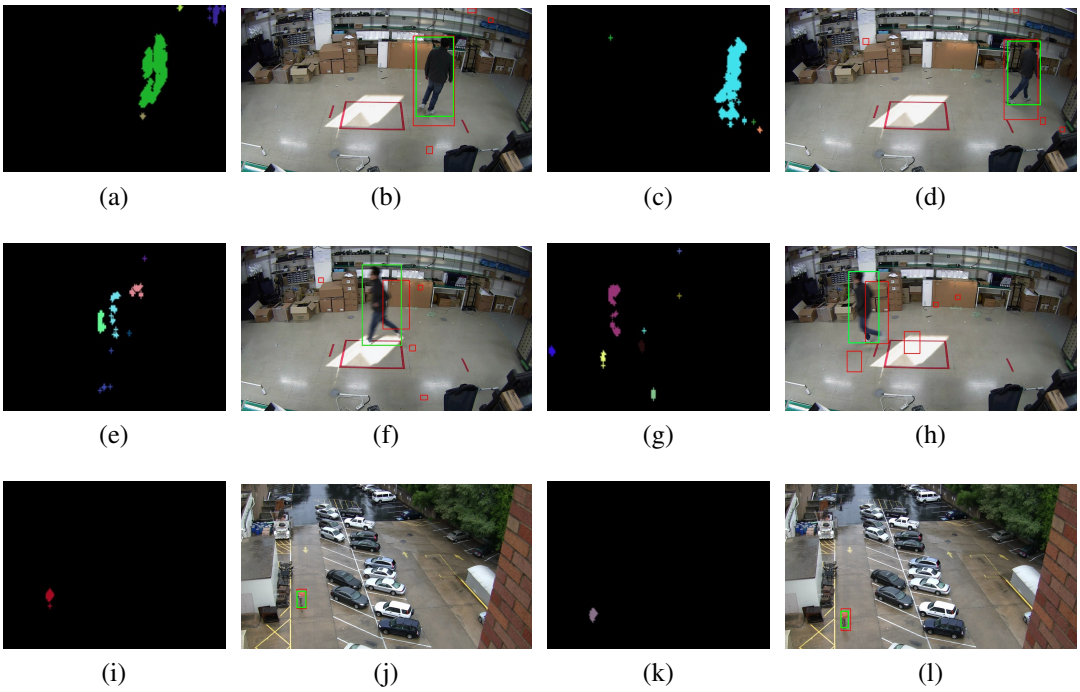


Fig. 4.4 Moving objects detection by the proposed method in different scenarios: (a, b, c, d) human walking; (e, f, g, h) human running; (i, j, k, l) test with a far distance of camera.

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)	**39 ms (26 FPS)**
1920x1080	N/A	N/A	**69 ms (14 FPS)**

