January 2022

Design Of Computer Vision Systems For Optimizing The Threat Detection Accuracy

Sina Gholamnejad Davani
Wayne State University

DESIGN OF COMPUTER VISION SYSTEMS FOR OPTIMIZING THE THREAT DETECTION ACCURACY

by

SINA GHOLAMNEJAD DAVANI

DISSERTATION

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

2022

MAJOR: COMPUTER ENGINEERING

Approved By:

_____

Advisor Date

_____

_____

_____

_____

DEDICATION

To all those who have been supportive of me in achieving my goals, especially my parents.

ACKNOWLEDGEMENTS

This research could not have been possible without the support of many people. I would like to show my sincere appreciation for my adviser Dr. Nabil Sarhan, whose guidance and directions were extremely helpful during my research. He always walked me through the extra mile and shared with me his knowledge and experience. I also would like to thank my committee members Dr. Mohammad Alhawari, Dr. Ekrem Murat, and Dr. Le Wang for their precious feedback.

I would like to extend my heartfelt gratitude to my parents who keep encouraging and supporting me on this track.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

1.1 Overview

Computer Vision (CV) is a science aimed at electronically perceiving and comprehending an image or a sequence of images (i.e., a video) [64]. The detection, recognition, and tracking of objects and events are common CV applications [71]. CV has been used in automated video surveillance [5, 40], Wireless Video Sensor Networks (WVSN) [35, 56, 10, 22, 57], mobile surveillance systems [41], Advanced Driving Assistance Systems (ADAS) [30], Vehicle-to-Vehicle/Vehicle-to-Infrastructure (V2V/V2I) video communication [75], traffic monitoring systems, and other Intelligent Transportation Systems (ITS) [42, 29]. According to Omdia [1], global CV market revenue is expected to grow across many use cases in different industries, increasing from $2.9bn in 2018 to $33.5bn by 2025. In 2021, 770 million video surveillance cameras were installed worldwide, according to Comparitech [2]. CV systems, including automated video surveillance, enable the real-time detection of threats by running CV algorithms as opposed to human observation.

This dissertation considers CV systems in which a central monitoring station receives and analyzes the video streams captured and delivered wirelessly by multiple cameras. It addresses how the bandwidth can be allocated to various cameras by presenting a cross-layer solution that optimizes the overall detection or recognition accuracy. In further contrast with prior work, it presents and develops a real CV system and subsequently provides a detailed experimental analysis of cross-layer optimization. Other unique features of the developed solution include employing the popular HTTP streaming approach, utilizing homogeneous cameras as well as heterogeneous ones with varying capabilities and limitations, and developing a new algorithm for estimating the effective medium airtime. The results show that the proposed solution significantly improves the accuracy of CV.

Additionally, this dissertation presents an enhanced object detection neural network system, called Enhanced YOLO, based on You Only Look Once Version 4 (YOLOv4) [17], which is one of the best per-

formers in object detection. By considering inherent video characteristics and employing different motion detection and clustering algorithms, the proposed system focuses on the areas of importance in consecutive video frames, thereby enabling the system to distribute the detection task dynamically and efficiently among multiple deployments of object detection neural networks. Our extensive experimental results show that the proposed solution is capable of providing improvements in mean average precision (mAP), execution time, and required data transmissions to the object detector networks.

Finally, as detecting an activity provides significant automation opportunities in CV systems, we present an efficient activity detection recurrent neural network (RNN) utilizing the object detection results of the Enhanced YOLO solution and taking advantage of fast pose estimation solutions. By combining generated object detection and pose estimation results, the domain of the activity detection problem moves from a volume of red, green, and blue (RGB) pixel values to a time series of relatively small one-dimensional arrays. This allows the activity detection solution to take advantage of competent neural networks that have already been trained for object detection and pose estimation over thousands of hours on large GPU clusters. As a result, activity detection solutions can be created with substantially fewer additional training sets and training processing hours.

1.2 Main Challenges

Most research on CV has focused primarily on developing CV algorithms [17, 58, 25, 15], with far fewer studies considering the design of CV systems. Bandwidth allocation, an important step in the CV system design process, has been generally addressed in many-to-one video streaming systems by only a few studies through cross-layer optimization [37, 7, 6]. However, these studies were all simulation-based. The objective in [37, 7, 6] was to optimize video distortion. Although distortion may be appropriate for certain systems, the detection/recognition accuracy is the most important metric in CV systems. The authors of study [7] considered the accuracy but only for simple face detection tasks. In addition, the aforementioned

studies assumed real-time transfer protocol (RTP) streaming instead of HTTP streaming, and none used H.264 encoding; they assumed MJPEG [7, 6] or abstract video data [37].

Object detection neural networks form a major portion of CV algorithms. YOLO [17] and ResNet [33] could be considered as two major members of these networks. Studies on CV algorithms have mostly focused on image object detection (considering every input image independently) but have avoided opportunities to research video (multiple sequential images/frames) design space [17, 34]. Image object detection frameworks like YOLO and ResNet are not optimized to be incorporated in video object detection tasks.

Studies on activity detection in videos have mostly considered a design that ignores the advantage of combining object detection neural networks and RNN [73, 19]. By not taking advantage of object detection results and designing the solution from scratch, these studies are prone to complex and long training procedures. There are many effective neural networks designed and trained for thousands of hours on enterprise-level hardware that could extract meaningful features from images. When the neural network is designed from scratch, it will be required to be trained on large datasets for long hours before it can be useful in any capacity as it has to learn how to extract features from input data (raw volume of RGB pixel values in this case).

1.3 Main Research Objectives

In this dissertation, we seek to address the aforementioned challenges. The main research objectives can be summarized as follows:

1. To optimize the bandwidth allocation to different wireless PTZ cameras in order to enhance the face detection and recognition objectives.

2. To develop an enhanced object detection solution using motion detection and clustering algorithms to reduce the computational resource requirements and enhance the detection results by incorporating ROIs with higher resolutions.

3. To develop an RNN network to classify an activity based on the sequential detection results from the object detection neural networks and the pose/limbs estimation results.

1.4 Detailed Research Plan

1.4.1 Optimal Bandwidth Allocation in CV Systems

This dissertation addresses the bandwidth allocation problem in *many-to-one* CV systems, which is a primary step in the design process of a CV system. As illustrated in Figure 1.1, the considered system consists of multiple video cameras capturing and delivering live video streams to a central monitoring station over a single-hop Wi-Fi LAN. Such multiple cells can be utilized to construct a larger system. The monitoring station runs CV algorithms to detect potential threats in the monitored site. This station is generally connected to the access point with a high-bandwidth wired link that is not deemed as a bottleneck. The main challenge in the considered system is the limited available network bandwidth, which should be estimated accurately and then distributed efficiently among various cameras.

In contrast with prior work, we build a real CV system for automated video surveillance to run actual experiments. We also consider both face detection and face recognition applications. In further contrast to prior studies, the system employs HTTP streaming and homogeneous cameras as well as heterogeneous ones with varying capabilities (including resolutions and frame rates) to mimic varying deployment scenarios. We primarily use H.264 but consider the co-existence of other encoders. We develop the system utilizing a variety of open-source libraries, including FFmpeg, Simple DirectMedia Layer (SDL), Snappy, FaceNet, and Curl. We also develop a customized video player to enable full control of the video decoding process and will provide the statistics required by the optimization solution.

Moreover, we propose a cross-layer optimization solution for allocating bandwidth to various cameras in a manner that optimizes the overall detection or recognition accuracy. The solution dynamically manages the application rates and transmission opportunities of various cameras based on the current network conditions,

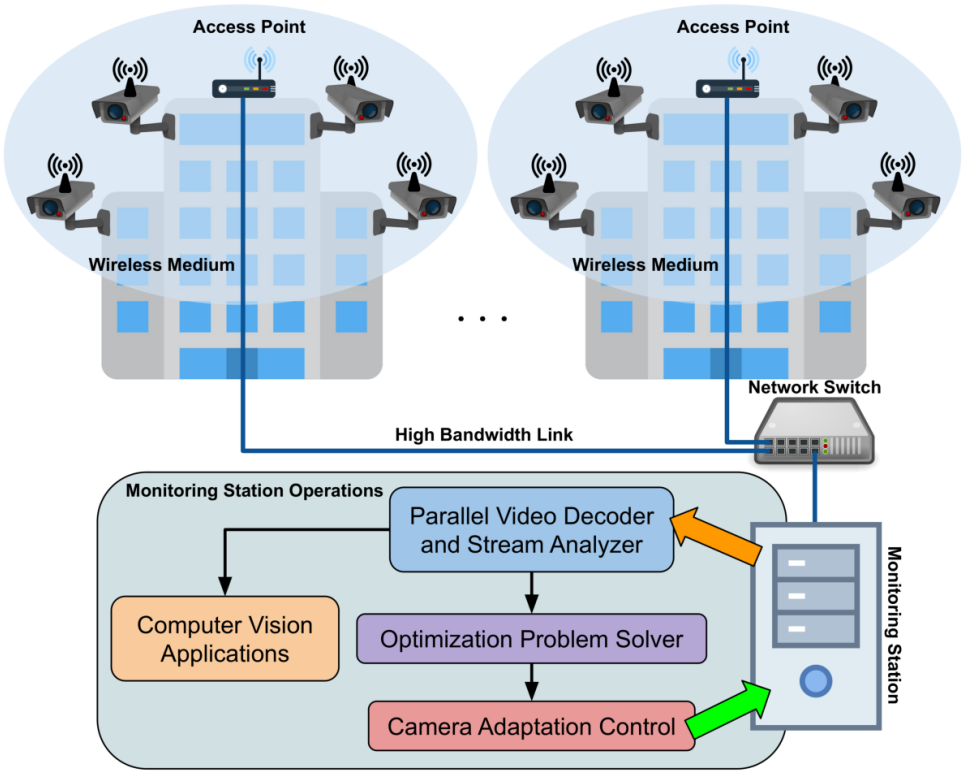


Figure 1.1: An Illustration of the Considered Computer Vision System

considering the capabilities and limitations of these cameras. Furthermore, the solution includes a new algorithm for determining the effective airtime of the network based on a novel method for estimating data dropping by using the smoothing calculations of the stream bitrates received by the monitoring station.

Research and development activities for our proposed bandwidth allocation solution can be categorized into three main phases.

In the first phase, we set up the required framework for our system. Different camera models and wireless access points should be considered in this phase to provide a development base for our proposed solution. As controlling the pan, tilt, zoom, and data transfer rate of PTZ cameras are necessary, different application interfaces should be implemented. A solution to limit the overall data transfer rate of an access point is provided.

In the second phase, we build a real CV system for automated video surveillance to run the actual exper-

iments. We develop a new algorithm for determining the effective airtime of the network. We also consider both face detection and face recognition applications. Our developed system employs HTTP streaming and homogeneous cameras as well as heterogeneous ones with varying capabilities (including resolutions and frame rates) to mimic varying deployment scenarios. The system is developed using a variety of open-source libraries, including FFmpeg, Simple DirectMedia Layer (SDL), Snappy, FaceNet, TensorFlow, and Curl. A customized video player is implemented to enable full control of the video decoding process.

In the third phase, we generate experimental results, using three different setups. The first experimental setup utilizes a real video surveillance dataset from campus, office, store, and street environments, collected from publicly available videos, while the second is based on a live laboratory environment, and the third utilizes the IARPA Janus Benchmark-B Face Challenge (IJB-B) dataset [74]. The results demonstrate that the proposed solution significantly improves CV accuracy.

1.4.2 Enhanced YOLO Solution

We develop an enhanced object detection solution based on YOLOv4 [17], which is one of the best performers in object detection tasks. The solution consists of a pre-processing step before activation of the inference tasks, as well as a YOLO deployment selection logic. This solution incorporates different methods of video motion detection and clustering algorithms to determine the patches of the input frame that require the inference call. This significantly reduces the volume of input pixels to the deep neural network, resulting in considerable power/computational-load savings. Moreover, the deployment selection logic could activate different YOLO inferences by using different network configurations (different input sizes). This enables an extra optimization space by considering different conditions and requirements (for example, satisfying minimum response time).

Additionally, to demonstrate the performance of the proposed Enhanced YOLO solution, we consider recorded street camera videos from five different cities (Lamai, Koh Samui, Thailand; Saint Petersburg, Rus-

sia; New Orleans, Louisiana, United States; New Laramie, Wyoming, US; Neath, Wales). The performance parameters are mAP, average execution time, and data transmission improvements, calculated by comparing the measured Enhanced YOLO results against the standard YOLO results.

The research and development activities for this part can be summarized as follows.

We first conduct an extensive examination of the available motion detection techniques, including Adaptive Background Learning [63, 60], Adaptive Selective Background Learning [63], CodeBook [39], Frame Difference [60], Local Binary Similarity Segmenter (LOBSTER) [65], Mixture Of Gaussian V2 [80], Pixel-based Adaptive Word Consensus Segmenter (PAWCS) [67], SigmaDelta [45], Static Frame Difference [63], Flexible Background Subtraction with Self-Balanced Local Sensitivity (SuBSENSE) [66], ViBe [13], Weighted Moving Mean [63], and Weighted Moving Variance [63]. Initially, the benchmarking and comparison of these different motion detection techniques are based on their performance independently, without considering the performance when they are used as an element within the Enhanced YOLO solution.

The output from the motion detection unit is a series of binary (black/white) images, where each white pixel represents a moving (foreground) pixel. By considering each white pixel as a single 2D data point and applying a clustering algorithm to the image, multiple clusters could be generated wherein each cluster represents a moving (foreground/region of interest) in the original video frame. In this stage, we run an extensive examination of the available clustering algorithms including KMeans [11], MiniBatch KMeans [59], Affinity Propagation [28], Mean Shift [23], Spectral [44], Agglomerative [48], DBSCAN [27], OPTICS [9], and Birch [79].

Scale reduction factors for video frames are considered in motion detection and clustering operations for reducing workload and improving execution time. We employ BGSLibrary [62] and scikit-learn [52] libraries to implement the motion detection and clustering functionalities, respectively.

After the region of interest (section containing the motion/patch) is determined by the clustering algo-

rithm, it can be fed to the YOLOv4 detection network as the input image. Because the patch size could be considerably smaller than the entire frame, there are two opportunities for performance improvements in the network operations. The YOLOv4 has a static input image size which could be smaller compared to the video frames. For example, the video frame could have a size of 1920×1080 pixels while the YOLOv4 input image could be set at 608×608 pixels. The network then resizes each 1920×1080 video frame to 608×608 and starts to operate on the image to produce the output detection results. Since a smaller patch (region of interest) is fed to the network, the resizing result will have more detail in comparison to the 608×608 sized entire video frame. This can improve detection performance as the moving objects are presented by more pixels to the network. The other opportunity for performance improvement comes from multiple deployments of YOLOv4 with different input resolutions.

For example, three separate simultaneously deployed networks with input sizes of 960×960, 608×608, and 304×304 could be considered. Based on the patch size to the original video frame size ratio and average detection probability from previous frames, one of these networks is selected and activated. The detection time of the network heavily depends on the input resolution, so by efficiently using a lower resolution network for smaller patches of the original video frame, a significant amount of time and processing power could be saved. We employ the Darknet [54] based implementation of the YOLOv4 from the original authors.

Now that the detection results in the smaller patch region are available, we have to create an algorithm/policy to deal with the combination of the detection results from the patch with the detection results from other regions of the frame from previous network executions.

The final step is the extensive examination of the mentioned Enhanced YOLOv4 system by utilizing different motion detection and clustering algorithms as input optimizers and also considering a configuration for the deployment of the YOLOv4 detectors with different input sizes. Different videos and datasets

are used in benchmarking the system. By having the original YOLOv4 configured with an input size of 1056×1056 resolution and generating the detection results, the optimal output is generated and is used as a comparison point against the results generated from the improved mentioned system. We measure the complexity of the incorporated techniques as execution time will have a major impact on our system.

Figure 1.2 demonstrates our proposed enhanced YOLOv4 system and summarizes the operations involved in each step.

1.4.3 Activity Detection Recurrent Neural Network

We implement an activity detection solution that exploits the video design space. This solution incorporates recurrent neural networks to efficiently determine the classification of an activity by considering multiple sequential detection results from the YOLO inferencing tasks, as well as pose/limbs estimation results.

The proposed activity detection RNN solution operates efficiently, considering the reduced/pruned input volume. This is provided by the output from the YOLO inferences in addition to pose/limbs estimation results. This capability is especially useful in surveillance monitoring stations to classify dangerous activities like leaving baggage behind at airports, engaging in violence/fighting, committing theft, etc. Another use case could be in sporting events, where the designed system could be incorporated to determine activities like passing, shooting, and heading a ball. This will be a significant step toward having a fully autonomous event-video-recording/directing agent.

The research and development activities for this part can be summarized as follows.

First, to train our proposed RNN for activity classification, we consider the action of leaving a backpack unattended as our target activity as this would have serious security implications if left undetected. As no suitable video dataset that covers this activity is publicly available, we create our own training data by having a person conduct the act multiple times in different ways (e.g., using the left or right hand to lay the backpack

on the ground) and recording the actions with a camera. As our system emphasizes the benefits of employing already well-designed and well-trained advanced neural networks, we consider only a single video file for the training phase, containing only 103 instances of the desired action. After the video is recorded, it should be labeled by marking the index number of the frame appearing after the targeted action is completed. By doing so, 103 frame indexes are generated. The next step is to extract the YOLO detection results and pose/limbs estimation results by feeding the consecutive video frames to the YOLO object detection network and MediaPipe Pose estimation tool, respectively. Detection and pose/limbs estimation results are combined and filtered for each frame. Our time-series data for training the activity detection network is composed of these elements together with the marked frame indices.

Second, we focus on designing and implementing an RNN for handling the activity classification task. We have selected TensorFlow [4] as our machine learning platform of choice to implement the RNN solution. Other options include Darknet [54] and PyTorch [51]. Sequential outputs from the YOLOv4 network are fed to our designed RNN solution to produce a final output as an activity label that could be immensely useful in many autonomous video-based activities like deploying security systems, autonomous driving, sporting events narration, etc.

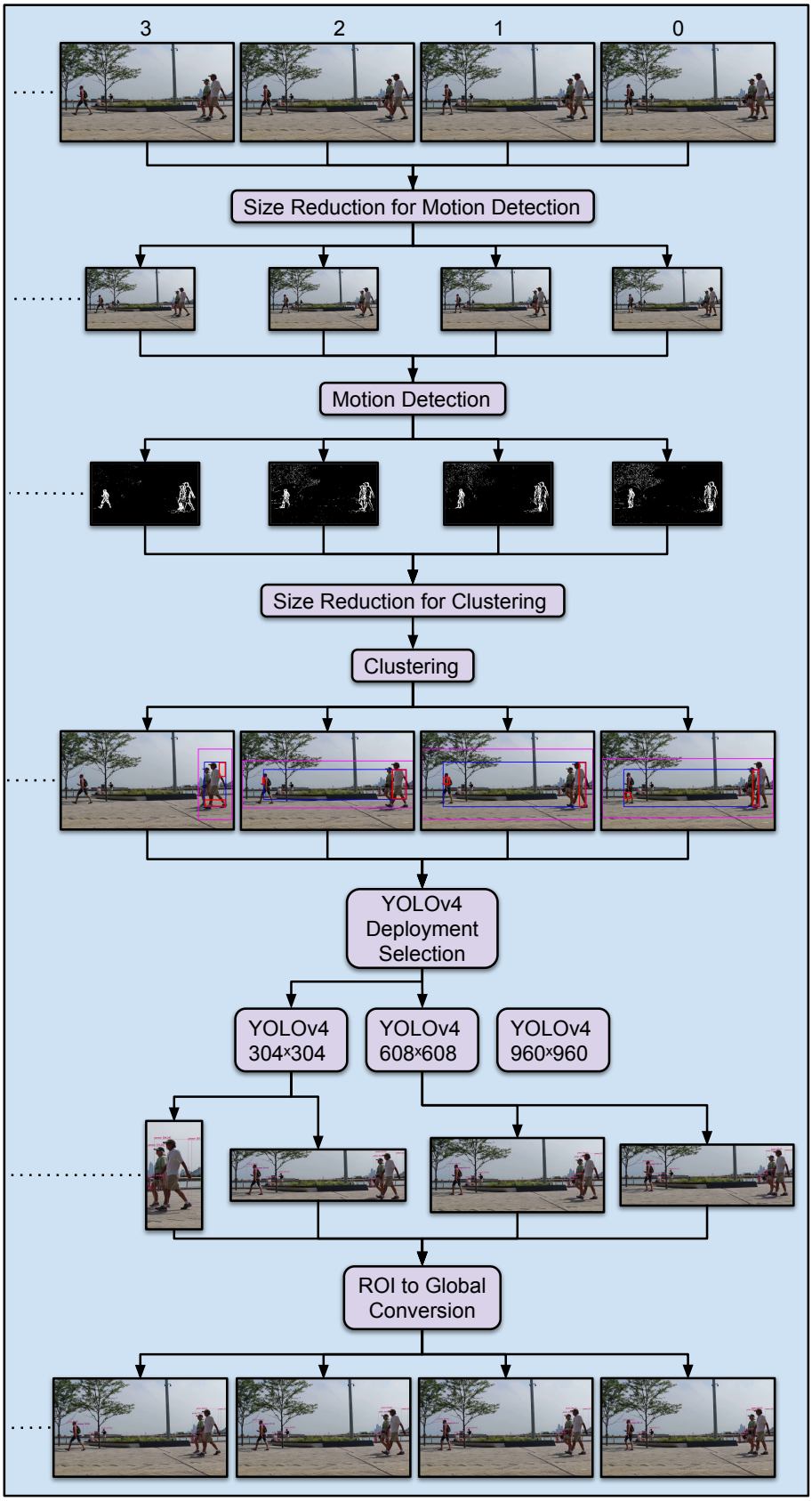


Figure 1.2: Proposed Enhanced YOLOv4 System

CHAPTER 2 BACKGROUND AND RELATED WORK

2.1 CV Systems and Cross-Layer Optimization

The *Enhanced Distributed Channel Access* (EDCA) mode of the 802.11e standard enables the provision of different quality-of-service levels among different access categories within the same station by adjusting parameters, such as the *Transmission Opportunity Time* (TXOP) [55].

Prior studies on cross-layer optimization in wireless video streaming have considered (i) systems in which only one station streams a video at a time [36, 12, 72] (and references within); (ii) systems in which a central video server streams to multiple stations [77] (and references within); and (iii) systems in which multiple stations deliver video streams to a central station [38, 37, 6, 7, 32, 61]. The latter category is most relevant to this thesis. In such many-to-one video streaming systems, the main objective has been minimizing the sum of video distortion in all received video streams [6, 37], instead of identifying accuracy error, which is the main concern in CV systems. Alsmirat and Sarhan in [7] explored optimizing only face detection, which is among the simplest of CV tasks. In addition, the problem formulation did not consider the limitation of the cameras in sending the encoded video streams. The aforementioned studies were simulation-based, used RTP streaming, and assumed MJPEG or abstract data streams. Video streaming in wireless ad-hoc networks was considered in [72].

As will be discussed in Subsection 3.2.1, the optimization solution requires an accurate estimation of the effective airtime, which can be defined as the fraction of the network time that is used for delivering useful data. Hsu and Hefeeda in [37] developed an analytical model for the effective airtime, whereas the authors in [6] developed an online estimation algorithm, addressing the shortcomings of that analytical model. In this thesis, we have enhanced the estimation algorithm by incorporating a novel method for estimating data dropping via smoothing calculations.

2.2 CV Algorithms

We have experimented with both face detection and face recognition. Face recognition is a major CV algorithm used in many applications, including authentication systems, personal photo enhancement, automated video surveillance, and photo search engines. Recent studies on face recognition employ deep learning using convolutional neural networks [70], with major algorithms including FaceNet [58] and ArcFace [25].

2.3 Neural Net Optimizations and Activity Detection

The authors in [78] proposed an object detection method for videos based on YOLOv3 and FlowNet 2.0, using multiple consecutive frames around the target frame from the video stream. First, the difference between the target frame and key frames (frames around the target frame) are calculated using optical flow operations: frames with small differences are directly fed to Flow-guided Bounding Box Transitions (FGBT) and skip running the YOLOv3 detection model, while frames with large differences pass through YOLOv3, and the bounding boxes are produced for them. If the bounding boxes have low confidence values, the target frame is sent to Flow-guided Feature Aggregation (FGFA) unit where multiple feature maps from multiple key frames are aggregated to enhance the target frame features; the result then is sent to YOLOv3 again to generate the final bounding boxes. The proposed technology is tested on three different videos, and the results are visually compared with pure YOLOv3 outputs.

Lu in [43] combined deep neural object detection with traditional object motion estimation to satisfy the real-time need for object tracking tasks. It is mentioned that slow computation speed limits the application of deep learning methods, and by combining the output of deep neural networks and the traditional methods of object tracking, they have balanced the detection speed and mAP. This feat is essentially achieved by reducing the frequency of deep neural network activations on the input images. This method was inspired by the natural way humans handle fast-moving object detection, namely by ignoring the details of

fast-moving objects but tracking objects by the movement laws and simple visual information at a glance. In this methodology, deep learning detection methods can be said to use glaring to accurately locate an object. Using traditional movement detection methods is like glimpsing a scene to locate an object. In other words, the study tries to reduce the workload on the execution framework by balancing the mAP and speed (marginally reducing mAP for higher speeds).

Alvar in [8] proposed a method whereby an object tracking task is accomplished by taking advantage of encoded moving vector information in video streams. Encoded moving vectors already exist in the compressed video bitstream, so the study attempted to use them to indicate the approximate location of the target object. By combining the semantic object detector results (from the decoded video frame), the object's location could be refined by accurately providing a bounding box on the decoded frame. The focus of the study was object tracking in already encoded video streams.

The authors in [76] suggested a method whereby the object detection task frequency at the edge was dynamically managed by a unit that handles the difference detection among frames. Here the deep neural net considers the whole image as ROI for the network input.

Oltean and Florea in [49] proposed a system that was used in traffic control related functions. It utilized the tiny-YOLO network to detect vehicles in a predefined ROI in the video stream. It also took advantage of motion estimation and tracking capabilities to account for cases where detection was not adequate in the incorporated neural net.

The authors in [46] introduced a system that assists visually impaired people by utilizing YOLO for handling the tracking and the occlusion compensation tasks. The system's output is delivered to the user as a warning through bone-conduction headphones.

In [73], the authors proposed a method for labeling an action by modifying YOLO and combining the results with the standard YOLO detections. The modification was applied by removing the last fully-

connected layer in YOLO and replacing it with an LSTM unit. The computational complexity is increased in comparison to the standard YOLO. The system, especially the training activities, has a dependency on the internal parts of the YOLO framework.

The authors in [19] suggested a solution where the action was classified using two modules operating and interconnecting simultaneously: a detection module which was based on the YOLOv2 and a recognition module which was based on the C3D network. The solution has convolutional units only, and the input is treated as a 3D stack of pixels.

In [50], the authors proposed an alternative method for visual tracking which incorporated a bi-directional LSTM network. It used preliminary location information and the appearance features of the target, produced by the YOLO algorithm, to improve space-temporal location predictions.

Piao and Inoshita in [53] introduced a method of carried object recognition and improved the performance of the conventional CNNs by introducing the additional knowledge of location relation between body parts and objects. The proposed method can help with the recognition task if carried objects are of small size or low resolution or if they are occluded by people.

The authors in [18] surveyed the current state-of-the-art CNN-based detection methods. Mask R-CNN, YOLO, and MOG were considered in the conducted experiments. YOLO was mentioned to have real-time satisfying results, and it was a better performer than Mask R-CNN in case of occlusion.

In order to mitigate the performance issue in deep neural networks, Chen in [20] proposed to cut a deep neural network in the mid-level and communicate between the two parts using a video encoder/decoder pipeline. Although this method could reduce the required transmitted data for the task at hand, the lossy nature of video encoding/decoding is unfavorable for neural network operations. Additionally, this method distributes the amount of required processing power as opposed to reducing the computational cost of object detection tasks.

Despite the contribution of the studies described above, the following two areas of research are needed:

- A general ROI-based method to reduce the input volume for a neural network, while simultaneously improving the detection accuracy by providing an opportunity to concentrate on the areas of importance.
- A general activity classification algorithm by taking advantage of object detection networks to categorize activities, resulting in an algorithm that is efficiently trainable and could operate on the object detection results.

In addition to providing an optimal bandwidth allocation solution for wireless cameras in CV systems, our proposed work includes a solution called Enhanced YOLO as a major CV application. This solution has an input filter for the neural network, reducing the input pixel volume which will invoke the inference calls on the network. This enables the system to focus on the areas of importance by considering higher resolution ROI patches. Additionally, it has multiple deployments of the YOLOv4 network with different input resolutions and dynamically selects the appropriate deployment in inference operations, resulting in significant execution time reductions. Our proposed work also includes an activity detection solution by taking advantage of the output recognition results from the YOLOv4 network to classify activities. RNNs are used to construct this unit.

CHAPTER 3 EXPERIMENTAL ANALYSIS OF OPTIMAL BANDWIDTH ALLOCATION IN COMPUTER VISION SYSTEMS

3.1 Developed Computer Vision System

We built a real many-to-one CV system and analyzed the effectiveness of cross-layer optimization by providing the results of actual experiments. As illustrated in Figure 1.1, the system consists of a monitoring station and various video cameras, including PTZ cameras, all of which are connected by a Wi-Fi network. The cameras include four Pan/Tilt/Zoom (PTZ) surveillance cameras (IPCam 7210W), one wide-angle camera (VivoTek IP7139), and two webcams (HP Truevision HD and Labtec PRO Webcam). The system employs HTTP streaming for delivering videos from the cameras to the monitoring station. To capture realistic deployment scenarios, we used both homogeneous and heterogeneous cameras. Although we primarily used H.264, we considered the co-existence of other encoders.

Figure 3.1 illustrates the overall system design, including built-in modules for performance evaluation. In the extensive system development process, we used the following libraries for developing various system aspects: FFmpeg, Simple DirectMedia Layer (SDL), Snappy, TensorFlow, and Curl. To turn the two webcams into functional IP cameras, we employed the VLC media streaming tool and developed a program in Python to act as a *virtual interface* for these cameras. Hence, we refer to these cameras as *virtualized IP cameras*. The virtual interface enables these two webcams to adjust their encoding bitrates according to the received control messages from the monitoring station, thereby allowing their treatment as any regular IP camera.

The monitoring station has the following main units.

- *Parallel Video Decoder and Stream Analyzer* – This unit receives video streams from IP cameras and decodes them. It also analyzes the streams to accurately determine all the system parameters involved in the effective airtime estimation, bitrate smoothing, and optimization solution. We developed a cus-

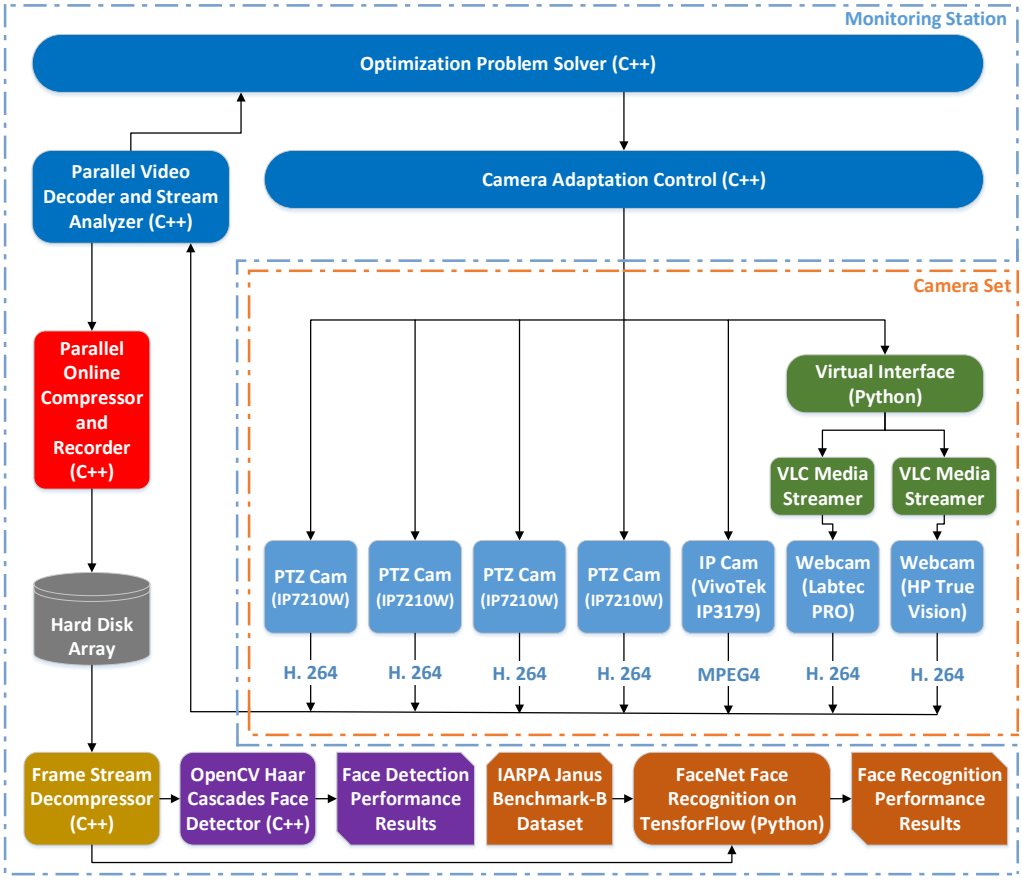


Figure 3.1: An Illustration of the Overall System Design Including Built-in Modules for Enabling Performance Evaluation

tomized multi-threaded video player in C++ using FFmpeg and SDL libraries to provide full control over the video decoding process and offer all the statistics required by the optimization solution. As SDL offers special multi-threading tools for media-rich applications, we incorporated it to handle parallelism and thread creation and to enforce mutual exclusion for different resources.

