

# **Dynamic Visual Data Prioritization in Automated Object Detection Systems for Multi-Camera Surveillance**

by

James A. D. Cameron

**Bachelors of Science in Engineering, University of New Brunswick, 2017**

**A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE  
REQUIREMENTS FOR THE DEGREE OF**

**Masters of Science in Engineering**

In the Graduate Academic Unit of Electrical and Computer Engineering

Supervisor(s): Mary E. Kaye, MScE, Electrical and Computer Engineering  
Erik J. Scheme, PhD, Electrical and Computer Engineering  
Examining Board: Julian Meng, PhD, Electrical and Computer Engineering  
Yevgen Biletskiy, PhD, Electrical and Computer Engineering  
External Examiner: Superio Ray, PhD, Computer Science

This thesis is accepted

Dean of Graduate Studies

**THE UNIVERSITY OF NEW BRUNSWICK**

**March, 2019**

©James A. D. Cameron, 2019

# Abstract

Modern automated object detection systems are key tools in surveillance applications. These systems rely on computationally expensive computer vision algorithms that perform object detection on visual data created by surveillance cameras. Due to the nature of surveillance systems, this visual data must be processed accurately and in real-time. However, many of the frames that are created by the surveillance cameras may be of low importance, providing no useful information to the object detection system. Sub-sampling the surveillance data by prioritizing important camera frames can greatly reduce unnecessary computation. Several works have been conducted on dynamic visual data sub-sampling using various modalities of information (ie. spatial or temporal information) for prioritization. However, few works have used different modalities of information together for visual data prioritization. Furthermore, given the fast pace of the research space, only a small subset of works have implemented visual data prioritization with modern computer vision algorithms in mind. This work evaluates several individual prioritization metrics, that use different modalities of information to prioritize visual data, for use with modern object detection algorithms. This thesis presents an ensemble method that uses a KNN regressor to combine the best of the previously evaluated metrics. This dynamic approach was shown to increase the detection rate in an indoor surveillance scenario by over 60% compared to a static sub-sampling baseline.

# Acknowledgements

I would first like to thank the University of New Brunswick (UNB) and the Natural Sciences and Engineering Research Council of Canada for supporting the fantastic experience that was my Master's degree. The faculty and staff of the Electrical and Computer Engineering department and Institute of Biomedical Engineering (IBME) are great people and I would encourage anyone interested in a graduate degree to speak with them.

Without my supervisors none of this would be possible. I am grateful to have had two excellent supervisors during my degree. Thank you Dr. Erik Scheme for the energy and knowledge that you brought to my education. Professor Mary Kaye, thank you for the wealth of time and experience that you gave.

This thesis was in part inspired by the security technology startup EhEye. My industry mentor and friend Phil Munz, along with the rest of the team, provided me with great feedback and support. I enjoyed working with them and hope to continue being a part of the team.

To all my friends from classes, residence, the IBME, and the lab: thank you. You have all made the long days shorter and the big challenges smaller. From my time at UNB I have met many amazing people, too many to mention here. However, I have spent a very large amount of time with a special group of friends. To Evan Campbell, Louis Richard, and Xavier St. Onge: Cheers! Also I want to thank my friend Shriram Raghu for all the advice and insightful chats in our lab during the course of my degree.

Outside of school I have a phenomenal group of people in my life. To Kathryn Chamberlain: thank you for keeping me together, you are the peanut butter to my strawberry jam. Finally, I would like to thank whole family for their love and support; my parents have always cheered for me from the front row.

# Table of Contents

<b>Abstract</b>	<b>ii</b>
<b>Acknowledgments</b>	<b>iii</b>
<b>Table of Contents</b>	<b>vii</b>
<b>List of Tables</b>	<b>viii</b>
<b>List of Figures</b>	<b>xii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background . . . . .	1
1.2 Problem Statement . . . . .	2
1.3 Contribution . . . . .	3
1.4 Thesis Structure . . . . .	4
<b>2 Literature Review</b>	<b>5</b>
2.1 Automated Object Detection System Design . . . . .	5
2.1.1 Object Detection . . . . .	6
2.1.2 Processing Hardware . . . . .	8
2.2 Visual Data Prioritization . . . . .	9
2.2.1 Motivation . . . . .	9
2.2.2 Methods . . . . .	13
<b>3 Methodology</b>	<b>15</b>

3.1	Datasets . . . . .	15
3.1.1	HDA Dataset . . . . .	15
3.1.2	DukeMTMC Dataset . . . . .	16
3.1.3	WILDTRACK Dataset . . . . .	16
3.2	Testbed . . . . .	16
3.2.1	Design . . . . .	16
3.2.2	Implementation . . . . .	17
3.2.2.1	Software . . . . .	17
3.2.2.2	Object Detector . . . . .	19
3.2.2.3	Processing Hardware . . . . .	21
3.2.2.4	Prioritization Schemes . . . . .	21
3.2.3	Parameter Tuning . . . . .	21
3.3	Performance Metrics . . . . .	26
3.3.1	Detection Count . . . . .	26
3.3.2	Processing Time . . . . .	27
<b>4</b>	<b>Individual Prioritization Schemes for Frame Selection</b>	<b>29</b>
4.1	Methods . . . . .	29
4.1.1	Spatial and Temporal Frame Information . . . . .	29
4.1.1.1	Frame Differencing . . . . .	29
4.1.1.2	Blob Detection . . . . .	31
4.1.1.3	Entropy . . . . .	32
4.1.1.4	Edge Detection . . . . .	33
4.1.1.5	Saliency . . . . .	35
4.1.2	Detection Confidence . . . . .	35
4.2	Implementation . . . . .	36
4.2.1	Spatiotemporal Prioritization . . . . .	36
4.2.2	Confidence-based Prioritization . . . . .	39

4.3	Results . . . . .	40
4.3.1	Detection Count . . . . .	41
4.3.2	Processing Time . . . . .	41
4.4	Discussion . . . . .	41
<b>5</b>	<b>Combining Prioritization Schemes for Frame Selection</b>	<b>47</b>
5.1	Methodology . . . . .	47
5.1.1	Prioritization Scheme Selection . . . . .	47
5.1.2	Methods of Information Fusion . . . . .	49
5.1.2.1	Weighted Average . . . . .	49
5.1.2.2	KNN Regressor . . . . .	50
5.2	Ensemble Implementation . . . . .	51
5.2.1	Weighted Average . . . . .	52
5.2.2	KNN Regressor . . . . .	54
5.3	Results . . . . .	56
5.3.1	Detection Count . . . . .	56
5.3.2	Processing Time . . . . .	56
5.4	Discussion . . . . .	59
<b>6</b>	<b>Final System Validation</b>	<b>60</b>
6.1	Processing Speed Optimization . . . . .	60
6.2	Results of Varying the Processing Batch Size . . . . .	61
<b>7</b>	<b>Conclusion</b>	<b>64</b>
7.1	Overview . . . . .	64
7.2	Contribution . . . . .	66
7.3	Limitations . . . . .	68
7.4	Recommendations for Future Work . . . . .	69

<b>Bibliography</b>	<b>71</b>
<b>A Definitions</b>	<b>80</b>
<b>B Dataset Information</b>	<b>82</b>
B.1 HDA Dataset . . . . .	82
B.1.1 Camera Topology . . . . .	82
B.1.2 Camera Information . . . . .	82
B.1.3 Reference Frames . . . . .	86
B.2 DukeMTMC Dataset . . . . .	86
B.2.1 Camera Topology . . . . .	86
B.2.2 Camera Information . . . . .	86
B.2.3 Reference Frames . . . . .	86
B.3 WILDTRACK Dataset . . . . .	88
B.3.1 Camera Topology . . . . .	88
B.3.2 Camera Information . . . . .	88
B.3.3 Reference Frames . . . . .	90
<b>C Additional Tables and Plots of Complete Results</b>	<b>92</b>
C.1 Dataset Detection Infomation . . . . .	92
C.1.1 Class Detection Count Line Plots . . . . .	92
C.1.2 Total Detection Count Per Camera . . . . .	92
<b>Curriculum Vitae</b>	

# List of Tables

3.1	Class Distribution of Detected Object for each Dataset . . . . .	21
3.2	Testbed algorithm and hardware configurations for each dataset . . .	26
3.3	Optimal detection counts for each dataset given the configurations presented in Table 3.2 . . . . .	27
5.1	Weights obtained from the Fitting of the Weighted Average Ensembles	50
5.2	$k$ values and their impact on detection performance, using the HDA Dataset . . . . .	54
B.1	HDA Dataset Camera Information . . . . .	82
B.2	DukeMTMC Camera Information . . . . .	86
B.3	WILDTRACK Camera Information . . . . .	88
C.1	Detection Count Per Camera for the HDA Dataset . . . . .	96
C.2	Detection Count Per Camera for the DukeMTMC Dataset . . . . .	97
C.3	Detection Count Per Camera for the WILDTRACK Dataset . . . . .	98

# List of Figures

2.1	A block diagram of a typical automated object detection system used for real-time surveillance . . . . .	5
2.2	An example image annotated by an object detector . . . . .	6
2.3	A block diagram of an automated object detection system with a Frame Selector that uses a prioritization scheme in order to dynamically sub-sample visual surveillance data . . . . .	10
2.4	A diagram showing a backpack detected in the same video sequence (WILDTRACK Camera 3) sampled at 10, 5, 2, and 1 FPS. The object detectors confidence is printed on the bottom right hand corner of each image. The thin green box outlining the backpack shows the bounding box created by the object detector. The color of the number represents if the confidence is greater than the minimum confidence threshold of 70%. . . . .	12
3.1	Two diagrams showing the two steps of the testbed: selecting frames and processing object detection count . . . . .	17
3.2	A flowchart of the how the Frame Selection Data is created as outlined in Figure 3.1(a) . . . . .	18
3.3	A Software Architecture Diagram showing the dependencies of each functional block on hardware and software components . . . . .	20
3.4	Processing Times vs. Batch Size . . . . .	23
3.5	Frames Processed vs. Batch Size for each dataset . . . . .	24

3.6	The relationship between the number of frames processed and the maximum number of objects detected. . . . .	25
3.7	An example image annotated by an object detector to demonstrate the challenges with detection confidence . . . . .	28
4.1	An example showing a delta frame demonstrating the difference between two frames. . . . .	30
4.2	The ‘blobs’ (represented by the red circles) detected in the binarized delta frame . . . . .	32
4.3	An example demonstrating the change in entropy between two frames from Camera 17 of the HDA Dataset ( $\Delta Entropy = 7.00 - 6.46 = 0.54$ ). . . . .	33
4.4	An example showing how edges are detected in images . . . . .	34
4.5	A demonstration of saliency . . . . .	35
4.6	A block diagram of an automated surveillance system with priority-aware computation that uses values provided from the object detector. . . . .	36
4.7	An example demonstrating the difference between a linear and a non-linear roulette . . . . .	40
4.8	The detection count achieved for each individual prioritization scheme expressed as percentage of achievable detections . . . . .	42
4.9	The processing time per image batch for each prioritization scheme . . . . .	43
5.1	The total number of object detections achieved per camera for each dataset using the best performing prioritization schemes of Chapter 4. The cells colored red represent <i>lower</i> than the optimal detection count; green cell represent counts <i>above</i> above the optimal detection count. The closer to white the cell color is, the closer the value is to the optimal detection count, shown in the top row. . . . .	48

5.2	An example of how a KNN Regressor predicts a label value based on previous feature data . . . . .	51
5.3	A block diagram outlining frame selection for a two camera automated surveillance system that uses three weighted prioritization schemes to prioritize frames. . . . .	52
5.4	A block diagram outlining a two camera automated surveillance system's frame selector that uses three weighted prioritization schemes and a roulette selection method to prioritize frames. . . . .	53
5.5	A block diagram outlining a two camera automated surveillance system's frame selector that uses four metrics to inform an estimator to prioritize frames. . . . .	55
5.6	The detection count achieved for each ensemble implementation expressed as percentage of achievable detections . . . . .	57
5.7	The processing times for each ensemble implementation . . . . .	58
6.1	A demonstration of the reduction in processing time of the $K_{C,P}$ ensemble implementation before and after optimization using the Cython compiler and multi-threading . . . . .	61
6.2	The HDA dataset's normalized detection results with the best performing Frame Selector implementations for an increasing number of frames processed per batch . . . . .	62
6.3	The DukeMTMC dataset's normalized detection results with the best performing Frame Selector implementations for an increasing number of frames processed per batch . . . . .	63
6.4	The WILDTRACK dataset's normalized detection results with the best performing Frame Selector implementations for an increasing number of frames processed per batch . . . . .	63

B.1	Floor plan of Floor 6 at ISR-Lisbon [39]	83
B.2	Floor plan of Floor 7 at ISR-Lisbon [39]	84
B.3	Floor plan of Floor 8 at ISR-Lisbon [39]	85
B.4	Reference Frames used for the HDA Dataset	87
B.5	Camera Topology of the Duke MTMC dataset [40]	88
B.6	Reference Frames used for the DukeMTMC Dataset	89
B.7	Camera Topology of the WILDTRACK dataset [41]	90
B.8	Reference Frames used for the WILDTRACK Dataset	91
C.1	Line plot of the number of class detections of the HDA dataset	93
C.2	Line plot of the number of class detections of the DukeMTMC dataset	94
C.3	Line plot of the number of class detections of the WILDTRACK dataset	95

# Chapter 1

## Introduction

### 1.1 Background

Modern surveillance networks have many industrial applications including monitoring patients in hospitals, detecting violence in stadiums, and identifying lost baggage in airports [1, 2, 3]. Regardless of application, these systems have the same underlying goals: to detect and report objects or people of interest, otherwise known as object detection. Traditional surveillance systems require a human to monitor several camera feeds. If and when an object of interest appears on the monitor, the human is expected to react quickly and accurately in response to the threat. With 24/7 surveillance and the growing numbers of cameras in these systems, it has become infeasible for a single human to monitor all of the camera streams simultaneously.

An example of this is in the borough of Islington in London England: two employees monitor 180 camera streams for 10 hour periods in order to report crimes [4]. Human operators often suffer from a condition called “operator blindness” which profoundly limits their ability to detect security threats when presented with large amounts of data for extended periods [1]. Automating object detection in surveillance systems alleviates the burden on operators analyzing large amounts of video data streams simultaneously.

With the proliferation of security cameras throughout the world, many methods have been used to create automated object detection systems. One such system with a total of 25 000 surveillance cameras was deployed in New York and Chicago in an effort to fight street crime [5]. More recently, Singapore has announced that

it intends to bring its 110 000 lamp post cameras together onto a single network to use them all for real time facial recognition to help combat terrorism [6].

## 1.2 Problem Statement

The two main metrics that are used to assess an object detector's performance are its speed and accuracy, both of which are essential in surveillance [2]. The recent emergence of Convolutional Neural Network (CNN) based object detectors has yielded a 30% improvement in accuracy over traditional object detectors [7]. The speed of the detector varies based on the required accuracy of the algorithm needed for the surveillance application and the size of images created by the surveillance cameras. The hardware used to process these algorithms can also greatly impact their processing times. Central processing units (CPUs) that are typically found at the center of modern computer systems are not well suited to efficiently process CNN-based algorithms. The co-processing hardware typically used for processing are general purpose graphics processing units (GPGPUs or GPUs) [8], which can range in cost from \$1000 USD to \$10000 USD, often making them the largest costs associated with these systems. With a typical commercial grade GPU, the processing speed of CNN-based object detectors is typically cited as being between 5 and 60 frames per second (FPS) for a single image or video source [9, 10, 11, 12, 13].

The average security camera used in modern surveillance systems is capable of capturing upwards of 30 FPS [2]. In order to process all of this data in real-time with a CNN-based object detector, multiple GPUs must be used. However, many of the frames that are created by a surveillance camera may be of low importance to the object detection system (ie. an empty hallway). Pre-processing the camera frames by prioritizing and selecting the most important images has been found to greatly reduce the unnecessary overuse of the object detection system [2, 14, 15].

Previous works, written before the popularity of CNNs, have proposed image pri-

oritization techniques to reduce the amount of visual data created by a camera surveillance network [2, 16, 17, 18, 19, 15]. The majority of these works, however, have done so with a focus on reducing wireless network traffic and have not readily considered the load on the object detection system. Recent papers on visual surveillance data prioritization have prioritized frames by training a CNN to pre-process video and only return frames with a high likelihood of containing a detectable object. Two works, one by Kang et al. [14] and another by Yu et al. [20], represent the state of the art. However, these works apply a single prioritization technique in retrospect (after the fact), and not in the context of real-time surveillance.

### 1.3 Contribution

Many works have been conducted on visual data prioritization using different modalities of information (ie. spatial or temporal information). However, given the nascence of CNN's in computer vision, and despite their overwhelming popularity, only a small subset of works [14, 20] have implemented visual data prioritization in the context of a CNN-based object detection algorithm. Of the works so far, only spatiotemporal information (created by looking at the pixel value difference between two frames) has been used. Furthermore, these systems are designed for analyzing visual data in retrospect (ie. forensics) and not in real-time. This work evaluates several individual prioritization schemes that use different modalities of information to prioritize visual data, for use with CNN-based object detection algorithms. Furthermore, the use of a CNN-based algorithm for prioritization increases the use of the GPU, which worsens the performance of the object detector. This dissertation presents fusion techniques designed to run on a CPU that combine the best of the evaluated priority metrics in order to increase the number of objects detected when sub-sampling visual data in a real-time automated object detection system. This constitutes a novel and meaningful contribution to a rapidly growing and widely

researched field.

## **1.4 Thesis Structure**

Chapter 2 presents a review of automated object detection systems for use in surveillance, followed by a review of visual data prioritization methods. Chapter 3 reviews the methodology used in this work by presenting the datasets, testbed, and performance metrics. Chapter 4 presents the methods and results of several individual priority metrics applied to the selected datasets. Chapter 5 presents results of a fusion of individual visual data priority metrics explored in Chapter 4. Chapter 6 provides the final results of the thesis work. Chapter 7 concludes this work by discussing the contributions and limitations of this work and offering ideas for future work.

# Chapter 2

## Literature Review

### 2.1 Automated Object Detection System Design

The design of an automated object detection system can be described as a producer/consumer relationship. The surveillance cameras produce image frames and the automated surveillance system processes them in order to identify objects of interest. In this work, automated surveillance systems are typically connected to a surveillance camera network via an internet protocol (IP) based network. Each camera transmits the encoded video (ie. H.264 [21]) over the network to the system. After the video packets are received from the network, the video decoders take the encoded camera streams and decode them to produce a series of frames to be stored in an image pool. The processing units are then used to perform object detection on this pool of image frames received from the network of surveillance cameras. This architecture is briefly summarized in Fig. 2.1.