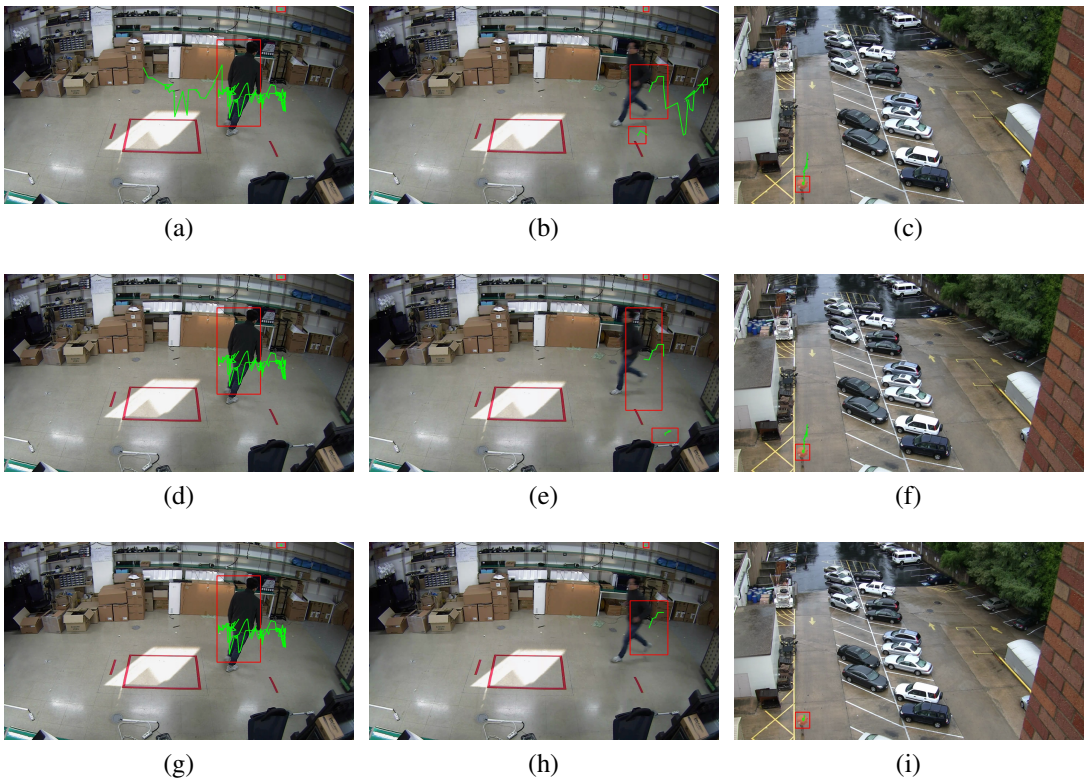


Fig. 4.5 Moving objects tracking by the proposed method with different α thresholds: α = 0.25 in (a, b, c), α = 0.4 in (d, e, f) and α = 0.5 in (d, e, f).

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 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 α of 0.25, 0.4 and 0.5 with the results are shown in Figure 4.5. In this experiment, each α threshold was applied for three same scenarios of human walking in Figure 4.5 (a, d, g), human running in Figure 4.5 (b, e, h) and the far camera distance in Figure 4.5 (c, f, i). We see that with each

test scenario, with lower α threshold value,the tracking object capability is better with longer the object trajactories. However, it will increase the number of false alarm detection if the noise MVs apprear 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 [**?**] [**?**]. The average comsuming time is messasured 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-clound platform will be implemented and evaluated. Our recored video test sequence is selected because of including different the moving object speeds. Base on the last experiment result shown in Table 4.4, we choose α is 0.25 to cover both walking and running case and compared computing resources with the conventional method (no frame filter). The entire demonstration was recorded and uploaded [**?**]. 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 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 1.2. The

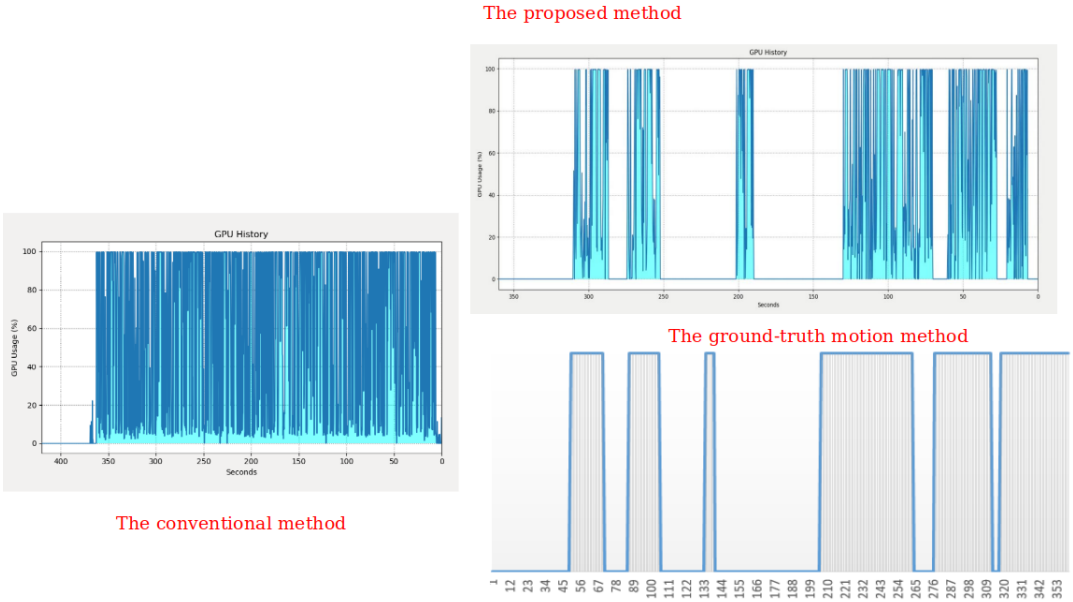


Fig. 4.6 GPU Monitoring with the conventional method and the proposed method ($\alpha = 0.25$).