- *Optimization Problem Solver* - Uses the estimated system parameters to determine the optimal distribution of the effective airtime among various cameras and then sends the allocations to the next unit.

- *Camera Adaptation Control* – This unit receives the optimal portion of the effective airtime for each

camera, generates a properly-formatted HTTP control message, and delivers the message to each camera. We utilized the HTTP message transfer function of Curl, a client-side URL data-transfer library supporting various protocols, including HTTP.

- *Parallel Online Compressor and Recorder* – We devised this unit in C++ to implement an offline approach for (i) analyzing the received video stream by applying CV algorithms and (ii) facilitating the comparative performance evaluation of various bandwidth allocation solutions. As expected, without the use of a specialized distributed processing system, the simultaneous analysis of all video streams can not be achieved in real-time, even when using a workstation with 8-core AMD Ryzen 7 running at 4 GHz with 32 GB of DDR4 RAM due to the aggregate computational complexity of CV algorithms. The devised offline approach helps bypass this challenge. Specifically, it records the video streams from various cameras for further analysis, without any video encoding or transcoding. However, this raises a new challenge: no cost-effective storage device can provide the necessary capacity and performance. Hence, we developed a recording process that performs fast lossless compression on the received data and enables the simultaneous handling of writing the video streams on multiple hard disk drives using the SDL and Snappy libraries. Snappy provides a toolkit for fast online compression. Note that the recording process and the offline examination approach are only for performance evaluation purposes and are not imposed by the proposed optimization solution. Real deployments of CV systems can address real-time detection and recognition by utilizing a distributed processing system and/or powerful GPUs.

- *Frame Stream Decompressor* – We developed this unit in C++ to uncompress and analyze the compressed recorded video streams. The uncompression task utilizes the Snappy library.

- *Face Detection* – We developed this unit in C++ using OpenCV to run the face detection function using Haar feature-based cascade classifiers.

- *Face Recognition* – We developed this unit in Python using FaceNet on TensorFlow to run the face recognition tasks. Specifically, we utilized the FaceNet 1.0.3 Python package, an open-source TensorFlow implementation of the face recognizer described in [58]. We used the pre-trained model named 20180402-114759, which is trained on the VGGFace2 data set and has the Inception ResNet v1 architecture.

When the cameras receive the HTTP control messages from the monitoring station, they act accordingly to adjust their video capturing and encoding parameters.

3.2 Proposed Cross-Layer Optimization Solution

We developed an enhanced cross-layer optimization solution, which dynamically distributes and allocates the wireless network bandwidth among various cameras in the considered many-to-one CV system, illustrated in Figure 1.1, with the objective of optimizing either the overall detection or recognition accuracy. The system includes S cameras and each one streams a different video at rate r_s over a bandwidth-limited WiFi medium to the access point, which in turn delivers the stream to the monitoring station typically through a high bandwidth link. Different video sources may have dissimilar physical rates. The CV system can be expanded by including and interconnecting such multiple cells.

3.2.1 Cross-Layer Optimization Problem Formulation

As in [7], the optimization problem can be formulated as the minimization of the sum of the accuracy error ($\mathcal{E}$) of all the video streams received by the central monitoring station. Since the wireless medium is shared by all S cameras, the problem can be formulated as to how to find the optimal portion of the airtime f_s to be assigned to each camera s. The set of allocations is given by $F^* = \{f_s^* | s = 1, 2, 3, ..., S\}$, where each allocation f_s is between 0 and 1, inclusive, and the sum of all allocations is equal to the effective medium airtime (A_{eff}). Hence, the application-layer transfer rate of camera s can be given by $r_s = f_s \times Y$, where Y is the total medium bandwidth (related to the access point). Subsequently, the optimization problem can

be formulated as follows:

$$F^* = \arg\min_{F} \sum_{s=1}^{S} \mathcal{E}(r_s), \tag{3.1a}$$

Subject to

$$\sum_{s=1}^{S} f_s = A_{eff}, \tag{3.1b}$$

$$0 \leq f_s \leq 1, \tag{3.1c}$$

$$r_s = f_s \times Y, \tag{3.1d}$$

$$f_s \leq \frac{y_s}{Y}, \quad \text{and} \tag{3.1e}$$

$$s = 1, 2, 3, ..., S. \tag{3.1f}$$

Assigning f_s and the resulting r_s to each camera s minimizes the overall accuracy error. We enhanced the formulation in [7] by (i) modifying Condition (3.1d) to consider the overall rate permitted by the access point (as opposed to just that perceived by the camera) and (ii) introducing Condition (3.1e). The latter guarantees that the assigned transfer rate for each camera s does not exceed that permitted by y_s, where y_s is the maximum application-layer rate that is allowed by the perceived physical rate of camera s. Note that channel fading and other reception conditions are inherently involved in the procedure determining A_{eff} (Subsection 3.2.2) and y_s. Note that r_s, y_s, and Y are all in terms of application-layer data.

As will be discussed in Subsection 3.4.3, by optimizing the accuracy, the system tries to increase the frame rate to the extent allowed, thereby decreasing latency. In addition, cross-layer optimization effectively decreases the latency, since each camera will send data at a rate that can be effectively received.

3.2.2 Effective Airtime Estimation

As required by the formulated optimization problem, we propose an enhanced algorithm for estimating the effective medium airtime (A_{eff}). The algorithm employs a novel method for determining the overall data dropping and corruption rate d_s for camera s using the smoothing calculations of the bitrates of the streams received by the monitoring station. The method uses only the video decoding statistics at the monitoring station, thereby avoiding the need to obtain dropped data statistics, which would not be accessible using standard APIs.

Figure 3.2 shows the simplified algorithm. First, with each camera sending data at a rate that is equal to the medium bandwidth divided by the number of cameras, the algorithm determines the effective throughput t_s for the video stream for each camera s as received by the application layer of the monitoring station. The algorithm then uses t_s to provide the initial value of A_{eff}: $A_{eff} = \sum_{s=1}^{S} t_s/Y$. Subsequently, during an estimation period, the algorithm assesses the overall data dropping and corruption rate d_s at the monitoring station while receiving the video stream from camera s, and then adjusts the estimated effective airtime accordingly. The algorithm determines d_s as the measured corrupted data rate for the stream plus a value capturing the difference between the announced frame rate for camera s and the frame rate that is actually received by the monitoring station. Specifically, we developed and employed the following equation to assess d_s:

$$d_S = [CorruptData_s + (SSDR_s \times SumFrameDelayVar_s) \times DW]/EP, \tag{3.2}$$

where $CorruptData_s$ is the total size of the received packets from source s that are corrupted; $SSDR_s$ is the smoothed stream bitrate for source s; $SumFrameDelayVar_s$ is the sum of the variations between the frames produced by source s and the corresponding one that is received by the monitoring station, with the variation being measured in terms of the delays between consecutive frames; DW is the delay weight

constant providing flexibility in adjusting the estimated value of the dropped data during EP; and EP is the estimation period. By multiplying $SumFrameDelayVar_s$ with $SSDR_s$, the size of the dropped data during EP can be estimated.

Through a *smoothing operation* over the evaluation time, rather than just using the momentary bitrate value, SSDR can be estimated more accurately as follows:

$$SSDR = SC \times PSSDR + (1 - SC) \times MSDR, \tag{3.3}$$

where SC, $PSSDR$, and $MSDR$ are the smoothing constant, previously estimated smoothed stream data rate, and momentary stream data rate, respectively. The delay weight, smoothing constant, and previous stream data rate help in estimating the dropping rate using the current observed frame rate.

The algorithm then determines the overall average corruption and dropping ratio perceived by the monitoring station for all streams as follows: $A_\Delta = \sum_{s=1}^{S} d_s/Y$. This value is used to adapt the current value of A_{eff} at the end of the current estimation period. If A_Δ is 0, the algorithm increases A_{eff} by $C \times A_{thresh}$. By contrast, if A_Δ is greater than some threshold A_{thresh}, it reduces A_{eff} by $\acute{C} \times (A_\Delta - A_{thresh})$, where A_{thresh} controls the allowable data dropping in the network and $\acute{C}$ and C are network-related constants. Otherwise, it increases A_{eff} by $\tilde{C} \times (A_{thresh} - A_\Delta)$, where $\tilde{C}$ is also a constant value.

To ensure better convergence and stability, we set C, $\acute{C}$, and $\tilde{C}$, respectively, to 20, 0.8, and 16 based on extensive experiments. Using these three parameters rather than just one greatly accelerates the convergence of A_{eff}. The algorithm, however, continues to update for the effective airtime value and does not stop upon convergence because the network conditions change dynamically. In our system, convergence occurs within 40 seconds initially and within a few seconds thereafter.

Input: $\{t_1, .., t_S, PSSDR_1, .., PSSDR_S, MSDR_1, .., MSDR_S,$
$CorruptData_1, .., CorruptData_S,$
$sumFrameDelayVar_1, .., sumFrameDelayVar_S\}$
Output: $\{A_eff\}$
if this is the first time to run the algorithm
$A_{eff} = \sum_{s=1}^{S} t_s/Y$;
At the end of each estimation period{
For each source s = 1 to S\{
$SSDR_s = SC \times PSSDR_s + (1 - SC) \times MSDR_s$;
$d_S = [CorruptData_s + (SSDR_s \times$
$SumFrameDelayVar_s) \times DW]/EP$; } // For
$A_\Delta = \sum_{s=1}^{S} d_s/Y$;
if ($A_\Delta == 0$) // Increase A_{eff} by $C \times A_{thresh}$
$A_{eff} = A_{eff} + C \times A_{thresh}$;
else if ($A_\Delta > A_{thresh}$) // Reduce A_{eff} by $\acute{C} \times (A_\Delta - A_{thresh})$
$A_{eff} = A_{eff} - \acute{C} \times (A_\Delta - A_{thresh})$;
else // Increase A_{eff} till first decrement
$A_{eff} = A_{eff} + \tilde{C} \times (A_{thresh} - A_\Delta)$; }// At

Figure 3.2: Simplified Algorithm for Dynamically Estimating the Effective Airtime

3.2.3 Cross-Layer Optimization Solution

Face Detection

According to [7], the accuracy error for face detection can be modeled as a linear function of the video data rate (r_s) for camera s: $\mathcal{E}(r_s) = a_s \times r_s^{b_s} + c_s$, where a_s, b_s, and c_s are camera-specific constants. Thus, the optimization problem (Equation (3.1)) is a budget-constrained convex problem that can subsequently be solved by Lagrangian Relaxation. Realistically assuming that the b_s values are the same for all cameras and equal to b, the solution can be given by:

$$f_s^* = (\frac{-\lambda}{a_s Y^b b})^{(1/(b-1))}, \tag{3.4}$$

where

$$\lambda = (\frac{A_{eff}}{\sum_{s=1}^{S} (\frac{-1}{a_s Y^b b y_s})^{(1/(b-1))}})^{(b-1)}. \tag{3.5}$$

We devised the following method to ensure that Condition (3.1e) is met: if f_s^* is larger than y_s/Y, we restart this solving process after setting f_s to y_s/Y, subtracting y_s/Y from A_{eff}, and eliminating that source from the problem domain.

Face Recognition

According to [31], the accuracy error for face recognition can be modeled as a sum of two exponentials of the video data rate (r_s): $\mathcal{E}(r_s) = a_s \times e^{b_s \times r_s} + c_s \times e^{d_s \times r_s}$, where a_s, b_s, c_s, and d_s are constants. Assuming $b_s = d_s$, the model can be simplified as $\mathcal{E}(r_s) = a_s \times e^{b_s \times r_s}$, where a_s and b_s are constants. As r_s can now be given as a function of the other parameters in the model, the optimization problem (Equation (3.1)) can be solved by Lagrangian Relaxation. The simplified model yields results similar to the original; based on actual experiments, the SSE, R-Square, Adjusted R-Square, and RMSE values in the original model compared with the actual data are 0.006267, 0.995, 0.9931, and 0.02799, respectively. In contrast, the values with the simplified model are 0.007374, 0.9941, 0.9935, and 0.02715, respectively. Assuming again that all the b_s values are the same for all cameras, the solution can be given by:

$$f_s^* = \frac{ln(\frac{-\lambda}{a_s b})}{b}, \tag{3.6}$$

where

$$\lambda = -(e^{\frac{A_eff}{b}} \prod_{s=1}^{S} a_s b)^{\frac{1}{S}}. \tag{3.7}$$

3.2.4 Proposed Method for Determining the Constant Values of the Accuracy Error Models

We presented the following method for determining the constant values of the analytical accuracy error models, namely a_s and b. These values are determined offline by first recording a video of the actual environment during typical operation at the highest supported resolution and bitrate by the related surveillance camera in the deployed system. Later, the video is transcoded to different combinations of resolution

and bitrate. Subsequently, the proportions of the detected/recognized faces relative to the original video are computed to find the accuracy error ($\mathcal{E}$) values. Finally, the values of the model constants are estimated based on the analytical models of the accuracy error. The system needs to recompute the constants only in the presence of significant changes in the system or environment. Furthermore, the readjustment of these constant values is a relaxed requirement and thus can be performed during the normal system operation.

3.3 Performance Evaluation Methodology

Table 3.1 summarizes the main parameters. Extensive analysis indicates that setting A_{thresh} and *Estimation Period* to 0.0075 and 5 seconds, respectively, improves performance in terms of both stability and convergence.

Table 3.1: Summary of Experimental Characteristics and Parameters

Parameter	**Model/Value(s)**
Number of Video Cameras	4, 7
Recording Period (minutes)	10
Application Rate	If not optimized: Max. Access Point Rate / # Cameras
Video Frame Rate (fps)	Camera Dependent: 7.5, 10, 25
Physical Characteristics	Extended Rate (802.11n)
Physical Data Rate (Mbps)	30, 25, 20, 15, 13, 10, 5
State Report Interval (seconds)	5
Detection Error Model	$b = -1.309$, $a_s = 3103$ (IP7210W Cam)
Recognition Error Model	$b = -88.35 \times 10^{-5}$, $a_s = 1.593$ (IP7210W Cam)
Camera Video Resolutions	1280×720, 800×600, 640×480

We conducted experiments using three different setups, as summarized in Table 3.2 and detailed later in this section. All experiments are performed using a TP-LINK TL-WR841N wireless router as the access point and a workstation with 8-core AMD Ryzen 7 running at 4 GHz with 32 GB of DDR4 RAM as the monitoring station. H.264 is used in all cameras except for VivoTek IP7139, which instead supports MPEG-4.

We compared the proposed solution, referred to as *New Optimization*, with the solution in [7], referred to as *Existing Optimization*. We also analyzed the case when the optimization is disabled, referred to as *No Optimization*. The main analyzed metrics were *face detection accuracy* and *face recognition accuracy*,

Table 3.2: Summary of the Three Experimental Setups

Setup	Recorded Resolution(s)	Cams	Video Content	Application
I	1280×720, 800×600, 640×480	7	Surveillance videos	Detection
II	1280×720	4	Live laboratory environment	Detection
III	1280×720	4	IJB-B data set	Recognition

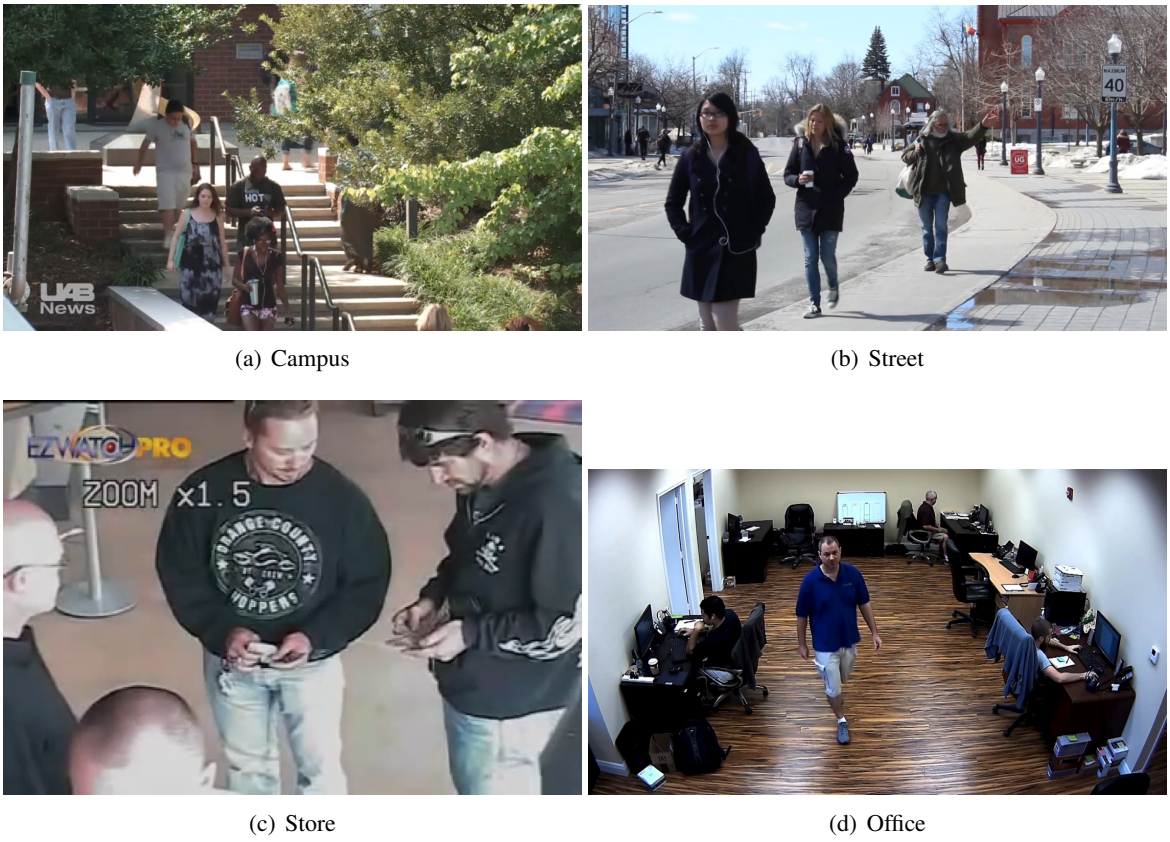
measured in terms of the overall number of detected/correctly recognized faces. *OpenCV* was used to run the Viola-Jones algorithm on the decoded video streams in Experimental Setups I and II. In contrast, *FaceNet* was utilized to run face recognition in Experimental Setup III.

3.3.1 Experimental Setup I: Using a Real Video Surveillance Data Set

Experimental Setup I uses various types of cameras (discussed in Section 3.1) to capture real surveillance videos rendered on separate monitors (model: Dell E2210Hc), thereby providing repeatable, realistic, and diverse scenery. Table 3.3 summarizes the main characteristics of the video surveillance data set, which were collected from YouTube and other sources and will be publicly available. The videos have different characteristics (including resolution and frame rate) and come from different environments: *office*, *campus*, *stores*, and *busy streets*. The original videos were truncated so that each category has nearly the same total video duration. Figure 3.3 shows sample video frames from this data set.

Table 3.3: Characteristics of the Real Surveillance Videos Used in Experimental Setup I

Type	Resolution	Duration (sec)	Frame Rate (fps)	Bitrate (Kbps)
Campus	1280 × 720	27	29	2704
Campus	1920 × 1080	44	23	3145
Campus	1280 × 720	77	30	2149
Office	1920 × 1080	10	30	2317
Office	480 × 360	8	30	343
Office	1280 × 720	57	30	2098
Office	1280 × 720	32	25	2088
Office	640 × 360	42	29	575
Store	480 × 360	57	6	355
Store	370 × 252	23	23	363
Store	1280 × 720	69	23	2550
Street	1280 × 720	14	30	768
Street	1920 × 1080	27	23	4215
Street	1920 × 1080	40	23	4035
Street	1280 × 720	68	29	2199



(a) Campus (b) Street

(c) Store (d) Office

Figure 3.3: Sample Frames from the Videos in Experimental Setup I

3.3.2 Experimental Setup II: Live Laboratory Environment

Experimental Setup II used our PTZ IP-Cam 7210W cameras to capture videos from a real lab environment. As discussed in Section 3.1, the monitoring station runs the optimization solution and sends the target bitrate to each camera, which in turn produces and transmits the adapted H.264 video stream. In this setup, the monitoring station also provides a controlled patrol movement for each camera. To allow for fair comparisons among various allocation solutions, a person in the lab acted according to a predefined script in each evaluation session. The script specified the paths that must be traversed by the acting person; the standing and walking directions; and the time spent on each path. Figure 3.4 shows sample concurrent views from two PTZ cameras.

(a) Camera 1 (b) Camera 4

Figure 3.4: Sample Concurrent Views of the Two Cameras in Experimental Setup II

3.3.3 Experimental Setup III: Using Videos from Janus Benchmark-B Face Challenge Data Set

Experimental Setup III is similar to Setup I, except for the use of a different dataset. Since the dataset used in Experimental Setup I did not contain any ground truth concerning the present faces, we utilized an unconstrained face recognition dataset, specifically the IARPA Janus Benchmark-B Face Challenge (IJB-B) dataset [74]. As the main objective of the face recognition experiment is to compare the performance of different bandwidth allocation solutions competing for the available effective airtime, we used the following criteria to select surveillance-like video files from the data set: (a) the presence of changing/moving backgrounds, (b) the presence of multiple subjects, and (c) the presence of a wide range of facial expressions/angles of the main subject in the scene. Table 3.4 summarizes the main characteristics of the selected video files, and Figure 3.5 shows sample frames.

Table 3.4: Characteristics of the Videos Used in Experimental Setup III

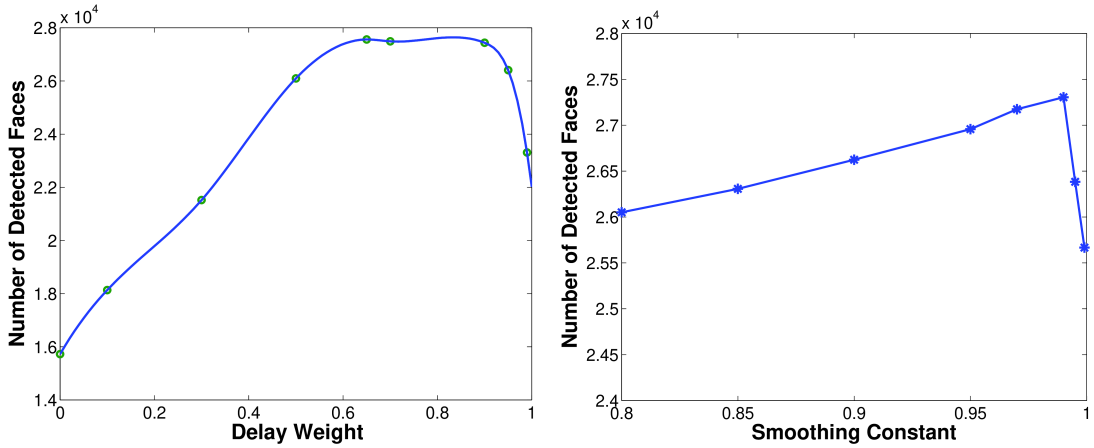
Type	**File No.**	**Resolution**	**Length (sec)**	**Frame Rate (fps)**	**Bitrate (Kbps)**
Street/Crowd	626	1280 × 720	6	29.97	2620
Sports	847	1280 × 720	13	25	2607
Street/Interview	1038	1280 × 720	20	25	3006
Politician Visit	1058	640 × 360	5	25	917
Street/Person	1668	368 × 300	7	29.97	268

Figure 3.5: Sample Frames from the Videos in Experimental Setup III

3.4 Results Presentation and Analysis

3.4.1 Tuning System Constants

Let us first discuss how to select the values of the constants of effective airtime estimation, namely the delay weight (DW) and smoothing constant (SC). Figure 3.6 illustrates how these constants impact face detection accuracy. The number of detected faces increases initially with the DW because of the tendency to produce higher frame rates, which reduces the stream bitrates and thus reduces both the contention for the medium bandwidth and the data dropping and corruption rate. Ultimately, the perceived frame rate by the monitoring station is increased. After a certain point, however, the generated high frame rates greatly reduce the stream bitrates and, consequently, the video quality. Likewise, the number of detected faces increases with SC up to a certain point and then starts to decrease. The increase occurs because the smoothing operations enable the system to estimate the effective airtime more accurately, whereas the subsequent decrease is due to aggressive smoothing, which greatly marginalizes the impacts of the momentary values of the stream bitrates. The best values of the DW and the SC in the considered system configuration are



(a) Delay Weight (b) Smoothing Constant

Figure 3.6: Effects of the Smoothing Constant and Delay Weight on Detection Accuracy [Experimental Setup I, 15 Mbps Medium Bandwidth]

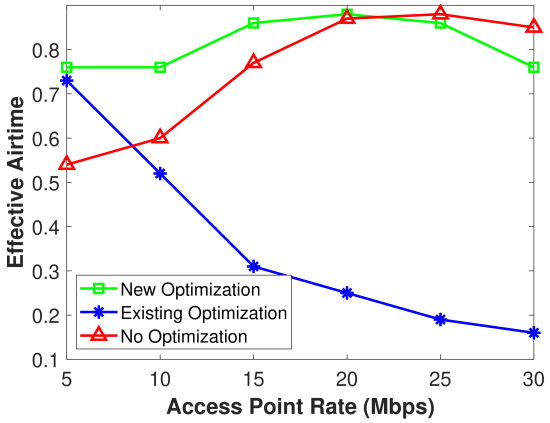


Figure 3.7: Comparing Various Solutions in the Overall Effective Airtime [Experimental Setup I]

0.65 and 0.99, respectively.

3.4.2 Comparing Effective Airtime Estimation under Different Solutions

Figure 3.7 shows that the proposed solution produces the largest area under the effective airtime curve; its area is 125% larger than that of the best existing solution and 8% larger than that with disabled optimization. As discussed in Subsection 3.2.1, the existing solution causes the system to use only a portion of the available bandwidth, thereby reducing the effective airtime to even lower values than those with disabled optimization.

3.4.3 Analysis of Cross-Layer Optimization for Face Detection

Let us now analyze the effectiveness of cross-layer optimization in terms of detection accuracy under Experimental Setup I. Figure 3.8 shows the number of detected faces versus bandwidth capacity for the entire video dataset and for each video category, as well as the percentage of false positives for each solution but for all categories. The proposed solution achieves 19%, 10%, 10%, and 10% higher accuracy than the existing solution in campus, office, store, and street environments, respectively, and 54%, 45%, 72%, and 69% higher accuracy than disabled optimization. As expected, the number of detected faces generally increases with the medium capacity. The occasional dips in the case of the existing solution are due to the aforementioned problem in utilizing the medium bandwidth. After a certain point, the proposed solution and disabled optimization converge to similar values, when the medium capacity is large enough to accommodate the maximum supported bitrates of the cameras, thereby eliminating bandwidth contention. *In actual systems, the number of employed cameras and the supported bitrates are larger, thereby raising the medium bandwidth at which convergence occurs.* Interestingly, the existing solution performs worse than disabled optimization when the contention among cameras falls below a certain level. Due to the lack of ground truth, the false positive results (Figure 3.8(f)) are manually determined based on 500 randomly-selected recorded frames from each camera in every experiment. The false positive percentage of the proposed solution is within 4% and is generally better than the other solutions. Figure 3.9 compares various solutions in detection accuracy for each camera. Cam1, Cam2, Cam3, and Cam4 refer to the IPCam 7210W wireless PTZ security cameras; Cam5 and Cam6 are the HP Truevision HD and Labtec PRO webcams, which are turned into functioning IP cameras using the VLC media player; and Cam7 is the VivoTek IP7139 camera. The proposed solution consistently performs the best, whereas the existing solution performs even worse than the disabled optimization with certain cameras due to its aforementioned problem.

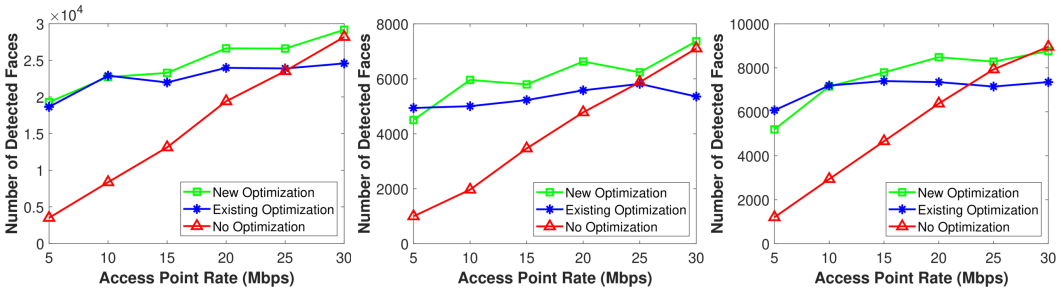
Figure 3.10(a) compares various solutions in terms of face detection accuracy under Experimental Setup

II. It shows the number of detected faces by the entire CV system and by each camera. Although there is only one person in the scene, the person appears in multiple frames of the video streams. Therefore, having higher quality video streams translates to a larger number of detected faces. The proposed solution achieves 123% higher accuracy than the existing solution and 148% higher than the disabled optimization.

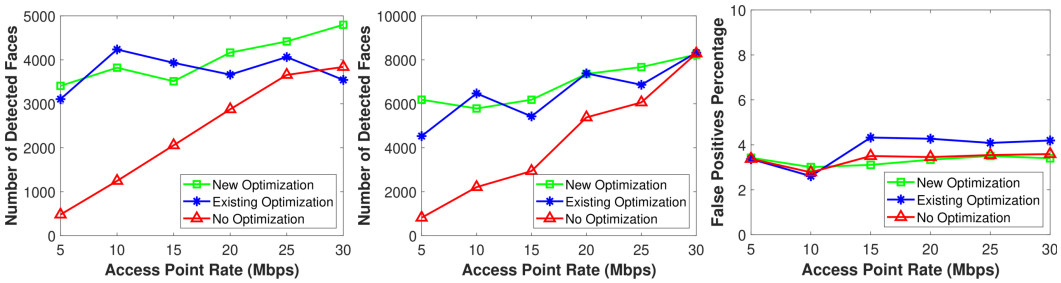
Let us now discuss the dynamics of the system as a result of the interplay of various factors. Figure 3.11(a) demonstrates the relationships among different system metrics, namely the average received rate by the monitoring station, effective airtime, frame rate, and the number of detected faces. To effectively display the different values of attributes together in the same chart, attribute values are normalized. The number of detected faces is the most important metric, and indeed the proposed solution continues to hold the lead in that metric. Achieving a high value in this metric depends on two main factors: the effective airtime and the average received frame rate. The proposed solution demonstrates a remarkable balance in improving these two main factors, resulting in producing the highest face detection accuracy. The existing solution becomes a viable choice only when the power consumption is of utmost significance and preferred over accuracy. The results also demonstrate the high effectiveness of cross-layer optimization. As expected, disabled optimization leads to the smallest number of detected faces because when the available medium bandwidth is limited and each camera sends at the highest rate without any governing policy, severe congestion in the network will result, thereby greatly increasing the probability of data packet loss and frame dropping.

3.4.4 Analysis of Cross-Layer Optimization for Face Recognition

Figure 3.10(b) compares the numbers of correctly recognized faces achieved by various solutions in Experimental Setup III. The proposed solution achieves 29% and 32% higher accuracy than the existing solution and disabled optimization, respectively. Figure 3.11(b) demonstrates the relationships among different system metrics: the average received rate by the monitoring station, effective airtime, frame rate, and the number of correctly recognized faces. The values are normalized.