With the growing number of cameras in security surveillance systems, the total number of image frames produced per second is rapidly increasing, giving rise to a big data processing challenge. Automated object detection systems are required to

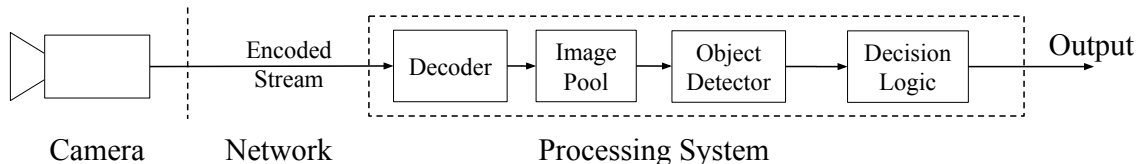


Figure 2.1: A block diagram of a typical automated object detection system used for real-time surveillance

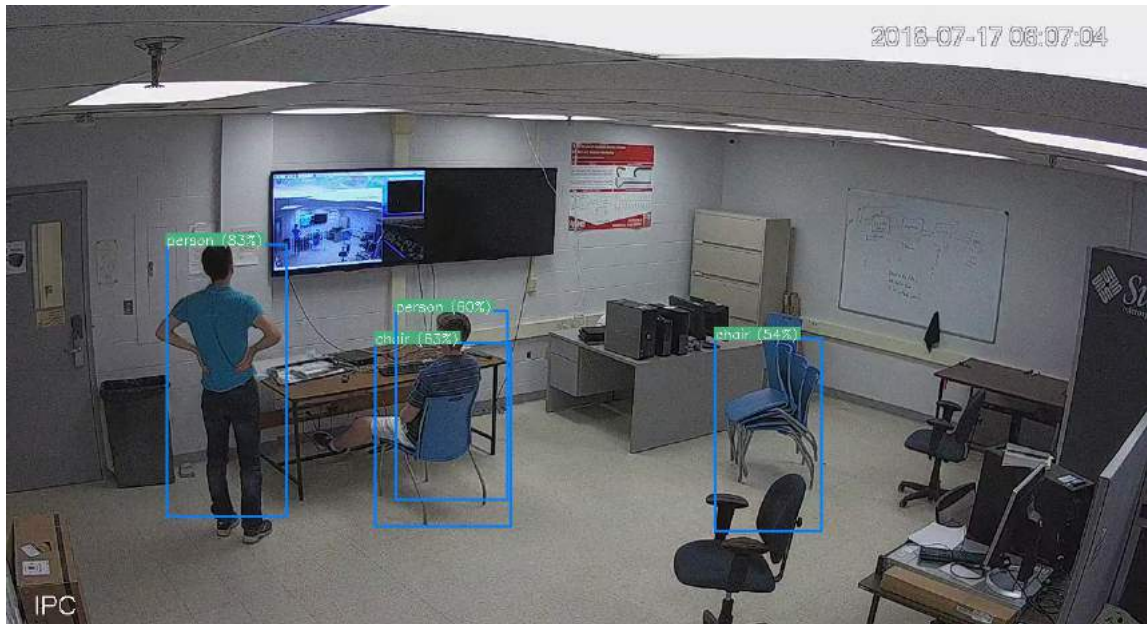


Figure 2.2: An example image annotated by an object detector

process large amounts of data in real-time so that any objects of interest will be properly and promptly identified. At their core, it is the object detection algorithms and their associated processing hardware that are predominately responsible for an automated object detection system’s detection accuracy and speed.

### 2.1.1 Object Detection

Object detectors have existed for many years in the field of computer vision [22] and have always been popular in automating surveillance systems [1, 2, 23]. Object detection algorithms create bounding boxes, simple rectangular shapes, around recognized objects within an image. Every box drawn by the detector is given an associated confidence based on how strongly the algorithm believes that the object within the box belongs to the identified class (ie. person, chair). Figure 2.2 shows an example of an image taken by the surveillance system in the Health Technologies lab at the University of New Brunswick, annotated by an object detection algorithm.

Until recently, object detectors relied on more traditional image processing methods such as edge detection or entropy [22]. However, with the emergence and subse-

quent widespread adoption of convolutional neural networks in the computer vision community, the best performing methods now use these deep learning approaches for object detection [8, 24].

Convolutional neural networks (CNN or ConvNet) are a new field of research which is rapidly being adopted with many video based technologies, including automated object detection systems. CNNs are a special subset of neural networks that have proven very effective in image processing [24], particularly for tasks related to object detection. CNNs have large network architectures, which account for their higher accuracy compared to traditional methods [25]. CNNs were largely ignored by the image processing community until the ImageNet competition in 2012, where CNNs nearly halved the error rates of other competing methods [8].

Regional CNN (R-CNN), the first notable CNN-based object detector, was found to outperform traditional algorithms used in detection with a 30% relative improvement over the best previous results demonstrated on the PASCAL VOC 2012 image dataset [7]. Within two years of R-CNN, many related algorithms had further improved object detection. One such work was Faster R-CNN, which demonstrated a speedup of 250 times over that of the original R-CNN [9]. Faster R-CNN is a two stage algorithm, relying on two separate CNNs to perform object detection. The first stage is dedicated to finding region proposals (bounding boxes) to identify potential regions containing an object. The second stage classifies the object, assigns an associated confidence, and refines the bounding box proposal to better fit around the object.

As opposed to the two stage approach of the R-CNN series, many other algorithms use only a single stage to perform objection detection. The single CNN provides a noticeably faster detection rate, but at the cost of accuracy [26]. The most notable single stage CNN-based object detection algorithms are the Single Shot Multi-box Detector (SSD) [10], RetinaNet [11], and YOLO [12, 13] algorithms. Of these object

detection algorithms, YOLO is noteworthy due to its high speed relative to other methods [12, 13]. This makes YOLO a desirable candidate for applications requiring real-time object detection. However, YOLO’s detection accuracy is generally modestly lower when compared to the other state-of-the-art algorithms [10].

For object detectors, mean accuracy precision (mAP) is the measure by which an object detector’s accuracy is compared for a given dataset. mAP is a measure of the overall accuracy for all objects present within the dataset. Smaller, harder to detect, objects often lead to lower mAP scores. The objects of interest for an automated surveillance system, however, differ with each application. People and cars are commonly detected objects, but smaller objects like guns and other weapons also present important security implications that may be relevant. In this context, accuracy is imperative given the significant ramifications of false positives or negatives.

The object detection algorithm chosen for an automated surveillance system determines its maximum achievable accuracy. The selection of algorithm also impacts the system design, as more computationally demanding algorithms require additional hardware resources to meet real-time system requirements.

### **2.1.2 Processing Hardware**

The most prominent hardware type used for CNN-based computer vision processing is the general purpose graphics processing unit (GPGPU or GPU) [8, 24]. A GPU’s highly parallelizable architecture with multiple identical processing cores, makes it a great fit for processing CNNs given the multiple neural network layers made up of multiple neurons [8]. When processing a single image, the GPU can process each neuron contained within a single neural network layer simultaneously. When processing a batch of images, several images can be computed simultaneously in order to decrease processing time. This is known as thread-level parallelism [27].

Field programmable gate arrays (FPGAs) also support parallelism, but implement it with hardware pipelines rather than with threads. FPGAs have shown great

promise for low latency processing with neural networks [27, 8]. The great advantage of FPGAs is the ability to reconfigure the design of their hardware pipelines. This can be used to control the number of hardware pipelines available, or to enable quantization [28] where smaller bit sizes are used. This is important for processing speed and for minimizing the usage of the FPGA’s fabric.

## 2.2 Visual Data Prioritization

### 2.2.1 Motivation

Automated surveillance systems necessarily have high accuracy constraints, which makes meeting real-time processing requirements a challenge for system designers. One way of meeting real-time requirements given these constraints is to use additional processing hardware units for the object detection system. The processing hardware units required, however, are very expensive, ranging from \$1000 USD to \$10000 USD per unit. This is an unattractive, and sometimes impractical, option that inflates the overall cost of automated surveillance systems.

A simple and more cost effective option would be to simply sub-sample the incoming video data, only processing a pre-determined portion of the camera frames. This would limit the need for more processing units and allow the system to meet the real-time requirement of an automated surveillance system. Using a static sub-sampling frequency for each camera stream, however, would be sub-optimal for maximizing object detection since it would consider all visual data to be the same (ie. an empty hallway and a busy lunch room would be considered the same priority). By pre-processing the frames of each surveillance camera stream, the automated surveillance system could intelligently and dynamically sub-sample the incoming data and prioritize it in order to maximize the number of objects detected. A block diagram of an automated surveillance system where intelligent sub-sampling is applied is shown in Figure 2.3. The Frame Selector intelligently sub-samples the incoming data and

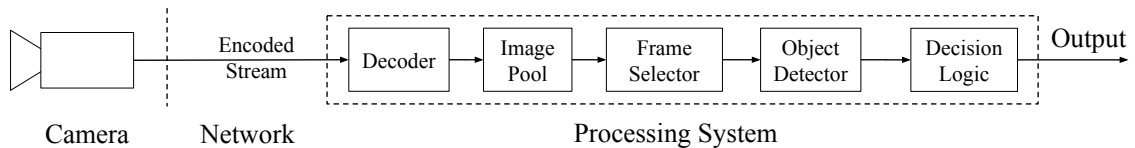


Figure 2.3: A block diagram of an automated object detection system with a Frame Selector that uses a prioritization scheme in order to dynamically sub-sample visual surveillance data

prioritizes it with a prioritization scheme.

Prioritization strategies for intelligent sub-sampling are not new in automated surveillance system research. They have been used for both priority-aware computation in automated surveillance processing systems and priority-aware communication for camera surveillance networks [2]. Priority-aware communication and computation apply to two bottlenecks in the complete automated surveillance system. In the case of priority-aware communication, the bandwidth between the cameras and the system may be limited, thus only allowing a fraction of possible content to be transmitted to the processing system. Priority-aware computation applies to the system described at the beginning of this chapter, where bandwidth of the processing system is the bottleneck. Despite these being two separate areas of research, many of the prioritization methods employed by either can be used interchangeably.

Static sub-sampling, such as reducing a camera’s frame rate from 60 down to 10 FPS, could greatly alleviate some of the computational requirement. However, making generalized assumptions could lead to missed detections of great importance. As mentioned previously, many of the frames created by a security camera may be of low priority and provide no useful information to the classification system. Pre-processing the camera frames by prioritizing them and only selecting important images can greatly reduce the unnecessary overuse of the object detection system [2, 14, 15]. The size of a camera’s field of view and area of observation greatly impact the minimum frame rate needed to ensure that no detections are missed. Furthermore, the movement speed of the objects of interest in the video are also

of importance when arbitrarily sub-sampling, as they may travel through the frame quickly and be missed altogether.

The work by Ramachandran et al. [2] discusses the challenges of object detection, as the appearance of objects varies with a plethora of factors. Among these factors are lighting, motion blur, orientation, object size, and occlusion. Despite the high accuracy of modern CNN-based object detection systems, these algorithms still struggle in these difficult conditions [24, 29]. The work by Sobti et al. [30] investigated this premise for pedestrian detection and demonstrated that object detection is dependent on the frame processing rate. This makes sense as higher frame rates help by providing more information to process; if an object is blurry or obscured in one frame, it may not be in the next frame.

In order to better understand the impact of frame rate on object detection, consider an automated surveillance system that only reacts when a backpack is detected with at least a confidence of 70% (a minimum confidence threshold of 70%). Figure 2.4 shows one second of annotated camera frames with the object detector. With a frame rate of 10 Hz (or FPS), there are a total of 10 images that could be processed by the object detector. The backpack appears in only 9 of the 10 images. When processed by the object detector, a backpack is detected with a confidence that is greater than 70% in only 3 of these frames. As the data is sub-sampled below 10 Hz, the backpack is eventually never detected with a confidence greater than the 70% threshold.

While processing every frame of a high frame rate video will maximize the possibility of detection, this is not always a practical solution. Assigning sub-sampling at a constant rate on a per camera basis, would be a waste of resources: crowded lunch rooms are not always full, and narrow hallways are not always empty. Since the goal of prioritization is to alleviate the bottleneck in a producer/consumer relationship, static methods may waste processing power on these unimportant frames. As shown



Figure 2.4: A diagram showing a backpack detected in the same video sequence (WILDTRACK Camera 3) sampled at 10, 5, 2, and 1 FPS. The object detectors confidence is printed on the bottom right hand corner of each image. The thin green box outlining the backpack shows the bounding box created by the object detector. The color of the number represents if the confidence is greater than the minimum confidence threshold of 70%.

in the next section, many dynamic methods have been created in order to maximize the effectiveness of automated surveillance systems.

### 2.2.2 Methods

The goal of prioritization is to dynamically sub-sample visual data, reducing the demand on the object detection system, ultimately reducing the cost of the automated surveillance system. Thus, methods of prioritization must not require expensive processing hardware, otherwise the benefit of sub-sampling is lost. Spatiotemporal data is one simple and popular modality often used in dynamic visual data prioritization. While spatial information is based on the pixel locations and values (typically grayscale) in a single image, spatiotemporal approaches consider the changes in these values over time (ie. from frame to frame). Many works use spatiotemporal data for wireless data priority-aware communication [16, 17, 18, 31, 32, 33, 34] and for priority-aware computation [2, 14, 15, 20, 35, 36, 37, 38].

Other recent works on visual data prioritization have prioritized frames by training a CNN to pre-process video and only return frames of importance. Two works, one by Kang et al. [14] and another by Yu et al. [20], represent the state of the art for this area. The work by Kang et al. is a standalone pre-processing method whose output is fed to an object detector such as YOLOv2. Yu et al. differentiates their work from Kang et al. by creating an end to end solution: taking an input camera frame and producing a detection output. Both Kang et al. and Yu et al. use spatiotemporal methods, specifically frame differencing (subtraction), as the basis for prioritization.

In both Kang et al.'s and Yu et al.'s work, their systems are trained on a small portion of a specified video and then applied on the rest of the video clip (ie. a model is trained with five minutes of surveillance video and then applied on an hour long video from the same camera feed). As a result, these systems are designed for analyzing visual data in retrospect (forensics) and not in real-time.

An alternative source of information that could be used to prioritize visual data is

the previous detection results of a camera. As demonstrated in Figure 2.4, the detection confidences are highly correlated over time (ie. previous detection confidences predict whether an object is likely to be detected in sequential frames). Thus, by using historical detection confidences, visual data may be prioritized. To the best of the author’s knowledge, the use of detection confidence is a novel approach to visual data prioritization in surveillance systems. Both the spatial-temporal methods popular in literature and the novel approach of using detection confidences are therefore reviewed further in Chapter 4.

This thesis specifically considers the application of visual data prioritization to real-time multi-camera surveillance. Despite the rapidly evolving literature in this area, no one has yet explored prioritization in modern surveillance systems that employ CNN-based detection algorithms for real-time applications. Furthermore, few have considered the fusion of information modalities for such systems. Consequently, the remainder of this work investigates several different individual and ensemble methods of prioritization for real-time visual data prioritization in modern multi-camera automated object detection systems.

# Chapter 3

## Methodology

### 3.1 Datasets

Three publicly available datasets were obtained in order to assess the performance and robustness of the different prioritization methods: the HDA Dataset [39], the DukeMTMC dataset [40], and the WILDTRACK dataset [41]. These datasets were chosen because they contain real world surveillance data of non-scripted scenes for extended periods of time. Each dataset is unique, improving their ability, as a set, to test the multiple factors impacting object detection performance and the use of frame prioritization.

#### 3.1.1 HDA Dataset

The HDA Dataset [39] was created by several indoor cameras placed over three floors of the Institute for Systems and Robotics (ISR-Lisbon) during lunch hour. A floor plan of the dataset can be seen in Figures B.1, B.2, and B.3. A total of 25 minutes of synced video data was used from 12 cameras spread over these floors. As shown in Table B.1, the cameras had significantly lower frame rates (as low as 1 FPS) than the other two datasets, which have frame rates of 60 FPS. The recorded video frame rates, however, varied depending on the resolution of the specific camera in the network.

### 3.1.2 DukeMTMC Dataset

The DukeMTMC dataset [40] had a total of 8 cameras, all recording 1080p frames at 59.94 FPS. The cameras were setup in the surrounding vicinity to Duke University’s chapel on a rainy afternoon. As seen in Figure B.5, the cameras were spread out over the area, with few overlapping zones. While this dataset provided a total of 85 minutes of synced camera footage to use, only the first 25 minutes was used in order to maintain consistency across the datasets.

### 3.1.3 WILDTRACK Dataset

The WILDTRACK dataset [41] had a total of 7 cameras recording 1080p frames at 60 FPS. The cameras were all mounted facing a busy courtyard located in Zürich, Switzerland on a busy, sunny afternoon. As seen in Figure B.7, the cameras are all pointed towards the center of the square, with many overlapping views. While this dataset provided a total of 35 minutes of synced camera footage to use, again, only the first 25 minutes was used in order to be consistent with the length of the HDA dataset.

## 3.2 Testbed

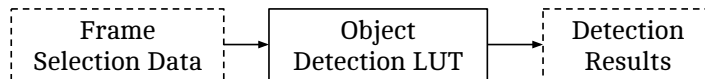
### 3.2.1 Design

A testing framework was created in order to simulate streaming any number of camera feeds at once from a dataset of prerecorded video. This allowed for the testing of different prioritization schemes with various datasets. Figure 3.1 shows a block diagram of the testbed functions.

The testbed processed each video of the chosen dataset separately with the Decoder and placed them into an Image Pool. After there was a predetermined number of images in the Image Pool ( $N_i$ ), the batch of images was passed to the Frame Selector.



(a) Diagram of creating Frame Selection Data



(b) Diagram of creating Object Detection Results given the Frame Selection Data

Figure 3.1: Two diagrams showing the two steps of the testbed: selecting frames and processing object detection count

The Frame Selector then processed each image based on a prioritization scheme and created a smaller image batch (the Frame Selection data) of the size  $N_p$  (where  $N_p \leq N_i$ ). An example of this process is shown in Figure 3.2. The detailed steps used to determine  $N_i$  and  $N_p$  for each dataset are outlined in Section 3.2.3.

The Frame Selection Data was then used with a Object Detection look up table (LUT) to determine the final object detection count. To avoid recomputing the object detection algorithm every time, the complete object detection results were first computed for every frame of every video from each dataset and stored in this LUT. The Frame Selection processes could then simply draw the result from the LUT. This process is represented in the block diagram in Figure 3.1(b).

## 3.2.2 Implementation

### 3.2.2.1 Software

The testbed software was developed in Python 2.7, using OpenCV [42], Caffe2 [43] and Detectron [44]. OpenCV is a computer vision library that provides primitive functions that can be used to prepare and process images in the testbed. Caffe2 is a deep learning framework created by Facebook Research that facilitates the fast development of deep neural networks. Detectron is a CNN-based object detection

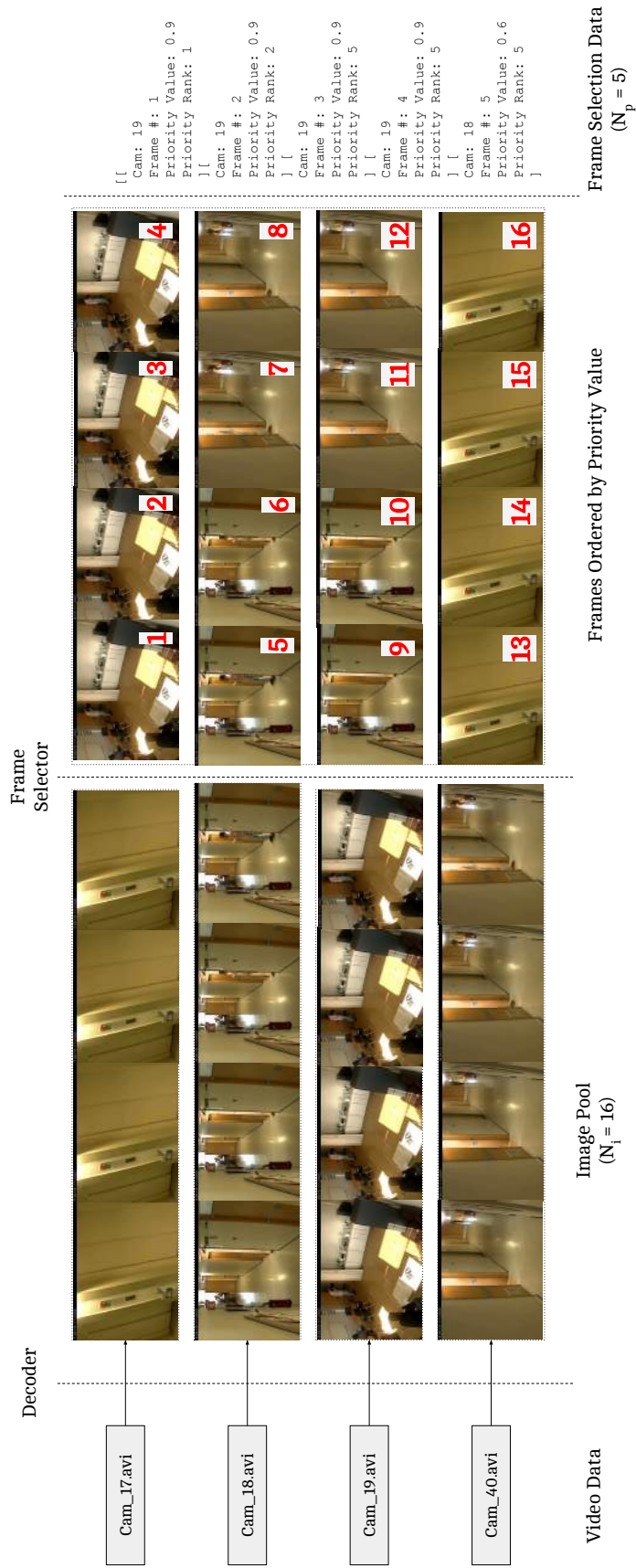


Figure 3.2: A flowchart of the how the Frame Selection Data is created as outlined in Figure 3.1(a)

framework that is implemented on top of Caffe2. A complete diagram showing the dependencies of each functional block on hardware and software components can be seen in Figure 3.3.