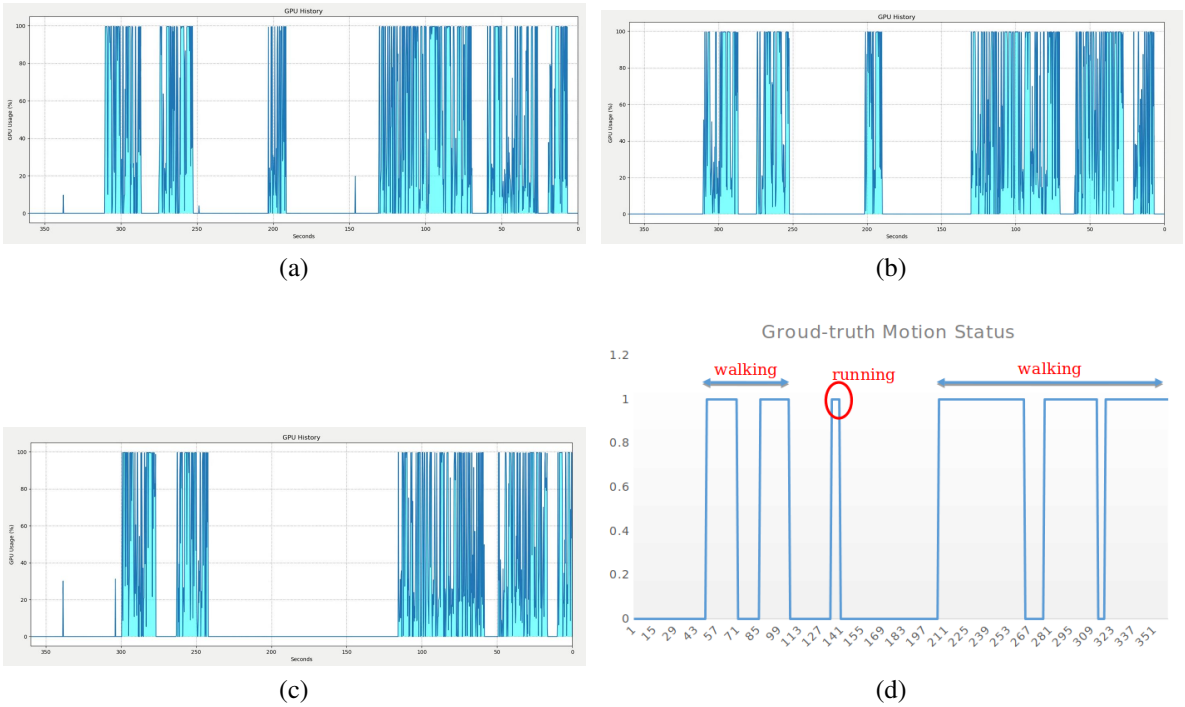


Fig. 4.7 GPU Monitoring: (a) $\alpha = 0.5$, (b) $\alpha = 0.25$, (c) $\alpha = 0.75$, (d) Ground Truth Motion Chart

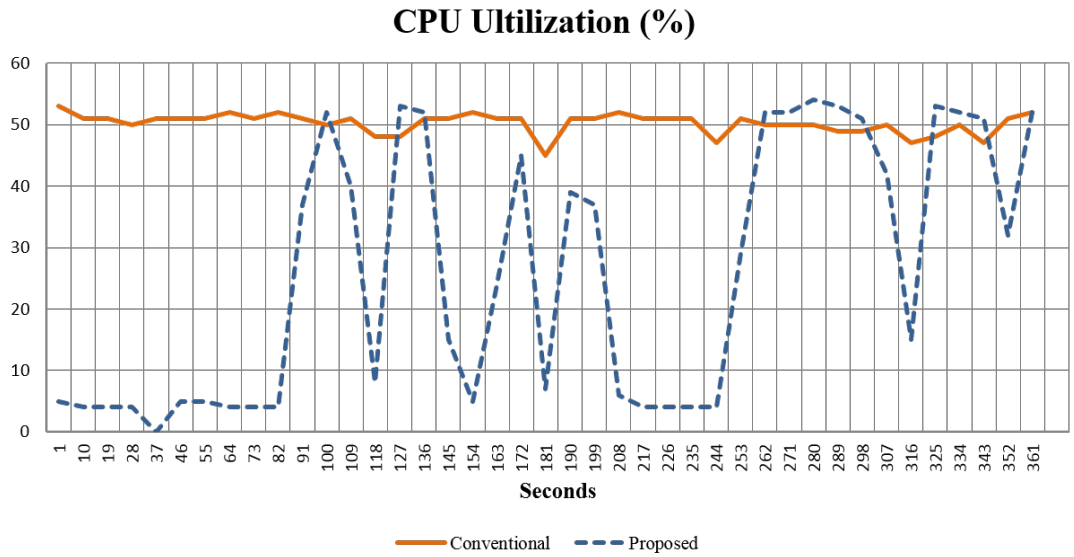


Fig. 4.8 CPU Monitoring with both the conventional method and the proposed method.