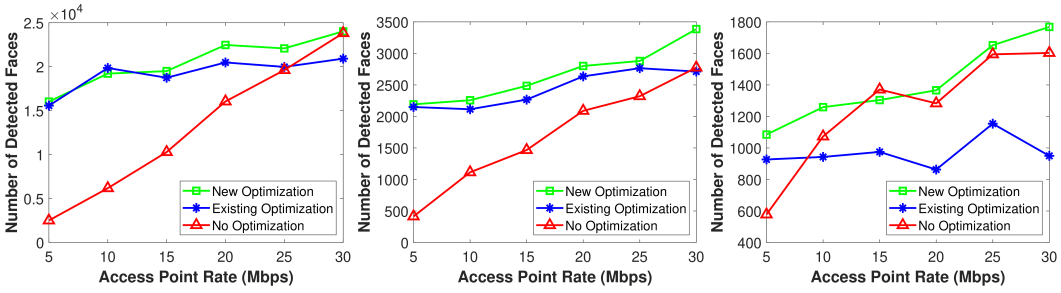


(a) Overall (b) Campus Environment (c) Street Environment



(d) Store Environment (e) Office Environment (f) False Positives (All Environments)

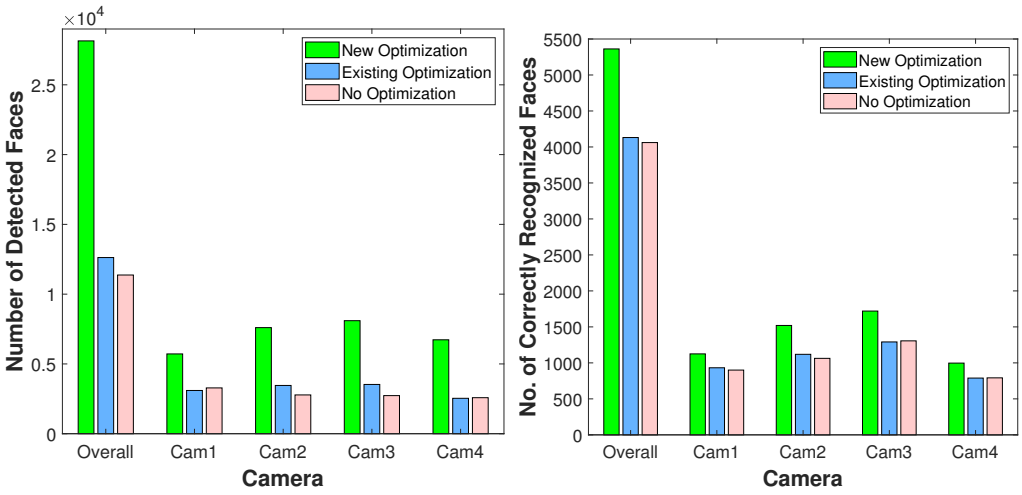
Figure 3.8: Comparing Various Solutions for All Video Categories and Each Category [Experimental Setup I]



(a) Combined Cameras 1-4 (IP7210W) (b) Combined Cameras 5 and 6 (Virtualized Web-cams) (c) Camera 7 (VivoTek IP7139)

Figure 3.9: Comparing Various Solutions in Detection Accuracy Using the Entire Video Data Set [Experimental Setup I]

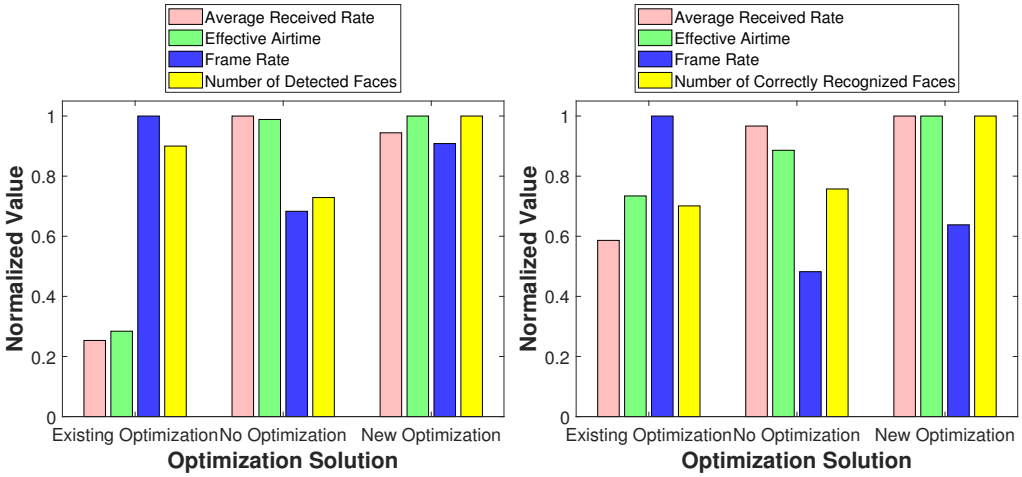
The proposed solution holds the lead in the number of correctly recognized faces, which is the most important metric. By balancing the average received frame rate, it significantly improves the effective airtime and average received rate, resulting in the capture of high-quality frames and the highest face recognition accuracy. The existing solution has the second-best results, with low medium bandwidth usage. Like Experimental Setup I, the existing optimization becomes a viable choice only when the power consumption



(a) Detection Accuracy in the Live Lab Environment (Experimental Setup II)

(b) Recognition Accuracy (Setup III)

Figure 3.10: Comparing Various Solutions in Accuracy



(a) Experimental Setup I

(b) Experimental Setup III

Figure 3.11: Relationships Among Different System Metrics

is of utmost significance and preferred over accuracy. The number of correctly recognized faces when the optimization is disabled is lower than the two optimization solutions; however, the improvement by using the existing optimization is marginal.

3.5 Conclusions

We have built a real computer vision system for automated video surveillance and have analyzed extensive results of the actual experiments using different video datasets as well as in a live laboratory environment. This work is published in [24]. The main results can be summarized as follows. (1) Cross-layer optimization in CV systems is highly effective in improving the detection/recognition accuracy. (2) By optimally distributing the available medium bandwidth and increasing the effective medium airtime, the system can successfully deliver high-quality video streams at high frame rates to the monitoring station. (3) The proposed optimization solution significantly enhances face detection and face recognition accuracy. (4) By properly assessing the overall data corruption and dropping rate through bitrate smoothing, the proposed effective airtime estimation algorithm achieves high accuracy. (5) The highest detection/recognition accuracy is achieved when the packet-dropping and error rates are very small. (6) A distributed processing system, or one with powerful GPUs, is required for the real-time detection of threats when a large number of video sources are supported.

CHAPTER 4 ENHANCED YOLO SOLUTION

4.1 Introduction

The first step in the proposed video input optimization solution is motion detection. Motion detection is also referred to as background subtraction (BS), which is an essential function in computer vision applications such as moving vehicles/people detection, multimedia applications, and video surveillance. BS essentially involves the comparison of an image with another image, which is an estimation of the background model. Moving foreground objects can be found in the image regions that have a significant difference from the reference image (background model). The BS process generally consists of three tasks: 1) background model initialization, 2) background model maintenance, and 3) foreground segmentation. Figure 4.1 shows the diagram of the BS process mentioned here.

The second step in our proposed video input optimization solution is clustering of the foreground points detected by the motion detection techniques described earlier. Clustering in general is the task of organizing a set of objects into groups or clusters, based on their shared similarities in some selected characteristics. This is a common technique for statistical data analysis and is employed in many fields, including pattern recognition, image analysis, information retrieval, bioinformatics, data compression, computer graphics, and machine learning.

Clustering can be done by various algorithms that differ significantly in what they define as a cluster and how to efficiently form them. One main notion of clusters is groups with small distances between cluster members. Clustering can be formulated as a multi-objective optimization problem. The appropriate clustering algorithm and parameter settings depend on the individual data set and intended use of the results. The parameters include, among others, the distance function to use, a density threshold, or the number of expected clusters. Modifying data preprocessing and model parameters should often be considered until the results demonstrate the desired properties. Although the problem of clustering is an NP-hard problem,

efficient heuristic algorithms converge quickly to a local optimum.

In our proposed solution, clustering is employed to determine a rectangular area in the current frames containing the dynamic objects in the scene. The input is a list of pixels that are classified as foreground and the output is a rectangular region of interest that will be fed to the YOLOv4 network to perform the object detection task.

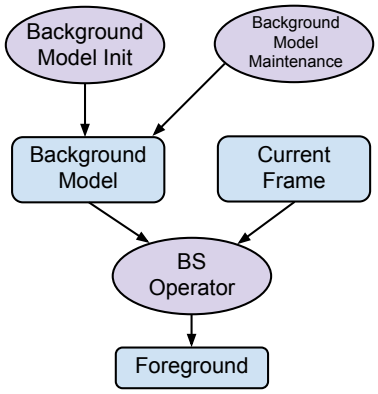


Figure 4.1: Background Subtraction Process Overview

4.2 Brief Description of Background Subtraction/Motion Detection Techniques

There are many background subtraction/motion detection techniques including Adaptive Background Learning [63, 60], Adaptive Selective Background Learning [63], CodeBook [39], Frame Difference [60], Local Binary Similarity Segmenter (LOBSTER) [65], Mixture Of Gaussian V2 [80], Pixel-based Adaptive Word Consensus Segmenter (PAWCS) [67], SigmaDelta [45], Static Frame Difference [63], Flexible Background Subtraction with Self-Balanced Local Sensitivity (SuBSENSE) [66], TwoPoints [3], ViBe [13], Weighted Moving Mean [63], and Weighted Moving Variance [63]. We examine these techniques, starting from Adaptive Background Learning, and measure their execution complexity and their performance in suitable videos that are common in our proposed system.

4.2.1 Adaptive Background Learning

This technique computes the average of the previous N frames to create the background. This means that to update the first background image, it considers new static objects in the video. The background image calculation is done using equation 4.1.

$$B_{t+1}(x, y) = 1/N \times \sum_{T=t-N}^{t} I_T, \tag{4.1}$$

Eq. 4.1 suggests that this method consumes a large amount of memory, which is not ideal for real-time implementations of the technique. To avoid this issue, it is better to compute the background using equation 4.2, where $\alpha \in [0, 1]$ is a constant that specifies how effective new information changes old observations.

$$B_{t+1}(x, y) = (1 - \alpha) \times B_t(x, y) + \alpha \times I_t(x, y), \tag{4.2}$$

Large values of α lead to the higher rates at which the background image is updated with new information in the video. If α is set too large it may lead to tail artifacts behind moving objects. The α value should be determined according to the observed scene and the size, speed, and distance of the moving objects from the camera to prevent tail artifacts. The issue of constant movement in small background objects, especially in outdoor environments (e.g., tree branches in a breeze), can be addressed by segmenting such objects with the moving objects.

Figure 4.2 provides an overview of the process in Adaptive Background Learning technique.

4.2.2 Adaptive-Selective Background Learning

The main advantage of Adaptive Background Learning is the dynamic updating of the background image while changes happen in the video. However, some foreground pixels tend to be included in the background image updating process. To counter this issue, an adaptive-selective algorithm is proposed wherein regions

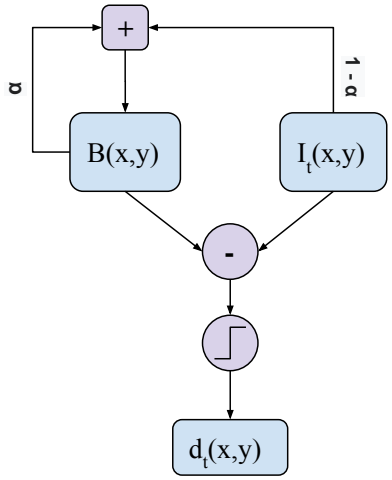


Figure 4.2: Overview of Adaptive Background Learning

with no moving pixels are only considered for the background updating process. This method is referred to as Adaptive-Selective Background Learning.

4.2.3 Codebook

This background subtraction/motion detection technique samples values over long periods while avoiding parametric assumptions. This technique has an adaptive background model that is capable of handling structural background motion over time while using a limited amount of memory. This model can also cope with illumination changes in the scene, either locally or globally. Additionally, this method allows moving foreground objects in the video during the initial training period where the background model is constructed. Codebook technique allows layered modeling and detection, which makes it possible to have multiple layers of background each representing a different background layer.

4.2.4 Frame Difference

The Frame Difference technique is one of the simplest ways to detect changes in pixel intensity in video frames. In a gray-scale frame, for each pixel with (x, y) coordinates in frame I_{t-1}, the absolute difference

with its corresponding pixel in the next frame I_t is calculated using:

$$d(x, y) = |(I_{t-1}(x, y) - I_t(x, y)|, \tag{4.3}$$

In an RGB image (colored image), this difference could be calculated by different methods, like Manhattan distance (4.4), where $I_t^C(x, y)$ is the pixel value/intensity in the C channel.

$$d(x, y) = |(I_{t-1}^R(x, y) - I_t^R(x, y)| + |(I_{t-1}^G(x, y) - I_t^G(x, y)| + |(I_{t-1}^B(x, y) - I_t^B(x, y)|, \tag{4.4}$$

Despite its simplicity, this method offers many advantages. It performs well in dynamic video scenes and operates quickly by satisfying the standard video frame rate. Additionally, the technique could be implemented easily, and has relatively low design complexity. This makes Frame Difference suitable for real-time systems.

4.2.5 Local Binary Similarity Segmenter (LOBSTER)

This spatiotemporal-based background subtraction/motion detection technique is based on the adaptation and integration of the Local Binary Similarity Patterns (LBSP) to a set of rules about model building and maintenance. This method starts off from the original ViBe [13] approach because it offered the prospect of a flexible method with potential for future improvements. The straightforward approach for this adaptation was to replace all pixel intensity-related concepts with their feature descriptor-based counterparts. As with ViBe, this method is based on a reference model that uses N background samples per pixel to independently determine which new pixels are foreground/moving. The difference is in the nature of these samples, which is replaced by LBSP binary string descriptors. To calculate the difference between the background model and the current frame, a Hamming distance operator is utilized. Additionally, there were multiple low-cost improvements to both the model's rules and the feature descriptor to improve performance.

4.2.6 Mixture of Gaussian V2

This method is an improvement on the background subtraction algorithm introduced by Stauffer and Grimson [69]. Instead of explicitly modeling the values of the pixels as a single particular type of distribution, they model the values of a particular pixel as a mixture of Gaussians. Using the persistence and the variance of each of the Gaussians in the mixture, they determine which Gaussians may be related to background colors. Pixel values that do not fit the background distributions will be categorized as foreground. Foreground pixels will be considered background when there is a Gaussian model that includes them. Their system handles the lighting changes, repetitive motions of elements in the scene, slow-moving objects, and adding or removing objects from the video frames, robustly. In this technique, slow-moving objects tend to take longer to be merged into the background. This is because their color has a greater variance than the background colors. Additionally, repetitive variations are learned, and a model for the background distribution is generally maintained even if it is temporarily replaced by another distribution. This method has two important parameters: the learning constant and the proportion of the data that should be accounted for by the background. Without needing to change the default value of these parameters, their proposed algorithm shows good performance in outdoor and indoor scenes.

4.2.7 Pixel-based Adaptive Word Consensus Segmenter (PAWCS)

This background subtraction/motion detection method can be utilized in a large variety of scenes without the need to manually readjust parameters. This method has a persistence-based word dictionary scheme for instance-based background modeling. Unlike the previously mentioned Codebook or other sample-sensitive methods, this non-parametric background modeling policy leads to online principled learning of static and dynamic background patches by having a low memory footprint. This is because it dynamically updates the minimal number of background samples (or words) that is required to properly categorize all the pixels in the scene. A word is an element consisting of RGB values and other items like brightness and the last

access time for that word. Persistence estimation is used to measure the importance of each background word over time by incorporating local match counts. Persistence values have an effect on the rate at which each word is updated. PAWCS requires no explicit training phase to generate its background models, and it keeps updating the models while processing new video frames.

This technique improves segmentation coherence by spreading information between neighboring pixel models. Additionally, it allows the capture of large-scale background change patterns. This method also automatically adjusts its primary parameters by incorporating closed-loop controllers into each pixel-level model. That way, each background region can exhibit its own modeling and classification behavior, which can also evolve over the analyzed sequences. Primary parameters are automatically adjusted and are regulated by monitoring multiple factors including segmentation noise; similarity between background models and new frames; and region instability by considering the frequency of label changes.

4.2.8 SigmaDelta ($\sum -\triangle$)

The first step of the SigmaDelta ($\sum -\triangle$) method is to compute $\sum -\triangle$ mean, after which the difference between the image and the $\sum -\triangle$ mean is calculated. The latter value is also referred to as motion likelihood measure. The next step is to calculate the $\sum -\triangle$ variance which is defined as the $\sum -\triangle$ mean of N times the non-zero differences. As the interest is in the pixels with a variation rate significantly higher than their temporal activity, the difference is multiplied by N. Finally, the motion label is produced using the comparison between the difference and the variance.

This algorithm is presented in Figure 4.3.

4.2.9 Static Frame Difference

This technique, also referred to as the basic model, manually sets a static image that represents the background. This image does not have any moving object. For each video frame, the absolute difference between the current frame and the static background image is calculated. A static image is not the best choice, if the

Step 1:
Initialization
  For each pixel x:
    $M_0(x) = I_0(x)$
For each frame t:
  For each pixel x:
    $M_t(x) = M_{t-1}(x) + sgn(I_t(x) - M_{t-1}(x))$
Step 2:
For each frame t:
  For each pixel x:
    $\triangle_t(x) = |M_t(x) - I_t(x)|$
Step 3:
Initialization
  For each pixel x:
    $V_0(x) = \triangle_0(x)$
For each frame t:
  For each pixel x such that $\triangle_t(x) \neq 0$:
    $V_t(x) = V_{t-1}(x) + sgn(N \times \triangle_t(x) - V_{t-1}(x))$
Step 4:
For each frame t:
  For each pixel x:
    if $\triangle_t(x) < V_t(x)$:
      then $D_t(x) = 0$
    else $D_t(x) = 1$

Figure 4.3: Simplified Algorithm for $\sum -\triangle$ Motion Estimation Technique

ambient lighting changes, then the foreground segmentation may fail dramatically. It is possible to solve this issue by using the previous frame rather than a static image. This enhanced technique is referred to as Frame Difference, which works with some background changes but has a weak performance if the moving object stops suddenly.

4.2.10 Flexible Background Subtraction with Self-Balanced Local Sensitivity (SuBSENSE)

This approach relies on the automatic adjustment of parameters, in addition to updating and pixel labeling rules for a non-parametric model. The goal is to achieve optimal segmentation results for different types of scenarios. Color and Local Binary Similarity Patterns (LBSP) is the basis for change detection in pixel values and is done by using spatiotemporal analyses [16, 68]. This leads to increased sensitivity for the detection of changes in pixel values. This method's flexibility is a result of its automatic adjustments of local sensitivity. Decision thresholds and state variables are adjusted by pixel-level feedback loops.

This method allows for the identification and isolation of areas where segmentation is more difficult. It is also capable of achieving excellent overall performance in difficult scenarios. The processing speed of SuBSENSE is still acceptable for real-time applications, although it is generally more expensive than other motion detection techniques.

4.2.11 TwoPoints

This method uses three frames to separate background and foreground pixels. These three frames are the current and the previous two frames, named history frames 1 and 2. The first step is the calculation of difference between the previous two frames. The result is used as a threshold value to determine moving pixels by comparing the current frame against history frame 1 and history frame 2. The result from each comparison is accumulated in the final output.

4.2.12 ViBe

This motion detection technique can be initialized with a single frame, eliminating the need to wait for several frames to initialize the background model. This is an advantage for image processing solutions embedded in digital cameras, which are required to work with short sequences. Instead of keeping samples in the pixel models for a fixed amount of time, the insertion time of a pixel in the model is ignored and a value is selected to be replaced randomly, resulting in a smooth fading lifespan for the pixel samples.

Additionally, this enables the technique to generate an efficient result for wider ranges of background changing rates and simultaneously reduces the required stored number of samples needed for each pixel model. The spatial consistency of the background model is guaranteed by allowing samples to diffuse between neighboring pixel models. This makes it more resilient to camera motions, while simultaneously eliminating the need to post-process segmentation maps in order to produce spatially coherent results. There is a strictly conservative update scheme in this method which dictates that no foreground pixel value should ever be merged into any background model.

4.2.13 Weighted Moving Mean

In this motion detection algorithm, the foreground/moving pixels is/are calculated using the following steps:

First, the weighted average of the previous l frames is calculated by Equation 4.5

$$Mean_l = \frac{1}{l}\sum_{t=1}^{l} W_t \times F_t \tag{4.5}$$

F_t is the frame at timestamp t in the video; W_t is the considered weight for each frame, which is generally higher for frames closer to the current frame being processed for foreground extraction (frame F_l).

Then the foreground is calculated by employing Equation 4.6.

$$Foreground_l = \sqrt{\sum_{t=1}^{l}(W_t \times |F_t - Mean_t|^2)} \tag{4.6}$$

The main advantage of this method is the adaptive maintenance of the background model while changes occur in the scene.

4.2.14 Weighted Moving Variance

This motion detection algorithm is similar to the Weighted Moving Mean approach. The difference is that it incorporates a weighted moving variance to directly calculate the foreground pixels.

4.3 Execution Complexity and the Visual Performance of Background Subtraction/Motion Detection Techniques

Figure 4.4 shows the execution time of different background/motion detection techniques for five different videos recorded from live street/traffic cameras in different cities. Figure 4.4(f) displays the average execution time over all the video files. As can be observed in this figure, three methods, namely LOBSTER,

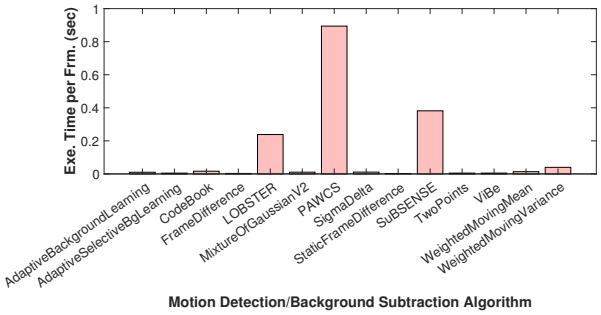
PAWCS, and SuBSENSE, are significantly more time-consuming and complex to execute than the rest of demonstrated methods. Therefore, these methods are not appropriate for use in our solution as they are more time-consuming than running the high-resolution YOLOv4 neural network on the input images without the application of any pre-processing. Thus, we have removed these methods from Figure 4.4(g) to properly display the average execution time per frame for other motion detection methods as the scaling is appropriate with the exclusion of the mentioned complex methods.

As demonstrated in this figure, all the methods except for Weighted Moving Variance are fairly light to execute and could reach frame rates of over 50fps running on a single thread on a regular CPU. This translates to easy implementation of these methods on edge devices without throttling the operating frame rate of these devices.

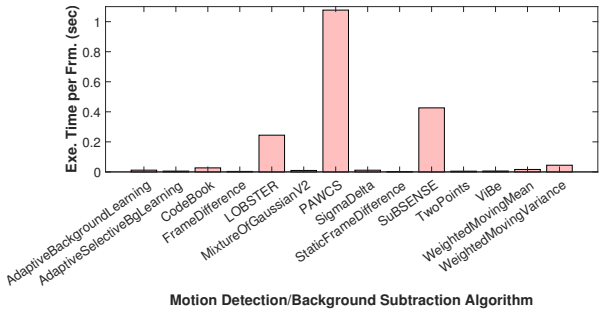
Figures 4.5, 4.6, 4.7, 4.8, and 4.9 illustrate the results/output of different motion detection algorithms on different input videos from live street/traffic cameras in multiple cities. In these figures, Subfigure x.a shows the original frame used as an input for the motion detection algorithm while the other subfigures show the output of different motion detection algorithms, with black pixels representing background and white pixels repressing foreground/motion.

The detected foregrounds in Weighted Moving Variance and Weighted Moving Mean methods mostly consist of edge pixels of the moving objects in the scene. This can be seen in Figures 4.5(b), 4.5(c), 4.6(b), 4.6(c), 4.7(b), 4.7(c), 4.8(b), 4.8(c), 4.9(b), and 4.9(c). The Weighted Moving Mean method seems to be less sensitive in generating the foreground image than the Weighted Moving Variance method.

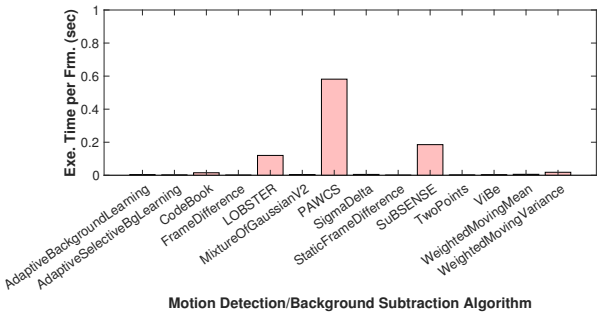
The foreground pixels in the ViBe method consist of the edge and internal pixels of the moving objects. By looking at Figures 4.5(d), 4.6(d), 4.7(d), 4.8(d), and 4.9(d), it can be observed that this method is more sensitive to moving pixels in comparison to Weighted Moving Average and Weighted Moving Variance and tends to produce more scattered white pixels in the results. The TwoPoint method behaves similarly to the



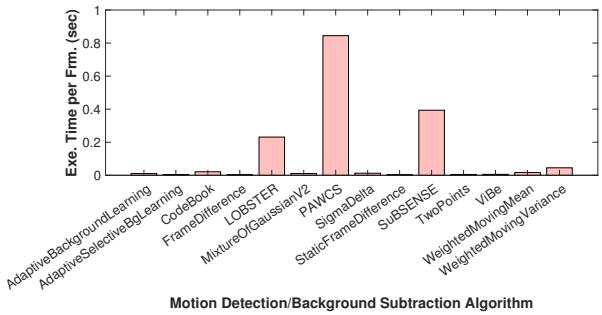
(a) Lamai, Koh Samui, Thailand



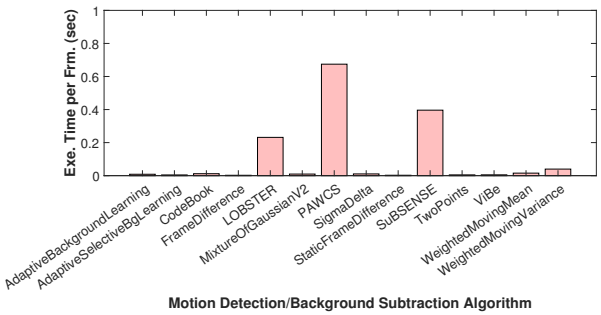
(b) Saint Petersburg, Russia



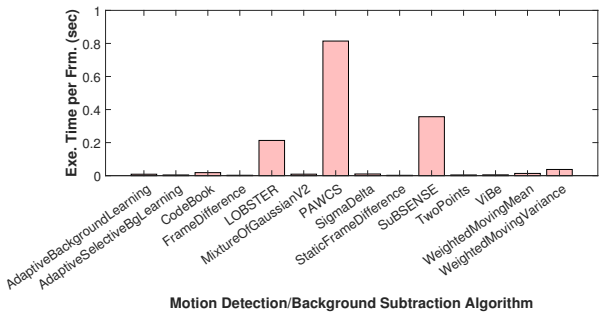
(c) New Orleans, Louisiana, United States



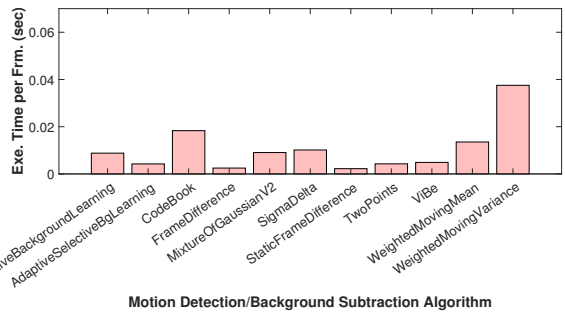
(d) Laramie, Wyoming, United States



(e) Renesse, Netherlands



(f) Average of all locations



(g) Average of all locations (without the computationally complex motion detection methods)

Figure 4.4: Average Execution Time per Frame for Different Motion Detection Algorithms, Considering Different Input Videos

Vibe method, but has considerably more white scattered pixels around the scene when no desired moving object exists. This fact can be observed by looking at Figures 4.5(e), 4.6(e), 4.7(e), 4.8(e), and 4.9(e).

The results from SuBSENSE, PAWCS, and LOBSTER motion detection techniques have clean and clearly defined patches of white pixels, with sharp edges, for the moving objects in the video frame. This is observable in Figures 4.5(f), 4.6(f), 4.7(f), 4.8(f), 4.9(f), 4.5(i), 4.6(i), 4.7(i), 4.8(i), 4.9(i), 4.5(k), 4.6(k), 4.7(k), 4.8(k), and 4.9(k). However, it should be noted that these three methods are extensively expensive to execute and would not be suitable for our proposed enhanced detection system; the LOBSTER method has more defined edges in comparison to SuBSENSE and PAWCS methods.

The Static Frame Difference method has loosely similar results to the TwoPoints method but with a larger amount of scattered white pixels around the scene. This can be seen in Figures 4.5(g), 4.6(g), 4.7(g), 4.8(g), and 4.9(g).

Trailing white pixels behind moving objects can be observed in the SigmaDelta motion detection results. By looking at Figures 4.5(h), 4.6(h), 4.7(h), 4.8(h), and 4.9(h) it can be deducted that the performance of the SigmaDelta method is similar to the TwoPoints method but with less scattered white pixels around the scene.

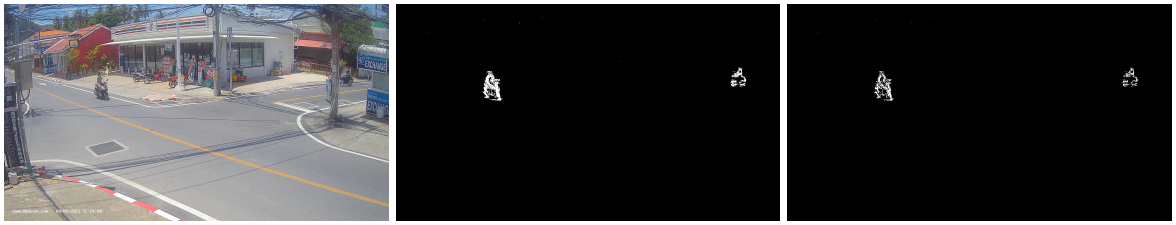
The Mixture of Gaussian V2 method has the most fluctuating performance in producing the foreground image, depending on the scene from the input video file. By comparing Figures 4.5(j), 4.6(j), 4.7(j), 4.8(j), and 4.9(j), it can be observed that this method on occasion produces a highly noisy image with a significant amount of scattered white pixels, while in other cases results in images with barely any white pixels visible in the scene.

The Frame Difference method (Figs. 4.5(l), 4.6(l), 4.7(l), 4.8(l), and 4.9(l)) has a performance similar to Weighted Moving Mean, while mostly being less sensitive to moving objects (less defined edges on the foreground image) and showing a higher number of scattered white pixels.

The highest number of scattered white pixels in the generated foreground images can be observed in the results collected from the CodeBook motion detection method. Figures 4.5(m), 4.6(m), 4.7(m), 4.8(m), and 4.9(m) show how noisy the results are compared to the results from other methods.

Adaptive Selective Background Learning has well-defined patches of white pixels, representing the moving objects in the scene; however, this method is prone to showing foreground pixels erroneously selected from previous frames. This causes a ghost object to be present in some results, chasing the moving object. Figures 4.5(n), 4.6(n), 4.7(n), 4.8(n), and 4.9(n) display the output of this method generated from different input video files.

Finally, Adaptive Background Learning (Figs. 4.5(o), 4.6(o), 4.7(o), 4.8(o), and 4.9(o)) can be observed to have the highest number of trailing white pixels dragging around the moving objects.