As a note, Caffe2 was recently merged with Pytorch [45] and consolidated into a single machine learning software package. This work, however, uses the term Caffe2 to refer to the original Caffe2 components used for deep neural networks processing.

It should also be noted that the rapid pace of development in the field of computer vision and object detection is in large part facilitated by the openness of the community to share their software, models and data. The availability of these resources was leveraged within this dissertation to enable the evaluations conducted herein.

### **3.2.2.2 Object Detector**

For this work, RetinaNet [11], implemented within Detectron, was chosen as the default object detector algorithm used in the testbed. RetinaNet is a single stage object detector, offering a balance between low processing time and high accuracy. Any CNN-based object detector, however, could be used instead of RetinaNet as the design of the automated object detection system presented in this work is object detector agnostic.

The Objection Detection LUT mentioned previously in Section 3.2.1 was created with the RetinaNet object detector. The RetinaNet implementation used a Resnet101 backbone for image classification and was trained using the open source Common Objects in COntext (COCO) dataset [46] created by Microsoft. Of the 81 classes used to train RetinaNet [11], only six were selected: ‘person’, ‘backpack’, ‘umbrella’, ‘handbag’, ‘suitcase’, and ‘cell phone’. These objects were chosen as they appear most commonly in the three datasets described in Section 3.1. The rate of occurrence of these objects in each of the three datasets can be seen in Table 3.1. These objects are defined as ‘objects of interest’ for the surveillance system and will be referred to as such throughout the remainder of this work.

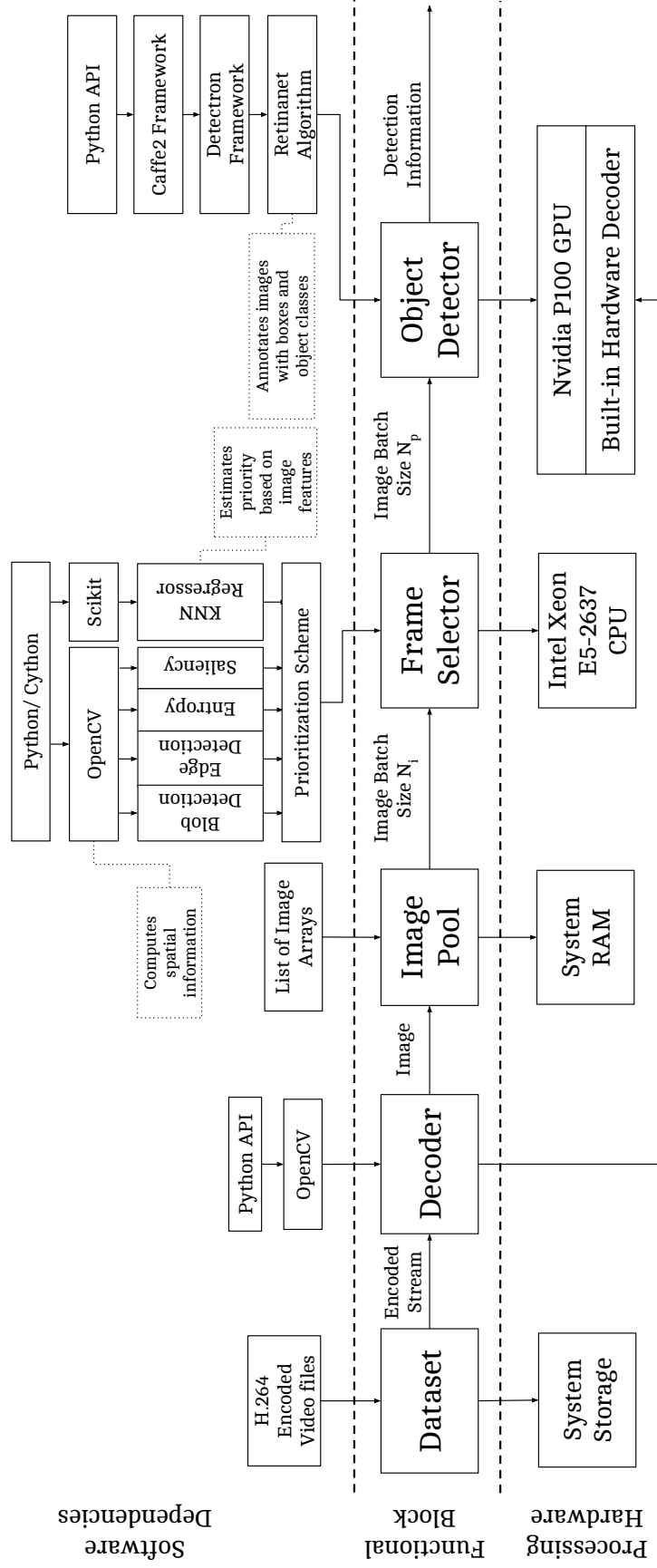


Figure 3.3: A Software Architecture Diagram showing the dependencies of each functional block on hardware and software components

Table 3.1: Class Distribution of Detected Object for each Dataset

Dataset	Object					
	Person	Backpack	Umbrella	Handbag	Suitcase	Cell
HDA	99.873%	0.095%	0.000%	0.007%	0.000%	0.025%
Duke	91.979%	6.128%	1.684%	0.208%	0.000%	0.000%
WILD	95.932%	3.515%	0.000%	0.444%	0.106%	0.002%

### 3.2.2.3 Processing Hardware

The processing hardware chosen for the testbed was as follows:

- Intel<sup>®</sup> Xeon<sup>®</sup> CPU E5-2637 v4 running at 3.50GHz (1 unit)
- Nvidia P100 GPU with 12 GB of HBM2 memory [47] (4 units)

### 3.2.2.4 Prioritization Schemes

Each prioritization scheme was implemented individually as functions in Python, but used a common interface to ensure uniform implementation. In all cases, a single prioritization scheme or an ensemble thereof was used within the Frame Selector to determine which frames had the highest priority (ie. the most objects).

### 3.2.3 Parameter Tuning

The batch size parameters  $N_i$  and  $N_p$  mentioned in Section 3.2.1 are dependent on the processing frequency ( $F_p$ ) of the system.  $F_p$  was defined as the rate at which data passed from the image pool through to the output of the automated object detection system. Each video (or camera stream) had an associated frame rate which determined how fast images were sent through the Decoder into the Image Pool. Thus  $F_p$  defined how many images ( $N_i$ ) could be placed in the Image Pool before having to be sub-sampled by the Frame Selector.

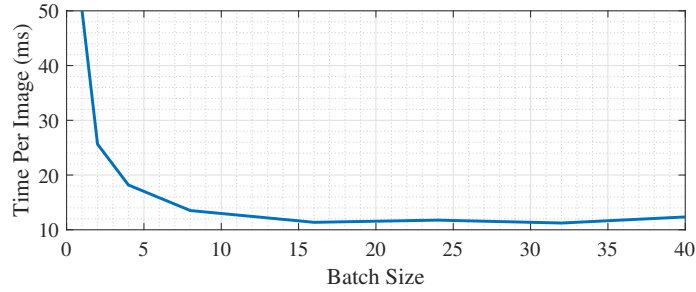
In order to determine the processing frequency ( $F_p$ ) of the system, a set of assumptions were made. Since the computation bottleneck of the system is the object detector, the object detector algorithm processing frequency was considered to be the overall processing frequency ( $F_p$ ). The speed in  $F_p$  is in part determined by the

object detection processing hardware. In this work a Nvidia P100 GPU with 12 GB of HBM2 memory [47] was used to process the CNN-based algorithm. This hardware was accessed via Compute Canada Cloud and was chosen due to it being the most advanced GPU accelerator card available to the author.

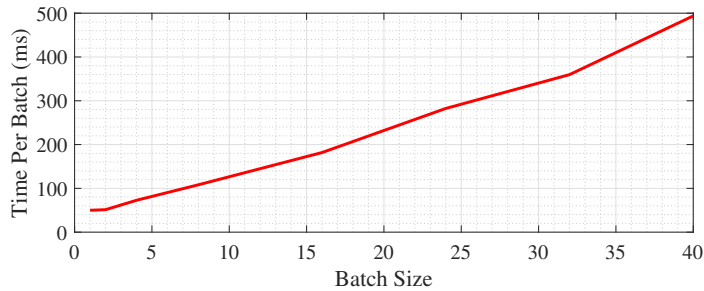
While object detection could be performed on one image at a time (single inference), several works have shown the benefit of batching multiple images and processing them all at once (batch inference) [8, 10]. To evaluate this notion, Figure 3.4 was created by varying the batch size of Resnet101 (the backbone of Retinanet [11]) and running the algorithm on the Nvidia P100. The Resnet101 backbone represents a fraction of the total processing time that the Retinanet algorithm will take to process an image. Setting this smaller amount of time as the goal for real-time requirements ensures that they will be met when implementing the full RetinaNet algorithm.

Figure 3.4 demonstrates the direct relationship between the object detector image batch size ( $N_p$ ) and the processing time ( $1/F_p$ ). As the batch size of the object detector is increased, the overall processing time increases. However, as shown in Figure 3.4, the processing time per image decreases as the batch size is increased. Note that, with the 1080p frames used to generate Figure 3.4 and the CNN architecture, the P100 could not process more than 40 images in a batch because it ran out of memory.

Despite the increase in per image processing speed obtained by increasing batch size, a single GPU card was not enough to process all of the surveillance data due to the higher frame rate of some of the datasets. Figure 3.5 shows the relationships between different batch sizes used by the object detector and the corresponding percentage of frames processed. Due to the higher frame rates of the DukeMTMC and WILDTRACK (60 FPS versus  $\leq 5$  FPS), these datasets required much more processing power. In order to achieve enough processing power, up to four GPUs were added in parallel. As shown by Figure 3.5(b) and 3.5(c), even four GPUs were



(a) Processing Time for each image vs. Batch Size



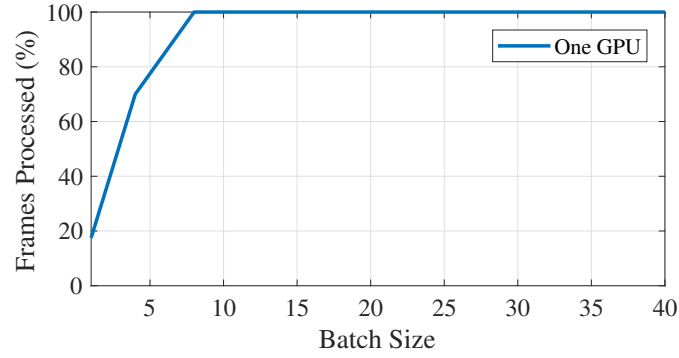
(b) Processing Time for total batch size vs. Batch Size

Figure 3.4: Processing Times vs. Batch Size

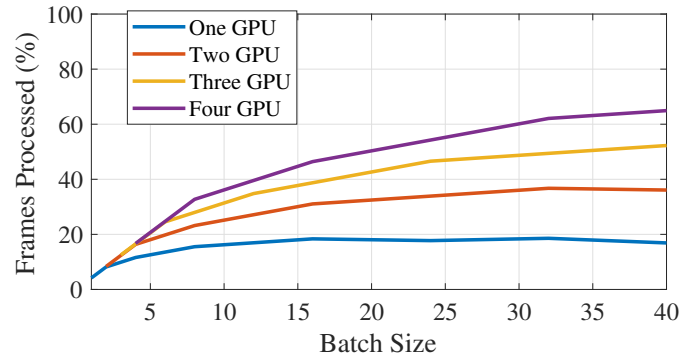
not powerful enough to fully process the WILDTRACK and DukeMTMC datasets in real-time.

After determining how the hardware and algorithm configurations impacted the number of frames processed for each dataset, the relationship between the number of frames processed and the maximum number of objects detected was determined. Since all of the frames for each dataset were processed previously with an object detector, the number of detections for each frame was known. Thus, when calculating the maximum number of object detections for a given number of possible frames, only the images with the highest object detection counts were counted. Figure 3.6 shows this relationship for each dataset. It can be seen that the HDA Dataset yielded 80% of all detections even when less than 30% images were processed. This is in stark contrast to the DukeMTMC and WILDTRACK datasets which required 53% and 65% of frames to be processed to achieve the same 80% of all detections.

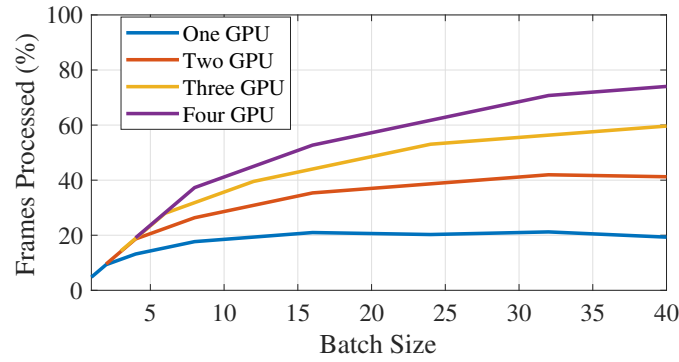
The differences in these rates is directly tied to the nature of the surveillance



(a) Frames Processed vs. Batch Size for the HDA Dataset



(b) Frames Processed vs. Batch Size for the Duke Dataset



(c) Frames Processed vs. Batch Size for the WILDTRACK Dataset

Figure 3.5: Frames Processed vs. Batch Size for each dataset

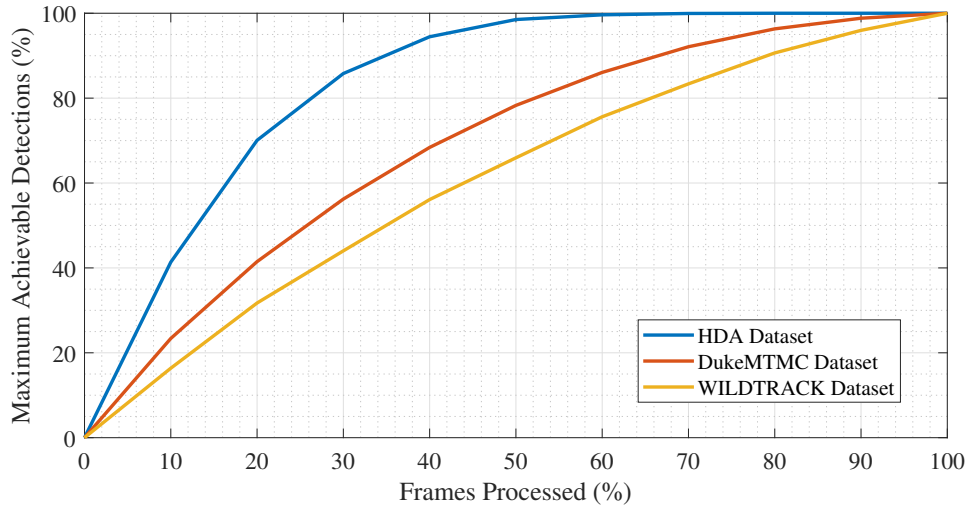


Figure 3.6: The relationship between the number of frames processed and the maximum number of objects detected.

datasets. The WILDTRACK dataset is very ‘busy’, with many humans never leaving the frame of view (FOV) of any camera. The DukeMTMC dataset varies camera by camera in terms of the number of humans, but there are typically always people visible in at least some cameras. Finally the HDA Dataset has many instances footage of hallways and stairwells containing no object of interest in the camera’s FOV.

Due to the difference between the three datasets (number of cameras, framerate, etc), the testbed had to be configured differently for each dataset in order to ensure that the results were comparable. Table 3.2 lists the batch size and GPU configurations for each dataset. The batch sizes and number of GPUs selected determined the percentage of frames processed and thus the maximum number of detections. For the DukeMTMC and WILDTRACK datasets, Figure 3.4(a) was consulted in order to choose the batch sizes that maximized efficiency of object detector: sizes of 32 and 16 respectively. However, as shown by Figure 3.6, more frames needed to be processed in the DukeMTMC and WILDTRACK datasets in order to have a comparable number of maximum number of achievable detections. Thus, as shown

Table 3.2: Testbed algorithm and hardware configurations for each dataset

	Dataset		
	HDA	DukeMTMC	WILDTRACK
<b>Number of GPUs</b>	1	4	4
<b>Processing Batch Size (<math>N_p</math>)</b>	4	128	64
<b>Batch Processing Time [ms]</b>	73	360	182
<b>Frames Processed</b>	70 %	74 %	83 %
<b>Max Number of Detections</b>	97 %	94 %	92 %

in Figure 3.5, four GPUs had to be used for the DukeMTMC and WILDTRACK datasets to increase the number of frames processed (to a batch size of 128 and 64 respectively) in order to ensure that the results would be comparable between the datasets.

### 3.3 Performance Metrics

To achieve the best possible performance tradeoff, the prioritization metrics should minimize the impact on overall system accuracy and not compromise the real-time performance of the system. Thus any prioritization scheme that meets computational time constraints, while maintaining the same number of detections, represents an improvement over the baseline. To measure the performance of the tested system, two performance metrics, *Detection Count* and *Processing Time*, were used to assess each prioritization scheme.

#### 3.3.1 Detection Count

The maximum number of possible detections in each dataset was pre-determined and are presented in Table 3.3. The *Total Detection Count* refers to the total number of objects of interest that could be detected in the datasets if every frame was processed. The *Optimal Detection Count* refers to the maximum possible number of detections achieved given some limitation in processing configuration (as outlined for each dataset in Section 3.2.3). The goal of each prioritization scheme was therefore

Table 3.3: Optimal detection counts for each dataset given the configurations presented in Table 3.2

Dataset	HDA	DukeMTMC	WILDTRACK
<b>Total Detection Count</b>	28,431	1,896,719	2,535,026
<b>Optimal Detection Count</b>	27,613	1,786,396	2,354,839
<b>Optimal Detection %</b>	97 %	94 %	92 %

to correctly choose the images with the highest number of detections, as to maximize the *Optimal Detection Count*.

Two methods were used to serve as a static sub-sampling baseline against which to compare the dynamic sub-sampling methods implemented by the prioritization schemes. The first method is a simple random method ( $B_R$ ), where camera frames were randomly chosen from the image pool. The second method, a scheme commonly used in many surveillance systems, applies prioritization in a ‘round robin’ method ( $B_{RR}$ ), ensuring all cameras are covered equally [15, 48, 49].

Only detections with confidence above 80% were included in the detection counts. This threshold was empirically determined by visual inspection of all processed datasets to ensure that nearly all objects detected above this threshold were true positive detections. If such a threshold is not set, many false positives would be detected, such as the falsely labeled ‘potted plant’ (actually a printer) with a detection confidence of 33% in Figure 3.7.

### 3.3.2 Processing Time

The processing time of any prioritization scheme(s) used in the Frame Selector were required not to exceed  $T$ , the maximum processing time, in order to meet the real-time requirements. For each dataset, a different  $T$  value was used according to the Batch Processing Time presented in Table 3.2.