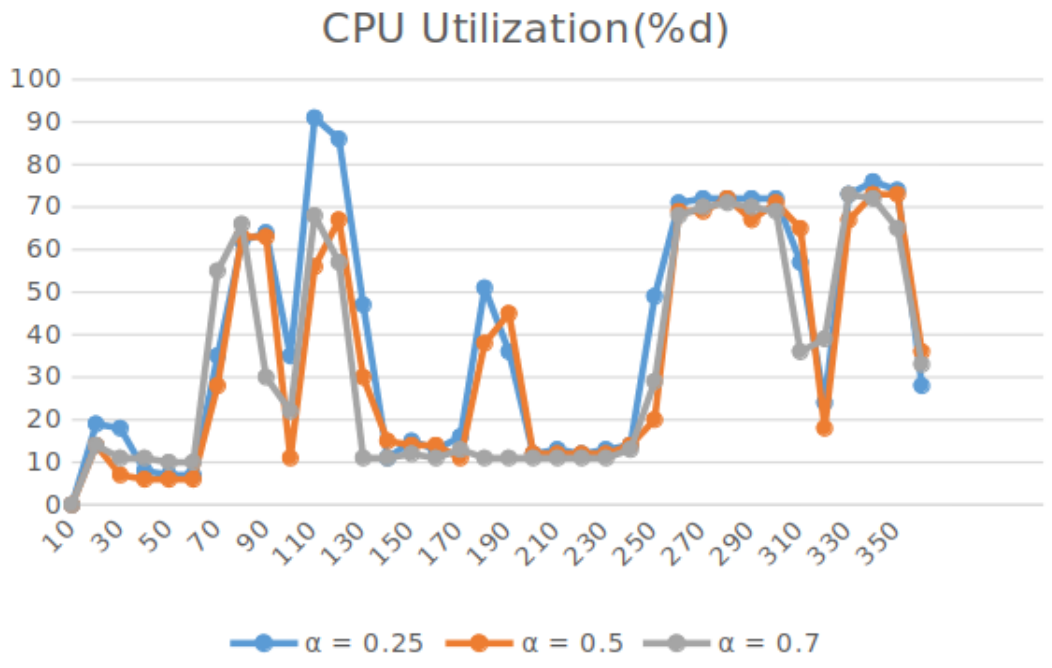


Fig. 4.9 CPU Ultilization with different alpha threshold.

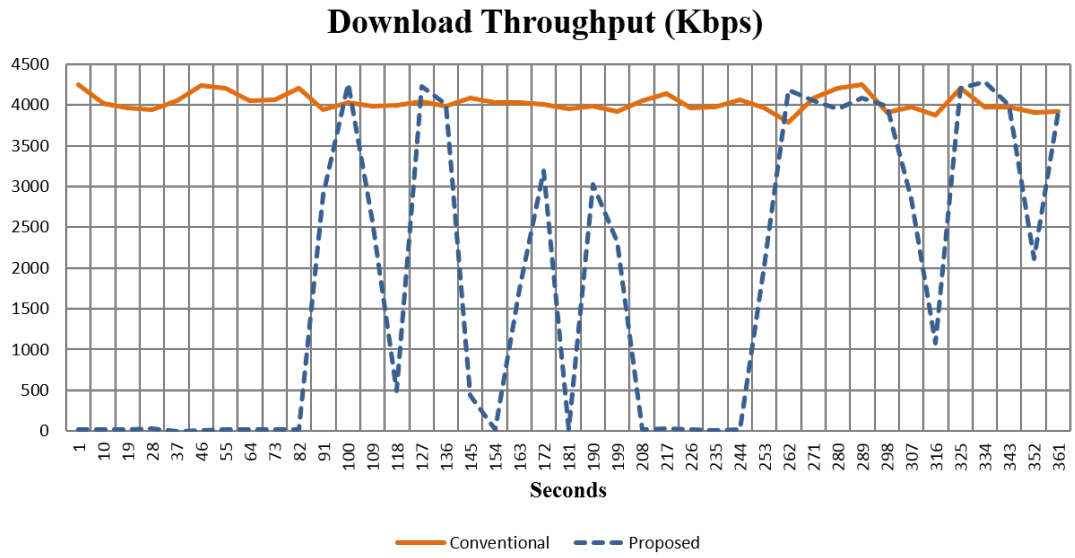


Fig. 4.10 Network download throughput monitoring with both the conventional method and the proposed method.