(a) Input Frame

(b) Weighted Moving Variance

(c) Weighted Moving Mean

(d) ViBe

(e) TwoPoints

(f) SuBSENSE

(g) Static Frame Difference

(h) SigmaDelta

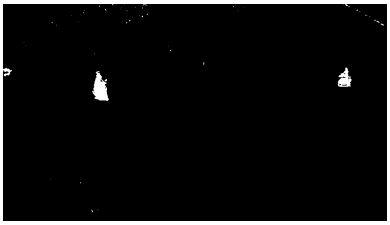
(i) PAWCS

(j) Mixture Of Gaussian V2

(k) LOBSTER

(l) Frame Difference

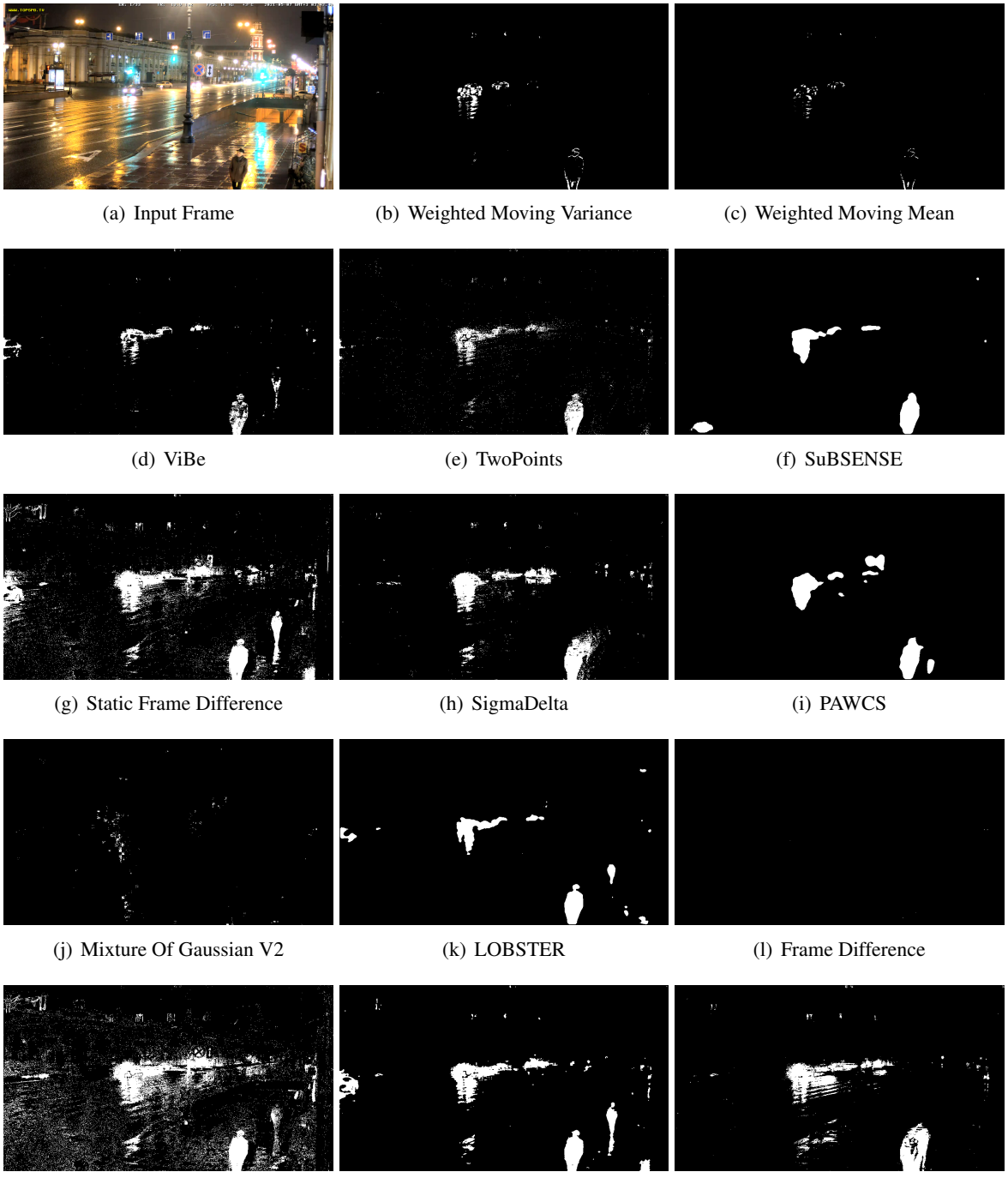
(m) CodeBook



(n) Adaptive Selective Background Learning

(o) Adaptive Background Learning

Figure 4.5: Results from Motion Detection Algorithms, Running on the Input Video from Lamai, Koh Samui, Thailand



(a) Input Frame (b) Weighted Moving Variance (c) Weighted Moving Mean

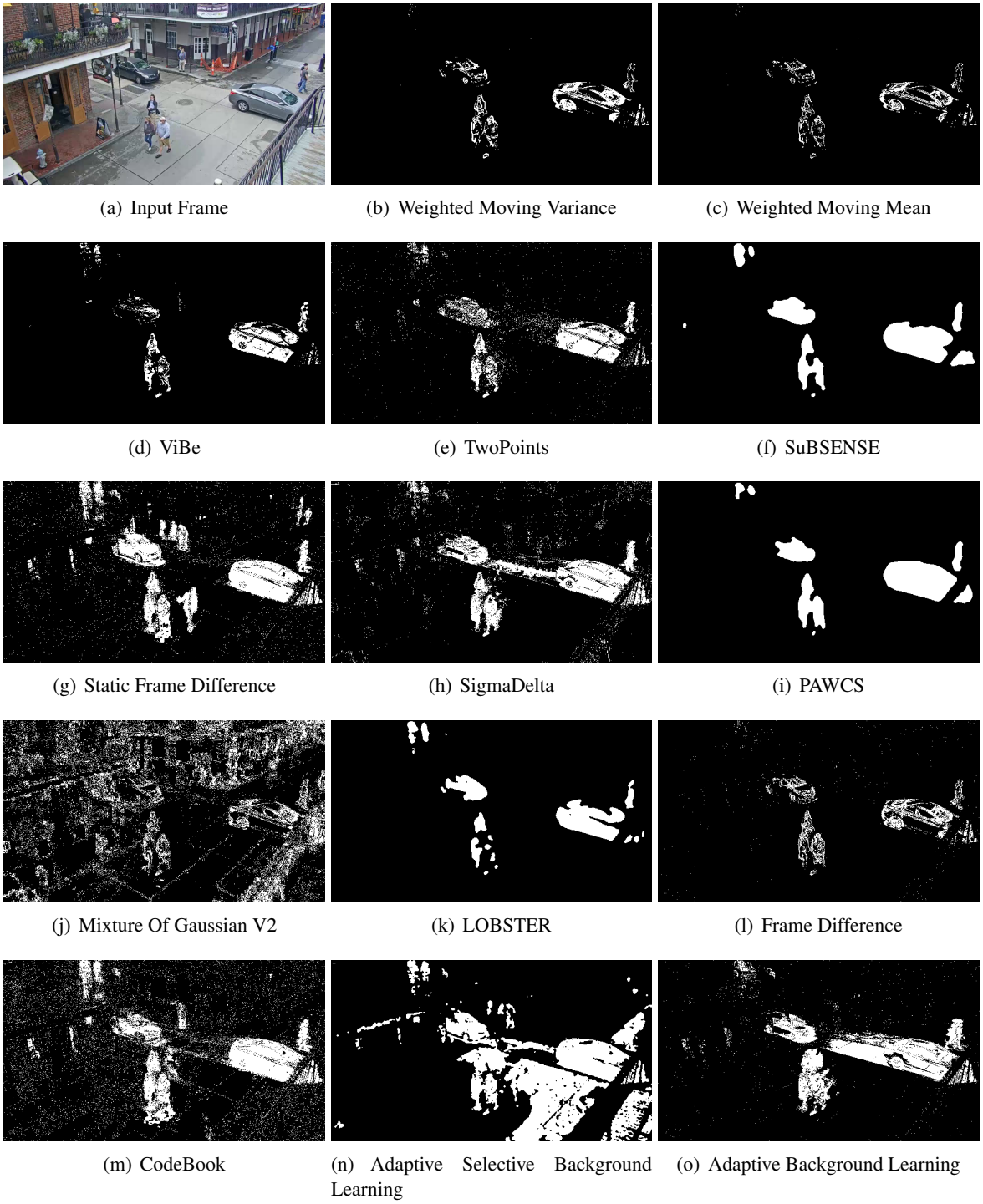
(d) ViBe (e) TwoPoints (f) SuBSENSE

(g) Static Frame Difference (h) SigmaDelta (i) PAWCS

(j) Mixture Of Gaussian V2 (k) LOBSTER (l) Frame Difference

(m) CodeBook (n) Adaptive Selective Background Learning (o) Adaptive Background Learning

Figure 4.6: Results from Motion Detection Algorithms, Running on the Input Video from Saint Petersburg, Russia



(a) Input Frame (b) Weighted Moving Variance (c) Weighted Moving Mean

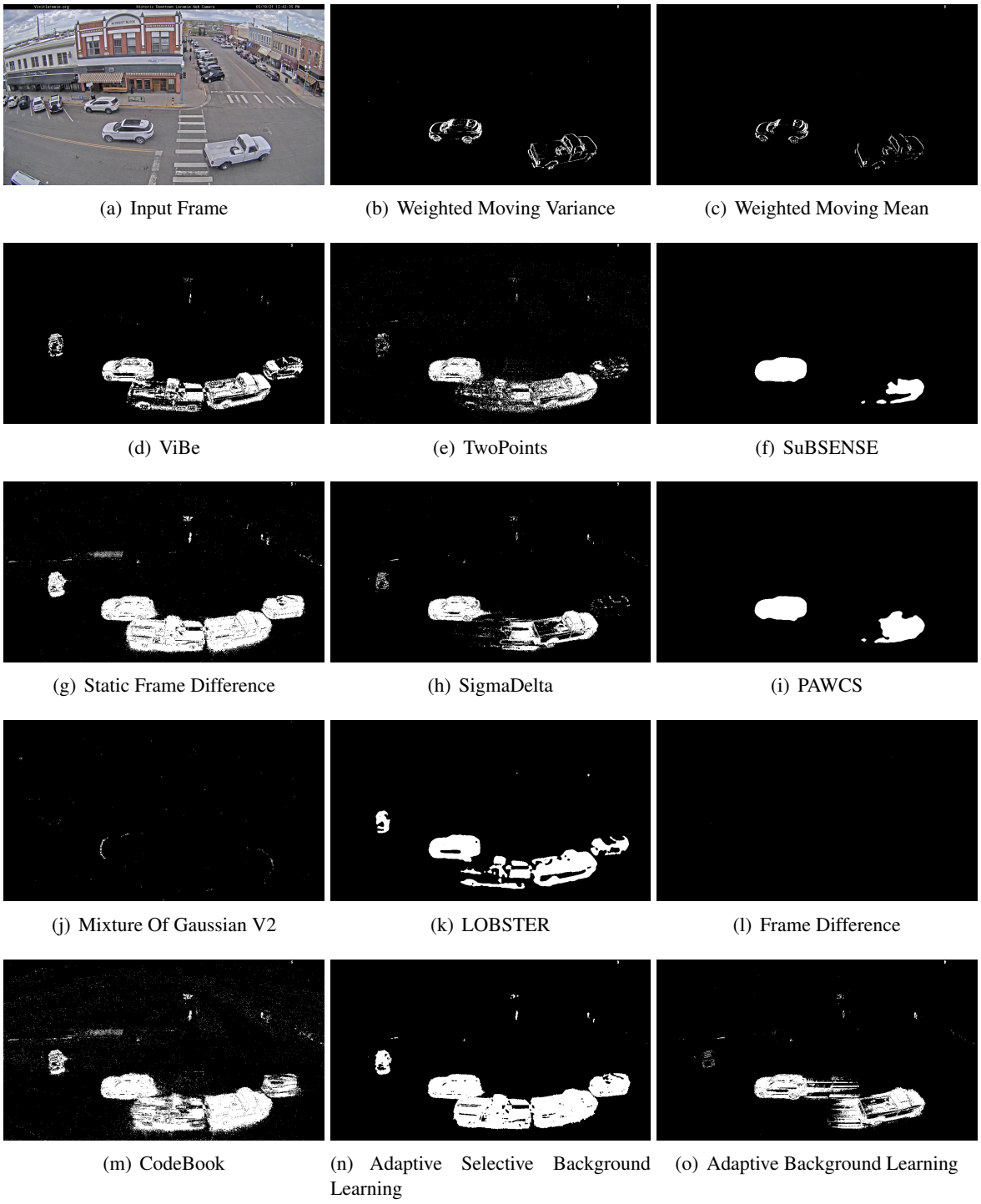
(d) ViBe (e) TwoPoints (f) SuBSENSE

(g) Static Frame Difference (h) SigmaDelta (i) PAWCS

(j) Mixture Of Gaussian V2 (k) LOBSTER (l) Frame Difference

(m) CodeBook (n) Adaptive Selective Background Learning (o) Adaptive Background Learning

Figure 4.7: Results from Motion Detection Algorithms, Running on the Input Video from New Orleans, Louisiana, United States



(a) Input Frame (b) Weighted Moving Variance (c) Weighted Moving Mean

(d) ViBe (e) TwoPoints (f) SuBSENSE

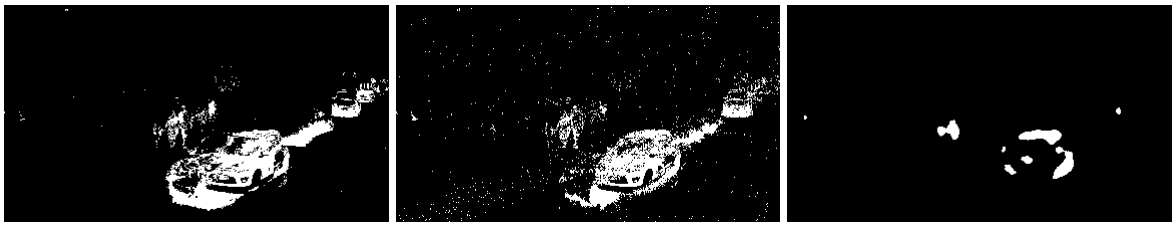
(g) Static Frame Difference (h) SigmaDelta (i) PAWCS

(j) Mixture Of Gaussian V2 (k) LOBSTER (l) Frame Difference

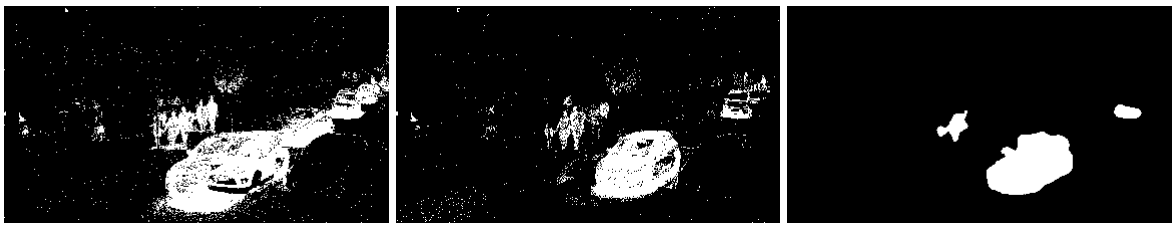
(m) CodeBook (n) Adaptive Selective Background Learning (o) Adaptive Background Learning

Figure 4.8: Results from Motion Detection Algorithms, Running on the Input Video from Laramie, Wyoming, United States

(a) Input Frame (b) Weighted Moving Variance (c) Weighted Moving Mean

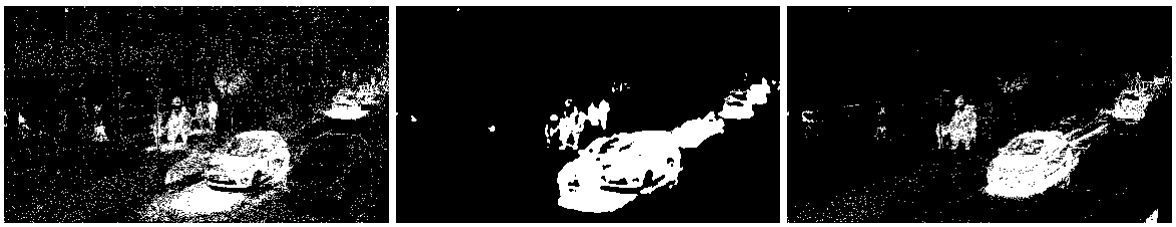


(d) ViBe (e) TwoPoints (f) SuBSENSE



(g) Static Frame Difference (h) SigmaDelta (i) PAWCS

(j) Mixture Of Gaussian V2 (k) LOBSTER (l) Frame Difference



(m) CodeBook (n) Adaptive Selective Background Learning (o) Adaptive Background Learning

Figure 4.9: Results from Motion Detection Algorithms, Running on the Input Video from Neath, Wales

4.4 Brief Description of Clustering Techniques

We are considering many clustering techniques including KMeans [11], Affinity Propagation [28], Mean Shift [23], Spectral [44], Agglomerative [48], DBSCAN [27], OPTICS [9], Birch [79], and MiniBatch KMeans [59]. We will examine these techniques, starting from KMeans, and measure their execution complexity and their performance in suitable videos that are common in our proposed system. By using a single motion detection algorithm and running our application and then measuring the execution time and recording the output from different clustering algorithms, this undertaking could be accomplished. We have selected the Frame Difference motion detection technique as the fixed algorithm in performing the related experiments.

4.4.1 KMeans

k-means clustering is a method that aims to partition n observations/points into k clusters. Each observation/point belongs to the cluster with the nearest mean. The mean is also called the cluster center or cluster centroid. This result of clustering is a partitioning of the data space into Voronoi cells. k-means clustering minimizes within-cluster variances.

The problem of clustering is an NP-hard problem; however, efficient heuristic algorithms converge quickly to a local optimum.

The most common implementation of this algorithm is an iterative refinement technique. This technique is also referred to as Lloyd's algorithm, particularly in the computer science community.

Given an initial set of k means $m_1^{(1)}$,..., $m_k^{(1)}$, the algorithm proceeds by alternating between two steps [26]:

Assignment step: Assign each point to the cluster with the nearest mean/center; the one with the least squared Euclidean distance. Mathematically, this translates to partitioning the points according to the Voronoi diagram generated by the means/centers.

$$S_i^{(t)} = \left\{ x_p : \left\| x_p - m_i^{(t)} \right\|^2 \leq \left\| x_p - m_j^{(t)} \right\|^2 \ \forall j, 1 \leq j \leq k \right\},$$

where each x_p is assigned to exactly one $S^{(t)}$, even if it could be assigned to two or more of them.

Update step: Recalculate means (centroids) for observations assigned to each cluster.

$m_i^{(t+1)} = \frac{1}{\left|S_i^{(t)}\right|} \sum_{x_j \in S_i^{(t)}} x_j$

The algorithm is converged when the assignments no longer change. The algorithm is not guaranteed to find the optimum solution.

4.4.2 Affinity Propagation

This clustering algorithm is based on the concept of message passing between data points [28]. This method does not require the number of clusters to be determined or estimated before running the algorithm. This is unlike other clustering algorithms such as k-means. Affinity propagation finds exemplar members of the input set that are representative of clusters.

The algorithm proceeds by alternating between two message-passing steps, which update two matrices. The responsibility matrix R has values $r(i,k)$ that quantify how well suited data point x_k is to serve as the exemplar for data point x_i, relative to other candidate exemplars for data point x_i. The availability matrix A contains values $a(i,k)$ that represent how appropriate it would be for data point x_i to pick data point x_k as its exemplar, taking into account other points' preference for x_k as an exemplar. Both matrices are initialized to all zeroes and could be considered as log-probability tables.

Iterations are executed until either the cluster boundaries remain unchanged, or some predetermined number of iterations is reached.

4.4.3 MeanShift

MeanShift clustering aims to discover blobs in a smooth density of data points. It is a centroid-based algorithm and operates by updating candidates for centroids to be the mean of the points inside a given region. These candidates are then filtered in a post-processing step to remove near-duplicates to produce the

final set of centroids.

Given a candidate centroid x for iteration t, the candidate is updated according to the following equation:

$x^{t+1} = m(x^t)$

Where m is the weighted mean of the density in the window and is calculated using the following equation: $m(x) = \frac{\sum_{x_i \in N(x)} K(x_i - x)x_i}{\sum_{x_i \in N(x)} K(x_i - x)}$ where K function measures the distance to the current estimate and $N(x)$ is the neighborhood of centroid x, a set of points for which $K(x_i) \neq 0$. The mean-shift algorithm now sets $x \leftarrow m(x)$ and repeats the estimation until m(x) converges.

4.4.4 Spectral

Spectral clustering is a popular algorithm due to its simple implementation and performance in many graph-based clustering applications. It can be solved efficiently by standard linear algebra software and is mostly capable of outperforming traditional algorithms such as the k-means. Spectral clustering is performed in these main steps:

- Create a similarity graph between the N data points to cluster.
- Compute the Laplacian L (or the normalized Laplacian) of the graph.
- Compute the first k eigenvectors (the eigenvectors corresponding to the k smallest eigenvalues of L). This defines a feature vector for each data point.
- Cluster the graph nodes based on these features.

4.4.5 Agglomerative

This is a hierarchical clustering algorithm. These clustering algorithms build nested clusters by merging or splitting them successively. These clusters could be represented as a tree. The root of the tree is the main cluster that gathers all the samples. The leaves in this tree are the clusters with only one data point. This

clustering method performs a hierarchical clustering using a bottom-up approach. Each data point starts as a single cluster, and clusters are successively merged together.

Different metrics could be employed for the merge strategy, including minimization of the sum of squared differences within all clusters. This is a variance-minimizing approach and it is similar to the k-means objective function but tackled with an agglomerative hierarchical approach.

Agglomerative clustering is capable of scaling to a large number of samples/data points when used jointly with a connectivity matrix, although it is computationally expensive when no connectivity constraints are considered between data points. If there is no connectivity constraint, it considers at each step all the possible merges.

4.4.6 DBSCAN

The DBSCAN algorithm assumes clusters are areas of high density separated by areas of low density. This generic assumption will lead to clusters found by DBSCAN to be of any shape. This is as opposed to k-means which assumes that clusters are convex shaped. The main component of the DBSCAN is the concept of core samples, which are samples that are in high density regions. A cluster is a set of core samples that are close to each other and a set of non-core samples which are close to a core sample but are not themselves core samples. Two parameters are configurable for this algorithm, *min samples* and *eps*. These two parameters formally define how dense the data points/samples should be in a cluster. Higher *min samples* or lower *eps* indicate higher density necessary to form a cluster.

Formally, a core sample is a sample in the data set wherein there exist *min samples* other samples within a distance of *eps*. These are called neighbors of the core sample. This indicates the core sample is in a dense area of the vector space. A cluster is a set of core samples that is built by recursively picking a core sample, finding all its neighbors that are core samples, finding all their neighbors that are core samples, etc. A cluster also has a set of non-core samples. These are samples that are neighbors of a core sample in the

cluster but are not core samples themselves. These samples are on the edges of a cluster.

By definition, any core sample is part of a cluster. Any sample/data point that is not a core sample, while being at least *eps* in distance from any core sample, is considered an outlier by the algorithm.

The *min samples* primarily controls how tolerant the algorithm is with respect to noise. The *eps* parameter is crucial to choose appropriately for the data set and the employed distance function and mostly should not be left at the default value. When chosen too small, most data points will not be clustered and will be considered as noise. When chosen too large, close clusters will be merged into a single cluster, and eventually, the entire data set will be considered as a single cluster.

4.4.7 OPTICS

OPTICS is short for Ordering Points to Identify Cluster Structure. This clustering method is closely related to the DBSCAN clustering algorithm. It adds two more terms to the concepts of DBSCAN clustering. The first one is the core distance, which is the minimum value of radius required to classify a given point/sample as a core point. If the given point is not a core point, then its core distance is undefined. The second term is reachability distance, which is defined with respect to another data point q. The reachability distance between a point p and a point q is the maximum of the core distance of p and the distance between p and q (Euclidean distance or some other distance metric). It should be noted that the reachability distance is not defined if q is not a core point.

This technique does not explicitly segment the data into clusters. Instead, it produces a visualization of reachability distances then uses this visualization to cluster the samples. This makes it different from other clustering techniques.

4.4.8 BIRCH

The BIRCH clustering algorithm stands for Balanced Iterative Reducing and Clustering using Hierarchies. It builds a tree called the Clustering Feature Tree (CFT) for the given data/samples. By using this

algorithm, the data is lossy compressed to a set of Clustering Feature nodes (CF Nodes). The CF nodes have a number of subclusters called Clustering Feature (CF) subclusters, which can have CF nodes as children.

The CF subclusters hold the required information for clustering. This eliminates the need to hold the entire input data in memory. This information includes:

- Number of samples in a subcluster.
- Linear Sum: an n-dimensional vector holding the sum of all samples.
- Squared Sum: sum of the squared L2 norm of all samples.
- Centroids: this helps avoid the recalculation of linear sum/number of samples.
- Squared norm of the centroids.

The BIRCH algorithm has two parameters. The first parameter is the branching factor which limits the number of subclusters in a node. The second parameter is the threshold which limits the distance between the entering sample and the existing subclusters.

This algorithm can be considered as a data reduction method since it reduces the input data to a set of subclusters that are obtained directly from the leaves of the CFT.

4.4.9 MiniBatchKMeans

The MiniBatchKMeans algorithm is a variation of the KMeans algorithm which reduces computation time by using mini-batches while still attempting to optimize the same objective function. In each training cycle, mini-batches are subsets of the input data that are randomly sampled. These mini-batches dramatically reduce the amount of processing required to converge to a local solution. MiniBatchKMeans delivers results that are just marginally worse than the conventional method, in contrast to other techniques that shorten the convergence time of k-means.

Similar to vanilla k-means, the algorithm iterates between two major phases. To generate a mini-batch, samples are randomly selected from the dataset in the first phase. These are then assigned to the centroid that is closest to them. The centroids are updated in the second phase. Unlike k-means, this is performed on a per-sample basis. The allocated centroid is updated for each sample in the mini-batch by taking the streaming average of the sample and all previous samples assigned to that centroid. As a result, the rate of change for a centroid over time is reduced. These steps are repeated until convergence or a set number of iterations has been reached.

MiniBatchKMeans converges faster than KMeans, but the results are of worse quality. In practice, the quality difference might be relatively minimal.

4.5 Execution Time Complexity and the Visual Performance of Clustering Techniques

Figure 4.10 shows the execution time of different clustering techniques for five different videos recorded from live street/traffic cameras in different cities.

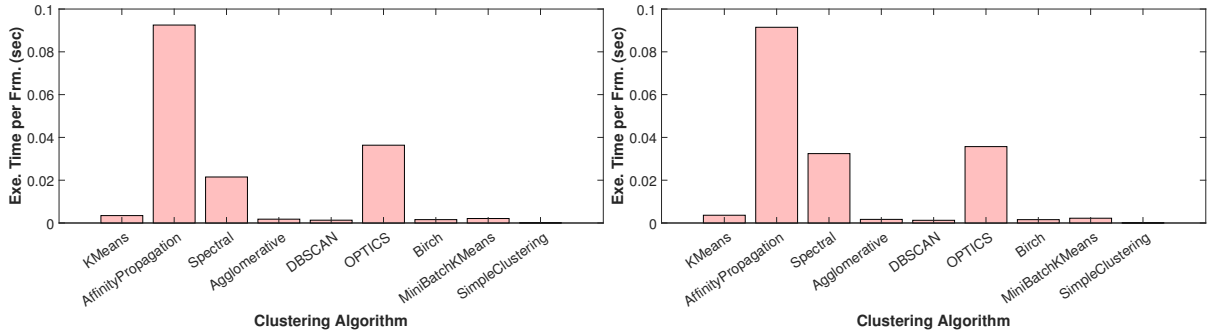
The average execution time over all the video files is shown in Figure 4.10(f). As shown in this diagram, Affinity Propagation takes substantially longer and is more complicated to implement than the other approaches. As a result, this method will not be suitable for use in our solution because it will consume a significant percentage of the overall time required to run the enhanced YOLO solution. Therefore, this method has been deleted from Figure 4.10(g) to appropriately display the average execution time per frame for other clustering methods, as the scaling is appropriate with the exclusion of the sophisticated approach stated. Although the Spectral and OPTICS clustering methods are still considerably more time consuming than the rest, we have included them in our experiments as they could still achieve decent frame rates while being utilized in our enhanced YOLO solution.

All the methods, with the exception of Affinity Propagation, are rather light to execute and may achieve frame rates of above 30fps when performed on a single thread on a regular CPU, as seen in this figure. As

a result, these methods can be easily implemented on edge devices without restricting their working frame rate.

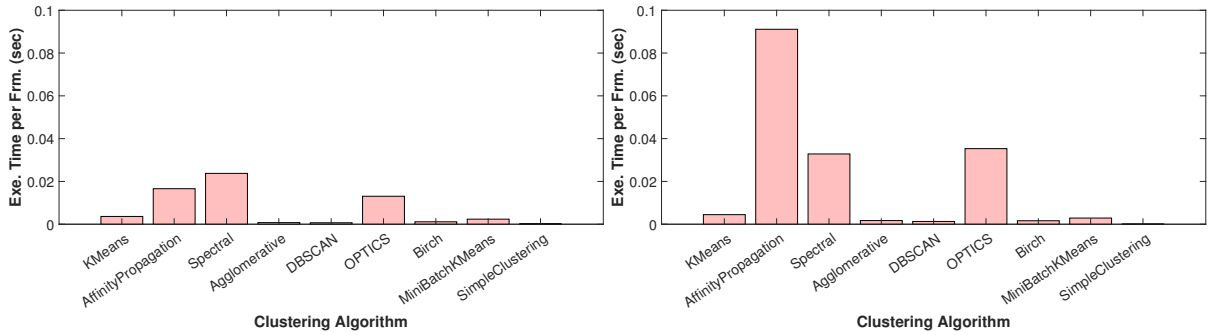
Figures 4.11, 4.12, 4.13, 4.14, and 4.15 illustrate the results/output of different clustering algorithms on different input videos from live street/traffic cameras in multiple cities. In these figures, Subfigure x.a shows the original frame used as an input for the clustering algorithm while the other subfigures show the output of different clustering algorithms, with red rectangles representing the encompassed area of formed single clusters, blue rectangles repressing the encompassing area of all the clusters, and magenta rectangles representing the extended area calculated from the blue rectangles that would be fed to the deployed YOLOv4 object detection solution.

Visually, most of the employed clustering methods behave similarly on the selected frame picked for comparison. There are some subtle differences, particularly in the generated individual clusters (red rectangles). For example, the DBSCAN method shown in Figures 4.11(j) and 4.15(j) generates clusters that encompass the entire frame, while other methods result in a small portion of the frame being selected as the region of interest. Another example could be the generated clusters in Figures 4.11(g) vs. 4.11(h), 4.12(b) vs. 4.12(c), 4.12(g) vs. 4.12(h), and 4.13(b) vs. 4.13(c) which are not generated similarly.