Each prioritization scheme was designed to be run on a CPU, as the main purpose of prioritization in this work was to reduce load on the GPU. The CPU used to

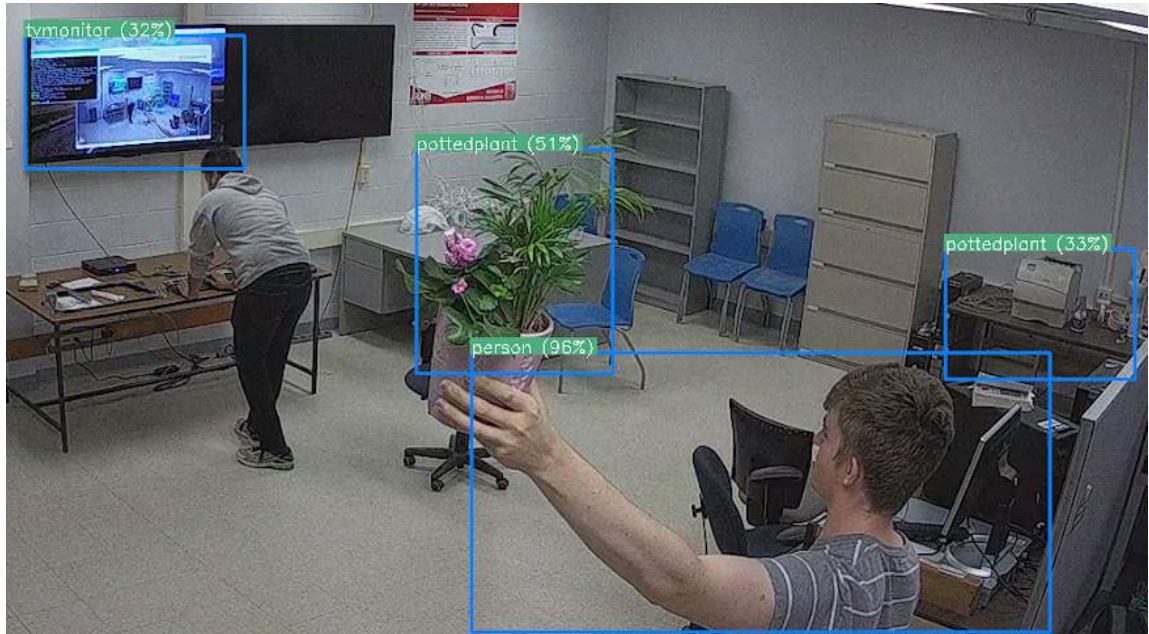


Figure 3.7: An example image annotated by an object detector to demonstrate the challenges with detection confidence

evaluate the processing time for each prioritization scheme was an Intel<sup>®</sup> Xeon<sup>®</sup> CPU E5-2637 v4 running at 3.50GHz.

# Chapter 4

## Individual Prioritization Schemes for Frame Selection

This section describes the various methods used for frame prioritization. To begin, the literature for each method is reviewed. This is followed by the implementation details for each prioritization scheme. Finally, the results of applying each prioritization scheme to the three datasets are presented.

### 4.1 Methods

#### 4.1.1 Spatial and Temporal Frame Information

##### 4.1.1.1 Frame Differencing

Frame differencing is one of the most intuitive methods of extracting temporal information. By observing the difference between two frames (consecutive or otherwise), information about the changes between images can be extracted. Recently, frame differencing has been used as the main information input for image prioritization methods [14, 20]. In this work, these so called delta frames are created in a two step process:

1. The two frames were compared on a pixel by pixel basis, and the difference was compared to a predefined threshold (here, set to 50 empirically). If the threshold was exceeded, the value was assigned a value of 1, and otherwise it was set to 0. This process created a binary array representing which pixels were sufficiently different to consider as changed. This binary array is referred to here as a binarized

delta frame.

2. The binarized delta frame was then multiplied element-wise with the current frame in order to create the complete delta frame. This delta frame can be thought of as a masked version of the original image emphasizing only the content that has changed.

An example of this delta frame extraction is shown in Figure 4.1.

The literature describes two main approaches to selecting which images are compared during differencing. The target frame being analyzed is typically compared to either an empty reference frame (such as the empty hallway in Fig. 4.1(a)) [14] or the previous frame in the sequence ( $n - 1$ ) [20]. Both methods have benefits and drawbacks within the context of automated surveillance systems.

When using consecutive frames, especially those from a high frame rate camera, the difference in frames may present no information gain if the object of interest has not moved. In this case, the delta frame would appear empty as no pixel values would have changed above the threshold for binary activation. This could lead to under prioritization of the current frame. An example of this could be an unattended suitcase or book bag, something that is often considered to be a security threat in airport settings [50].

A delta frame created in relation to an empty frame alleviates this problem by always showing the maximum information difference compared to some known baseline. However, having a true empty frame can also be problematic if the environment

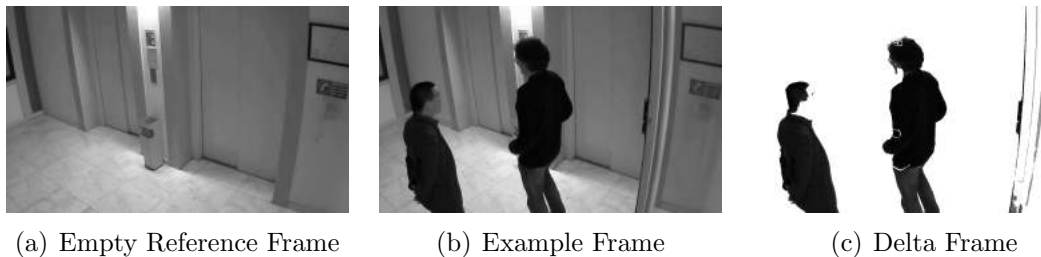


Figure 4.1: An example showing a delta frame demonstrating the difference between two frames.

is likely to change (consider day versus night, or a Canadian winter compared to a baseline summer image, for example) [51]. In the case of the shorter length datasets used in this work, this is not likely to be significant, however, some of the datasets contain camera footage that never contains such an ‘empty’ baseline (e.g. Camera 54 of the HDA dataset, shown in Fig. B.4(g)).

#### 4.1.1.2 Blob Detection

A ‘blob’ is a binary large object, which refers to a group of connected pixels in a binary image. Blob detection was used previously as a primitive form of object detection [22]. After two images are differenced (absolute subtraction) and binarized (thresholded to 0 or 1), any differences between the two images appear as a blob with the outline of the object. An example of two people blobs is shown in Figure 4.2. After applying a blob detector, as described below, a count of the number of blobs can suggest how many objects may be present within the frame [52].

The OpenCV library [42] provides a popular blob detector called `SimpleBlobDetector` which is used in this work. It begins by converting the source image to binary images by applying thresholding on pixel values with several thresholds from `minThreshold` (inclusive) to `maxThreshold` (exclusive) with distance `thresholdStep` between neighboring thresholds. The algorithm then extracts any pixel clumps/groupings from every binary image using `findContours()` and calculates their centers. The centers from several binary images are then grouped by their coordinates. Close centers are combined into one group, forming a single blob. The distance between blobs is controlled by the `minDistBetweenBlobs` parameter. Finally, from each of the groups, the estimated final centers of blobs and their radii are returned. In this work the `minThreshold` and `maxThreshold` values were set to 50 and 255 respectively, while `thresholdStep` was set to 50.

The `SimpleBlobDetector` used in this work cannot tell the difference between a human blob or any other blob. This means that the blob detector can be susceptible



Figure 4.2: The ‘blobs’ (represented by the red circles) detected in the binarized delta frame

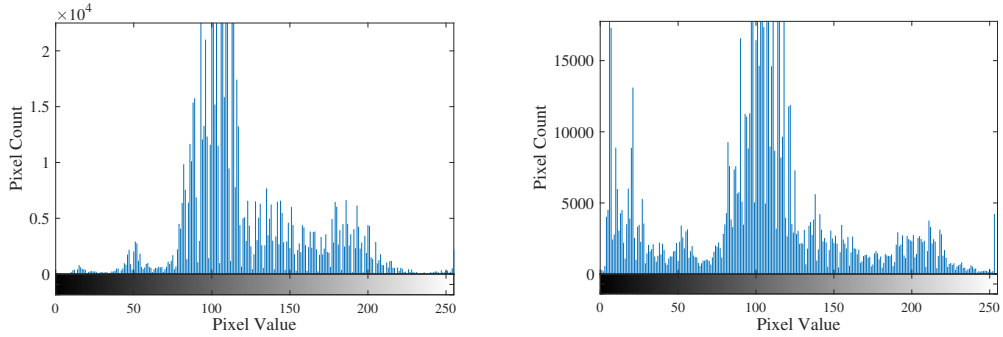
to drastic changes in lighting or non important objects. Upon close inspection, it can be observed in Figure 4.2, that a blob was found in the bottom right hand of the frame, caused by the door that had been swung open. While some noise can be removed by tuning the parameters of the blob detector (such as the minimum allowed average activation of pixels within the blob), the blobs detected may be unrepresentative of the information present within the frame. Nevertheless, this approach has found popularity as a simple and functional approach to activity detection.

#### 4.1.1.3 Entropy

An alternative approach derived from information theory, entropy is a measure of the randomness in an image. In computer vision, entropy is typically determined by the variation of pixel values (0-255) in an image. A histogram of values demonstrating this concept can be seen in Figure 4.3. Entropy is calculated from this histogram as described in Equation 4.1 [52]. As an image changes so does its entropy, which can be seen in Figure 4.3.

$$entropy = \sum_{p=0}^{255} (p * \log_2(p)) \quad (4.1)$$

In this work an image’s histogram is computed using OpenCV’s `calcHist` function. From this histogram the entropy of the image is then calculated following Equation



(a) Histogram of Figure 4.1(a) which corresponds to an entropy of 6.46 (b) Histogram of Figure 4.1(b) which corresponds to an entropy of 7.00

Figure 4.3: An example demonstrating the change in entropy between two frames from Camera 17 of the HDA Dataset ( $\Delta Entropy = 7.00 - 6.46 = 0.54$ ).

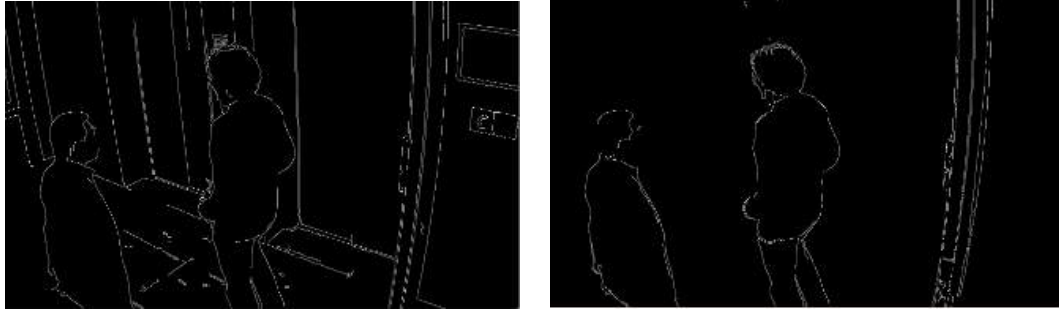
#### 4.1.

While entropy was once used to perform object detection [22], it has more recently been used to prioritize and select frames for various surveillance applications [53, 17, 54, 33, 55]. A single entropy value provides information about the randomness within an image frame. If there are several people wearing different clothing in a white hallway, for example, the entropy value would typically be higher than the empty hallway alone. The challenge with absolute entropy measurements, however, is that the background has a large impact on the extracted value, often leading to highly skewed values for backgrounds with a greater randomness.

By calculating the difference in entropies between frames, a delta entropy representing the variation in pixel values (and therefore information content) between two frames, can be calculated. Subtracting two entropies helps to eliminate the common elements of the background reducing their impact. Delta entropy was used by [17] and has proven to be a very effective method to prioritize frames.

##### 4.1.1.4 Edge Detection

Edge detection within images is a well understood area in computer vision [52], and several previous works have used detected edges for image prioritization [17, 54, 33, 32]. Edge pixels often describe the outline of objects, suggesting that the



(a) The result of Figure 4.1(b) processed with an edge detector (b) The result of Figure 4.1(c) processed with an edge detector

Figure 4.4: An example showing how edges are detected in images

more edge pixels an image contains, the more likely that the image contains more objects. Irgan et al. [17], therefore, used the number of edge pixels as a metric to prioritize visual information. There are, however, many approaches to extracting edge information. The Hough transform [56], for one, is quite popular for detecting specific geometries, but is relatively computationally expensive, and so has not been frequently used to assess the priority of images in real-time systems.

In this work, a Canny edge detector [57], one of the most popular edge detectors, was used to detect edges present in images. The Canny detector first applies a 5 by 5 Gaussian filter in order to reduce the noise in the image. It then finds the intensity gradient of the image and applies non-maximum suppression to create a binary image with thin edges. It then uses hysteresis thresholding to determine which edges are true edges.

In this work, the Canny edge detector provided by the OpenCV library [42] was used with the lower and upper pixel value threshold set to 35 and 255 respectively. Figure 4.4(a) shows the results of processing an image with this edge detector. An edge detector can also be applied after extracting a delta frame to improve information density, as shown in Figure 4.4(b).



(a) Color Frame

(b) An example saliency frame created by processing with Figure 4.5(a)

Figure 4.5: A demonstration of saliency

#### 4.1.1.5 Saliency

Saliency is often used in computer vision to describe prominent features within an image. An example of a salient object is a white boat on a blue ocean, or a football player with a red jersey on a green field. Saliency algorithms have been applied to image prioritization in several works [16, 35]. The saliency method used in this work was based upon that of Hou and Zhang [58] and was provided by an OpenCV library [42] function called `StaticSaliencySpectralResidual()`. This function analyzes the log spectrum of a color image frame to obtain a spectral residual map. The spectral residual map is then converted to the spatial domain to obtain the salient frame, which suggests the positions of potential objects.

#### 4.1.2 Detection Confidence

In automated surveillance systems, thresholds for object detection confidences must be set to a value that finds a good balance between false negatives and false positives. To be practical, false positives must usually be avoided at all costs, otherwise excessive human intervention is required, counteractive to the benefit of the system. In this work, the confidence threshold was set to 80%, as visual inspection of the processed datasets determined a low rate of false positives above this level. If an object of interest is not detected due to insufficient confidence when the object first

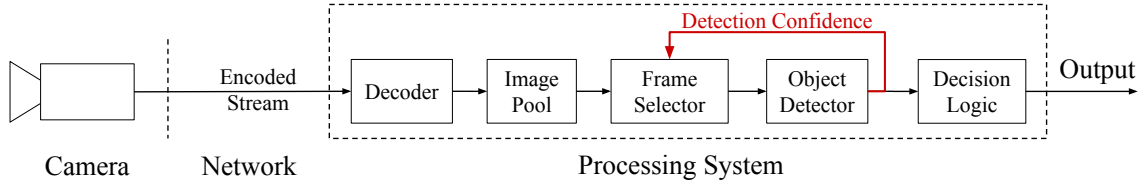


Figure 4.6: A block diagram of an automated surveillance system with priority-aware computation that uses values provided from the object detector.

enters a camera’s FOV, it may still be detected in subsequent frames. An example of this was shown in Figure 2.4 where the backpack’s detection confidence changes from frame to frame. While the object of interest may not exceed the threshold, it may still yield a higher than random confidence. Furthermore, frames with high confidence are likely to be followed by more frames with high confidence while the object is being tracked. For this reason, the confidence of a detection (whether sufficient for an alerted detection or not) may be used to prioritize subsequent frames.

As shown in Figure 4.6, the detection confidences created by the object detector were also stored by the Frame Selector. The prioritization metric used here was the total sum of detection confidences over 50 % for all objects of interest detected in the camera’s previously processed frame.

## 4.2 Implementation

### 4.2.1 Spatiotemporal Prioritization

Using the methods presented in the Section 4.1.1, various combinations of the spatial (blob, edge, entropy, saliency) and temporal (delta frames) methods were combined to create the following prioritization metrics:

- **Blob - Salient Frame, Current** ( $S_{B,S,C}$ ): This implementation begins by pre-processing the current frame with a saliency filter, and then applies a blob detector to the salient frame. The prioritization metric is the number of blobs detected in the salient frame.

- **Blob - Delta Frame, Reference** ( $S_{B,F,R}$ ): A delta frame is created by comparing the current frame to a pre-determined reference frame with no objects of interest. A blob detector is then applied to the delta frame. The prioritization metric is the number of blobs detected in the delta frame.
- **Blob - Delta Frame, Consecutive** ( $S_{B,F,C}$ ): A delta frame is created by comparing the current frame to the previous frame ( $n - 1$ ). A blob detector is then applied to the delta frame result. The prioritization metric is the number of blobs detected in the delta frame.
- **Edge - Delta Count, Reference** ( $S_{Ed,C,R}$ ): The current frame is processed with a Canny edge detector; summing the number of edge pixels. The absolute difference is then calculated between the current edge pixel count and the edge pixel count of a pre-determined (empty) reference frame. The prioritization metric is the absolute difference in edge pixels.
- **Edge - Delta Count, Consecutive** ( $S_{Ed,C,C}$ ): The current frame is processed with a Canny edge detector; summing the number of edge pixels. The absolute difference is then calculated between the current edge pixel count and the edge pixel count of the previous frame ( $n - 1$ ). The prioritization metric is this absolute difference in edge pixels.
- **Edge - Delta Frame, Reference** ( $S_{Ed,F,R}$ ): A delta frame is created by comparing the current frame with a pre-determined (empty) reference frame. A Canny edge detector is then applied to the delta frame. The prioritization metric is the number of edge pixels found in the delta frame.
- **Edge - Delta Frame, Consecutive** ( $S_{Ed,F,C}$ ): A delta frame is created by comparing the current frame with the previous frame ( $n - 1$ ). A Canny edge detector is then applied to the delta frame. The prioritization metric is the number of edge pixels found in the delta frame.
- **Entropy - Delta Count, Reference** ( $S_{En,C,R}$ ): The entropy of the current

frame is calculated. The absolute difference is then calculated between the current entropy value and the entropy value of a pre-determined (empty) reference frame. The prioritization metric is this absolute difference in entropy.

- **Entropy - Delta Count, Consecutive** ( $S_{En,C,C}$ ): The entropy of the current frame is calculated. The absolute difference is then calculated between the current entropy value and the entropy value of the previous frame ( $n - 1$ ). The prioritization metric is this absolute difference in entropy.
- **Entropy - Delta Frame, Reference** ( $S_{En,F,R}$ ): A delta frame is created by comparing the current frame with a pre-determined (empty) reference frame. The prioritization metric is the calculated entropy of the delta frame.
- **Entropy - Delta Frame, Consecutive** ( $S_{En,F,C}$ ): A delta frame is created by comparing the current frame with the previous frame ( $n - 1$ ). The prioritization metric is the calculated entropy of the delta frame.
- **Pixel Count - Delta Frame, Reference** ( $S_{P,F,R}$ ): A binarized delta frame is created by comparing the current frame with a pre-determined (empty) reference frame. The prioritization metric is the number of non-zero values in the delta frame.
- **Pixel Count - Delta Frame, Consecutive** ( $S_{P,F,C}$ ): A binarized delta frame is created by comparing the current frame with the previous frame ( $n - 1$ ). The prioritization metric is the number of non-zero values in the delta frame.

Each Frame Selector that used a spatiotemporal prioritization scheme pre-processed the frames down to a size of 640 by 480, before calculating its priority value. This was done to ensure that processing times remained consistent across cameras and datasets. Furthermore, as is standard with image processing methods, the camera frames were converted to gray-scale before being processed by spatiotemporal prioritization schemes. Both the computation time of the re-scaling and gray-scale conversions are included in the measured processing times. Finally, the delta frames

were all calculated as described at the end of Section 4.1.1.1.