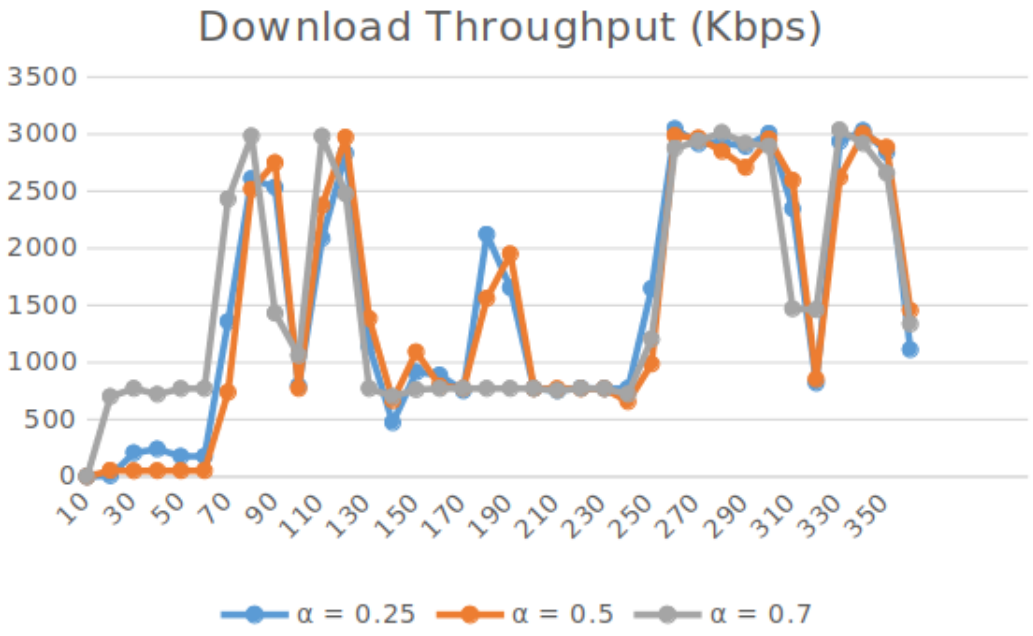


Fig. 4.11 Download Throughput with different alpha threshold.

Table 4.6 Average computing resources of both the conventional method and the proposed method with α=0.25

Computing Resources	**Conventional Method**	**Proposed Method**	**Performance Ratio**
GPU Utilization (%)	65.61	30	0.45
CPU Utilization (%)	50.24	39.64	0.51
Download Throughput (Kbps)	4028	1509	0.35

Table 4.7 Average computing resources of both the conventional method and the proposed method with α=0.5

Computing Resources	**Conventional Method**	**Proposed Method**	**Performance Ratio**
GPU Utilization (%)	65.61	22.3	0.34
CPU Utilization (%)	50.24	34	0.45
Download Throughput (Kbps)	4028	1478	0.36

Table 4.8 Average computing resources of both the conventional method and the proposed method with α=0.7

Computing Resources	**Conventional Method**	**Proposed Method**	**Performance Ratio**
GPU Utilization (%)	65.61	33.73	0.33
CPU Utilization (%)	50.24	25.9	0.43
Download Throughput (Kbps)	4028	1808.2	0.35

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 [**?**], [**?**]. In terms of consumption of computational resources, GPU, CPU utilizations, and download throughput of both methods are measured. In figure 4.6, We measured the GPU computing resource of both the conventional method and proposed method. In conventional method, all frame is pushed into by deep learning model to detect object. So GPU was always at high loading status during video time. In contrast, the proposed method filters all static frame so GPU load status will change according to moving object status of the video. Compare the ground-truth motion chart and GPU load consumption chart, we can see the similarity between them. We also test the same experiment with other threshold alpha to evaluate the change of GPU load consumption graph. The results was shown in Figure 4.7. We found that when alpha value is big ($\alpha = 0.7$), the moving object at high speed will be missed. Because the IoU score is small when object move fast. Similar with GPU load consumption, CPU load and Download throughput have same story as presented in Figure 4.8 and 4.10 . With conventional method always be high consumption load and the proposed method, consumption load is varied following the ground- truth motion chart. And, alpha value have same effect to CPU and download throughput. When alpha is small, it can detect well for moving object at different speed (running and walking), however can be included false alarm. When alpha is too big value, some fast moving motion was missed detection. 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 disappear-

ance of motion in ground-truth table.

In order to evaluate the efficiency of the proposed method, we calculate the performance ratio of the conventional method and the proposed method. 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:

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

Compared with the performance ratio, which is calculated in real scenario test in Table 4.6, 4.7, 4.8, our performance theory model and real measurement are matching and reasonable.