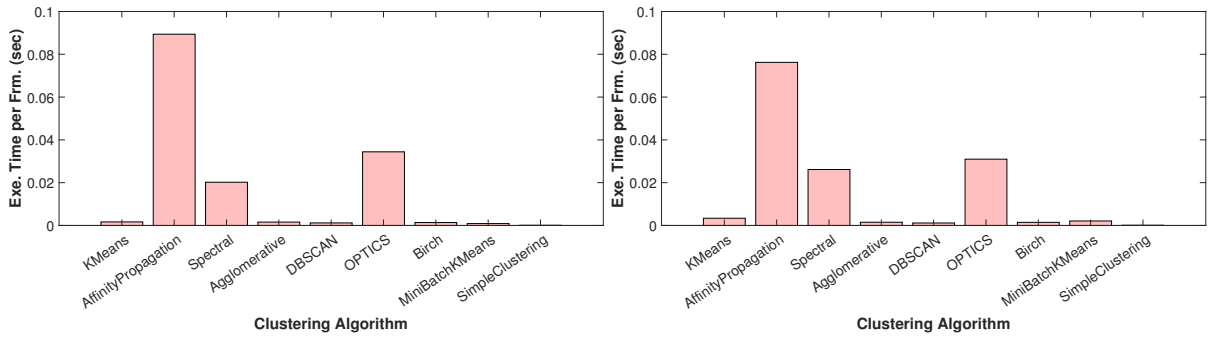
(a) Lamai, Koh Samui, Thailand

(b) Saint Petersburg, Russia



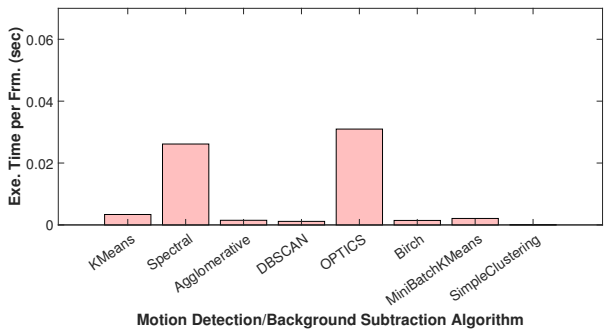
(c) New Orleans, Louisiana, United States

(d) Laramie, Wyoming, United States



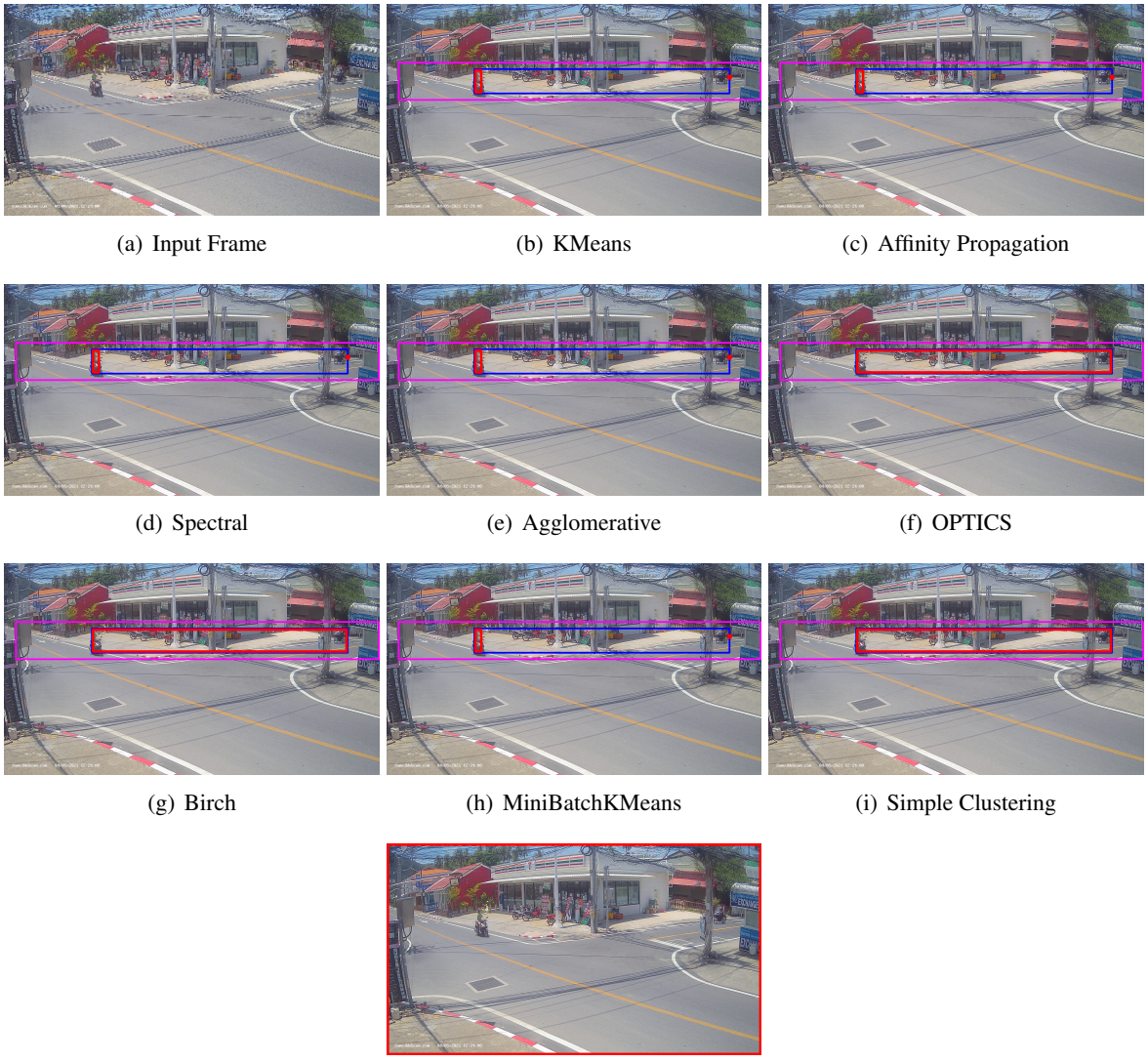
(e) Renesse, Netherlands

(f) Average of all locations



(g) Average of all locations (without the computationally complex clustering detection methods)

Figure 4.10: Average Execution Time per Frame for Different Clustering Algorithms, Considering Different Input Videos



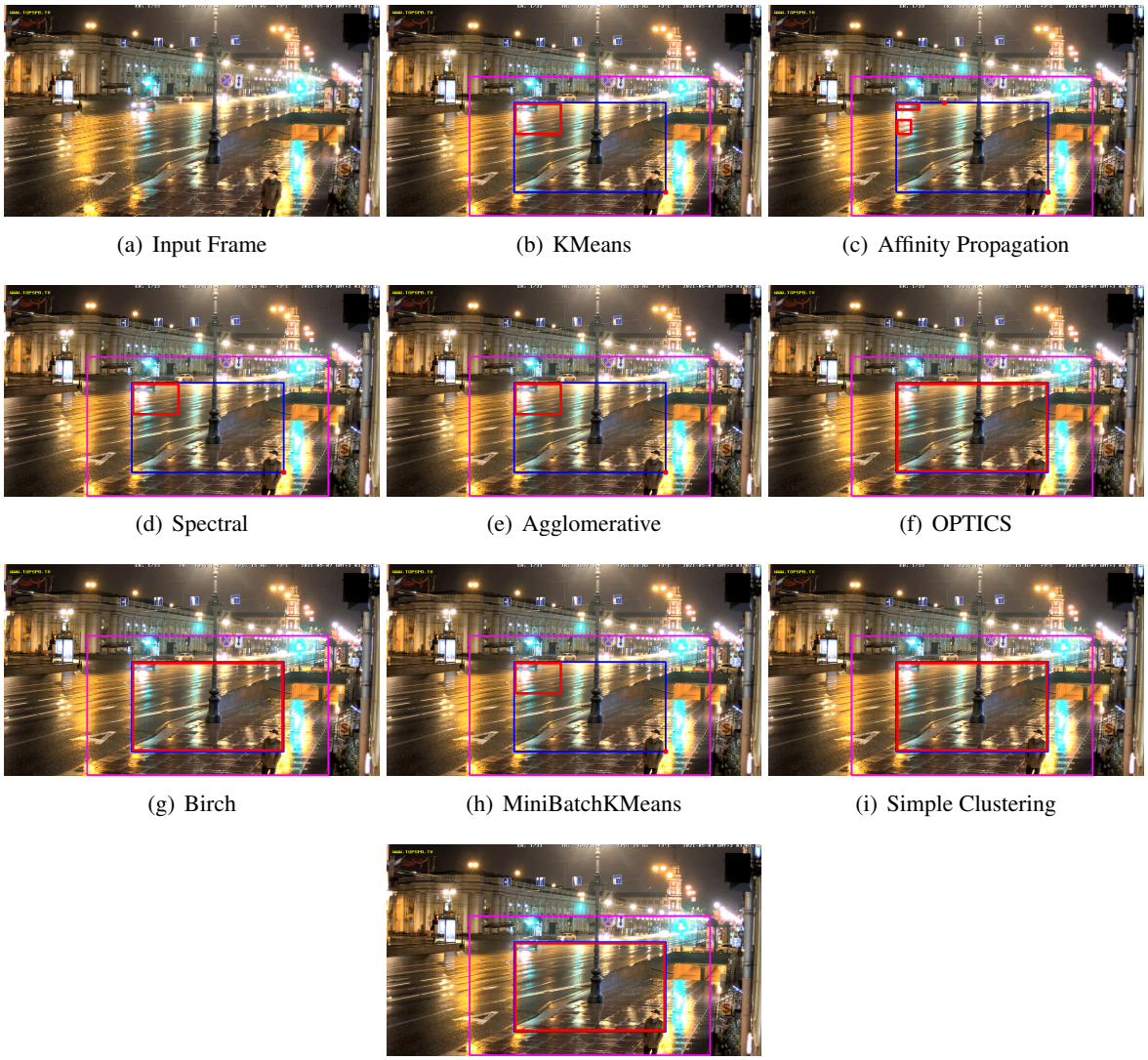
(a) Input Frame (b) KMeans (c) Affinity Propagation

(d) Spectral (e) Agglomerative (f) OPTICS

(g) Birch (h) MiniBatchKMeans (i) Simple Clustering

(j) DBSCAN

Figure 4.11: Results from Clustering Algorithms, Running on the Input Video from Lamai, Koh Samui, Thailand



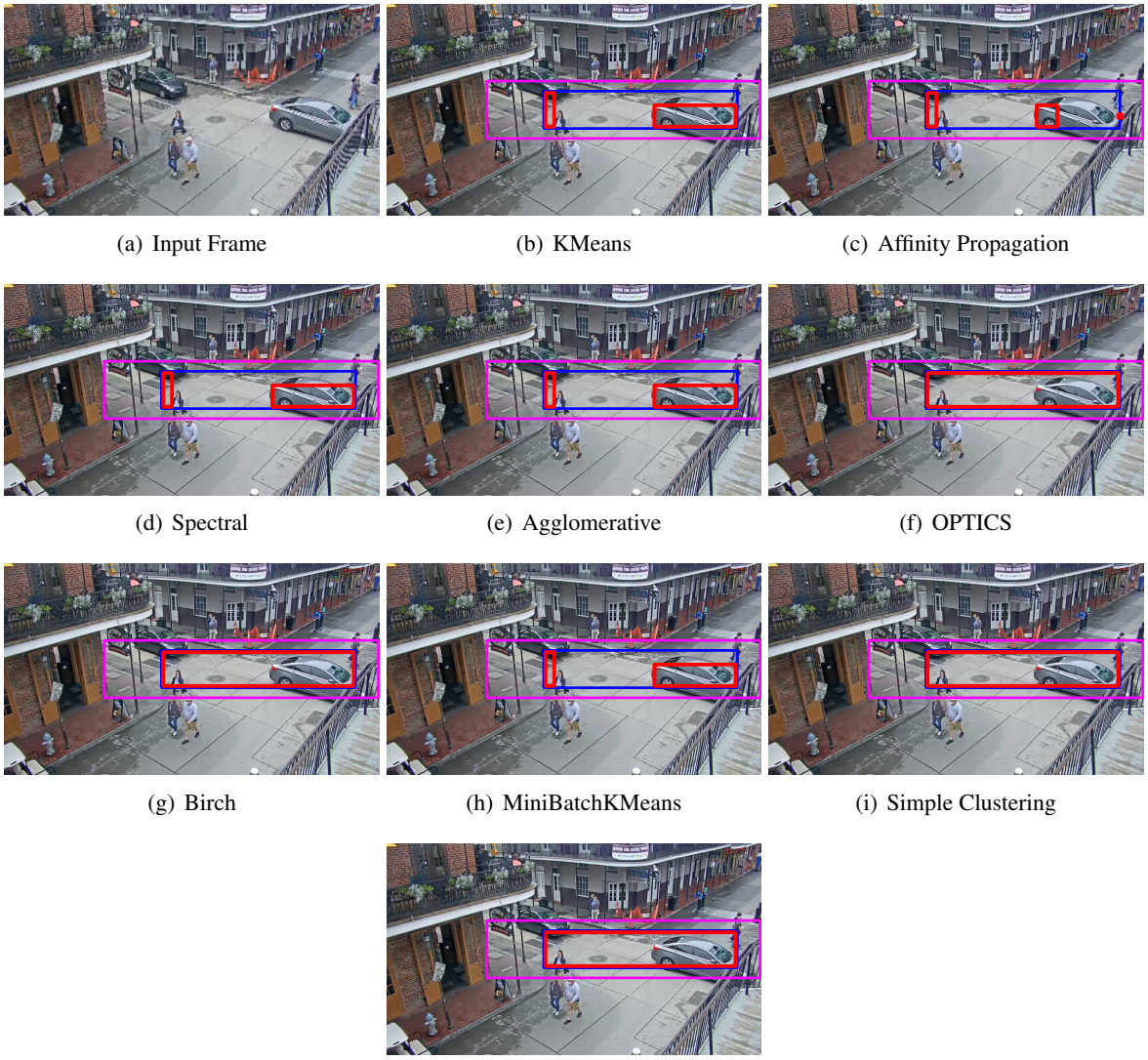
(a) Input Frame (b) KMeans (c) Affinity Propagation

(d) Spectral (e) Agglomerative (f) OPTICS

(g) Birch (h) MiniBatchKMeans (i) Simple Clustering

(j) DBSCAN

Figure 4.12: Results from Clustering Algorithms, Running on the Input Video from Saint Petersburg, Russia



(a) Input Frame (b) KMeans (c) Affinity Propagation

(d) Spectral (e) Agglomerative (f) OPTICS

(g) Birch (h) MiniBatchKMeans (i) Simple Clustering

(j) DBSCAN

Figure 4.13: Results from Clustering Algorithms, Running on the Input Video from New Orleans, Louisiana, United States

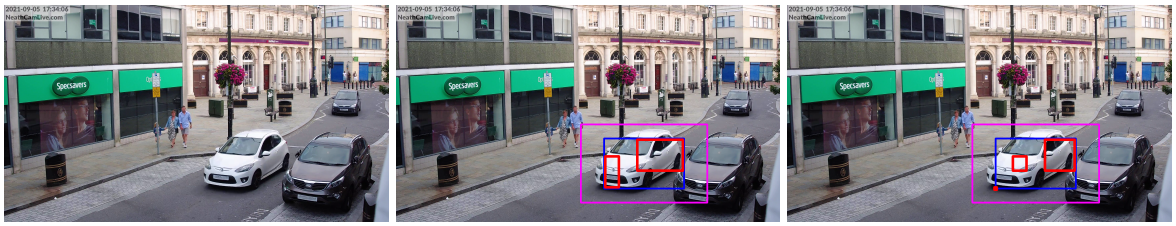
(a) Input Frame (b) KMeans (c) Affinity Propagation

(d) Spectral (e) Agglomerative (f) OPTICS

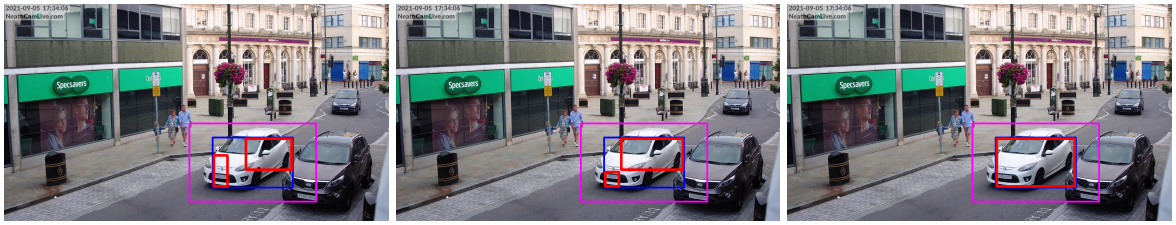
(g) Birch (h) MiniBatchKMeans (i) Simple Clustering

(j) DBSCAN

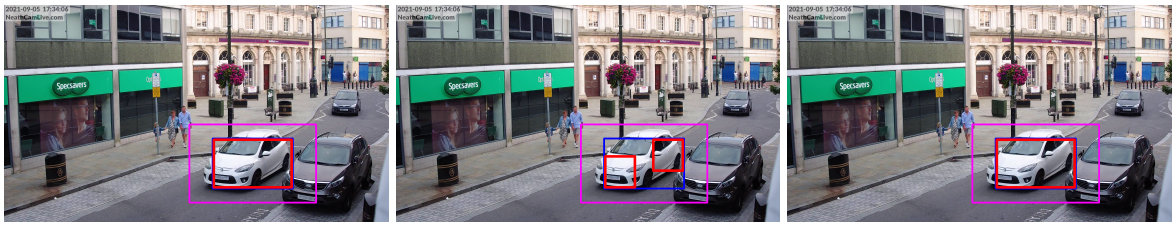
Figure 4.14: Results from Clustering Algorithms, Running on the Input Video from Laramie, Wyoming, United States



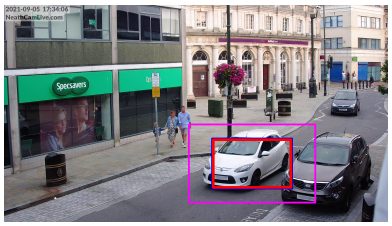
(a) Input Frame (b) KMeans (c) Affinity Propagation



(d) Spectral (e) Agglomerative (f) OPTICS



(g) Birch (h) MiniBatchKMeans (i) Simple Clustering



(j) DBSCAN

Figure 4.15: Results from Clustering Algorithms, Running on the Input Video from Neath, Wales

4.6 Performance Evaluation Methodology

Figure 4.16 illustrates an overview of the performance evaluation methodology used in conducting different experiments and producing the comparison results.

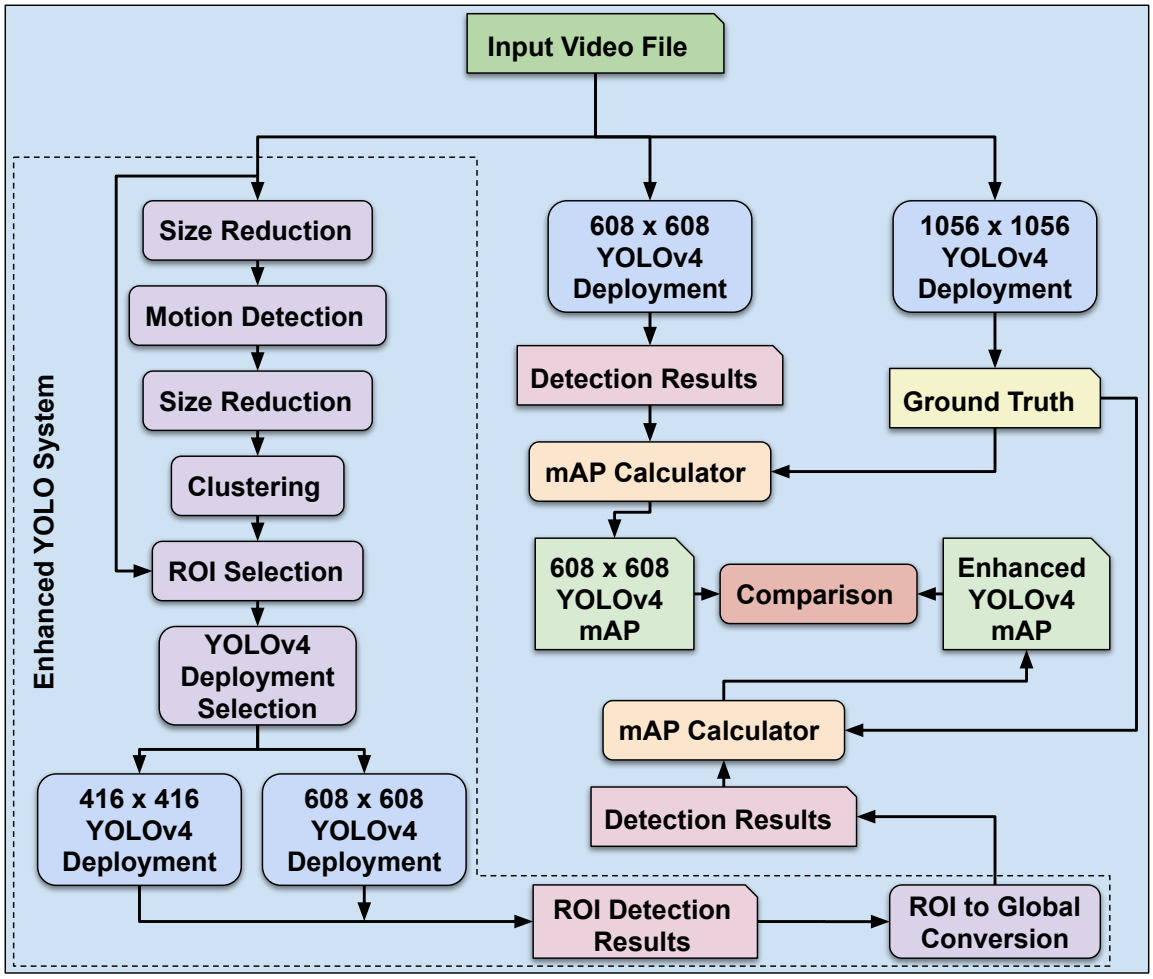


Figure 4.16: Overview of Performance Evaluation Methodology

First, the frames from the input video file are fed to a YOLOv4 deployment configured with a high resolution (e.g., 1056×1056) input. The generated detection results from each individual frame are stored in a file to be used later for comparison. These detection results will be considered as ground truth in our experiments. The frames from the same video file are fed to another YOLOv4 deployment configured with a lower resolution input (e.g., 608×608). Similarly, the detection results from each individual frame are

stored in a file.

The mentioned input video file will also be used in our proposed Enhanced YOLO system. Each frame from the video file will be reduced in size to be prepared for the motion detection algorithm. The execution time required for this algorithm is reduced substantially by reducing the size of the original input frame. Not only is a reduced size input image detailed enough for our intended application, but the reduction in the number of pixels and miniature movements also contributes to a better and cleaner result in the motion/background-foreground detection algorithm.

After generating the motion detection image (a binary image where each white pixel represents a moving/foreground element), another size reduction operation is applied to the result to prepare it for the clustering algorithm. As was the case with the motion detection step, the execution time required for the clustering algorithm is reduced substantially by reducing the size of binary background/foreground input frame. Similarly, a reduced size input frame is sufficiently detailed for our intended application and the reduction in the number of pixels/samples contributes to a better and cleaner result in the clustering algorithm.

Using the clustering results, the region of interest (ROI) could be calculated by considering an area that covers all generated clusters. We also consider an additional configurable border that expands the selected ROI calculated from the formed clusters. The calculated ROI is used in cropping the original input image.

The next step in the process is to decide which YOLOv4 deployment to select as the object detector. We are making this decision based on two factors. The first is the ratio of the cropped region/ROI to the original input image and the second is the previous average recognition probability values calculated from previous input images fed to the YOLOv4 deployments. This calculation is done using the following equations.

$$IndexIndicator = C \times \frac{CA}{IA} + (1 - C) \times (1 - SRP_t), \tag{4.7}$$

$$SRP_t = S \times SRP_{t-1} + (1 - S) \times RP_t \tag{4.8}$$

Where C, CA, IA, SRP, S, and RP are constant, Cropped Area, Image Area, Smoothed Recognition Probability, constant, and Recognition Probability, respectively. C and S values are between 0 and 1 and Recognition Probability is calculated by averaging all the recognition probability values of all objects in the ROI extracted from the YOLOv4 detection results.

A larger CA means a larger $\frac{CA}{IA}$ which contributes to a higher $IndexIndicator$. Similarly, a lower SRP increases the value for $1 - SRP_t$ which leads to a higher $IndexIndicator$. $IndexIndicator$ value is between 0 and 1. The closer the value to 1, the higher the input resolution for the selected deployed YOLOv4 detector. After determining the $IndexIndicator$ value and assuming $ThresholdValues = [th_1, th_2, th_3, ..., th_n]$ ($n + 1$ being the number of deployed YOLOv4 detectors), the following equation could be used to find the selection index for the deployed YOLOv4 detectors.

$$Index = x \;\; where \;\; th_x <= IndexIndicator \;\; and \;\; th_{x+1} > IndexIndicator \tag{4.9}$$

where $th_1, th_2, ..., th_n$ are constant threshold values assigned to control the distribution of the cropped images between different YOLOv4 deployments with different input resolutions. th_x is greater than th_{x-1} and smaller than or equal to 1.

As an example, by deploying two YOLOv4 detectors ($n + 1 = 2$), one with a higher resolution, the $ThresholdValues$ array could be assumed as $[0.5]$. This means that if the calculated $IndexIndicator$ value is for instance 0.42, the YOLOv4 deployment with index 0 will be selected as the object detector for the current cropped image. On the other hand, if the calculated value is 0.68, the YOLOv4 deployment with index 1 will be selected as the object detector (the one with a higher input resolution). It should be noted that inclusion $th_0 = 0$ is implicit in calculating the index value.

After selecting the proper YOLOv4 deployment and feeding the cropped ROI image to it, the detection result in the ROI is generated. This is still not the desired final detection result as it only covers the ROI region (the region containing the motion/foreground). This regional detection result needs to be converted to the global detection result. The conversion takes advantage of the previous detection results to populate the entire current frame with the detected objects. Here we check every detected object bounding box from the previous frame by considering the ROI bounding box and calculating the intersection over union value. If the calculated value is smaller than a configurable threshold, it will be transferred to the new frame detection results. The ROI detection result is converted to the whole frame coordinates and is also added to the new frame detection result. This calculated detection result is stored in a file to be used later for comparison.

By using a mAP (described in the next section) calculation module and the stored ground truth detection results, the mAP results for both 608×608 YOLOv4 deployment and our Enhanced YOLO system are generated. The mAP calculation module takes two input arguments. One is the ground truth detection results and the other is the stored detection results from an object detection system. The generated mAP results determine how close the results from the object detection system are to the ground truth information.

By comparing the mAP results from our Enhanced YOLO solution and the mAP results from the 608×608 YOLOv4 deployment, the performance of our proposed solution is demonstrated.

Additionally, we are reporting the average execution time per frame and the number of pixels delivered to the YOLOv4 deployment as these two items are important in any object detection system. The first item directly translates to the processing power requirements and the second item is directly related to the required bandwidth and data transferring to a cloud-based neural network deployment.

4.7 Mean Average Precision (mAP)

The mean average precision (mAP), often known as AP, is a widely used metric for assessing the performance of models handling document/information retrieval and object detection tasks. Wikipedia defines

the mean average precision (mAP) of a group of queries as follows:

$$mAP = \frac{\sum_{q=1}^{Q} AveP(q)}{Q} \tag{4.10}$$

where Q is the total number of queries in the set and $AveP(q)$ denotes the average precision (AP) for a single query, q.

The algorithm basically says that for each query, q, we calculate its corresponding AP, and then take the mean of all of these AP scores to get a single value, termed the mAP, which measures how well our model performs on the query.

Precision and recall are two regularly used metrics to assess the effectiveness of a classification model. To comprehend mAP, we must first examine precision and recall.

The precision of a particular class in classification, also known as positive predicted value, is defined as the ratio of true positives (TP) to the total number of predicted positives in the field of statistics and data science. The formula is as follows:

$$Precision = \frac{TP}{TP + FP} \tag{4.11}$$

Similarly, the recall of a particular class in classification is defined as the ratio of true positive rate (TP) to the total of ground truth positives. The formula is as follows:

$$Recall = \frac{TP}{TP + FN} \tag{4.12}$$

We would need to reduce our number of FP to achieve high precision, which would reduce our recall. Similarly, lowering the number of FN would boost recall while lowering precision. In many circumstances, such as information retrieval and object detection, we want our precision to be great (our predicted positives

to be TP).

We must first grasp IoU to calculate AP for object detection. The IoU is defined as the ratio of the area of intersection over the area of union for the predicted and ground truth bounding boxes. Figure 4.17 shows a visual representation of IoU.

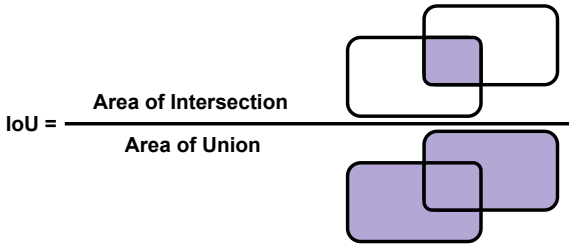


Figure 4.17: Visual Representation of IoU

To establish if a predicted bounding box (BB) is TP, FP, or FN, the IoU would be employed. The TN is not calculated because each image is presumed to include an object. Consider Figure 4.18:

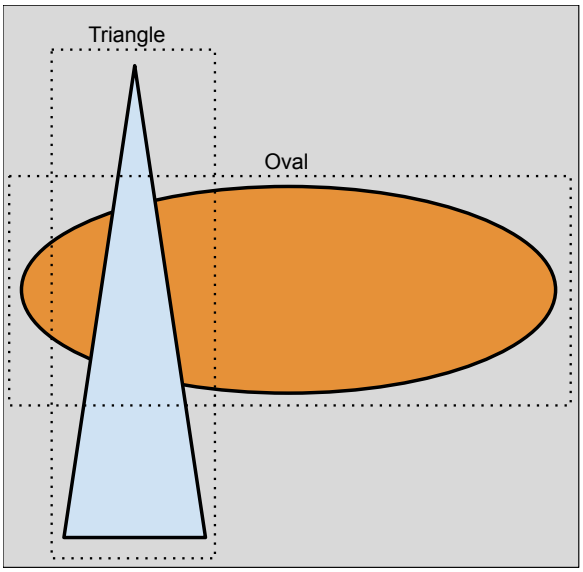


Figure 4.18: Image with a Triangle and an Oval Labeled with Ground Truth Bounding Boxes

A triangle and an oval are depicted in the image, together with their ground truth bounding boxes. For the time being, we will disregard the oval. On this image, we run our object detection model and get a predicted bounding box for the triangle. If the IoU is more than 0.5, we call a prediction a TP. Figure 4.19

describes a possible scenario for TP.

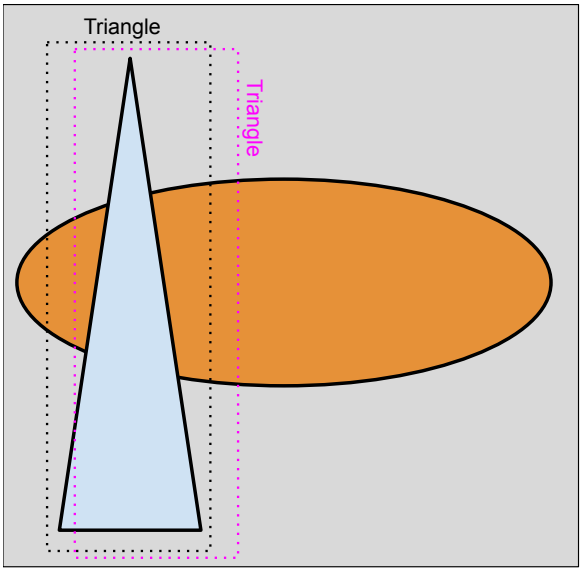
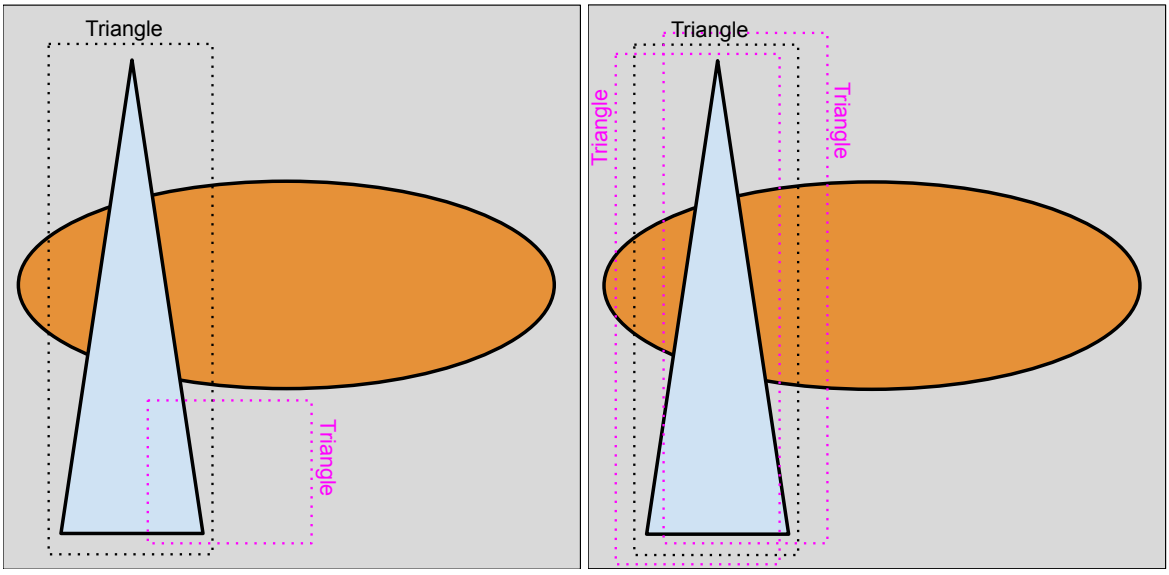


Figure 4.19: IoU of Predicted BB (cyan) and GT BB (black) > 0.5 with the Correct Classification

There are two instances in which a BB could be classified as FP: when IoU < 0.5 and when the predicted BB is duplicated.

Figure 4.7 shows these two instances.



(a) IOU < 0.5 (b) Duplicated BB

Figure 4.20: Illustrating the Different Scenarios a Predicted BB (cyan) Would be Considered as FP

When our object detection model fails to detect the target, we call it a false negative. Two scenarios could occur: when there is no detection at all and when the predicted BB has an IoU > 0.5 but the wrong classification.

We can now calculate the precision and recall of our detection for a particular class over the test set because the TP, FP, and FN have been clearly established. The confidence level of each BB, which is normally determined by its Softmax layer, would be used to rank/score the output.

The precision-recall curve is calculated using the model's detection output for a specific class (e.g., "person") by adjusting the model score threshold that specifies what is counted as a model-predicted positive detection of the class.

To find a point on the precision-recall curve, treat all objects above a specified model score threshold as positive predictions, then calculate the precision and recall for that threshold.

The average precision value across all recall values is the final step in computing the AP score. This is the single value that sums up the precision-recall curve's form. The AP score is defined as the mean precision at a set of 11 equally spaced recall levels; for clarification, recall values = [0, 0.1, 0.2,..., 1.0]. As a result, the precision at $recall_i$ is assumed to equal the highest precision measured at a recall that is greater than $recall_i$.

The mean Average Precision or mAP score is calculated by taking the mean AP over all classes.

4.8 Results

Figures 4.21 to 4.35 compares mAP, the average execution time per frame, and the transferred data from our proposed Enhanced YOLO solution against the results generated from the raw YOLO solution as described in Figure 4.16. This means if a bar has a positive value and occupies the right-hand portion of the figure, the performance related to that bar in our proposed solution is better in comparison to the raw solution; on the other hand, if a bar has a negative value and occupies the left-hand portion of the figure,

the performance related to that bar in our proposed solution is worse than the raw solution. By considering all combinations of different motion detection algorithms and different clustering techniques, all the tuples of blue, orange, and red bars are generated. The first and the second word in each y-axes label determines the employed motion detection algorithm and the clustering technique, respectively. As an example, *StaticFrameDifference MinbatchKmeans* indicates that the results are related to the Enhanced YOLO system where the configured motion detection algorithm is Static Frame Difference, while the employed clustering algorithm is Min-batch Kmeans.

Figure 4.21 shows the relative results for our Enhanced YOLO solution by considering the input video from Lamai, Koh Samui, Thailand. As can be seen in the figure, the overwhelming portion of the bars and the area occupied by the bars belong to the right-hand portion of the graph, meaning the Enhanced YOLO solution generally outperforms the raw solution by a considerable margin. As an example, by looking at the methods with *WeightedMovingMean* as the selected motion detection algorithm, generally more than 1200% (12 times) reduction in the required data transmissions to the object detection neural network could be achieved. Furthermore, the execution time reduction is around 800% which means that only one eighth of the processing power will be required to run the enhanced YOLO solution in comparison to the raw method. These large improvements come by some comparatively minor reduction in mAP results (around 20%).