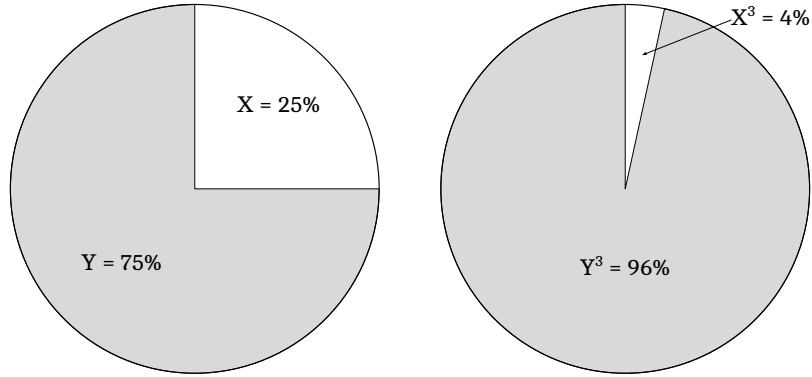
## 4.2.2 Confidence-based Prioritization

All of the spatiotemporal prioritization schemes described in the previous section calculate priority on a per frame basis, based on each frame’s own pixel value information. Conversely, detection confidence values are updated on a per camera basis and are determined by previously selected frames. If a simple sorting method is used to select frames with the highest priority, the cameras with the highest detection counts will always be selected, causing the system to always succumb to local minima; a particular camera may never be able to build up enough confidence to be selected. To mitigate this risk, a weighted roulette selection method was used. This method uses the camera’s previous detection confidence value as the weighting for each frame and randomly selects a frame based on the weight. After a frame is chosen, it is removed from the roulette. The roulette is ‘spun’  $N_p$  times in order to create a selection of chosen frames.

The weighted roulette adds randomness so that the system avoids these local minima. This benefits the system by insuring that no one camera dominates the selection process, and that lower priority camera are viewed at least occasionally, allowing the system to update its detection count/confidence information. It was found empirically that a cubic non-linear roulette weighting resulted in the highest detection values. A visual depiction of this is provided in Figure 4.7.

The following confidence-based prioritization schemes were created based on historical confidence values:

- **Confidence - Non-Linear Roulette ( $C_{NL}$ ):** For each camera, the priority value was determined by summing the detection confidences of the most recently processed images that were greater than 50%. This value was then normalized by the number of frames the camera had in the batch. These values were treated as prioritization metrics and passed to a non-linear weighted roulette as described at



(a) Linear Roulette:  $X = 2$ ,  $Y = 6$  (b) Non-Linear Roulette:  $X^3 = 8$ ,  $Y^3 = 216$

Figure 4.7: An example demonstrating the difference between a linear and a non-linear roulette

the beginning of this section. The results of the roulette were used as the ranking results passed to the object detector. After each prioritized frame was processed, the detection information was stored. Only the most recent detection confidence information for each camera was kept, and the older information was discarded.

- **Confidence - Linear Roulette ( $C_L$ ):** This scheme behaves identically to the previous scheme with the exception that a linear weighted roulette was used.
- **Confidence - Sorted ( $C_S$ ):** This scheme behaves identically to the previous scheme with the exception that the prioritization metrics were sorted from highest to lowest and the frames with the highest values were passed to the object detector.

### 4.3 Results

The performance of each of the prioritization schemes presented in the previous section were evaluated on each of the surveillance datasets presented. The parameter settings (ie. batch sizes) for the testbed were consistent with those previously outlined in Table 3.2.

### 4.3.1 Detection Count

Figure 4.8 shows a summary of the detection count performance for each prioritization scheme on all three datasets. The  $B_R$  and  $B_{RR}$  methods were included to show the relative performance compared to baseline random and round robin approaches, respectively.

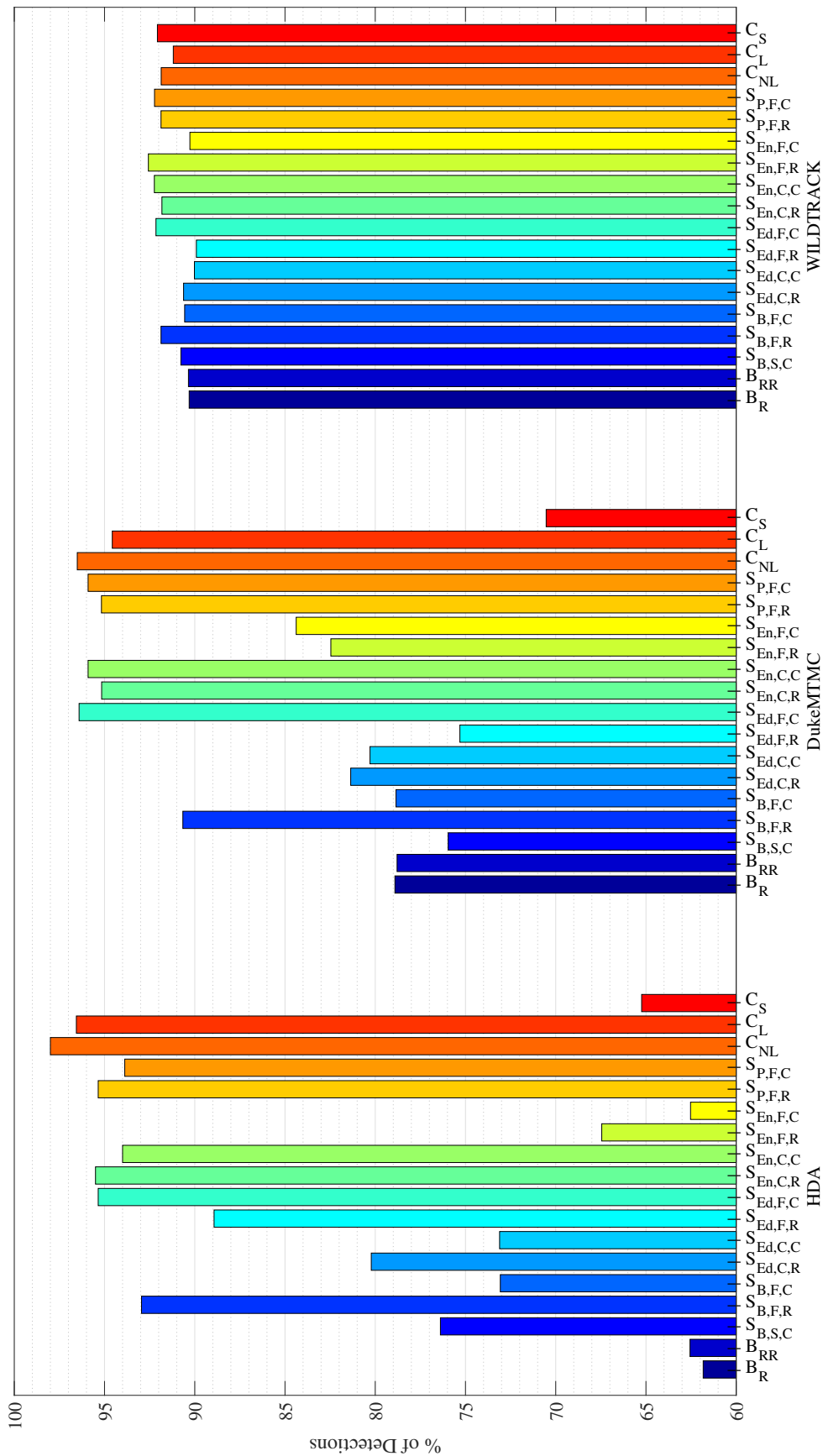
### 4.3.2 Processing Time

Figure 4.9 shows the processing times taken for the Frame Selector to select the most important frames from one batch with each prioritization scheme. The time  $T$  is included as a dotted line across the graph to show the processing time of the image batch with an object detector  $T = \frac{1}{F_p}$ . The processing time should be lower than this value in order to meet real-time requirements. Note that the processing times for  $C_{NL}$ ,  $C_L$ , and  $C_S$  were much smaller than the other prioritization schemes, making them difficult to see by comparison.

## 4.4 Discussion

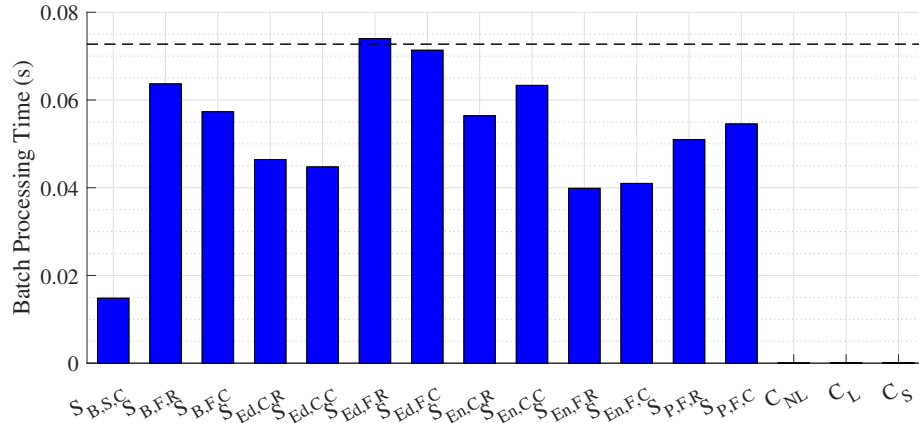
The goal of the work described in this chapter was to test several methods of frame selection and to evaluate their effectiveness in prioritizing surveillance camera frames. Three datasets were used to provide a variety of surveillance environments in order to better contrast the effectiveness of each prioritization scheme.

The baseline detection count results presented in Figure 4.8 demonstrate the object of interest density in each of the datasets. By observing the difference between the results of the *Random* ( $B_R$ ) and *Round Robin* ( $B_{RR}$ ) baseline prioritization scheme, it is clear that the HDA-Dataset has the lowest density of objects of interest. When choosing random frames to process, 61% of possible detections within the HDA dataset were achieved. The DukeMTMC dataset had a baseline of 78%, demonstrating that this dataset has a higher frequency of objects of interest. The

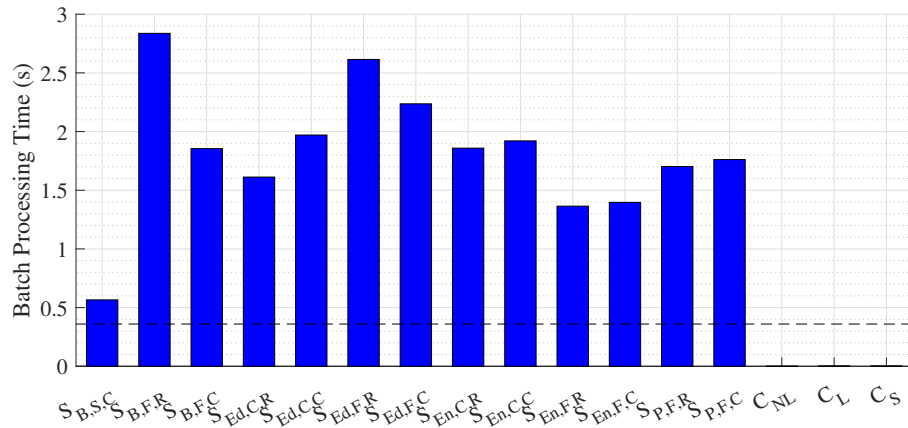


Legend:  $B_R$ : Baseline - Random |  $B_{RR}$ : Baseline - Round Robin |  $S_{B,S,C}$ : Blob - Salient Frame, Current |  $S_{B,F,R}$ : Blob - Delta Frame, Reference |  $S_{B,F,C}$ : Blob - Delta Frame, Consecutive |  $S_{Ed,C,R}$ : Edge - Delta Count, Reference |  $S_{Ed,C,C}$ : Edge - Delta Count, Consecutive |  $S_{Ed,F,R}$ : Edge - Delta Frame, Reference |  $S_{Ed,F,C}$ : Edge - Delta Frame, Consecutive |  $S_{En,C,R}$ : Entropy - Delta Count, Consecutive |  $S_{En,C,C}$ : Entropy - Delta Count, Reference |  $S_{En,F,R}$ : Entropy - Delta Frame, Reference |  $S_{En,F,C}$ : Entropy - Delta Frame, Consecutive |  $S_{P,F,R}$ : Pixel Count - Delta Frame, Reference |  $S_{P,F,C}$ : Pixel Count - Delta Frame, Consecutive |  $C_{NL}$ : Confidence - Non-Linear Rou. |  $C_L$ : Confidence - Linear Rou. |  $C_S$ : Confidence - Sorted

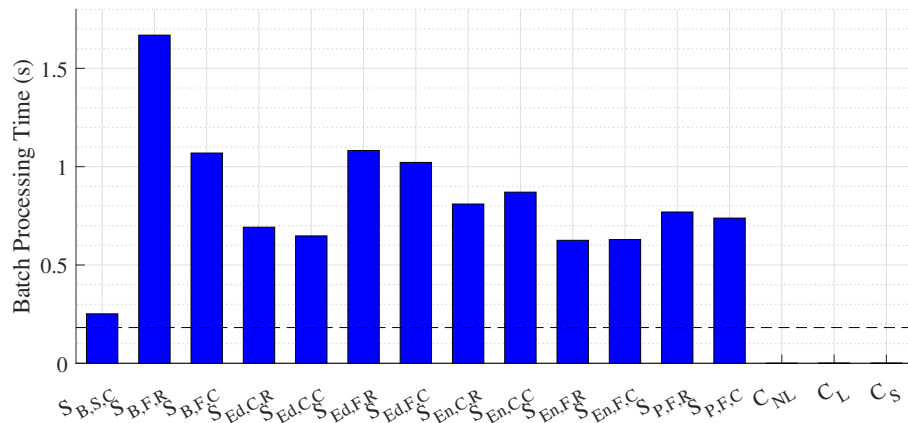
Figure 4.8: The detection count achieved for each individual prioritization scheme expressed as percentage of achievable detections



(a) Average Prioritization Scheme Processing Time per Image Batch for the HDA Dataset



(b) Average Prioritization Scheme Processing Time per Image Batch for the DukeMTMC Dataset



(c) Average Prioritization Scheme Processing Time per Image Batch for the WILDTRACK Dataset

Figure 4.9: The processing time per image batch for each prioritization scheme

WILDTRACK dataset, whose object density is the highest of three datasets, had a baseline of 90%.

The first prioritization schemes presented in this chapter were based on spatial and temporal information created by analyzing individual camera frames. The best performing spatiotemporal prioritization schemes ( $S_{B,F,R}$ ,  $S_{Ed,F,C}$ ,  $S_{En,C,R}$ ,  $S_{En,C,C}$ ,  $S_{P,F,R}$ , and  $S_{P,F,C}$ ) brought both the HDA and DukeMTMC detection values up to approximately 95%. The WILDTRACK dataset appears to present the greatest challenge out of all the datasets, with no spatiotemporal prioritization schemes greatly outperforming the baseline. However, across all datasets, the prioritization schemes  $S_{B,F,R}$ ,  $S_{Ed,F,C}$ ,  $S_{En,C,R}$ ,  $S_{En,C,C}$ ,  $S_{P,F,R}$ , and  $S_{P,F,C}$  had the best performance across each of the datasets proportional to the baseline methods  $B_R$  and  $B_{RR}$ .

The spatiotemporal prioritization schemes that used delta frames achieved overall better results than the prioritization schemes that did not ( $S_{B,S,C}$ ,  $S_{B,F,C}$ ,  $S_{Ed,C,R}$ ,  $S_{Ed,C,C}$ ,  $S_{En,F,R}$ , and  $S_{En,F,C}$ ). Delta frames created by consecutive frames and empty reference frames were both tested, and the success of a prioritization scheme appears to be impacted more significantly by the method of extracting spatial information (ie. entropy or blob count) than the choice in temporal information (consecutive or reference frames). Most prioritization schemes based on the same spatial method, such as  $S_{En,C,R}$  and  $S_{En,C,C}$  or  $S_{P,F,R}$  and  $S_{P,F,C}$ , yielded similar performance for both types of delta frames. The exceptions to this were the  $S_{B,F,R}$  and  $S_{Ed,F,C}$  prioritization schemes, which outperformed their counterparts.

The  $S_{B,F,R}$  prioritization scheme was based on a binarized delta frame created with an empty reference frame and a simple blob detector. It was found that objects in a delta frame based on a reference frame would appear as larger blobs, as opposed to in the consecutive delta frames ( $S_{B,F,C}$ ), where the changes would be smaller. These smaller blobs are less likely to be detected by the blob detector, and tended to be confused with any noise in the delta frame.

The  $S_{Ed,F,C}$  prioritization scheme was based on a delta frame created with consecutive frames, which was then processed by an edge detector. The consecutive delta frame in this case greatly outperformed the reference frame ( $S_{Ed,F,R}$ ). This was likely due to the consecutive frame approach giving important information about cameras with large changes between frames, allowing the frame selector to choose frames with moving objects. The edge pixel count of the reference delta frames also suffered from the bias of any perceived edges on an object (ie. light and dark clothing, or plaid), leading to poorer prioritization.

These detection count results validate the effectiveness of each of the spatiotemporal prioritization schemes in prioritizing visual data for maximum detection counts. The results of the most successful prioritization schemes are further examined in order to create effective ensemble implementations in Chapter 5.

Of the methods that produced the best detection counts,  $S_{P,F,R}$  and  $S_{P,F,C}$  had the lowest detection processing times as shown in Figure 4.9. Processing time is an important factor in deciding which prioritization schemes to use in the ensemble implementations since the batch processing time of the frame selector must be less than that of the object detector in order to maintain real-time performance.

As shown by the bar charts in Figure 4.9, however, only the HDA dataset yielded prioritization scheme processing times below the required timing threshold. This is due to the HDA dataset having a much lower frame rate on average ( $\leq 5$  FPS) versus the DukeMTMC and WILDTRACK datasets that had higher frame rates (60 FPS). As previously discussed, the prioritization schemes used here were implemented in Python 2.7 and were only single threaded. Given the insufficient speed performances found here, further optimization was performed to the ensemble implementations described in Chapter 6.

Conversely, the processing times of the confidence prioritization schemes were much less than both the spatiotemporal prioritization schemes and the batch processing

time threshold ( $T$ ). The detection count performance of  $C_{NL}$  was the highest of all individual prioritization schemes on the HDA and DukeMTMC datasets. However, it yielded no real impact on the WILDTRACK detection count performance over baseline. This  $C_{NL}$  prioritization scheme uses a non-linear weighted roulette, marginally outperforming the linear weighted roulette  $C_L$ , but substantially outperforming the sorted method  $C_S$ .

Based on the overall processing time and detection count performances,  $C_{NL}$  proved to be the best individual prioritization scheme in this chapter. The WILDTRACK dataset, regardless of scheme, proved to be a challenge for all prioritization schemes given its high rates of objects of interest.

# Chapter 5

## Combining Prioritization Schemes for Frame Selection

### 5.1 Methodology

#### 5.1.1 Prioritization Scheme Selection

Chapter 4 demonstrated several prioritization schemes that individually selected frames of higher interest in order to maximize object detection. By observing the number of detections per camera for each prioritization scheme and comparing them to the optimal number of detections, an understanding of the relative merits of the different prioritization schemes can be gained. Figure 5.1 shows the a pseudo heat-map of the number of detections per camera for each dataset using the best performing prioritization schemes from Chapter 4 ( $S_{B,F,R}$ ,  $S_{Ed,F,C}$ ,  $S_{En,C,R}$ ,  $S_{En,C,C}$ ,  $S_{P,F,R}$ ,  $S_{P,F,C}$ , and  $C_{NL}$ ).

By evaluating the results presented in Figure 5.1, common trends between the prioritization schemes can be identified. The  $S_{Ed,F,C}$ ,  $S_{En,C,C}$ , and  $S_{P,F,C}$  metrics, and the  $S_{En,C,R}$ , and  $S_{P,F,R}$  metrics share two separate characteristics that are apparent in the performance of the prioritization schemes. The  $S_{Ed,F,C}$ ,  $S_{En,C,C}$ , and  $S_{P,F,C}$  metrics rely on consecutive delta frames and the  $S_{En,C,R}$ , and  $S_{P,F,R}$  metrics use empty reference frames. Despite using different spatiotemporal methods, the prioritization schemes that use the same type of delta frames have very similar results.

These trends in detection count between prioritization schemes suggest that com-

Prioritizer	Cam 02	Cam 17	Cam 18	Cam 19	Cam 40	Cam 53	Cam 54	Cam 56	Cam 57	Cam 58	Cam 59	Cam 60	Total
<b>Optimal</b>	390	2209	5668	3010	2978	242	8758	210	1640	1287	471	750	27613
<b>S - B,F,R</b>	353	2065	5699	2442	2752	208	8732	171	1235	824	452	733	25666
<b>S - Ed,F,C</b>	381	2165	5106	2979	2830	251	8244	224	1733	1301	485	630	26329
<b>S - En,C,R</b>	337	2006	5562	2745	2792	236	8515	209	1647	1252	519	551	26371
<b>S - En,C,C</b>	381	2181	5082	2979	2877	251	7826	229	1715	1274	496	665	25956
<b>S - P,F,R</b>	339	2131	5527	2800	2786	236	8277	208	1666	1226	518	614	26328
<b>S - P,F,C</b>	380	2152	5077	2981	2899	251	7802	229	1713	1271	495	672	25922
<b>C - NL</b>	379	2214	5571	3018	2993	201	8764	139	1606	1185	376	616	27062

(a) The detection count for each camera of the HDA dataset

Prioritizer	Cam 1	Cam 2	Cam 3	Cam 4	Cam 5	Cam 6	Cam 7	Cam 8	Total
<b>Optimal</b>	262931	323383	120468	126118	185153	337844	168786	261713	1786396
<b>S - B,F,R</b>	267044	297690	89311	116774	183125	285513	121725	258417	1619599
<b>S - Ed,F,C</b>	280591	319987	125168	81907	179773	317771	168439	248584	1722220
<b>S - En,C,R</b>	282187	305670	76581	88246	190273	334773	161686	260496	1699912
<b>S - En,C,C</b>	279339	319533	128720	75748	180605	312452	170314	246594	1713305
<b>S - P,F,R</b>	282212	307387	75938	88958	190526	333695	160374	261042	1700132
<b>S - P,F,C</b>	279347	319559	128744	75729	180654	312316	170324	246613	1713286
<b>C - NL</b>	268165	322543	87199	123953	171929	334423	144194	271679	1724085

(b) The detection count for each camera of the DukeMTMC dataset

Prioritizer	Cam 1	Cam 2	Cam 3	Cam 4	Cam 5	Cam 6	Cam 7	Total
<b>Optimal</b>	277662	397290	328016	261931	404632	312866	372442	2354839
<b>S - B,F,R</b>	285101	314197	341775	251873	329487	256111	384787	2163331
<b>S - Ed,F,C</b>	285648	296918	321599	254142	350742	286633	374375	2170057
<b>S - En,C,R</b>	307392	266580	343399	251244	335892	267410	390286	2162203
<b>S - En,C,C</b>	284657	299553	321234	261899	359474	272388	372840	2172045
<b>S - P,F,R</b>	307178	274129	345345	251775	335663	259109	390307	2163506
<b>S - P,F,C</b>	284564	300001	321536	262118	359139	271808	372724	2171890
<b>C - NL</b>	287282	295411	320606	249828	319034	318265	372649	2163075

(c) The detection count for each camera of the WILDTRACK dataset

Figure 5.1: The total number of object detections achieved per camera for each dataset using the best performing prioritization schemes of Chapter 4. The cells colored red represent *lower* than the optimal detection count; green cell represent counts *above* above the optimal detection count. The closer to white the cell color is, the closer the value is to the optimal detection count, shown in the top row.

binning dissimilar schemes (such as the two different types of delta frames - consecutive and reference) may further improve the detection count. The delta frame created between the current frame and a reference frame generate complete objects in the cameras FOV. This is contrasted by the consecutive delta frame which only provides information on how much has changed over a short period of time.

Because the performance of these systems appeared to be dominated by the method of delta extraction, a representative scheme from each category was selected for further evaluation. Since all behaved with relatively equal performance, the fastest spatiotemporal prioritization schemes according to the results in Chapter 4,  $S_{P,F,R}$  and  $S_{P,F,C}$ , were chosen to be used in the ensemble implementations. The  $S_{B,F,R}$  and  $C_{NL}$  prioritization schemes, however, did not follow the same detection count patterns as any of the other well performing prioritization schemes from Chapter 4. Consequently, these two schemes were also evaluated, resulting in a total of four individual prioritization schemes ( $C_{NL}$ ,  $S_{B,F,R}$ ,  $S_{P,F,R}$ , and  $S_{P,F,C}$ ) used to create ensemble implementations.

## 5.1.2 Methods of Information Fusion

### 5.1.2.1 Weighted Average

Information fusion has been used in countless works to increase system performance by combining information from separate sources [59, 60]. As a tractable implementation of information fusion, a weighted average was used to combine the input from different prioritization schemes. Here, the weights were determined using a regression approach to map the fusion of the normalized ranking outputs from each of the prioritization schemes ( $C_{NL}$ ,  $S_{B,F,R}$ ,  $S_{P,F,R}$ , and  $S_{P,F,C}$ ) to the normalized ranked output of the optimal priority values. With the rankings for each frame from each camera across all datasets, several linear regressions were fit for each combination of the prioritization schemes (see Table 5.1).

Table 5.1: Weights obtained from the Fitting of the Weighted Average Ensembles

Prioritization Scheme Symbol	Individual Metric	Weight
$W_{P,A}$	$S_{P,F,R}$	0.5022
	$S_{P,F,C}$	0.4372
$W_{S,A}$	$S_{B,F,R}$	0.6163
	$S_{P,F,R}$	0.0917
	$S_{P,F,C}$	0.3378
$W_A$	$C_{NL}$	0.5192
	$S_{B,F,R}$	0.2186
	$S_{P,F,R}$	-0.0355
	$S_{P,F,C}$	0.2391
$W_{B,C,P}$	$C_{NL}$	0.5155
	$S_{B,F,R}$	0.2022
	$S_{P,F,C}$	0.2214
$W_{C,P}$	$C_{NL}$	0.6046
	$S_{P,F,C}$	0.2924

In total, five combinations of prioritization schemes were created. Their computed weights are displayed in Table 5.1. The first two combinations ( $W_{P,A}$  &  $W_{S,A}$ ) were chosen as complementary spatiotemporal methods, according to the distribution of chosen cameras shown in Figure 5.1.  $W_A$  was created by combining all of the best chosen schemes. By observing the weights of  $W_A$  and noting that  $S_{P,F,R}$  had a negligible weighting associated with it, thus  $W_{B,C,P}$  was created with the  $S_{P,F,R}$  scheme excluded.  $W_{C,P}$  was then created with the  $S_{B,F,R}$  scheme removed as to significantly decrease overall computation time of prioritization.

### 5.1.2.2 KNN Regressor

After testing the Weighted Average implementations and noting their poor increase in detection count performance over their individual prioritization schemes, it was determined that an adaptive fusion technique was needed. Thus the  $k$  nearest neighbors (KNN) regressor was also evaluated [61]. The KNN regressor acts as an estimator for object detection count based on the feature values (priority values) created by the individual prioritization schemes. It uses  $k$  of the previously observed object detection results on a per camera basis in order to estimate the value of the

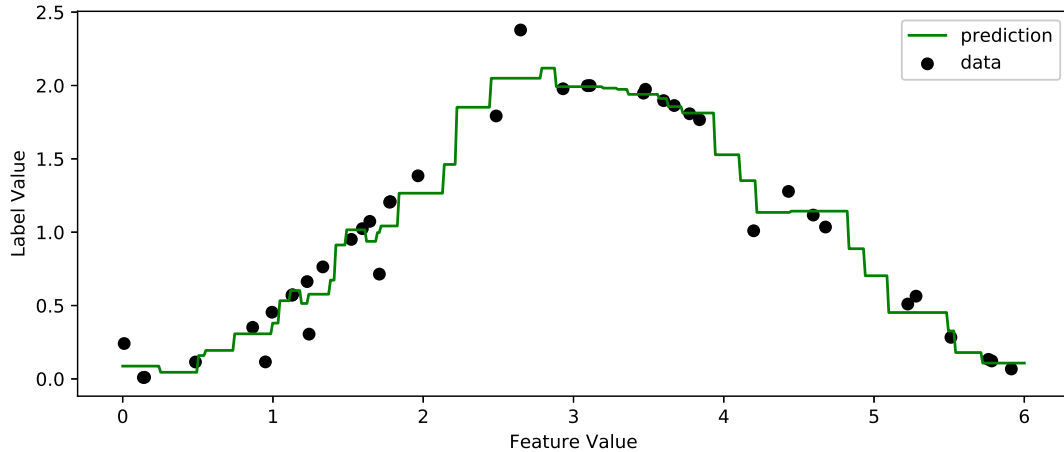


Figure 5.2: An example of how a KNN Regressor predicts a label value based on previous feature data

current observation. Figure 5.2 shows an example of how a KNN Regressor predicts a label value based on previous feature data.

Since the KNN regressor is non parametric and an instance-based classifier, there is no training that can be done before implementation on a particular surveillance system. This is a good quality for automated surveillance systems as the KNN regressor will tune itself to the camera system. Furthermore, implementing the estimator on a per camera basis, ensures that accurate estimations of detection counts are given for every camera which has a differing FOV and lighting.

A nearest neighbour approach has some limitations in terms of computational performance. With each estimation, a regression must be fit to the dataset. Thus with a large dataset there is a significant amount of computation needed. However, by keeping the number of sample points low, the computation time will remain low.

## 5.2 Ensemble Implementation

Using the methodologies presented in the previous section, a total of eight ensembles were implemented.

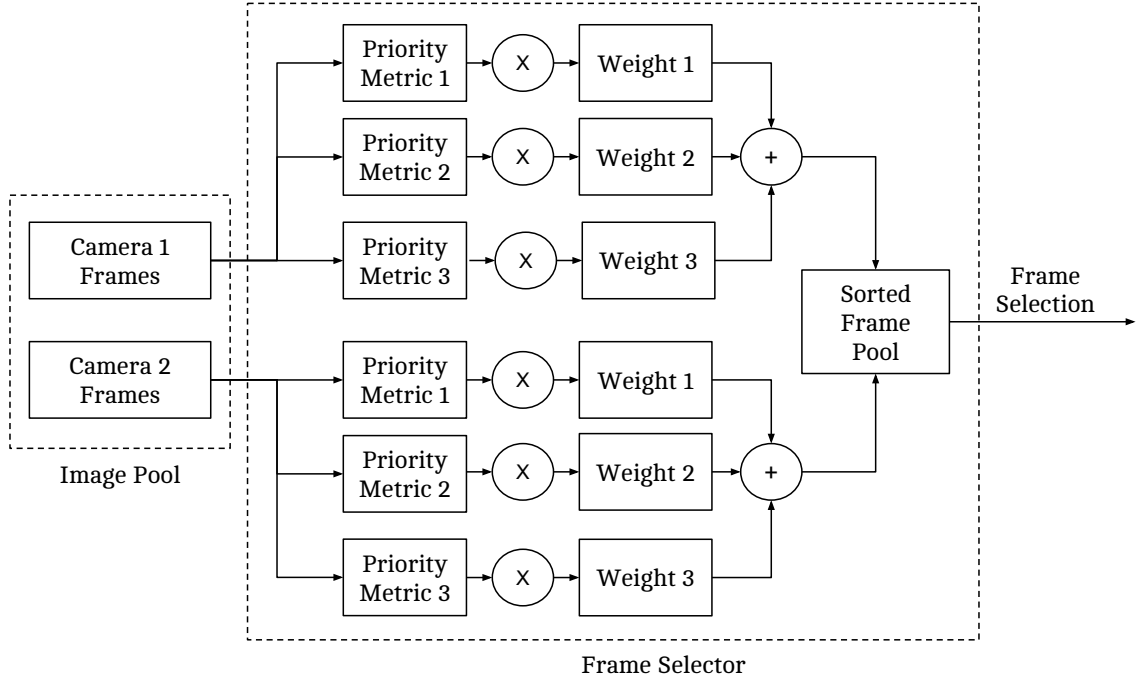


Figure 5.3: A block diagram outlining frame selection for a two camera automated surveillance system that uses three weighted prioritization schemes to prioritize frames.

### 5.2.1 Weighted Average

For the ensemble implementations that use only spatial temporal information ( $W_{P,A}$  and  $W_{S,A}$ ) the frame selector was constructed corresponding to the diagram in Figure 5.3. These ensembles did not need the roulette method as they do not rely on previous detection information; only spatiotemporal information, as outlined below.

- **Weighted Average** -  $S_{P,F,R} + S_{P,F,C}$  ( $W_{P,A}$ ): This ensemble is a pairing of two prioritization schemes that use the same method of extracting spatial information (pixel count) but use differing temporal data (delta frames). The weights used in this ensemble are given in Table 5.1. This implementation follows the design shown in Figure 5.3.
- **Weighted Average** -  $S_{B,F,R} + S_{P,F,R} + S_{P,F,C}$  ( $W_{S,A}$ ): This ensemble uses all three spatiotemporal prioritization schemes chosen in the previous section. The

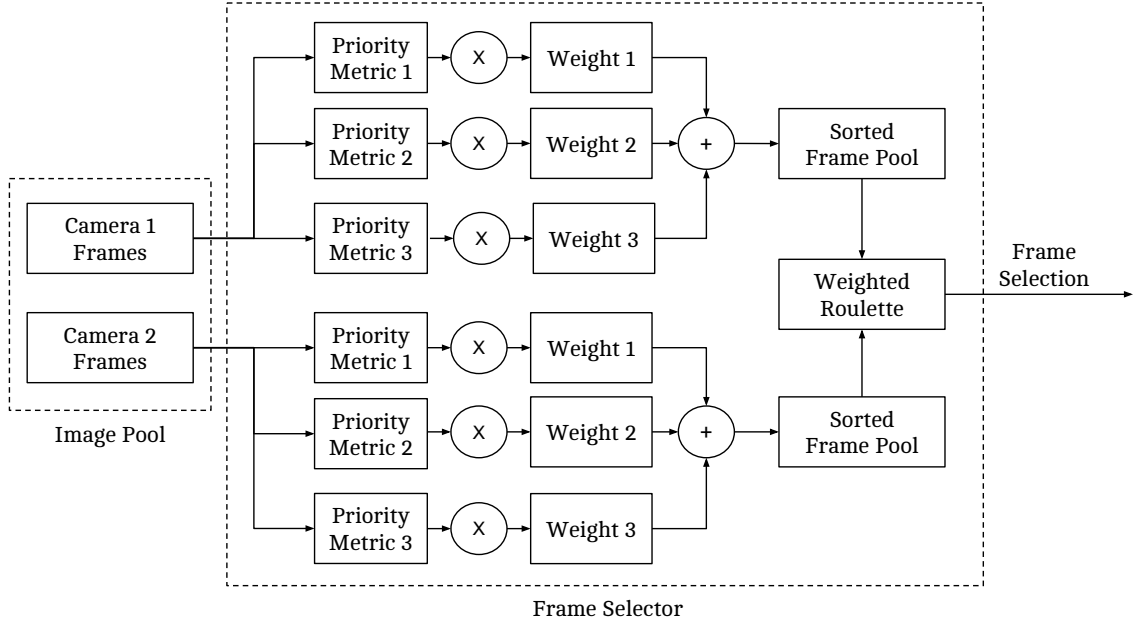


Figure 5.4: A block diagram outlining a two camera automated surveillance system’s frame selector that uses three weighted prioritization schemes and a roulette selection method to prioritize frames.

weights used in this ensemble are given in Table 5.1. This implementation follows the design shown in Figure 5.3.

The ensemble implementations that do rely on previous detection information ( $W_{C,P}$ ,  $W_A$ ,  $W_{B,C,P}$ ) followed the structure shown in Figure 5.4. For these implementations, a non-linear weighted roulette was used with one modification. Instead of using the roulette method with all frames from the image pool and their associated weighting, only a single frame with the highest priority value from each camera was used in the roulette. As frames were selected, they were removed from the associated camera queue and the next highest priority frame from that camera was put into the roulette. The outline for these three ensemble implementations are as follows.

- **Weighted Average -  $C_{NL} + S_{P,F,C}$  ( $W_{C,P}$ ):** The  $C_{NL}$  and  $S_{P,F,C}$  prioritization schemes were chosen as a fusion of spatiotemporal information and detection confidence.  $S_{P,F,C}$  was chosen because it was given a higher weighting than  $S_{P,F,R}$  (as shown in in Table 5.1 by the weights of  $W_A$ ) and requires less processing time than

Table 5.2:  $k$  values and their impact on detection performance, using the HDA Dataset

$k$ value	1	2	3	4	5	10	15
% of Detections	95.87	97.07	98.15	97.64	97.58	96.84	96.52

$S_{B,F,R}$ . The weights used in this ensemble implementation are given in Table 5.1. This implementation followed the design shown in Figure 5.4.

- **Weighted Average -  $C_{NL} + S_{B,F,R} + S_{P,F,R} + S_{P,F,C}$  ( $W_A$ ):** This ensemble implementation used all four prioritization schemes chosen in the previous section. The weights used in this ensemble are given in Table 5.1. This implementation also followed the design shown in Figure 5.4.
- **Weighted Average -  $C_{NL} + S_{B,F,R} + S_{P,F,C}$  ( $W_{B,C,P}$ ):** From Table 5.1, it is evident that the weight for  $S_{P,F,R}$  in  $W_A$  is negligible. With this in mind, only the three more prominent prioritization schemes were included. The weights used in this ensemble implementation are given in Table 5.1. This implementation also followed the design shown in Figure 5.4.

### 5.2.2 KNN Regressor

This work used the KNN Regressor implemented in Python by Scikit-learn [62]. A diagram for the design of this ensemble implementation is shown in Figure 5.5, where the Estimator block in the diagram is a KNN Regressor. Prioritization schemes and detection confidences were stored for each camera. The prioritization schemes were used as the data points for regression while the detection confidences were used as data labels. Observed data points were stored in a queue, with the oldest ones being replaced by newer ones whenever a frame from the camera was processed by the object detector. From empirical experimentation, it was found that storing a total of 30 sample points and a value of  $k = 3$  gave the best detection count results.