The *StaticFrameDifference* motion detection method could be of interest if strictly no decrement in mAP performance is desired. Figure 4.22 is a slice of Figure 4.21. It displays the relative results for *StaticFrameDifference* methods. *StaticFrameDifference AgglomerativeClustering* combination provides around 24% improvement in both data transmission and execution time metrics while enabling 2% improvement in mAP results; a win on all three fronts. Another interesting option is *StaticFrameDifference DBSCAN* as for less than 1% decrement in mAP results, more than 75% improvement in execution time and data transmission metrics are achieved.

Figure 4.21: Relative Performance Results for Enhanced YOLO Solution against the Raw YOLO Considering the Input Video from Lamai, Koh Samui, Thailand

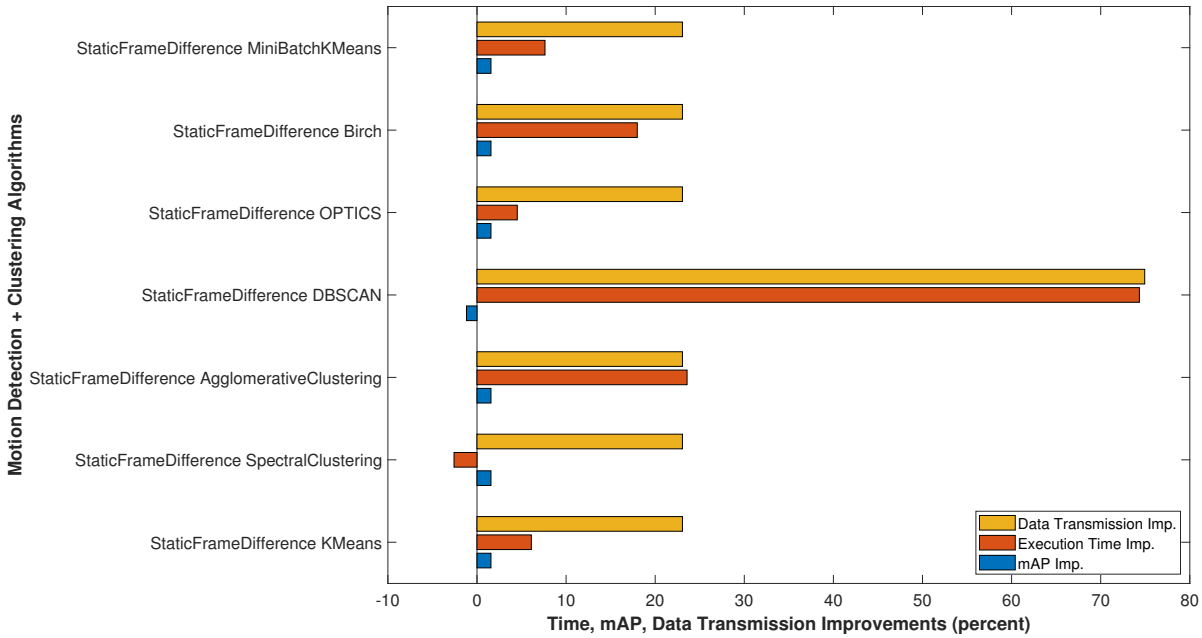


Figure 4.22: StaticFrameDifference Slice of Relative Performance Results for Enhanced YOLO Solution against the Raw Yolo for the Input Video from Lamai, Koh Samui, Thailand

Figure 4.23 demonstrates the relative results for our Enhanced YOLO solution by considering the input video from Saint Petersburg, Russia. It can be observed that, similarly to the results from the previous input video file, the vast majority of the bars and the area occupied by the bars are located on the righthand side of the graph, implying that the Enhanced YOLO solution outperforms the raw solution by a significant margin. Let us again consider the methods with *WeightedMovingMean* as the selected motion detection algorithm. These methods generally reduce the required data transmissions to the object detection neural network by more than 1200% (12 times). Furthermore, the execution time reduction is around 1000%. In comparison to the raw technique, only a tenth of the computing power will be required to run the enhanced YOLO solution. As was the case before, these large improvements come by some comparatively minor reduction in mAP results (around 20%).

The *StaticFrameDifference* and *CodeBook* motion detection methods could be considered if strictly no decrement in mAP performance is desired. Figure 4.24 is a slice of Figure 4.23 and displays the relative

results for *StaticFrameDifference* methods. *StaticFrameDifference DBSCAN* combination provides around 14% improvement in both data transmission and execution time metrics while improving the mAP results by 24%, improving all the metrics simultaneously. Figure 4.25 is another slice of Figure 4.23 and displays the results related to the *CodeBook* combination methods. Here, all the combinations lead to great performance improvements in all the three measured metrics of mAP, execution time, and data transmission. Especially, the *CodeBook DBSCAN* combination improves the mAP, execution time, and required data transmission by 150%, 113%, and 23%, respectively.

Figure 4.26 shows the relative results for our Enhanced YOLO solution by considering the input video from New Orleans, Louisiana, United States. Like all previous input video files, it can be observed that the vast majority of the bars and the area occupied by the bars are located on the right-hand side of the graph, meaning the Enhanced YOLO solution largely outperforms the raw solution. By considering the methods with *WeightedMovingMean* as the selected motion detection algorithm, the required data transmissions to the object detection neural network are reduced by more than 1600% (16 times). The execution time reduction is around 1200% which indicates that only 8% of the computing power will be required to run the Enhanced YOLO solution. Again, these large improvements come by a small reduction in mAP results (around 10%).

Similarly, here the *StaticFrameDifference*, *CodeBook*, and *AdaptiveSelectiveBackgroundLearning* motion detection methods could be considered if strictly no decrement in mAP performance is desired. Figure 4.27 is a slice of Figure 4.26 and demonstrates the relative results for *StaticFrameDifference* methods. *StaticFrameDifference DBSCAN* combination provides around 8% improvement in data transmission, 4% improvement in execution time, and 5% improvement in mAP results. This means that improvements are present in all metrics simultaneously. Figure 4.28 is another slice of Figure 4.26 and displays the results related to the *CodeBook* combination methods. As was the case with *CodeBook* in the previous input video file, here all the combinations lead to great performance improvements in the three measured metrics of

Figure 4.23: Relative Performance Results for Enhanced YOLO Solution against the Raw YOLO Considering the Input Video from Saint Petersburg, Russia

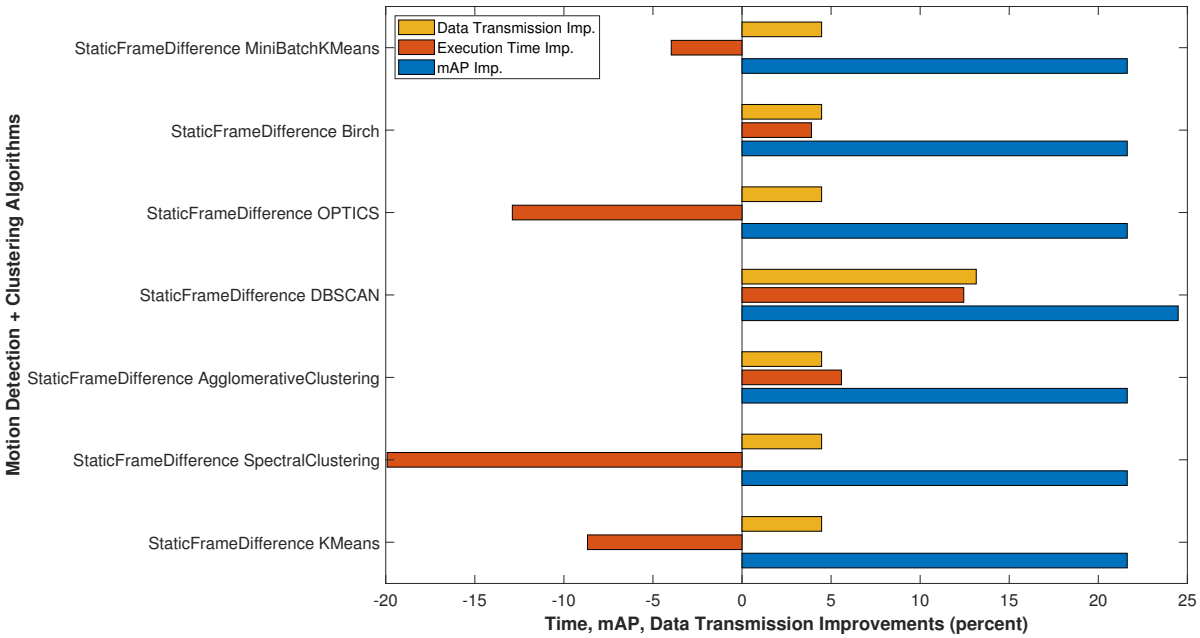


Figure 4.24: StaticFrameDifference Slice of Relative Performance Results for Enhanced YOLO Solution against the Raw YOLO for the Input Video from Saint Petersburg, Russia

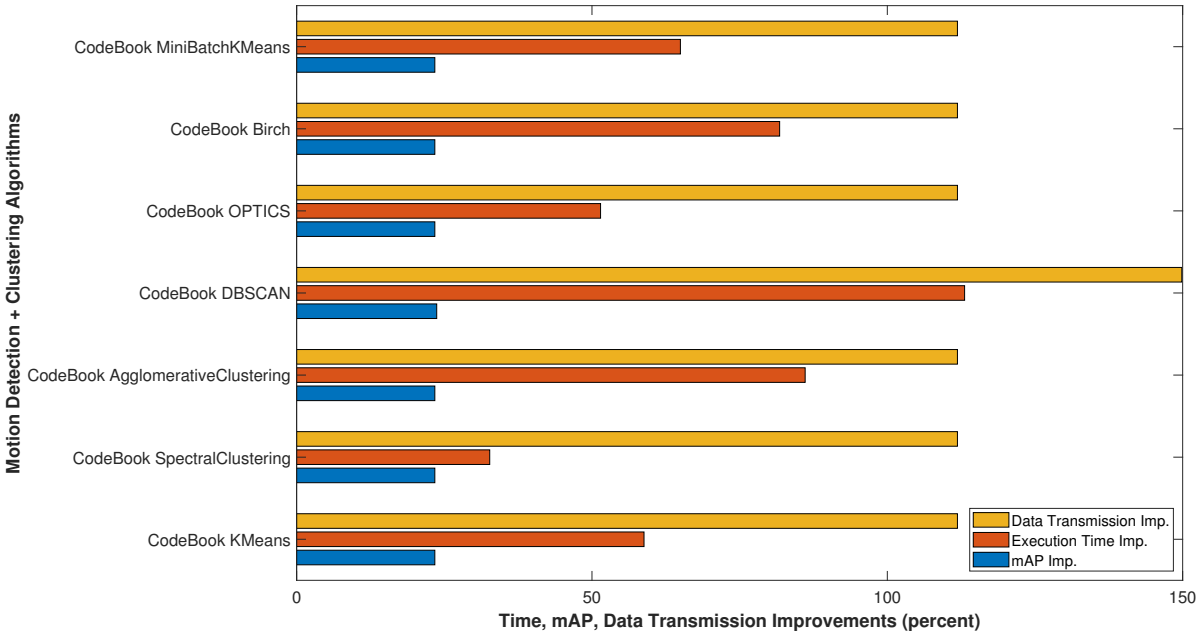


Figure 4.25: CodeBook Slice of Relative Performance Results for Enhanced YOLO Solution against the Raw YOLO for the Input Video from Saint Petersburg, Russia

mAP, execution time, and data transmission. Especially, the *CodeBook DBSCAN* combination improves the execution time and required data transmission by more than 155% and 125%, respectively. This consid-

erable performance improvement comes at a negligible cost of 1% degradation in mAP results. Another interesting combination in this graph is *CodeBook AgglomerativeClustering*. This method improves the mAP, execution time, and required data transmission by 2%, 65%, and 80%, respectively.

Lastly, the relative results for *AdaptiveSelectiveBackgroundLearning* methods are displayed in Figure 4.29 which is a slice from 4.26. Here, the two combinations with *DBSCAN* and *AgglomerativeClustering* as clustering algorithms improve the mAP, execution time, and required data transmission by 6%, 6%, and 3%, respectively.

Figure 4.30 shows the relative results for our Enhanced YOLO solution by considering the input video from Laramie, Wyoming, US. It can be observed that, similarly to the previous results, the vast majority of the bars and the area occupied by the bars are located on the righthand side of the graph, indicating that the Enhanced YOLO solution outperforms the raw solution by a large margin. By considering the methods with *WeightedMovingMean* as the selected motion detection algorithm, the required data transmissions to the object detection neural network are reduced by more than 1600% (16 times). The execution time reduction is around 1000%, which indicates that only 10% of the computing power will be required to run the Enhanced YOLO solution. Again, these substantial improvements come by a small reduction in mAP results (around 13%).

As was the case in previous input video files, here the *StaticFrameDifference* and *CodeBook* motion detection methods could be considered if strictly no decrement in mAP performance is desired. Figure 4.31 is a slice of Figure 4.30 where it focuses on the relative results for *StaticFrameDifference* methods. *StaticFrameDifference DBSCAN* combination provides around 8% improvement in execution time and around 2% improvement in mAP. Another ideal combination is *StaticFrameDifference AgglomerativeClustering*, which provides around 9% improvement in execution time and around 2% improvement in mAP, while at the same time not decrementing the data transmission results. Figure 4.32 is another slice of Figure 4.30

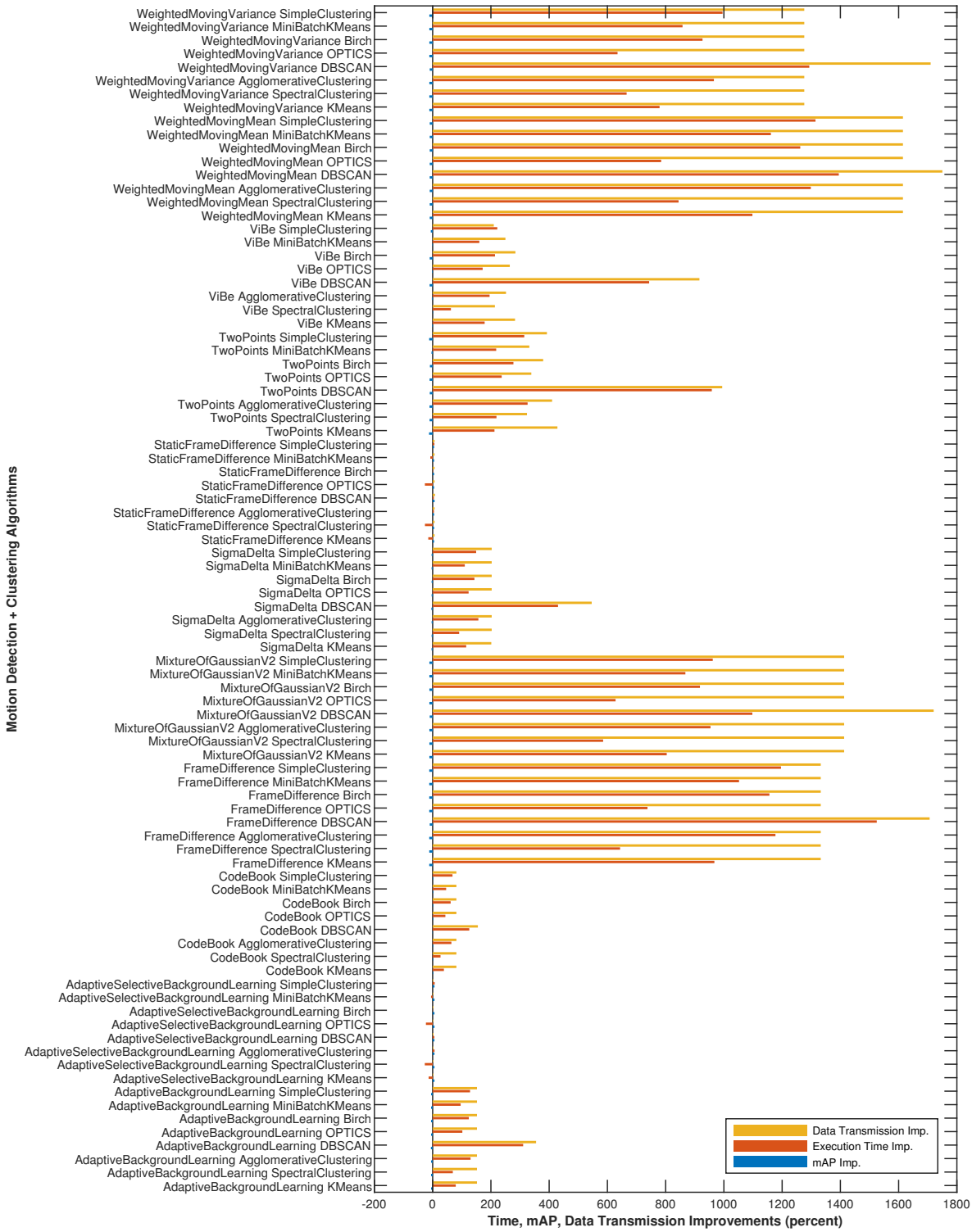


Figure 4.26: Relative Performance Results for Enhanced YOLO Solution against the Raw YOLO Considering the Input Video from New Orleans, Louisiana, United States

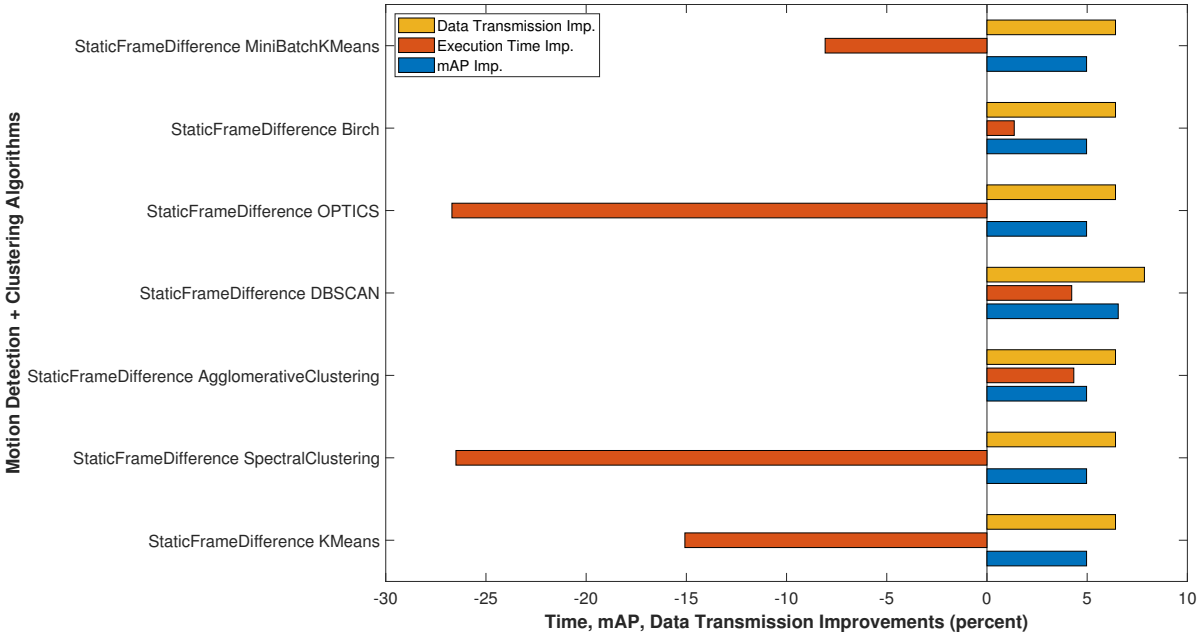


Figure 4.27: StaticFrameDifference Slice of Relative Performance Results for Enhanced YOLO Solution against the Raw YOLO for the Input Video from New Orleans, Louisiana, United States

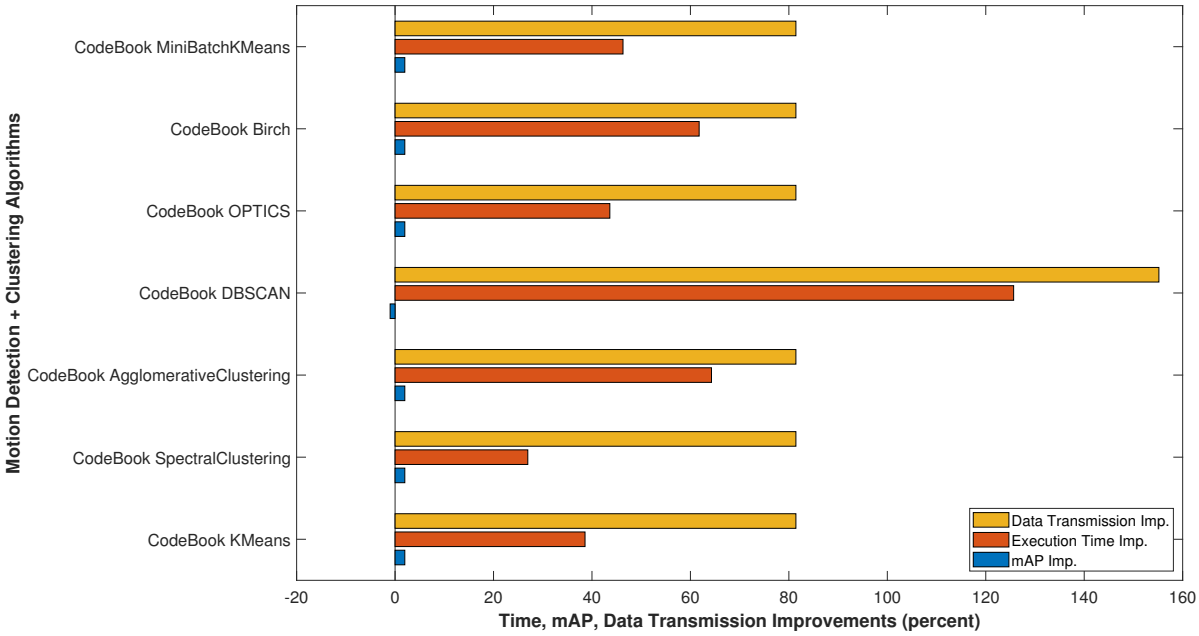


Figure 4.28: CodeBook Slice of Relative Performance Results for Enhanced YOLO Solution against the Raw YOLO for the Input Video from New Orleans, Louisiana, United States

and displays the results related to the *CodeBook* combination methods. Similar to previous input video files, here all the combinations lead to substantial performance improvements in the two measured metrics of exe-

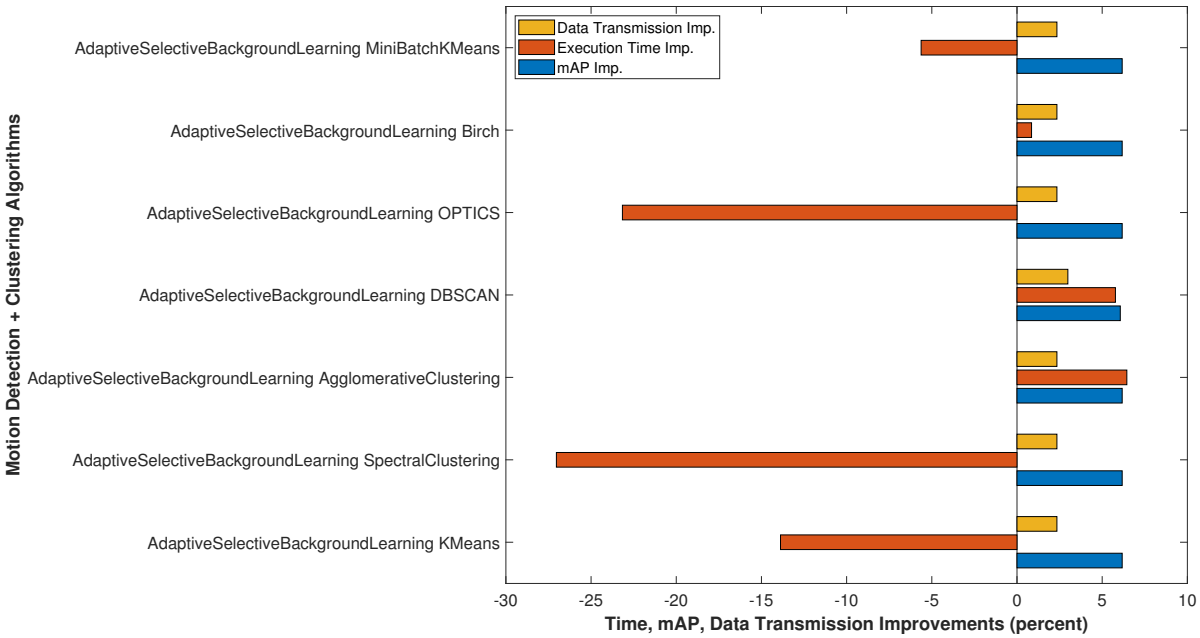


Figure 4.29: AdaptiveSelectiveBackgroundLearning Slice of Relative Performance Results for Enhanced YOLO Solution against the Raw YOLO for the Input Video from New Orleans, Louisiana, United States

cution time and data transmission. Especially, the *CodeBook DBSCAN* combination improves the execution time and required data transmission by more than 269% and 304%, respectively. Furthermore, it improves the mAP results by more than 6%.

The last figure in this section is Figure 4.33 which shows the relative results for our Enhanced YOLO solution by considering the input video from Neath, Wales. The vast majority of the bars and the area occupied by the bars are located on the righthand side of the graph, indicating that the Enhanced YOLO solution outperforms the raw solution by a large margin as was the case with previous input video files. By choosing the methods with *WeightedMovingMean* as the employed motion detection algorithm, the required data transmissions to the object detection neural network are reduced by around 500% (5 times). The execution time reduction is around 400% which means that only 25% of the computing power will be required to run the Enhanced YOLO solution. Again, these large improvements come by a small reduction in mAP results (around 20%). This mAP reduction is only 5% in the *WeightedMovingMean DBSCAN*

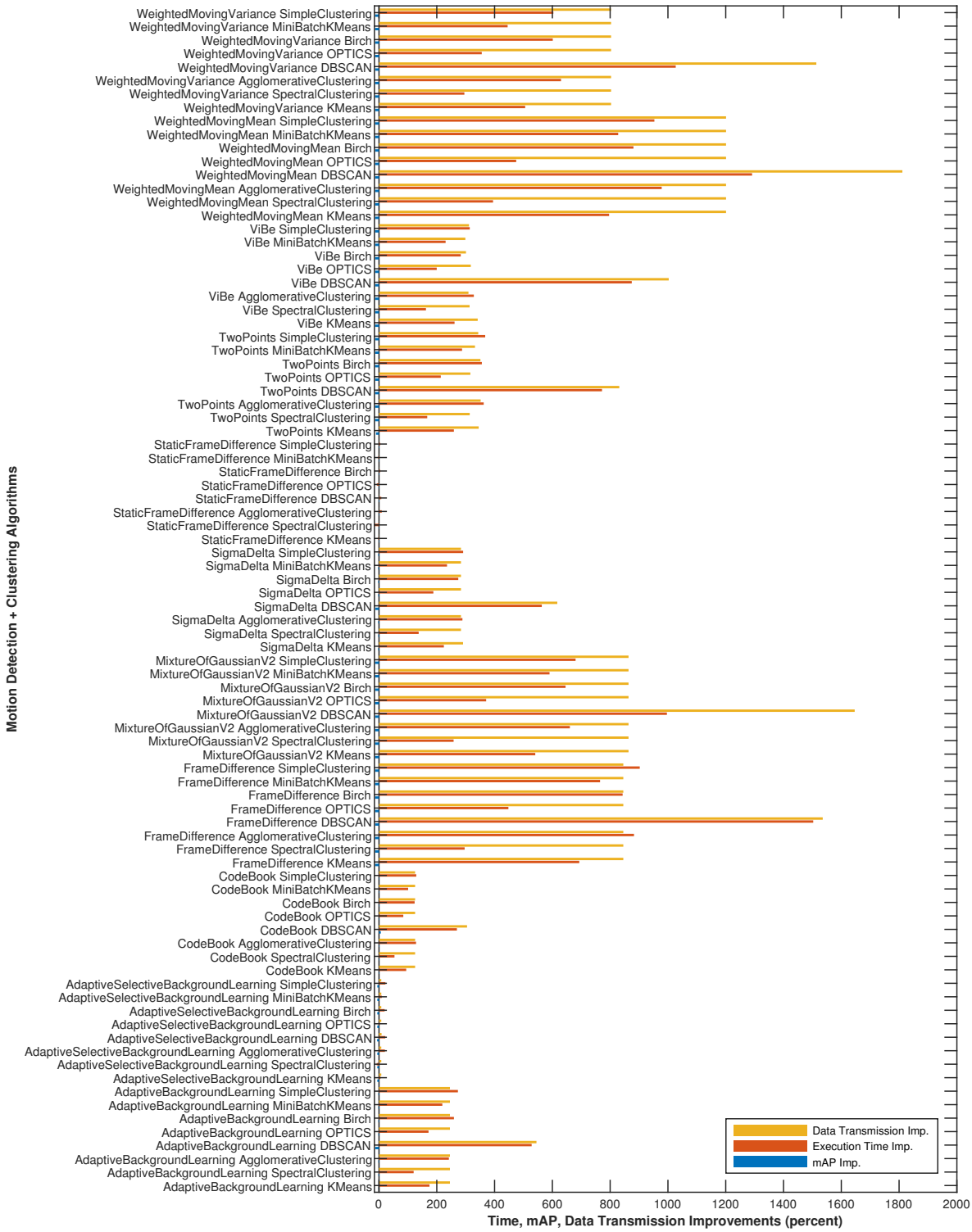


Figure 4.30: Relative Performance Results for Enhanced YOLO Solution against the Raw YOLO Considering the Input Video from Laramie, Wyoming, United States

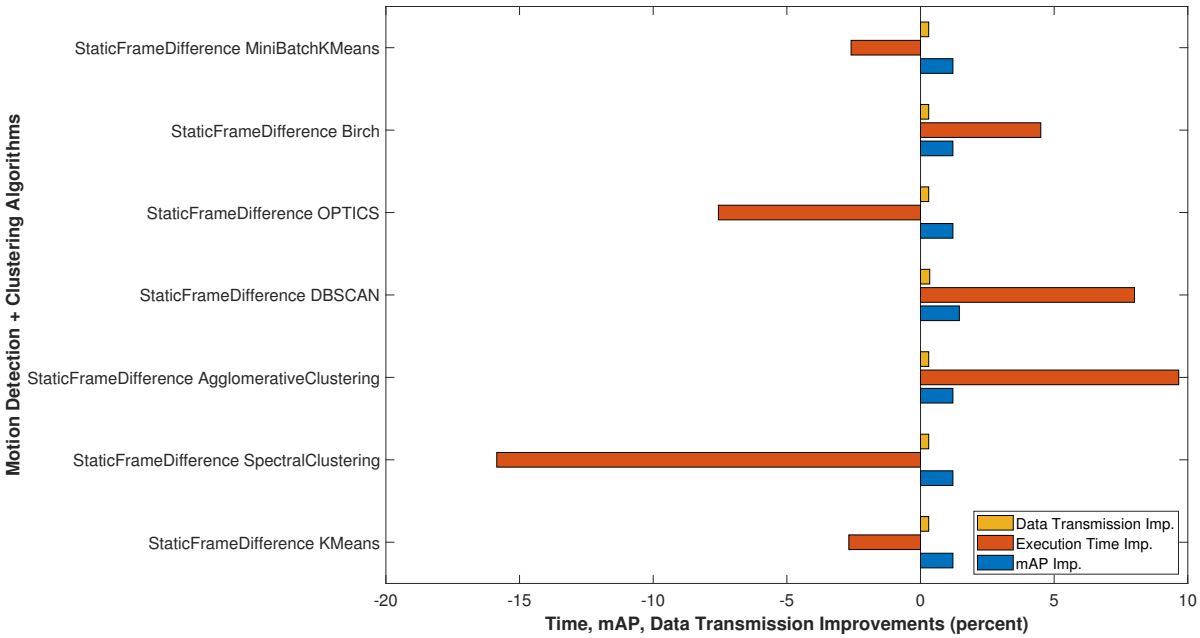


Figure 4.31: StaticFrameDifference Slice of Relative Performance Results for Enhanced YOLO Solution against the Raw YOLO for the Input Video from Laramie, Wyoming, United States

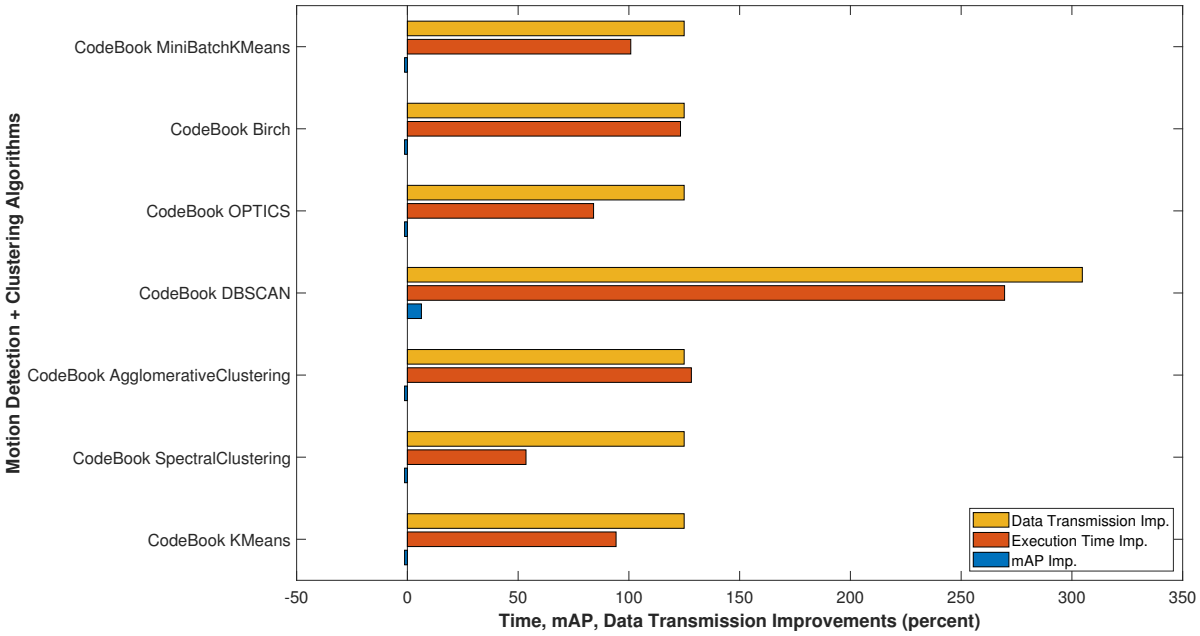


Figure 4.32: CodeBook Slice of Relative Performance Results for Enhanced YOLO Solution against the Raw YOLO for the Input Video from Laramie, Wyoming, United States

method, while the improvements in the required data transmissions and execution time are 568% and 721%, respectively.

The *ViBe*, *StaticFrameDifference*, and *CodeBook* motion detection methods could be considered if strictly no decrement in mAP performance is desired. Figure 4.34 is a slice of Figure 4.33 and focuses on the relative results for *ViBe* methods. *ViBe DBSCAN* combination enables around 255% improvement in required data transmissions, 204% in execution time, and around 8% in mAP. If a higher mAP improvement is desired, the *ViBe AgglomerativeClustering* method could be considered, resulting in improvements in required data transmissions, execution time, and mAP of 13%, 138%, and 173%, respectively.

Figure 4.35 is a slice of Figure 4.33 and focuses on the relative results for *StaticFrameDifference* methods. *StaticFrameDifference AgglomerativeClustering* combination provides around 4%, 12%, and 26% improvements in required data transmissions, execution time, and mAP, respectively. Figure 4.35 is another slice of Figure 4.33 and displays the results related to the *AdaptiveSelectiveBackgroundLearning* combination methods. *AdaptiveSelectiveBackgroundLearning AgglomerativeClustering* combination improves the execution time and required data transmission by more than 10% and 2%, respectively. Furthermore, it improves the mAP results by 35%.

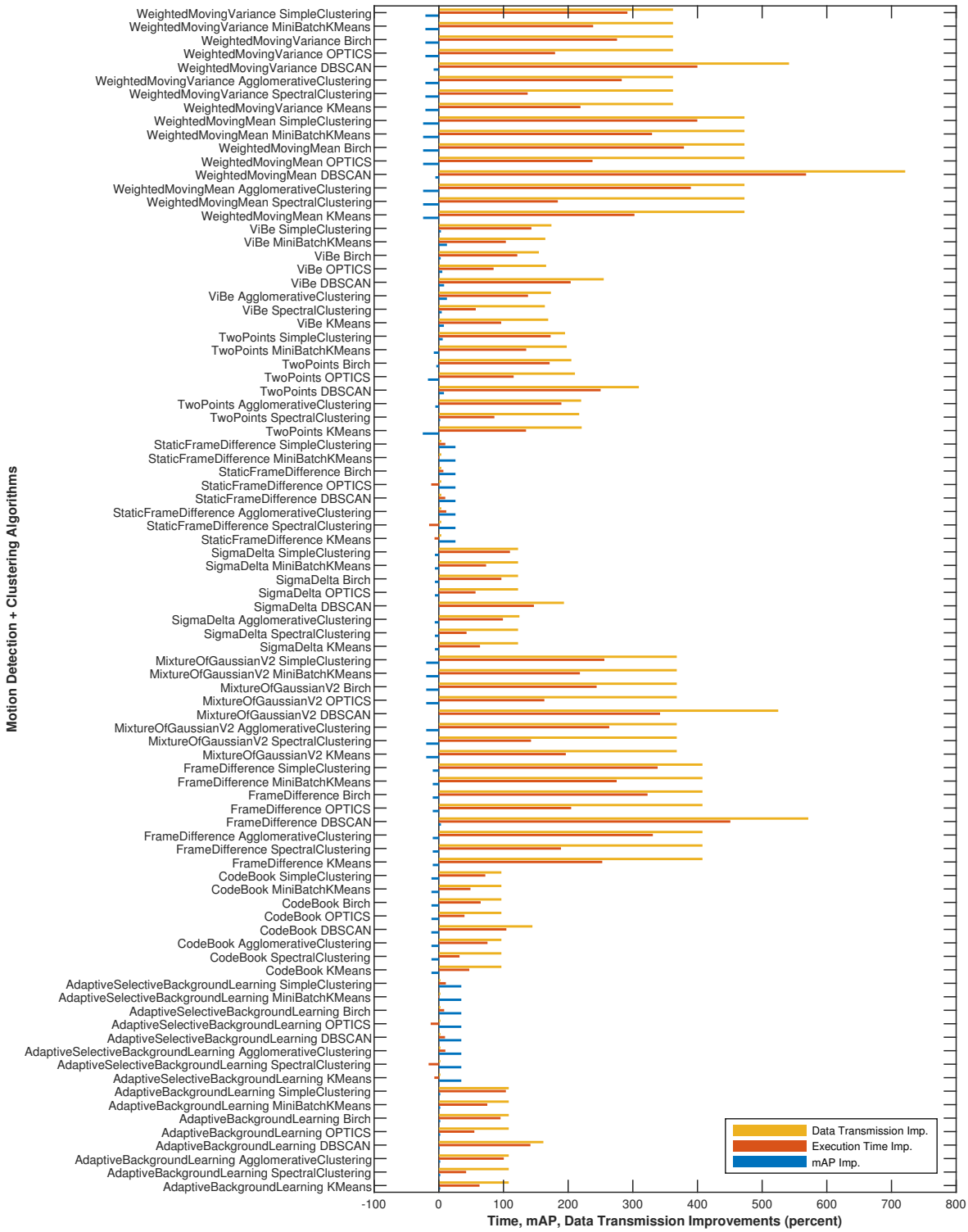


Figure 4.33: Relative Performance Results for Enhanced YOLO Solution against the Raw YOLO Considering the Input Video from Neath, Wales

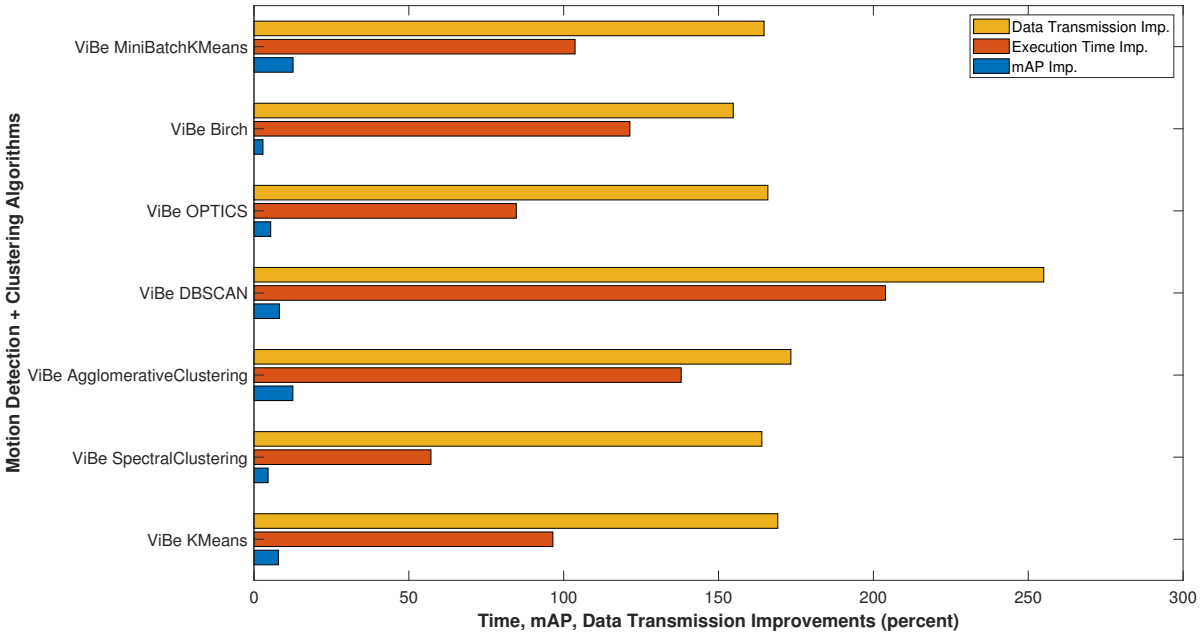


Figure 4.34: ViBe Slice of Relative Performance Results for Enhanced YOLO Solution against the Raw YOLO for the Input Video from Neath, Wales

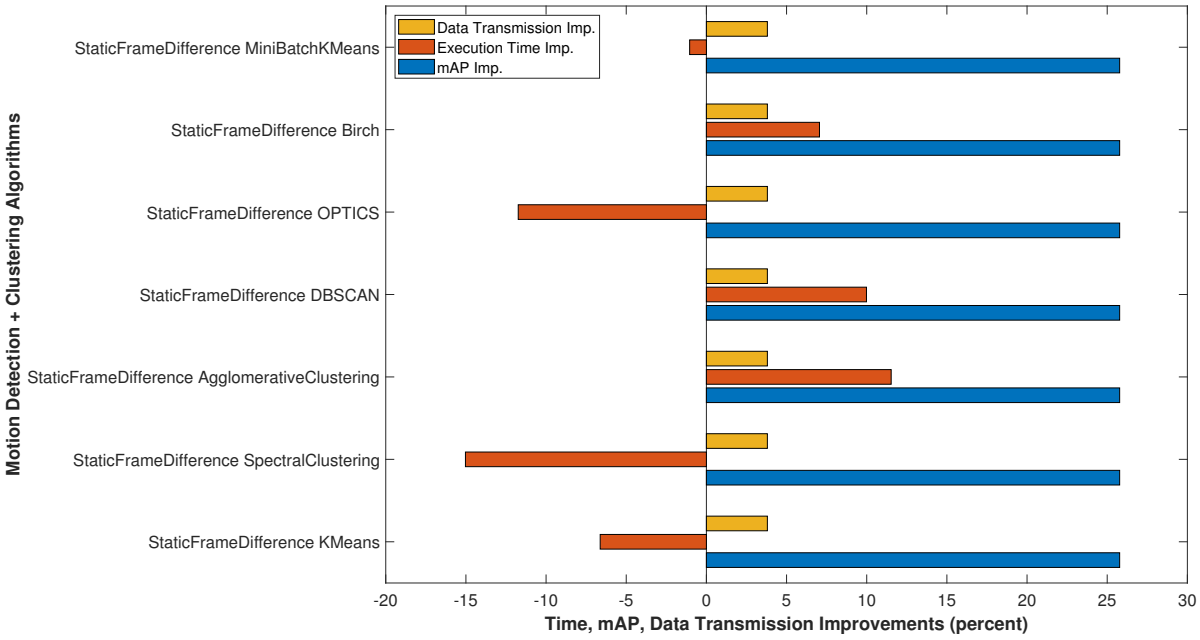


Figure 4.35: StaticFrameDifference Slice of Relative Performance Results for Enhanced YOLO Solution against the Raw YOLO for the Input Video from Laramie, Wyoming, United States

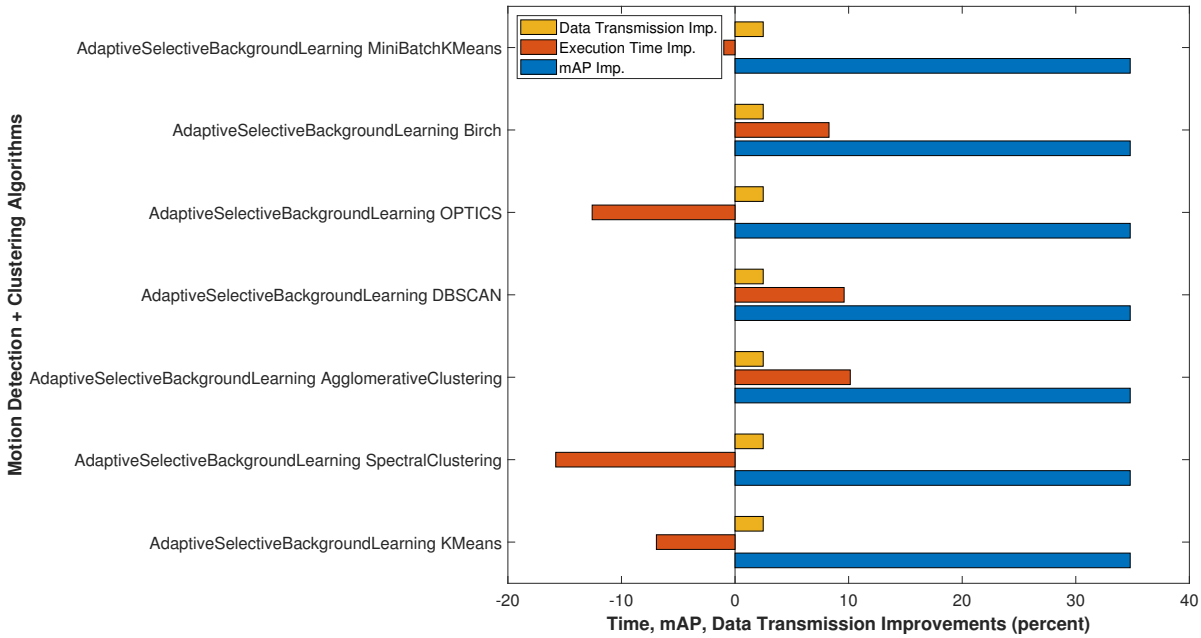


Figure 4.36: StaticFrameDifference Slice of Relative Performance Results for Enhanced YOLO Solution against the Raw YOLO for the Input Video from Laramie, Wyoming, United States

4.9 Conclusion

By taking advantage of video (related consecutive frames) characteristics, Enhanced YOLO could focus on the areas of importance in video frames and reduce the pixel volume delivered to the object detection neural network. This substantially reduces the required data transmission for cloud-based object detection neural network deployment. Additionally, the focus on areas of importance in video frames enables the system to represent an important object with more pixel information to the object detection neural network. This in turn gives the flexibility to choose a lower input resolution object detection deployment and, as a result, substantially reduces the required processing power for object detection tasks. Another effect of having more pixel information to represent an important object in video frames is enabling the object detection task to generate better and more accurate results.

To determine the performance of our proposed Enhanced YOLO solution, we have conducted 445 experiments by considering different motion detection algorithms, clustering techniques, and input video files. We have measured the mAP, average per-frame execution time, and transmitted data to the YOLO object detection neural network as performance-determining parameters. These numbers are compared by the results generated from the raw YOLO solution where no input filter was employed in the data input pipeline for the network.

The extracted results from our experiments show that the proposed Enhanced YOLO solution provides significant improvements in object detection tasks, especially in the required execution time (processing power) and data transmissions. We observed that by employing *WeightedMovingMean* as the motion detection algorithm and *DBSCAN* as the clustering technique, the improvement in required data transmissions is between 540% and 1810%. This means that the application of the Enhanced YOLO solution in object detection tasks could reduce the amount of data transmissions by a significant 94%.

By employing the aforementioned combination, the improvement in required execution time (processing

power) is between 400% and 1300%. This means the amount of required processing power could be cut by a significant 92% if the Enhanced YOLO solution is utilized in object detection tasks. These substantial improvements in the stated combination come with a relatively small cost of 8 to 30% in mAP results.

We have also observed that the Enhanced YOLO solution could be employed if strictly no decrement in mAP results is desired. In these cases, *StaticFrameDifference* and *CodeBook* could be considered as the selected motion detection algorithms. In some cases, *AdaptiveSelectiveBackgroundLearning* could also be considered. *DBSCAN* as the clustering technique is an excellent choice for improving the performance of object detection operations. By considering the mentioned combinations, it can be seen from the reported results that up to 35% improvement in mAP performance could be achieved. Furthermore, up to 304% improvement in the required data transmissions and up to 270% improvement in the required execution time (processing power) could be provided by these combinations.

CHAPTER 5 ACTIVITY DETECTION RECURRENT NEURAL NETWORK

5.1 Introduction

By taking advantage of the framework provided by Enhanced YOLO, object detection results could be efficiently generated for an input video file. Having access to consecutive video frames and the related detection results for each individual frame provides an excellent opportunity for an activity detection solution. This solution could treat the generated results with different timestamps (from different frames in the video) as time-series input data.

There are two elements involved in determining an activity. The first one is where each limb is located and how different limbs are moving during the activity. The second element is the objects around the involved person. What these objects are and where they are located are the two essential questions that clarify the second element in determining an activity. As an example, we can consider the act/activity of heading a ball where a soccer player jumps in the air and strikes the soccer ball with their head. First, the player prepares for the jump by bending their knees, bringing their hands toward their chest, and raising their elbows. When in the air, they move their head toward the ball and strike it. The object, namely the soccer ball, gets close to the player with a relatively high altitude, collides with their head (most probably the forehead), and gets away with a sudden change in direction and speed. These two mentioned elements (1: limb/joint positions and movements; 2: objects' classes, locations, and movements) are the most defining and important entities that could clearly classify an activity. Here, it is not important what the player is wearing or what haircut they have, what pattern is printed on the ball, or what kind of field the game is happening on (grass, in-door, asphalt, etc.).

If an RNN could be designed to take advantage of these two sources of information/data, not only can it process significantly less raw input data (as opposed to dealing with a high volume of pixels presented as multiple multi-channel frames), but it can also be trained with a significantly smaller amount of training

data as many variables are hidden from the equation/solution. For example, the previously mentioned soccer player's clothes are not important. So, by not considering this factor in the training process, the required labeled data to represent different clothing will be eliminated.

It is worth mentioning that we are not claiming that this is the way the human brain operates in determining an activity. This is our best guess in providing a highly efficient method for classifying different activities.

The second element in detecting an activity (objects' classes, locations, and movements) could be provided by our Enhanced YOLO (or any other object detection) solution. On the other hand, the first element should be provided by a pose estimation solution.

In this thesis, we have designed and implemented the aforementioned RNN solution. Section 5.2 describes different concepts and employed items in the design process. There, we describe the MediaPipe pose estimation toolkit from Google in addition to other building units of our RNN activity detection solution (like Convolution, GRU, Dropout, BatchNormalization, etc.). Next, in the Proposed Activity Detection RNN Solution section (5.3), we describe data preparation and the training processes in detail. Finally, we visually demonstrate some output samples of our proposed network in the Results section (5.4).

5.2 Background

5.2.1 Pose Estimation

In applications like measuring physical activities, sign language recognition, and full-body gesture control, human position estimation from a video is crucial. It can be used as the foundation for yoga, dance, and fitness applications, to name a few examples. In augmented reality, it can also enable the overlay of digital content and information on top of the physical world.

We have utilized MediaPipe Pose as our limbs/joints position and movement detector. As previously described, this satisfies the need for the first element involved in determining/classifying an activity. Me-

diaPipe is a set of cross-platform, customizable machine learning solutions for live and streaming media. Using Google BlazePose research [14], which also drives the ML (Machine Learning) Kit Pose Detection API in Android and IOS, MediaPipe Pose is a machine learning solution for high-fidelity body pose tracking, inferring 33 3D landmarks and a background segmentation mask on the full body from RGB video frames. For inference, most current state-of-the-art algorithms rely on powerful desktop environments, whereas Google's method delivers real-time performance on most recent mobile phones, desktops/laptops, in Python, and even on the web.

A two-step detector-tracker ML pipeline is used in the solution, which has been demonstrated to be effective in their MediaPipe Hands and MediaPipe Face Mesh solutions. The pipeline initially locates the person/pose region-of-interest (ROI) within the frame using a detector. Using the ROI-cropped frame as input, the tracker then predicts the pose landmarks and segmentation mask within the ROI. It's worth noting that in video use cases, the detector is only used when necessary, such as for the first frame, and when the tracker cannot detect body pose presence in the preceding frame. For other frames, the pipeline simply calculates the ROI based on the pose landmarks from the previous frame.

The detector is based on Google's lightweight BlazeFace model, which is utilized as a proxy for a human detector in MediaPipe Face Detection. It predicts two more virtual key points that accurately characterize the human body's center, rotation, and scale as a circle. Inspired by Leonardo da Vinci's Vitruvian Man, they forecast the midpoint of a person's hips; the radius of a circle circumscribing the entire person; and the incline angle of the line linking the shoulder and hip midpoints.

The landmark model in MediaPipe Pose estimates the location of 33 pose landmarks, as shown in Figure 5.1.

We have incorporated two different outputs from MediaPipe Pose in our activity detection network. The first one is $POSE\_LANDMARKS$ which is a list of pose landmarks. Each landmark consists of the

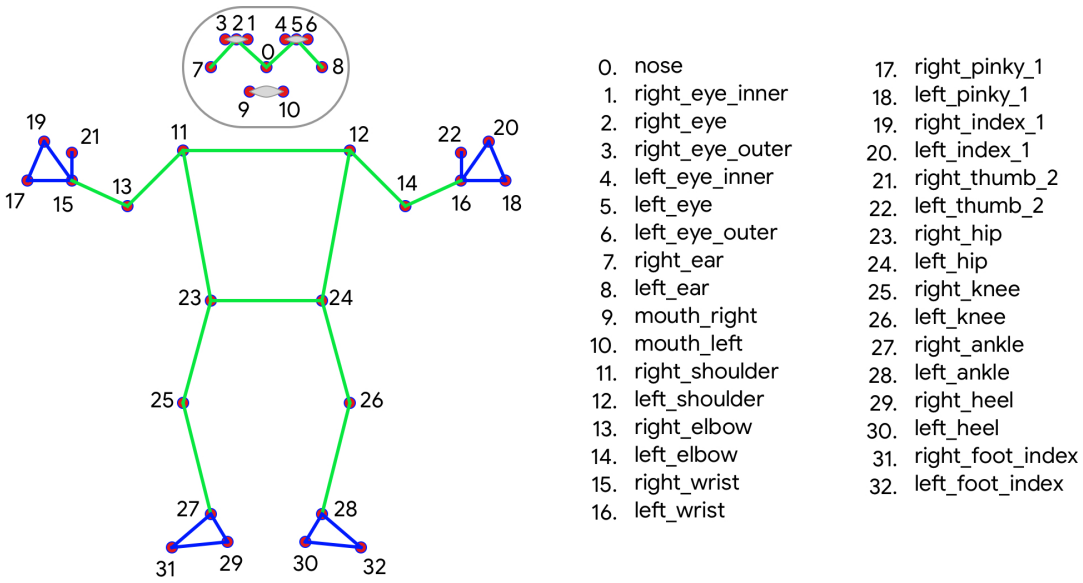


Figure 5.1: Location of 33 Pose Landmarks in MediaPipe Pose (Courtesy of Googblogs)

following:

- x and y: Landmark coordinates normalized to [0.0, 1.0] by the image width and height respectively.
- z: Defines the depth of a landmark, with the origin being the depth at the midpoint of the hips; the smaller the value, the closer the landmark is to the camera. The magnitude of z is measured using a scale that is similar to that of x.
- visibility: The likelihood of the landmark being seen (present and not obstructed) in the image is expressed as a number between 0.0 and 1.0.

The second output is $POSE\_WORLD\_LANDMARKS$. This is another set of world coordinates for pose landmarks. The following are the components of each landmark:

- x, y, and z: Three-dimensional real-world coordinates in meters, with the origin being in the middle of the hips.
- visibility: Identical to the visibility defined in the related pose landmarks.

5.2.2 Different Building Layers of Our Activity Detection Network

1D convolution (temporal convolution): This layer creates a convolution kernel (a window/array of weights) that is convolved with the layer input over a single spatial (or temporal) dimension to produce a tensor of outputs.

Batch normalization: This layer normalizes its inputs. Batch normalization is a transformation that keeps the mean output close to 0 and the standard deviation of the output close to 1.

ReLU Activation: Applies the rectified linear unit activation function. This returns the typical ReLU activation: max(x, 0), the element-wise maximum of 0, and the input tensor.

Dropout: The Dropout layer, which helps minimize overfitting, sets input units to 0 at random with a configurable frequency at each step during training time. Inputs that aren't set to 0 are scaled up so that the total sum remains the same for the layer.

GRU: One of the three built-in RNN layers in Keras, as introduced in [21]. Essentially, this unit could facilitate the traverse of a feature throughout different time steps. Keras is an open-source software library for artificial neural networks that provides a Python interface. It serves as a user interface for TensorFlow.

Dense: Regular densely connected Neural Networks layer. Dense implements the following operation: output = activation (dot(input, kernel) + bias), where activation is the element-wise activation function supplied as the activation parameter, kernel is the layer's weights matrix, and bias is the layer's bias vector.

5.3 Proposed Activity Detection RNN Solution

5.3.1 Labeled Data Preparation

The first step in creating our activity detection network is generating labeled data for a desired activity. Leaving a bag/baggage unattended is a sensitive activity that, if detected, could have important implications. As so, we have decided to select this activity as an example to show how preparation, training, and post-processing procedures are done. We have recorded a video of a person leaving a backpack unattended

multiple (precisely 103) times, each time in a slightly different way while walking in a different direction. The main directions are rear to front, front to rear, rear-left to front-right, and rear-right to front-left; additional angles between these main walking directions were also considered in the movement patterns. Different backpack holding positions (using a single shoulder strap or the top handle) and operating arms (left or right) were used in conducting the activity. Furthermore, different bending angles towards left or right was considered when leaving the backpack unattended. The total length of the recorded video file is 15 minutes and 26 seconds. The frame width and height are 1920 and 1080 pixels, respectively, and the video was recorded using a Panasonic DMC-LF1 camera. Figs. 5.2 and 5.3 show sample frames from this video. The video file will be publicly accessible online.

Figure 5.2: Sample Frame from the Recorded Video for Labeled Data Generation

We have recorded/stored the object detection results in addition to the pose estimation results for each individual frame in this video (x/input values). When that person completes an activity (leaving the bag unattended), we have marked the frame for that specific timestamp using our labeling tool. A single frame could have a target value of 0 or 1: 0 means that a specific frame/timestamp is not marking the completion of

the activity, while 1 means the opposite, indicating that a specific frame/timestamp is marking the completion of the activity (y/target values). Figure 5.3 shows some sample frames from the recorded video with the related x/input and y/target values stored as our labeled data. Starting from the left, the first frame is right after the activity is complete. In other words, it is right after the bag is left unattended. Accordingly, we consider the y/target value of 1 for this frame. The x/input values are detection and pose estimation results. The detection results are a set of detection records, each containing the label of the object, detection probability, and the bounding box encompassing the object. As was mentioned earlier, the pose estimation results are pose landmarks and pose world landmarks. Each contains 33 points, each of which in turn includes four values determining the location (in two different ways) and the visibility of the point. The second frame is not related to the completion of the desired activity. Therefore, the selected y/target values for this frame should be 0. The pose estimation in the last frame from the left is failing to produce any results. Because of this, we simply ignore this frame and do not include the related object detection results in the labeled data, despite the presence of detection results.

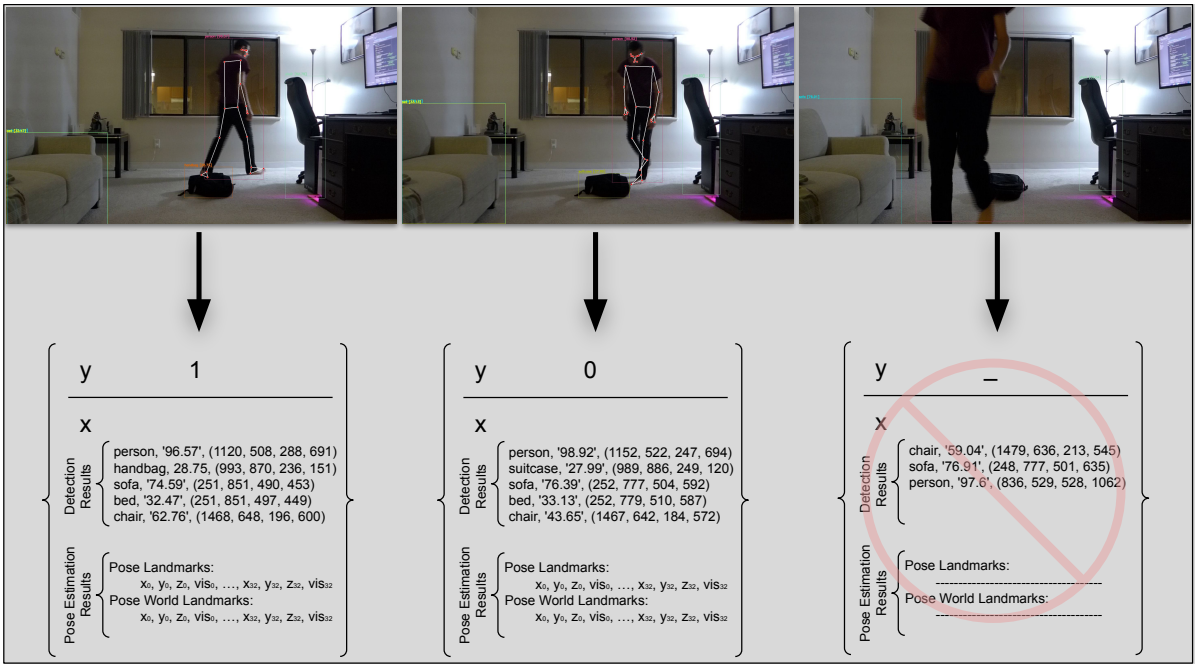


Figure 5.3: Sample Frames from the Recorded Video with the Related X/Input and Y/Target Values

5.3.2 Training/Test Sets Creation

We have created a tool to convert our labeled data to training/test sets. By using this conversion tool, we have generated our training/test sets using the following two-step process.

In the first step, each recorded datum (stored x/input and y/target values related to a single frame) is converted to a training/test record using the following operations:

1. Find the center of the hip in the 2D frame using the Pose Landmark points index 23 and 24.
2. Sort the detected objects in the frame using their distance from the hip center as the sorting key.
3. Consider the top N (configurable number of objects to include) objects in the sorted detection list.
4. Calculate the one-hot representation of each object label in the considered top N objects.
5. Use the Pose World Landmarks points with index 0, 11, 12, 13, 14, 15, 16, 23, 24, 25, 26, 27, and 28 to create an array consisting of x,y,z values from these world points.
6. Concatenate the top N one-hot representation of detected objects from the sorted detection list and the array created in the previous operation as input array for the detection network.
7. Consider y to be the target value for the array generated in the previous step.

A one-hot representation of an object label is calculated by finding the index value for that label in the list of the supported classes in YOLO and then creating a zero vector with a length of the mentioned list and setting the found index item in the zero vector to 1. Figure 5.4 shows an example of turning a label (handbag) to a one-hot representation.

The mentioned indexes for Pose World Landmarks are corresponding to the points representing nose, shoulders, elbows, wrists, hips, knees, and ankles (Figure 5.1).

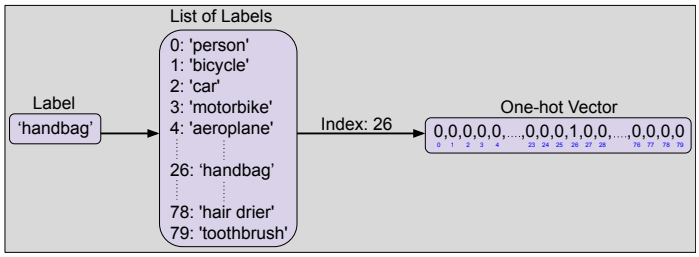


Figure 5.4: One-hot Representation of the Label "handbag"

Figure 5.5 demonstrates how stored x/input and y/target values related to a single frame are converted to a record suitable for training/test sets. Here the N (configurable number of objects to include) value is 3. As can be observed, the length of the x/input values for a single frame is 279. This length is calculated by multiplying N (3) with the length of a single one-hot vector (80) and adding the length of the array representing the item number 5 in the previous numbered list ($13 \times 3 = 39$).