Based on the detection results of the Weighted Average ( $W_n$ ) ensemble implementations shown in Figure 5.6, the best performing combination of individual priori-

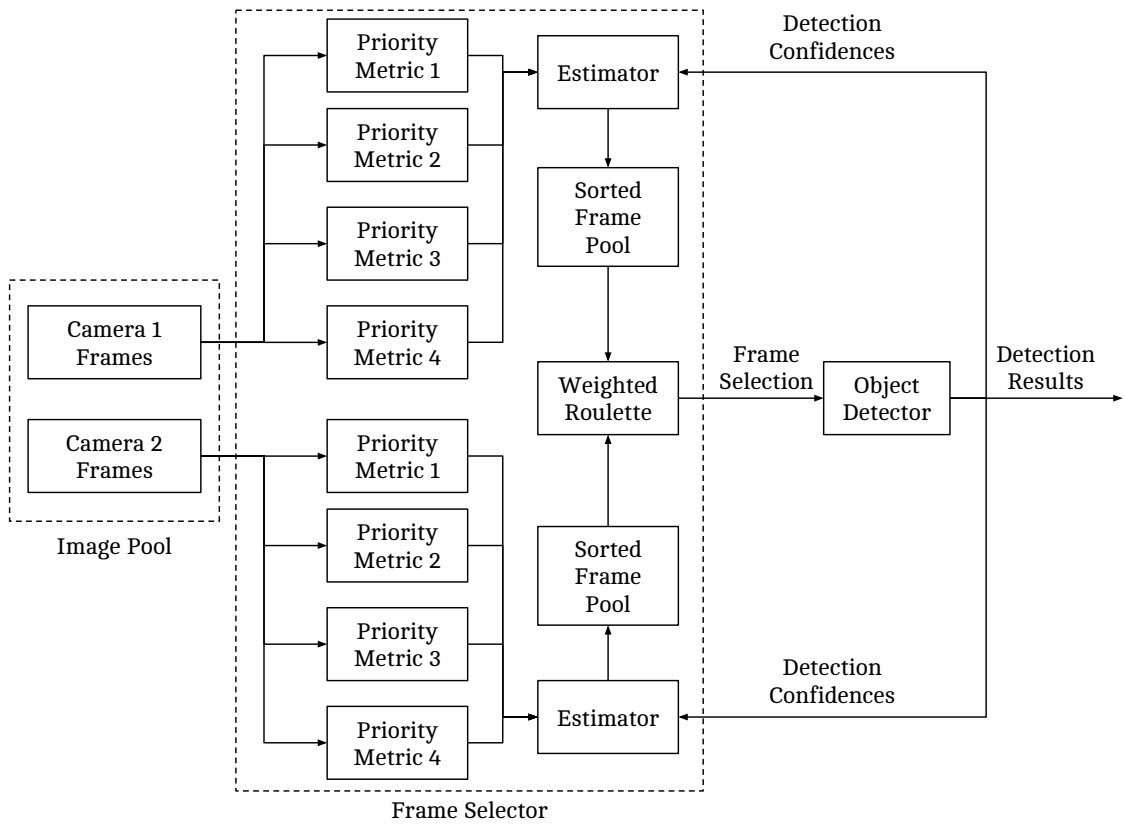


Figure 5.5: A block diagram outlining a two camera automated surveillance system's frame selector that uses four metrics to inform an estimator to prioritize frames.

tization schemes were used to create the following KNN Estimator ensemble implementation:

- **KNN Estimator** -  $C_{NL}, S_{B,F,R}, S_{P,F,R}, S_{P,F,C} (E_1)$
- **KNN Estimator** -  $C_{NL}, S_{B,F,R}, S_{P,F,C} (K_{B,C,P})$
- **KNN Estimator** -  $C_{NL}, S_{P,F,C} (K_{C,P})$

## 5.3 Results

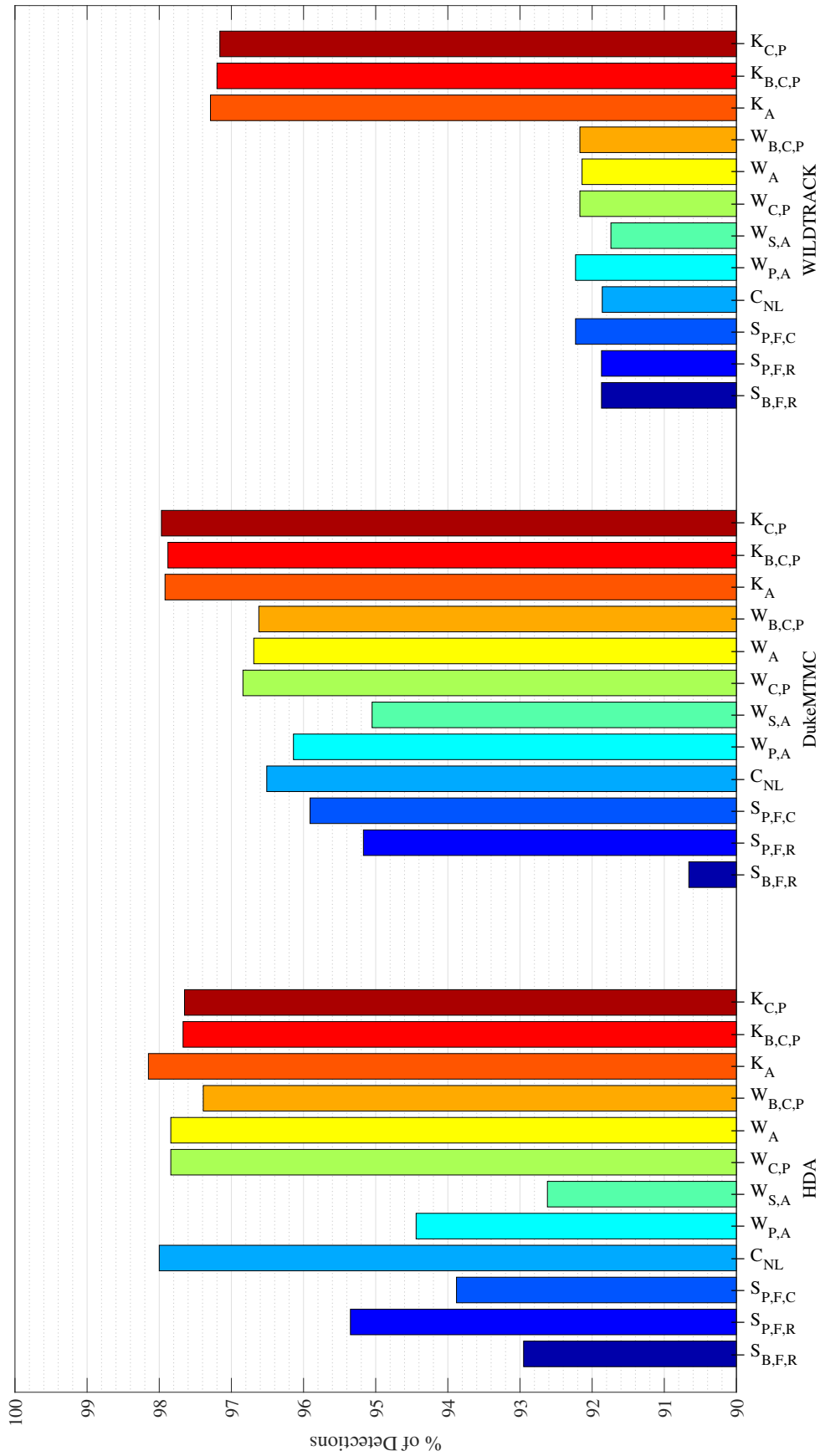
This section presents the results of the ensemble implementations alongside the results of the top performing individual prioritization schemes from Chapter 4. The parameter settings (ie. batch size) for the testbed were as previously outlined in Table 3.2.

### 5.3.1 Detection Count

The detection count for each ensemble implementation is presented in Figure 5.6. The scale of this plot (ranging from 90-100% of detections) was changed to better visualize the performance differences between the individual prioritization schemes and the ensemble methods. Figure 5.6 also includes the detection count results of the four individual schema ( $C_{NL}, S_{B,F,R}, S_{P,F,R}, S_{P,F,C}$ ) that were used to create the ensemble implementations.

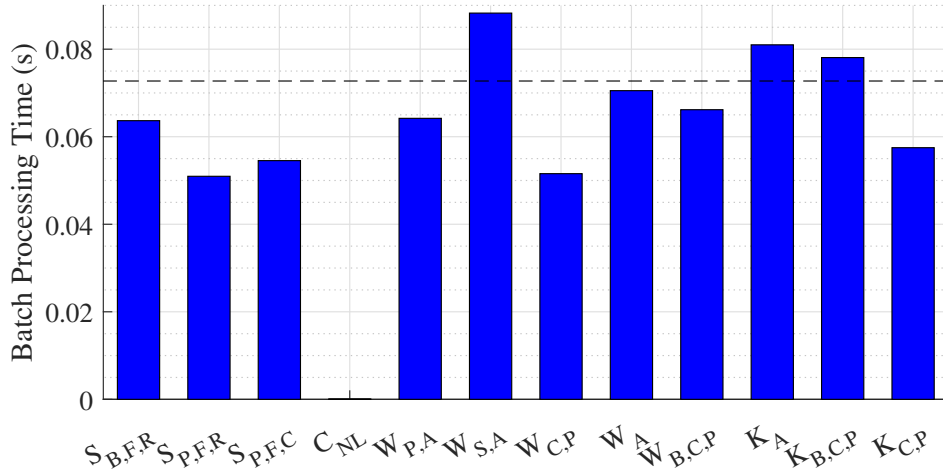
### 5.3.2 Processing Time

The average processing times are presented in Figure 5.7. The processing times for each dataset represent how long it took the Frame Selector to prioritize all of the frames in the image batch. It should be noted that  $C_{NL}$ 's process time was so small in comparison to the other times that it does not appear on the plot. The time  $T$  is included as a dotted line across the graph to show the processing time of the image batch with an object detector  $T = \frac{1}{F_p}$ .

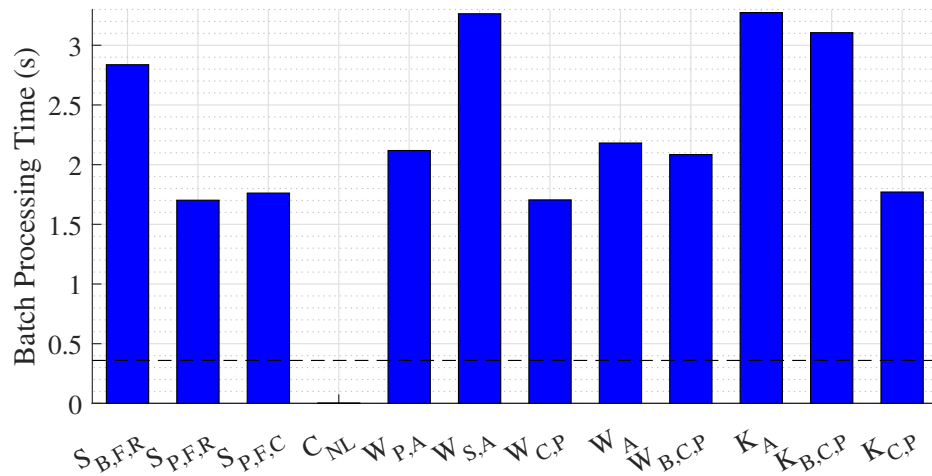


Legend:  $S_{B,F,R}$ : Blob - Delta Frame, Reference |  $S_{P,F,R}$ : Pixel Count - Delta Frame, Reference |  $S_{P,F,C}$ : Pixel Count - Reference |  $C_{NL}$ : Confidence - Non-Linear Roulette |  $W_{P,A}$ : Weighted Average ( $S_{P,F,R}, S_{P,F,C}$ ) |  $W_{S,A}$ : Weighted Average ( $S_{B,F,R}, S_{P,F,R}, S_{P,F,C}$ ) |  $W_C$ : Weighted Average ( $C_{NL}, S_{P,F,C}$ ) |  $W_A$ : Weighted Average ( $C_{NL}, S_{B,F,R}, S_{P,F,R}, S_{P,F,C}$ ) |  $W_{B,C,P}$ : Weighted Average ( $C_{NL}, S_{B,F,R}, S_{P,F,R}, S_{P,F,C}$ ) |  $K_A$ : KNN Estimator ( $C_{NL}, S_{B,F,R}, S_{P,F,C}$ ) |  $K_{B,C,P}$ : KNN Estimator ( $C_{NL}, S_{B,F,R}, S_{P,F,C}$ ) |  $K_{C,P}$ : KNN Estimator ( $C_{NL}, S_{P,F,C}$ )

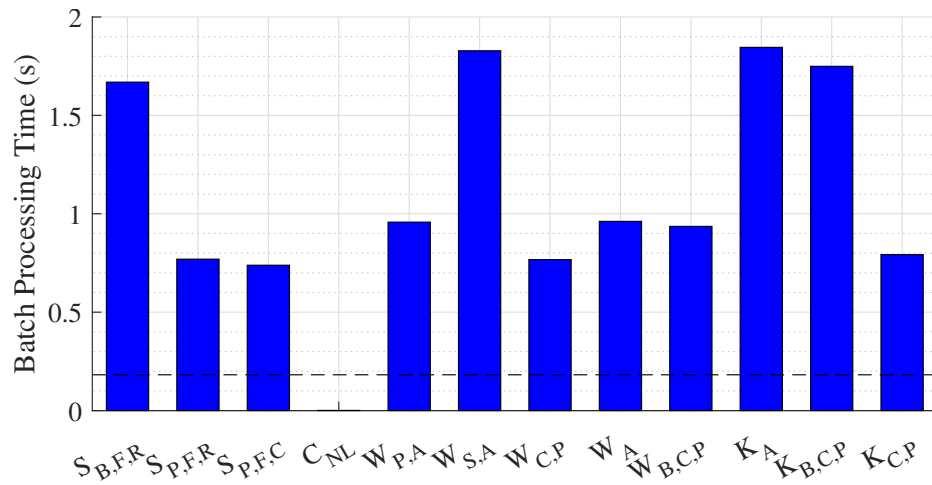
Figure 5.6: The detection count achieved for each ensemble implementation expressed as percentage of achievable detections



(a) Average Processing Time per Image Batch for the HDA Dataset



(b) Average Processing Time per Image Batch for the DukeMTMC Dataset



(c) Average Processing Time per Image Batch for the WILDTRACK Dataset

Figure 5.7: The processing times for each ensemble implementation

## 5.4 Discussion

Across all datasets, the weighted average ensemble implementations that only used spatiotemporal information ( $W_{P,A}, W_{S,A}$ ) performed relatively poorly, surpassed in detection count by most of their individual counterparts. Conversely, the DukeMTMC dataset benefited from the addition of detection confidence prioritization scheme  $C_{NL}$  to ensemble  $W_{C,P}$ ,  $W_A$ , and  $W_{B,C,P}$ . However, the detection counts for the HDA and WILDTRACK datasets remained unchanged by the additional computation of the weighted average ensemble.

The KNN regressor method yielded better results than the weighted average fusion method. Although it did not improve the detection count of the HDA dataset relative to the individual prioritization scheme, the DukeMTMC and WILDTRACK datasets benefited from this ensemble implementation. Specifically, with the KNN regressor, the WILDTRACK's detection count performance improved by 6% relative to the previously tested individual prioritization schemes presented in Chapter 4. This is in stark contrast to the previously reported result that these individual schemes could not increase the detection count above the random baseline ( $B_R$ ).

If detection count were the only performance metric, the KNN estimator method would therefore clearly be the best prioritization implementation. Its detection counts outperformed all other implementations, individual or ensemble, across all datasets. However, there are real-time constraints imposed by the processing speed of the object detector. As seen by Figure 5.7, none of the ensemble implementations were able to meet the timing requirements. As mentioned in the previous chapter, however, there is room for optimization. The next chapter will cover the process of optimizing of the KNN ensemble implementation to meet the real-time constraints of each dataset.

# Chapter 6

## Final System Validation

### 6.1 Processing Speed Optimization

As noted at the end of the last chapter, the KNN Estimator-based ensemble implementations were too slow to be used in real-time for the DukeMTMC and WILDTRACK datasets. In order for these ensemble implementations to be viable in a real-world online system, they must be optimized to reduce processing times to be less than the threshold ( $T$ ). Because of the superior performance of the KNN estimators ( $K_A, K_{B,C,P}, K_{C,P}$ ) in terms of detection counts across the datasets, the rest of the implementations were not further optimized in this work. Of these ensemble implementations,  $K_{C,P}$  had a much better trade-off between computational complexity and detection count - it only needed to process two prioritization schemes ( $C_1$  and  $S_{P,F,C}$ ) compared to  $K_A$  and  $K_{B,C,P}$  which process three and four prioritization schemes respectively. Consequently, only  $K_{C,P}$  was chosen to be optimized to meet the real-time requirements of this final system validation.

The first step of optimization is to find out which code instructions are the slowest and represent the speed bottleneck. After analyzing  $K_{C,P}$ , it was determined that the slowest section was the processing of the estimation values, specifically the  $S_{P,F,C}$  information. To optimize this section of code, Cython [63] was used to create the custom package `speedy_estimations`. Cython is a compiler that converts Python code into C code that can still be easily used within the other pre-existing Python programs. This method was chosen to ensure that the optimized code integrated easily within the testbed, which was built in Python.

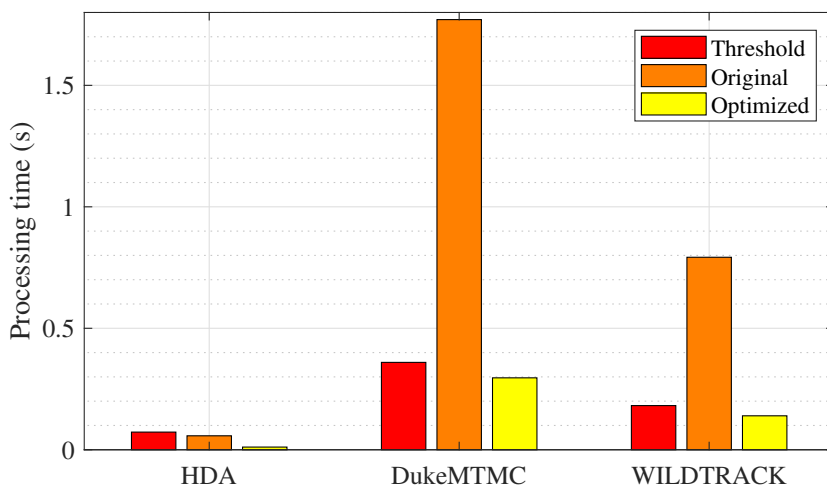


Figure 6.1: A demonstration of the reduction in processing time of the  $K_{C,P}$  ensemble implementation before and after optimization using the Cython compiler and multi-threading

The Cython implementation of `speedy_estimations` sped up the  $K_{C,P}$  implementation by a factor of 2.5 times. While an improvement, further optimization was needed in order to meet the real-time requirement. Due to the historical nature of the  $S_{P,F,C}$  prioritization metric, the frames from each camera had to be processed in sequence. However, the groupings of each camera’s frames can be processed concurrently. Thus, in order to obtain a greater speed gain, the estimation processes for each camera were threaded. After combining these two optimization steps, the performance of the system was able to surpass that of the target timing thresholds. The final timings of the optimized  $K_{C,P}$  implementation can be seen in Figure 6.1.

## 6.2 Results of Varying the Processing Batch Size

The detection results presented in previous chapters were produced using only a single processing batch size ( $N_p$ ), which translated to 70 %, 74 %, and 83 % of frames processed for the HDA, DukeMTMC, and WILDTRACK datasets, respectively (as specified in Table 3.2). To evaluate the performance of this final optimized system relative to some of the best alternatives, a comparison across batch sizes was con-

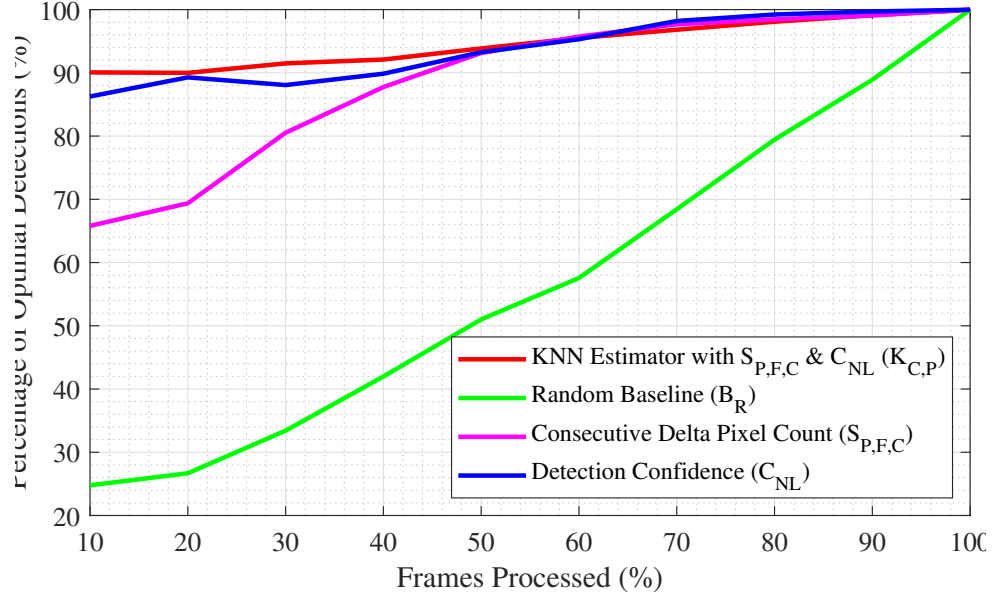


Figure 6.2: The HDA dataset’s normalized detection results with the best performing Frame Selector implementations for an increasing number of frames processed per batch

ducted. Figure 3.5 and 3.6 shows how the number of frames processed in each batch impacts the maximum achievable detection count. Figures 6.2, 6.3, and 6.4 demonstrate how results of the best performing Frame Selector implementations vary as the number of frames processed is changed. Each dataset differs, but in general, the detection count rises as the numbers of frames processed is increased due to the greater likelihood that frames with higher detection counts will be selected.