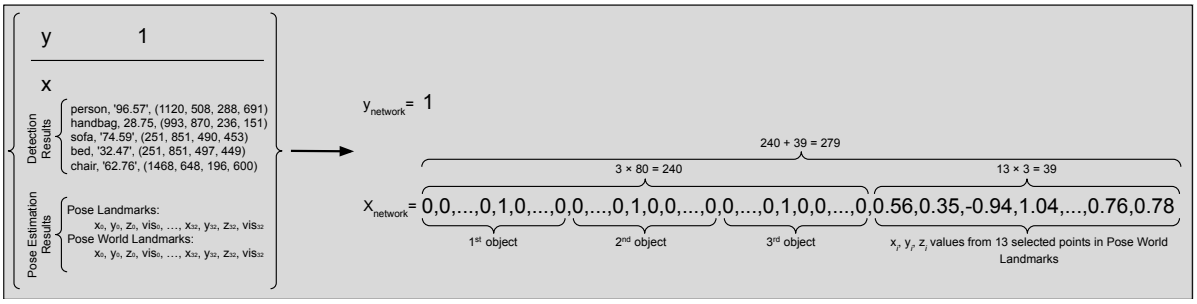


Figure 5.5: Converting a Labeled Record to a Training/Test Set Record

The second step handles the y/target values in the converted labeled data, now that our stored labeled data are converted to a proper format for the network consumption. The number of zeros in the set containing all the y/target values (Y) is significantly higher than the number of ones in that set. This is because a value of 1 for y/target represents the completion of the desired activity, meaning that every other frame that is not marked with completion of the desired task will have a target value of zero in the converted labeled data set.

Having a significantly higher number of zeros in Y will pose a challenge in the training and deployment of an RNN network that has to work with the mentioned data set. It should be noted that since the RNN will

consider the converted labeled data set as a time-series, the number of timestamps with y value as 1 will be very small in intended output from the network.

To mitigate this problem, as is the convention with time-series and RNN networks, we have to adjust our converted labeled data. The adjustment is done by iterating through the Y set and, whenever a y value of 1 is encountered, converting a fixed number of y values to 1. For example, if the Y set is {0,0,0,0,0,0,1,0,0,0,0,0} and the fixed number is set to 3, after adjustment, the Y set will be {0,0,0,0,0,0,1,1,1,1,0,0}.

After adjusting the Y set, the only item left is to distribute our labeled, converted, and adjusted data between a training and a test set (sometimes test set is referred to as development set if only two sets are considered in the programming process). Here, because our data is a time-series (the order of x/input and y/target values related to the consecutive frames is important in drawing conclusions from the data set), we cannot shuffle our items in the data set and distribute them between the training and test collections.

5.3.3 RNN Architecture

Figure 5.6 shows our implemented network architecture to determine the occurrence of an activity from the input time-series. This network is an RNN with a many-to-many configuration. The length of the input time-series is equal to the length of the output layer. This network is structurally similar to the networks that could be used in voice activation features [47]. The first convolution layer extracts low-level features from the input while reducing the number of required calculations for the remaining layers. Batch normalization operations are helpful in balancing the output values from each layer. This is especially important in our case, as we concatenate one-hot vectors related to detected objects with pose estimation results. Dropout layers help prevent the network from over-fitting the training set during the training procedure. GRU units are incorporated as the network has to process and make sense of time-series. It should be noted that the last layer (Dense + Sigmoid) is time-distributed, where all the units with different timestamps share the same parameters.

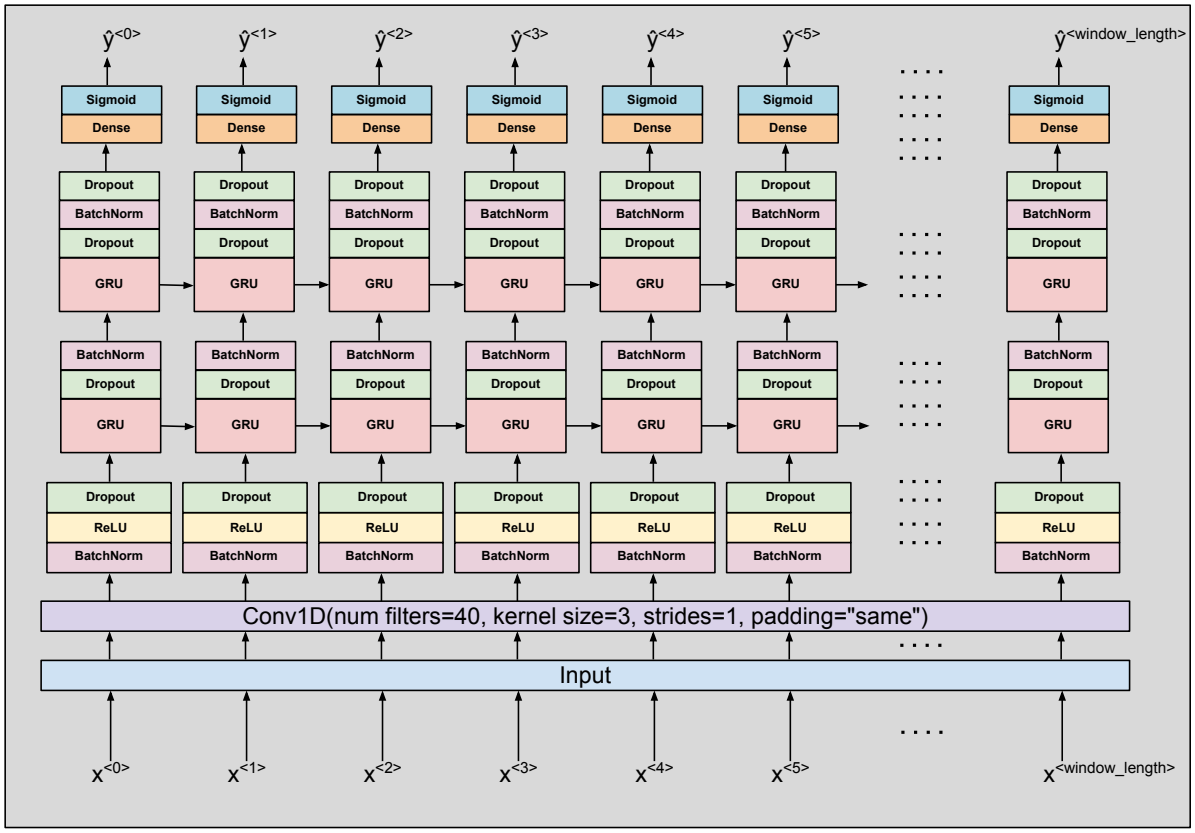


Figure 5.6: Architecture Overview for the Proposed Activity Detection Neural Network

5.3.4 RNN Training Procedure

By having access to the properly formatted training set and the implemented RNN architecture detailed in the previous section, the training procedure could be undertaken. As our input is a time-series and our network is designed as an RNN, a window/sample size should be considered. This window slides on the training data set and the present items in the window are fed to the network to be used in the training. The sliding window requires a sequence stride as well, where the value determines how many timestamps (different frames) the window should slide between two consecutive activations of the forward propagation in the network. Table 5.1 summarizes the parameters and their values involved in the training process.

Table 5.1: Summary of Activity Detection Training Parameters

Parameter	**Model/Value(s)**
Window/Sample Length	40
Sequence Stride	10
Optimizer	Adam with lr=0.001, β_1=0.9, β_2=0.999, $decay$=0.01
Loss Function	Binary Crossentropy
Model Accuracy After 100 Epochs	0.9447
Model Loss After 100 Epochs	0.1822
Dropouts Rate	0.8
Number of Units in GRU	128
Total Number of Parameters in the Network	199,185
Number of Trainable Parameters	198,593
Number of Non-trainable Parameters	592

5.4 Activity Detection Results

The same sliding and sequence stride mechanism mentioned in the training procedure is utilized in the deployment of the proposed RNN. The output is generated from the forward propagation operation on the sliding window. The length of the generated output array is 40 (the same as the window/sample length). Each item inside this array is a floating-point number with a value between 0 and 1 (sigmoid activation output).

By iterating through the output array and considering a threshold value, a new array could be generated where each element is 1 if the corresponding item in the output array is greater than the threshold value, and is 0 otherwise. We could consider this operation as an adjusted Hardmax function. By summing the numbers in this new array, a single integer value is generated. The larger this number, the higher the probability that the desired action was conducted in the time window used to generate the forward propagation output. By using a configurable threshold value and comparing it against the calculated sum value, the system can decide if the desired activity was conducted during the time window.

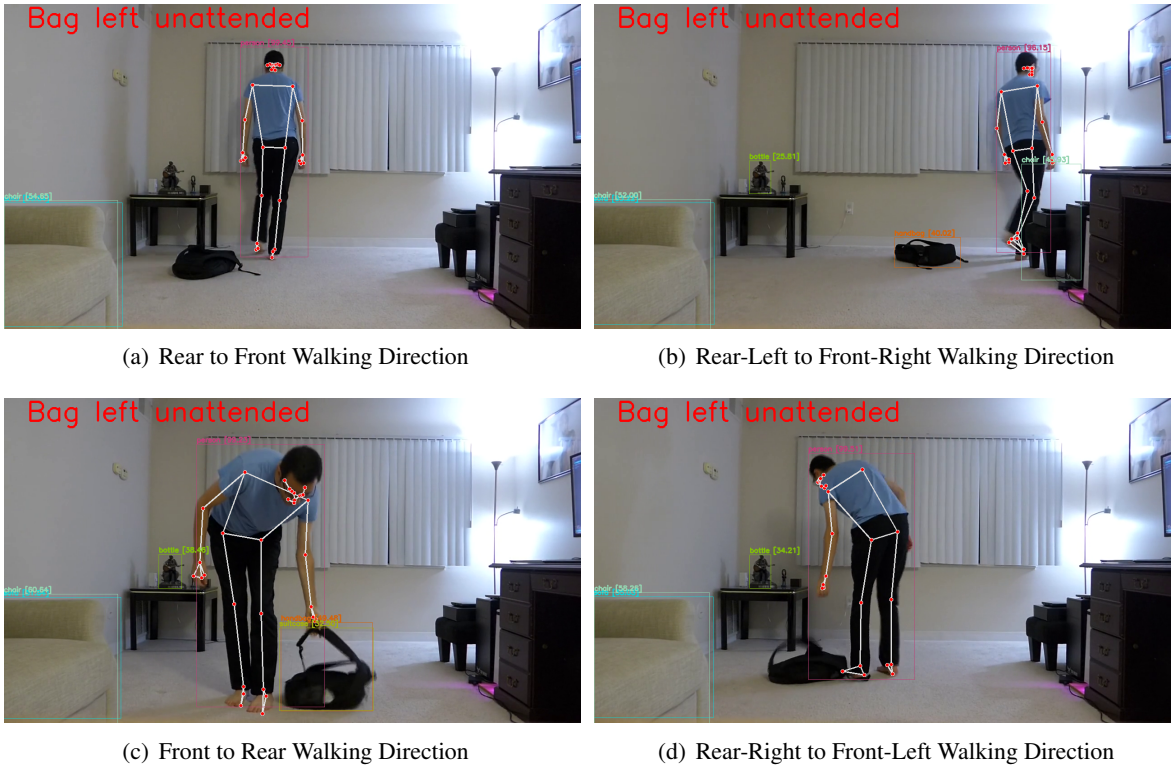
We have set the first threshold value used in the adjusted Hardmax function to 0.2 and the second threshold value used in the adjusted output array's sum evaluation to be 1.

To test our network, we have recorded a video with different clothing and lightning conditions from the

video used in the training procedure. The same movement patterns described earlier in Subsection 5.3.1 were used to record this video. The length of the video is 60 seconds and the frame width and height are 1280 and 720 pixels, respectively. The same Panasonic DMC-LF1 camera used in the training phase was utilized to record the evaluation video. The video file will be publicly available online.

The results show that our network is capable of correctly recognizing the activity with 86% accuracy. Here, the accuracy was calculated by counting the number of times the activity was correctly categorized, divided by the total number of activities in the video that were covered by the camera. Unlike the percentage of correctly estimated y/target values for individual frames which yield a higher accuracy value, this is a realistic measurement. The former measurement could have a higher value as the target values are dominated by the number of zeros.

The achieved accuracy value is very promising, especially when considering that our network was trained on only a single training video. Additionally, our TensorFlow implementation of the network took less than 4 minutes to be trained for this activity on an average consumer-level desktop PC (Ryzen 7 2700x and GTX 1080). Solutions that consider a volume of raw pixel values to provide the detection results are simply incapable of attaining these accuracy values when trained on as much data as we have utilized in our solution. We have intentionally chosen to train our presented network with just a single video file to demonstrate the benefits of using powerful and already trained neural networks to design and implement a solution. Figure 5.7 shows four sample frames when the network is detecting the activity (leaving a bag/backpack unattended). Each frame shows a different walking direction when the activity is being detected. Figures 5.7(a), 5.7(b), 5.7(c), and 5.7(d) are related to the rear to front, rear-left to front-right, front to rear, and rear-right to front-left walking directions, respectively.



(a) Rear to Front Walking Direction

(b) Rear-Left to Front-Right Walking Direction

(c) Front to Rear Walking Direction

(d) Rear-Right to Front-Left Walking Direction

Figure 5.7: Sample Frames When the Action is Being Detected by the Activity Detection Network

5.5 Conclusion

Detection results from object detection neural networks can provide an opportunity for building efficient solutions for activity detection tasks. By using fast pose estimation frameworks like MediaPipe Pose in addition to captured detection results, the domain of the activity detection problem shifts from a volume of RGB pixel values to a time-series of relatively small one-dimensional arrays. This enables the activity detection solution to take advantage of very capable neural networks being backed by thousands of hours of training on massive clusters of GPUs. Thus, it is possible to create capable activity detection solutions by adopting significantly smaller training sets and training processing hours.

CHAPTER 6 SUMMARY AND FUTURE WORK

6.1 Summary

This dissertation considers computer vision (CV) systems in which a central monitoring station receives and analyzes video streams captured and transmitted wirelessly by multiple cameras. It addresses how bandwidth can be allocated to different cameras by presenting a cross-layer solution that improves overall detection or recognition accuracy. In addition, unlike previous work, it presents and develops a real CV system before providing a detailed experimental analysis of cross-layer optimization. Other distinguishing characteristics of the developed solution include the use of the popular HTTP streaming approach, the use of homogeneous as well as heterogeneous cameras with varying capabilities and limitations, and the inclusion of a new algorithm for estimating the effective medium airtime. The results show that the proposed solution improves CV accuracy significantly.

Furthermore, because neural networks are a major component of CV algorithms, this dissertation includes an improved object detection neural network system. The proposed system focuses on the areas of importance in consecutive frames by considering inherent video characteristics and employing different motion detection and clustering algorithms, allowing the system to distribute the detection task dynamically and efficiently among multiple deployments of object detection neural networks. Our thorough experimental results suggest that our proposed solution can improve mAP, execution time, and necessary data transmissions to object detector networks.

Finally, we present an effective activity detection RNN by having access to object detection results and taking advantage of quick posture estimation techniques, as identifying an activity provides considerable automation prospects in computer vision systems. The domain of activity detection shifts from a volume of RGB (red, green, and blue) pixel values to a time-series of relatively small one-dimensional arrays by integrating object detection and pose estimation results. This enables the activity detection system to use

highly capable neural networks that have been trained over thousands of hours on massive GPU clusters. As a result, capable activity detection methods can be developed with far fewer training sets and processing hours.

6.2 List of Publications

6.2.1 Published:

- Experimental Analysis of Optimal Bandwidth Allocation in Computer Vision Systems. *IEEE Transactions on Circuits and Systems for Video Technology*, Accepted in December 2020.

- Experimental analysis of bandwidth allocation in automated video surveillance systems. *ACM Multimedia*, pages 1457-1464, October 2017.

- A clustering approach for controlling PTZ cameras in automated video surveillance. *IEEE International Symposium on Multimedia*, pages 333 – 336, December 2016.

6.2.2 Under Review:

AWARE: An Autonomous System for Optimal Control of PTZ Cameras. Second Revision submitted to *ACM Transactions on Autonomous and Adaptive Systems* in October 2021.

6.3 Future Work

There are areas in our proposed solutions that could be additionally investigated to further improve the performance of our system:

- In the Enhanced YOLO project, rather than a singular region of interest (ROI) in video frames, multiple ROIs could be investigated (with each one fed to a YOLO deployment with a relatively lower input resolution compared to the singular case).

- Using word embedding in the activity detection network as opposed to one-hot vectors for different classes of objects.

- Experimenting with other network architectures for activity detection (including tuning hyperparameters like the window width and the gap size for time-series).

- Multiple activity detection on a single network by expanding the output to a one-hot vector form.

REFERENCES

[1] Computer vision technologies and markets. https://omdia.tech.informa.com/OM011959/Computer-Vision-Technologies-and-Markets. Accessed: 2021-10-01.

[2] Surveillance camera statistics: Which city has the most CCTV cameras? https://www.comparitech.com/vpn-privacy/the-worlds-most-surveilled-cities. Accessed: 2021-06-01.

[3] TwoPoints background subtraction algorithm. https://github.com/andrewssobral/bgslibrary/tree/master/src/algorithms/TwoPoints. Accessed: 2021-02-01.

[4] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[5] M. S. Al-Hadrusi, N. J. Sarhan, and S. G. Davani. A clustering approach for controlling PTZ cameras in automated video surveillance. In *2016 IEEE International Symposium on Multimedia (ISM)*, pages 333–336, 2016.

[6] M. Alsmirat and N. Sarhan. Cross-layer optimization for many-to-one wireless video streaming systems. *Multimedia Tools and Applications*, 77(19):24789–24811, 2018.

[7] M. Alsmirat and N. J. Sarhan. Intelligent optimization for automated video surveillance at the edge: A cross-layer approach. *Simulation Modelling Practice and Theory*, 105:102171, 2020.

[8] S. R. Alvar and I. V. Bajić. MV-YOLO: Motion vector-aided tracking by semantic object detection. In *Proceeding of IEEE 20th International Workshop on Multimedia Signal Processing (MMSP)*, pages

1–5, 2018.

[9] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. Optics: Ordering points to identify the clustering structure. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, SIGMOD '99, page 49–60, New York, NY, USA, 1999. Association for Computing Machinery.

[10] A. Arar, A. A. El-Sherif, A. Mohamed, and V. C. M. Leung. Optimum power and rate allocation in cluster based video sensor networks. In *2015 International Conference on Computing, Networking and Communications (ICNC)*, pages 183–188, 2015.

[11] D. Arthur and S. Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, page 1027–1035, USA, 2007. Society for Industrial and Applied Mathematics.

[12] Y. S. Baguda. Energy-efficient biocooperative video-aware QoS-based multiobjective cross-layer optimization for wireless networks. *IEEE Access*, 8:127034–127047, 2020.

[13] O. Barnich and M. Van Droogenbroeck. ViBe: A universal background subtraction algorithm for video sequences. *IEEE Transactions on Image processing*, 20(6):1709–1724, 2010.

[14] V. Bazarevsky, I. Grishchenko, K. Raveendran, T. Zhu, F. Zhang, and M. Grundmann. BlazePose: On-device real-time body pose tracking. 2020.

[15] M. Benetti, M. Gottardi, T. Mayr, and R. Passerone. A low-power vision system with adaptive background subtraction and image segmentation for unusual event detection. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 65(11):3842–3853, 2018.

[16] G.-A. Bilodeau, J.-P. Jodoin, and N. Saunier. Change detection in feature space using local binary similarity patterns. In *2013 International Conference on Computer and Robot Vision*, pages 106–112, 2013.

[17] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. YOLOv4: Optimal speed and accuracy of object detection. *arXiv*, 2004.10934, 2020.

[18] M. Burić, M. Pobar, and M. Ivašić-Kos. Object detection in sports videos. In *Proceeding of 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1034–1039, 2018.

[19] X. Chen and Y. Han. Multi-task CNN model for action detection. In *Proceeding of IEEE Visual Communications and Image Processing (VCIP)*, pages 1–4, 2018.

[20] Z. Chen, K. Fan, S. Wang, L.-Y. Duan, W. Lin, and A. Kot. Lossy intermediate deep learning feature compression and evaluation. In *Proceedings of the 27th ACM International Conference on Multimedia*, MM '19, page 2414–2422, New York, NY, USA, 2019. Association for Computing Machinery.

[21] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation, 2014.

[22] S.-P. Chuah, Y.-P. Tan, and Z. Chen. Rate and power allocation for joint coding and transmission in wireless video chat applications. *IEEE Transactions on Multimedia*, 17(5):687–699, 2015.

[23] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 24(5):603–619, 2002.

[24] S. G. Davani and N. J. Sarhan. Experimental analysis of optimal bandwidth allocation in computer vision systems. *IEEE Transactions on Circuits and Systems for Video Technology*, 31(10):4121–4130, 2021.

[25] J. Deng, J. Guo, N. Xue, and S. Zafeiriou. ArcFace: Additive angular margin loss for deep face recognition. In *Proceeding of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4685–4694, 2019.

[26] W. Deng, R. Patil, L. Najjar, Y. Shi, and Z. Chen. Incorporating community detection and cluster-

ing techniques into collaborative filtering model. *Procedia Computer Science*, 31:66–74, 2014. 2^{nd} International Conference on Information Technology and Quantitative Management, ITQM 2014.

[27] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, 1996.

[28] B. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315(5814):972–976, 2007.

[29] A. Guan, S. H. Bayless, and R. Neelakantan. Connected vehicle insights. *Trends in computer vision. Technology scan series*, 2012, 2011.

[30] M. Guo, M. Ammar, and E. Zegura. V3: a vehicle-to-vehicle live video streaming architecture. In *Third IEEE International Conference on Pervasive Computing and Communications*, pages 171–180, 2005.

[31] H. R. Hamandi and N. J. Sarhan. Novel analytical models of face recognition accuracy in terms of video capturing and encoding parameters. In *Proceedings of the 2020 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, 2020.

[32] J. He, D. Wu, X. Xie, M. Chen, Y. Li, and G. Zhang. Efficient upstream bandwidth multiplexing for cloud video recording services. *IEEE Transactions on Circuits and Systems for Video Technology*, 26(10):1893–1906, 2016.

[33] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

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

[35] Z. He, Y. Liang, L. Chen, I. Ahmad, and D. Wu. Power-rate-distortion analysis for wireless video communication under energy constraints. *IEEE Transactions on Circuits and Systems for Video Tech-*

nology, 15(5):645–658, 2005.

[36] Z. He and D. Wu. Resource allocation and performance analysis of wireless video sensors. *IEEE Transactions on Circuits and Systems for Video Technology*, 16(5):590–599, 2006.

[37] C.-H. Hsu and M. Hefeeda. A framework for cross-layer optimization of video streaming in wireless networks. *ACM Transactions on Multimedia Computing Communications and Applications*, 7:5:1–5:28, 2011.

[38] J. Huang, Z. Li, M. Chiang, and A. K. Katsaggelos. Pricing-based rate control and joint packet scheduling for multi-user wireless uplink video streaming. In *Proceeding of 15th International Packet Video Workshop (PV2006)*, Dec 2006.

[39] K. Kim, T. Chalidabhongse, D. Harwood, and L. Davis. Real-time foreground–background segmentation using codebook model. *Real-Time Imaging*, 11:172–185, 2005.

[40] P. Korshunov. Rate-accuracy tradeoff in automated, distributed video surveillance systems. In *Proceedings of the 14^{th} ACM International Conference on Multimedia*, MM '06, page 887–889, New York, NY, USA, 2006. Association for Computing Machinery.

[41] W. A. Latif and C. C. Tan. SmartArgos: Improving mobile surveillance systems with software defined networks. In *2015 IEEE Conference on Communications and Network Security (CNS)*, pages 763–764, 2015.

[42] B.-F. Lin, Y.-M. Chan, L.-C. Fu, P.-Y. Hsiao, L.-A. Chuang, S.-S. Huang, and M.-F. Lo. Integrating appearance and edge features for sedan vehicle detection in the blind-spot area. *IEEE Transactions on Intelligent Transportation Systems*, 13(2):737–747, 2012.

[43] Y. Lu, Y. Chen, D. Zhao, and H. Li. Hybrid deep learning based moving object detection via motion prediction. In *Proceeding of Chinese Automation Congress (CAC)*, pages 1442–1447, 2018.

[44] S. C. Madeira and A. L. Oliveira. Biclustering algorithms for biological data analysis: a survey. *IEEE/ACM transactions on computational biology and bioinformatics*, 1(1):24–45, 2004.

[45] A. Manzanera and J. C. Richefeu. A new motion detection algorithm based on sigma-delta background estimation. *Pattern Recognition Letters*, 28(3):320–328, 2007.

[46] B. Mocanu, R. Tapu, and T. Zaharia. Seeing without sight — an automatic cognition system dedicated to blind and visually impaired people. In *Proceeding of IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 1452–1459, 2017.

[47] A. Ng, K. Katanforoosh, and Y. B. Mourri. Sequence models [MOOC]. Coursera. `https://www.coursera.org/learn/nlp-sequence-models/home/info`, 2019.

[48] F. Nielsen. *Hierarchical Clustering*, pages 195–211. 02 2016.

[49] G. Oltean, C. Florea, R. Orghidan, and V. Oltean. Towards real time vehicle counting using YOLO-tiny and fast motion estimation. In *Proceeding of IEEE 25th International Symposium for Design and Technology in Electronic Packaging (SIITME)*, pages 240–243, 2019.

[50] C. Pan, D. Shi, N. Guan, Y. Zhang, L. Wang, and S. Jin. Learning to track by bi-directional long short-term memory networks. In *Proceeding of IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pages 783–790, 2019.

[51] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates,

Inc., 2019.

[52] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[53] J. Piao, T. Inoshita, and K. Iwamoto. Carried object recognition via location relation with body parts. In *Proceeding of IEEE International Conference on Image Processing (ICIP)*, pages 3058–3062, 2019.

[54] J. Redmon. Darknet: Open source neural networks in c. `http://pjreddie.com/darknet/`, 2013–2016.

[55] M. H. Sanan, K. A. Alam, M. Z. Rafique, and B. Khan. Quality of service enhancement in wireless LAN: A systematic literature review. In *Proceeding of 13th International Conference on Mathematics, Actuarial Science, Computer Science and Statistics (MACS)*, pages 1–8, 2019.

[56] B. A. B. Sarif, M. Pourazad, P. Nasiopoulos, and V. C. M. Leung. A study on the power consumption of H.264/AVC-based video sensor network. *International Journal of Distributed Sensor Networks*, 11(10):304787, 2015.

[57] B. A. B. Sarif, M. T. Pourazad, P. Nasiopoulos, and V. C. M. Leung. Analysis of power consumption of H.264/AVC-based video sensor networks through modeling the encoding complexity and bitrate. In *ICDS 2014*, 2014.

[58] F. Schroff, D. Kalenichenko, and J. Philbin. FaceNet: A unified embedding for face recognition and clustering. *CoRR*, abs/1503.03832, 2015.

[59] D. Sculley. Web-scale k-means clustering. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, page 1177–1178, New York, NY, USA, 2010. Association for Computing Machinery.

[60] K. Sehairi, F. Chouireb, and J. Meunier. Comparative study of motion detection methods for video surveillance systems. *Journal of Electronic Imaging*, 26(2):1–29, 2017.

[61] H. Shiang and M. van der Schaar. Information-constrained resource allocation in multicamera wireless surveillance networks. *IEEE Transactions on Circuits and Systems for Video Technology*, 20(4):505–517, 2010.

[62] A. Sobral and T. Bouwmans. BGS library: A library framework for algorithm's evaluation in foreground/background segmentation. In *Background Modeling and Foreground Detection for Video Surveillance*. CRC Press, Taylor and Francis Group, 2014.

[63] A. Sobral and A. Vacavant. A comprehensive review of background subtraction algorithms evaluated with synthetic and real videos. *Computer Vision and Image Understanding*, 122:4–21, 2014.

[64] M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis, and Machine Vision*. 2014.

[65] P. St-Charles and G. Bilodeau. Improving background subtraction using local binary similarity patterns. In *IEEE Winter Conference on Applications of Computer Vision*, pages 509–515, 2014.

[66] P. St-Charles, G. Bilodeau, and R. Bergevin. Flexible background subtraction with self-balanced local sensitivity. In *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 414–419, 2014.

[67] P. St-Charles, G. Bilodeau, and R. Bergevin. Universal background subtraction using word consensus models. *IEEE Transactions on Image Processing*, 25(10):4768–4781, 2016.

[68] P.-L. St-Charles and G.-A. Bilodeau. Improving background subtraction using local binary similarity patterns. In *IEEE Winter Conference on Applications of Computer Vision*, pages 509–515, 2014.

[69] C. Stauffer and W. Grimson. 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, 1999.

[70] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi. Inception-v4, Inception-ResNet and the impact of residual connections on learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 4278–4284, 2017.

[71] R. Szeliski. *Computer Vision: Algorithms and Applications*. Texts in Computer Science. Springer London, 2010.

[72] J. Tian, H. Zhang, D. Wu, and D. Yuan. Interference-aware cross-layer design for distributed video transmission in wireless networks. *IEEE Transactions on Circuits and Systems for Video Technology*, 26(5):978–991, 2016.

[73] R. Vial, H. Zhu, Y. Tian, and S. Lu. Search video action proposal with recurrent and static YOLO. In *Proceeding of IEEE International Conference on Image Processing (ICIP)*, pages 2035–2039, 2017.

[74] C. Whitelam, E. Taborsky, A. Blanton, B. Maze, J. Adams, T. Miller, N. Kalka, and A. K. Jain. IARPA Janus benchmark-b face dataset. In *Proceeding of IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 592–600, 2017.

[75] E. Yaacoub and F. Filali. Cluster based V2V communications for enhanced QoS of SVC video streaming over vehicular networks. In *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 678–683, 2014.

[76] Y. Yang, K. Han, S. Lee, and J. Lee. Temporal difference based adaptive object detection (ToDo) platform at edge computing system. In *Proceeding of IEEE 17th Annual Consumer Communications Networking Conference (CCNC)*, pages 1–2, 2020.

[77] H. Zhang, Y. Zheng, M. A. Khojastepour, and S. Rangarajan. Cross-layer optimization for streaming scalable video over fading wireless networks. *IEEE Journal on Selected Areas in Communications*, 28(3):344–353, 2010.

[78] S. Zhang, T. Wang, C. Wang, Y. Wang, G. Shan, and H. Snoussi. Video object detection base on RGB

and optical flow analysis. In *Proceeding of 2nd China Symposium on Cognitive Computing and Hybrid Intelligence (CCHI)*, pages 280–284, 2019.

[79] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. *ACM SIGMOD Record*, 25(2):103–114, 1996.

[80] Z. Zivkovic and F. Van Der Heijden. Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern Recognition Letters*, 27(7):773–780, 2006.

ABSTRACT

DESIGN OF COMPUTER VISION SYSTEMS FOR OPTIMIZING THE THREAT DETECTION ACCURACY

by

SINA GHOLAMNEJAD DAVANI

May 2022

Advisor: Dr. Nabil Sarhan

Major: Computer Engineering

Degree: Doctor of Philosophy

This dissertation considers computer vision (CV) systems in which a central monitoring station receives and analyzes the video streams captured and delivered wirelessly by multiple cameras. It addresses how the bandwidth can be allocated to various cameras by presenting a cross-layer solution that optimizes the overall detection or recognition accuracy. The dissertation presents and develops a real CV system and subsequently provides a detailed experimental analysis of cross-layer optimization. Other unique features of the developed solution include employing the popular HTTP streaming approach; utilizing homogeneous cameras as well as heterogeneous ones with varying capabilities and limitations; and including a new algorithm for estimating the effective medium airtime. The results show that the proposed solution significantly improves the CV accuracy.

Additionally, the dissertation features an improved neural network system for object detection. The proposed system considers inherent video characteristics and employs different motion detection and clustering algorithms to focus on the areas of importance in consecutive frames, allowing the system to distribute the detection task dynamically and efficiently among multiple deployments of object detection neural networks. Our experimental results indicate that our proposed method can enhance the mAP (mean average precision), execution time, and required data transmissions to object detection networks.

Finally, as recognizing an activity provides significant automation prospects in CV systems, the dissertation presents an efficient activity-detection recurrent neural network that utilizes fast pose/limbs estimation approaches. By combining object detection with pose estimation, the domain of activity detection is shifted from a volume of RGB (red, green, and blue) pixel values to a time-series of relatively small one-dimensional arrays, thereby allowing the activity detection system to take advantage of highly capable neural networks that have been trained on large GPU clusters for thousands of hours. Consequently, capable activity detection systems with considerably fewer training sets and processing hours can be built.

AUTOBIOGRAPHICAL STATEMENT

Sina G. Davani is an ADAS Senior Software Engineer at CONTINENTAL AG. While working in the automotive industry, he has been tasked with developing software solutions for advanced driver assistance systems. He received his B.S. degree in Computer Engineering from Isfahan University of Technology in 2012 and his M.S. degree in the same field from Wayne State University in 2017. His main research interests are automated video surveillance; systems simulation; bandwidth adaptations for video streaming systems; machine learning; computer vision; and parallel and distributed system design and implementation.