It is clear from these figures that ensemble  $K_{C,P}$  is superior in maximizing detection. While there is a clear improvement across all tested schemes as the percentage of frames is increased, the  $K_{C,P}$  ensemble scheme outperforms the others at all levels. This is especially evident in the HDA (Figure 6.2) and DukeMTMC (Figure 6.3) datasets for lower batch sizes, where the final  $K_{C,P}$  KNN Estimator-based scheme yielded more than, and almost double the detections as compared to the baseline, respectively.

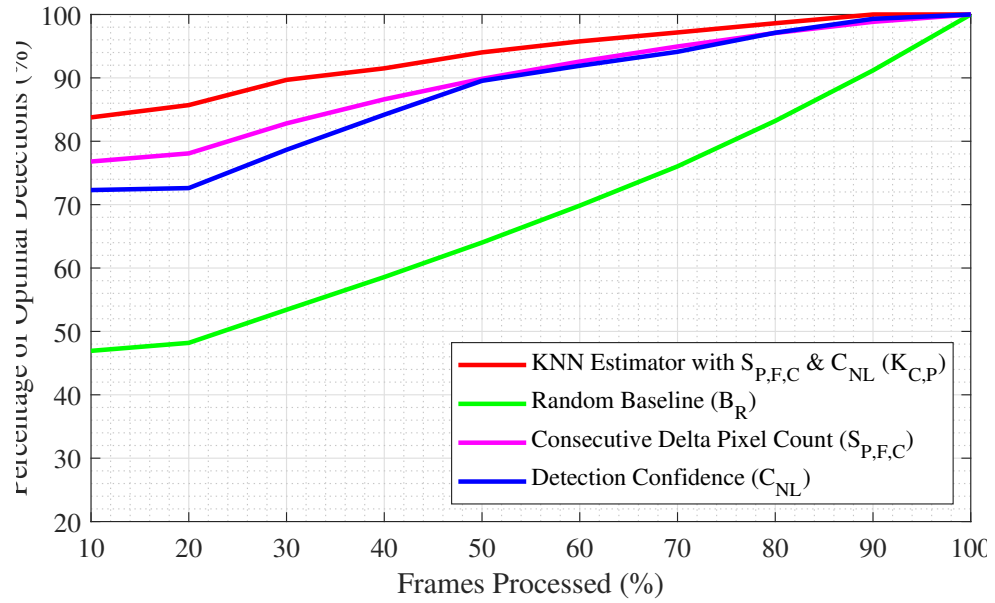


Figure 6.3: The DukeMTMC dataset’s normalized detection results with the best performing Frame Selector implementations for an increasing number of frames processed per batch

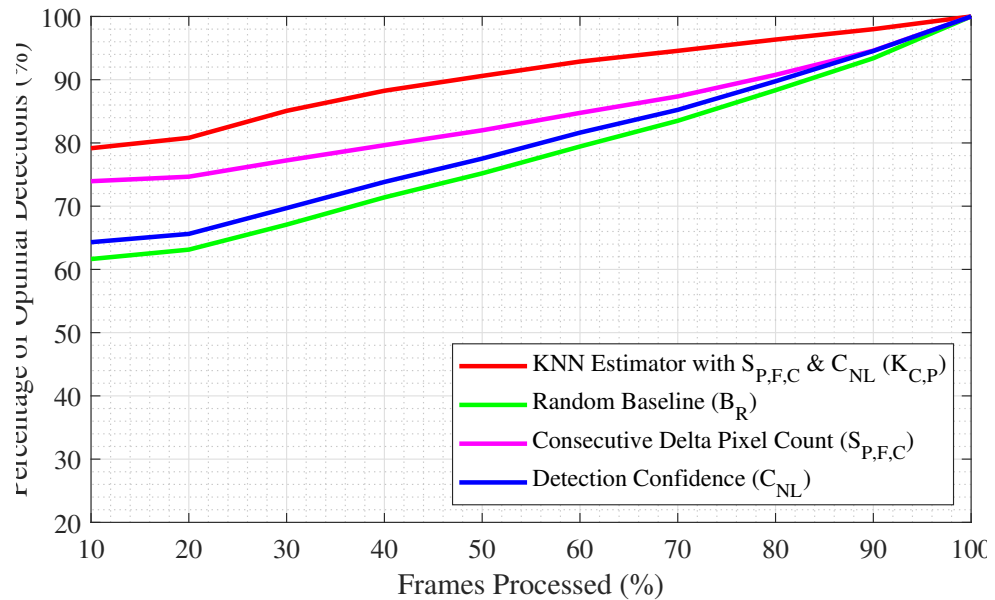


Figure 6.4: The WILDTRACK dataset’s normalized detection results with the best performing Frame Selector implementations for an increasing number of frames processed per batch

# Chapter 7

## Conclusion

### 7.1 Overview

Surveillance requires automated object detection systems to perform in real-time, and with the average security camera capturing upwards of 30 FPS, the processing requirements can result in a large cost in hardware (GPUs). The goal of prioritization is to dynamically sub-sample visual data, reducing the demand on the object detection system, ultimately reducing the cost of the automated surveillance system. Many works have explored the use of visual data prioritization with different modalities of information. However, only a small subset of works have implemented visual data prioritization in the context of a CNN-based object detection algorithm. Of the works so far, only spatiotemporal information (created by looking at the pixel value difference between two frames) has been used in this context. Furthermore, these methods are designed for analyzing visual data in retrospect (ie. forensics) and not in real-time. This dissertation has presented fusion techniques designed to run on a CPU that combine the best of the evaluated priority metrics in order to increase the number of objects detected when sub-sampling visual data in a real-time automated object detection system.

In Chapter 4, several modalities of spatiotemporal information were used to create a variety of prioritization schemes. Temporal information was extracted by creating delta frames that represented how pixel values had changed over time. Spatial information relied on the actual pixel values in the delta frame. By combining spatial methods with the temporal frame information, spatiotemporal prioritization schemes

were created to select frames of projected importance. The temporal metrics based on delta pixel counts proved to be the most efficient and effective for maximizing object detection count. Additionally within Chapter 4, detection confidence was used as another modality of prioritization information. This modality of information also proved to be effective, but only marginally better than the results of using spatiotemporal information. When the detection results of the best performing individual prioritization schemes were compared on a per camera basis against the optimal detection count (Figure 5.1), it was found that not all prioritization schemes were prioritizing the same information content in the camera frames. It was theorized that by combining prioritization schemes that under-prioritized some camera frames and over-prioritized others (when compared to the optimal case), an ensemble approach might be more effective in maximizing object detection count. Based on this theory, Chapter 5 presented ensemble implementations that were created from the best performing individual prioritization schemes evaluated in Chapter 4.

Two different methods were used to fuse the different information modalities in the ensemble implementations: a weighted average and a KNN regressor. The weights for the weighted average were determined by solving an over-specified system of equations created with the normalized results of the calculated prioritization schemes and optimal priority values. The results of the weighted average ensemble were found to be only marginally better than that of the individual prioritization schemes. Based on these results it was speculated that using static weights might lead to sub-optimal results due to the dynamic nature of surveillance data and the variability in the FOV between cameras. It was therefore decided to implement a KNN regressor on a per camera basis in order to address these challenges.

Due to the adaptive nature of the KNN regressor, the resultant implementation demonstrated greater detection count performance. Additionally, it proved to handle difficult surveillance scenarios, such as the detection dense WILDTRACK dataset,

more effectively than the individual and weighted average ensemble implementations.

Although the KNN Estimator implementations were effective at maximizing the object detection count, their processing time initially did not meet the real-time requirements associated with each dataset. In Chapter 6 the ensemble method was optimized to meet these timing requirements using multi-threading and by re-writing the functions in Cython. The optimized implementation ran five times faster than the original, making it a viable option for real-time use.

Chapter 6 also presented the results of varying the processing batch size with multiple implementations and its impact on detection count performance. The developed prioritization scheme was shown to greatly outperform the baseline random frame selection, but was especially advantageous when the batch size was decreased. Across all datasets the KNN ensemble implementation proved to be the best at selecting the frames that maximized the number of objects detected.

## 7.2 Contribution

Modern surveillance cameras provide a constant stream of visual data that may contain objects of security importance. Each camera produces upwards of 30 high definition image frames every second. Automated object detection systems have been created in order to ease the burden of analyzing the incredible amount of visual data that is created. However, the cost associated with processing all of this data is very large, often resulting in the static sub-sampling of video data. While static sub-sampling the camera streams lowers the number of frames, it disproportionately decreases the number of objects found by the automated object detection system. As an example, for an indoor surveillance scenario (HDA Dataset), it was demonstrated that sampling only 10% of frames lead to a decrease of 80% of the number of possible detection (as shown in Figure 6.2).

This thesis has presented and contrasted several methods of visual data priori-

zation that can efficiently and effectively sub-sample surveillance data to maximize the object detection count. While other modern works in the field have focused on forensically analyzing surveillance data (post-processing data at a later time), this work focused on creating prioritization schemes that could be employed in real-time applications, such as automated security systems. Each frame selector implementation was built, timed and tested in a custom-built test framework on a total of three publicly available datasets, each with a different surveillance scenario and associated challenges.

Individual prioritization schemes were created to assign a priority value to each camera frame so that the frame selector could choose the highest ranked frames and pass them along to the object detector. Previous works on visual data prioritization have mainly used spatiotemporal data, and so several of the prioritization schemes leverage spatial and temporal information. Additionally, a novel approach that leveraged the historical detection confidences created by the CNN-based object detector was also implemented here for the first time. The two prioritization schemes that performed the best in terms of computational time and detection count were the consecutive delta pixel count, and the proposed detection confidence method.

Another important contribution of this thesis was the use of prioritization techniques that were created by fusing the best performing individual prioritization schemes. The best ensemble implementation used a KNN regressor to dynamically fuse the prioritization schemes together. The resultant ensemble implementation demonstrated a performance increase of up to 60% when compared to the baseline static sub-sampling methods. Additionally, the ensemble proved to handle crowded surveillance scenarios better than any individual prioritization scheme and a weighted average ensemble, increasing the number of detection counts from 90% to 97%. Finally, the KNN Estimator ensemble implementation for frame selection was optimized to run within the real-time constraints of the automated object detection

system, making it a viable option for real-world applications.

This work was designed, from the ground up, to be readily adopted by industry. The use of Python and popular libraries such as OpenCV and Caffe2 made this work possible and ensure that implementation of the prioritization schemes presented in this work can be translated for their use. Modern and emerging object detection systems can benefit greatly from the use of the Frame Selectors explored here, reducing cost while ensuring that fewer objects of interest are missed in real-world surveillance systems.

### 7.3 Limitations

A limitation of this work is the lack of public multi-camera surveillance datasets available to be used. While three unique datasets, which had a breadth of varying characteristics, were used to test the Frame Selector, they each represented only 25 minutes of a particular camera configuration each. Despite their differences and associated challenges, each can only provide limited insight into the robustness of frame selection.

When using detection confidence as a prioritization scheme, the accuracy of the object detection algorithm used to provide detection confidences limits the ability of the Frame Selector to correctly prioritize the incoming visual data. If a false positive or negative is introduced to the system by the inaccuracy of the object detector, the Frame Selector will incorrectly prioritize the data. While today's object detectors are powerful and improving, this does represent a possible weakness in such a system.

While this work investigated all of the individual computational pieces of an automated object detection system, there is further work that can be done to verify the real-time nature of a complete system. The threshold time  $T$  did not reflect all of the aspects of the system. Video decoding time and inter-process communication are two examples that could impact a fully developed and scalable automated object

detection system. Furthermore, the code created in this thesis was written primarily in Python. Using a compiled language such as C++ could significantly reduce computation time and improve the real-time performance.

This thesis validated the performance of the object detector by counting the number of occurrences of pre-selected objects of interest ('person', 'backpack', 'umbrella', 'handbag', 'suitcase', and 'cell phone'). The vast majority of detected objects are of the 'person' class. Further work could investigate the ability to prioritize small objects of interest. Furthermore, this thesis used a frame by frame detection rate for validation. By tracking what objects enter and leave the surveillance scene (leveraging temporal information), it is possible that the objects in individual frames that were undetected by the prioritization schemes may still be identified. This context could provide further insight into the detection likelihoods of the prioritization schemes for a given object size and time in a camera's FOV.

## 7.4 Recommendations for Future Work

For many prioritization schemes, delta frames that used an empty reference frame were used to denote the temporal changes in the camera's FOV. However, lighting changes or large objects that remain in the FOV may incorrectly bias the camera frames over time. In order to account for changes in the FOV, dynamic reference frames should be used. Dynamic reference frames will provide a more accurate representation of the important changes in a camera's FOV.

This work explores the use of two different modalities of prioritization information: detection confidence and spatiotemporal. Future work could be done to identify additional modalities of information that improve detection count when combined with the other methods that were proven effective in this work. Further research could also evaluate other data fusion techniques such as neural networks or Kalman filters.

Tracking objects is an important aspect in surveillance and could be incorporated into prioritization to further improve performance. Additionally, by tracking which objects are detected over time, objects that are missed in a particular frame may still be identified. This would provide further insight into how prioritization impacts missed detections and would further reduce the system's false positives.

# Bibliography

- [1] N. G. Kiewiet, Y. Roodt, H. Roos, and W. A. Clarke, “Automated surveillance and detection of foreign stationary objects,” in *AFRICON, 2011*, Sep. 2011, pp. 1–6.
- [2] U. Ramachandran, K. Hong, L. Iftode, R. Jain, R. Kumar, K. Rothemel, J. Shin, and R. Sivakumar, “Large-Scale Situation Awareness With Camera Networks and Multimodal Sensing,” *Proceedings of the IEEE*, vol. 100, no. 4, pp. 878–892, Apr. 2012. [Online]. Available: <http://ieeexplore.ieee.org/document/6155584/>
- [3] X. Wang, “Intelligent multi-camera video surveillance: A review,” *Pattern Recognition Letters*, vol. 34, no. 1, pp. 3–19, Jan. 2013. [Online]. Available: <http://linkinghub.elsevier.com/retrieve/pii/S016786551200219X>
- [4] Robert Draper, “They Are Watching You and Everything Else on the Planet,” *National Geographic*, no. February 2018, Feb. 2018. [Online]. Available: <https://www.nationalgeographic.com/magazine/2018/02/surveillance-watching-you/>
- [5] T. Williams, “Facial Recognition Software Moves From Overseas Wars to Local Police,” *The New York Times*, Dec. 2017. [Online]. Available: <https://www.nytimes.com/2015/08/13/us/facial-recognition-software-moves-from-overseas-wars-to-local-police.html>
- [6] A. Aradhana, “Singapore to test facial recognition on lampposts, stoking privacy...” *Reuters*, Apr. 2018. [Online]. Available: <https://www.reuters.com/article/us-singapore-surveillance/singapore-to-test-facial-recognition-on-lampposts-stoking-privacy-fears-idUSKBN1HK0RV>

- [7] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation,” in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2014, pp. 580–587.
- [8] V. Sze, Y. H. Chen, T. J. Yang, and J. S. Emer, “Efficient Processing of Deep Neural Networks: A Tutorial and Survey,” *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.
- [9] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017.
- [10] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “SSD: Single Shot MultiBox Detector,” *arXiv:1512.02325 [cs]*, vol. 9905, pp. 21–37, 2016.
- [11] T. Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollr, “Focal Loss for Dense Object Detection,” in *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017, pp. 2999–3007.
- [12] J. Redmon and A. Farhadi, “YOLO9000: Better, Faster, Stronger,” *arXiv:1612.08242 [cs]*, Dec. 2016.
- [13] —, “YOLOv3: An Incremental Improvement,” *arXiv:1804.02767 [cs]*, Apr. 2018.
- [14] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia, “NoScope: Optimizing Neural Network Queries over Video at Scale,” *Proc. VLDB Endow.*, vol. 10, no. 11, pp. 1586–1597, Aug. 2017.
- [15] T. Zhang, A. Chowdhery, P. V. Bahl, K. Jamieson, and S. Banerjee, “The Design and Implementation of a Wireless Video Surveillance System,” in *Pro-*

- ceedings of the 21st Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '15. New York, NY, USA: ACM, 2015, pp. 426–438.
- [16] I. Mehmood, M. Sajjad, W. Ejaz, and S. W. Baik, “Saliency-directed prioritization of visual data in wireless surveillance networks,” *Information Fusion*, vol. 24, pp. 16–30, Jul. 2015.
- [17] K. Irgan, C. nsalan, and S. Baydere, “Low-cost prioritization of image blocks in wireless sensor networks for border surveillance,” *Journal of Network and Computer Applications*, vol. 38, pp. 54–64, Feb. 2014.
- [18] S. Nasir, C. T. E. R. Hewage, M. Mrak, S. Worrall, and A. M. Kondo, “Depth based object prioritisation for 3d video communication over Wireless LAN,” in *2009 16th IEEE International Conference on Image Processing (ICIP)*, Nov. 2009, pp. 4269–4272.
- [19] J. T. B. Fajardo, K. Yasumoto, and M. Ito, “Content-based data prioritization for fast disaster images collection in delay tolerant network,” in *2014 Seventh International Conference on Mobile Computing and Ubiquitous Networking (ICMU)*, Jan. 2014, pp. 147–152.
- [20] R. Yu, H. Wang, and L. S. Davis, “ReMotENet: Efficient Relevant Motion Event Detection for Large-Scale Home Surveillance Videos,” in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, Mar. 2018, pp. 1642–1651.
- [21] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, “Overview of the H.264/AVC video coding standard,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, Jul. 2003.
- [22] A. Andreopoulos and J. K. Tsotsos, “50 Years of object recognition: Directions forward,” *Computer Vision and Image Understanding*, vol. 117, no. 8,

- pp. 827–891, Aug. 2013. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S107731421300091X>
- [23] M. Shah, O. Javed, and K. Shafique, “Automated Visual Surveillance in Realistic Scenarios,” *IEEE MultiMedia*, vol. 14, no. 1, pp. 30–39, Jan. 2007.
- [24] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.
- [25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: <http://www.deeplearningbook.org/>
- [26] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, “Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017, pp. 3296–3297.
- [27] R. Guerra, E. Martel, J. Khan, S. Lopez, P. Athanas, and R. Sarmiento, “On the Evaluation of Different High-Performance Computing Platforms for Hyperspectral Imaging: An OpenCL-Based Approach,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 10, no. 11, pp. 4879–4897, Nov. 2017.
- [28] P. Gysel, J. Pimentel, M. Motamedi, and S. Ghiasi, “Ristretto: A Framework for Empirical Study of Resource-Efficient Inference in Convolutional Neural Networks,” *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–6, 2018.
- [29] K. He, G. Gkioxari, P. Dollr, and R. Girshick, “Mask R-CNN,” *arXiv:1703.06870 [cs]*, Mar. 2017. [Online]. Available: <http://arxiv.org/abs/1703.06870>

- [30] A. Sobti, C. Arora, and M. Balakrishnan, "Object Detection in Real-Time Systems: Going Beyond Precision," in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, Mar. 2018, pp. 1020–1028.
- [31] J. H. Ko, B. A. Mudassar, and S. Mukhopadhyay, "An Energy-Efficient Wireless Video Sensor Node for Moving Object Surveillance," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 1, no. 1, pp. 7–18, Jan. 2015.
- [32] Y. Wang, D. Wang, X. Zhang, J. Chen, and Y. Li, "Energy-Efficient Image Compressive Transmission for Wireless Camera Networks," *IEEE Sensors Journal*, vol. 16, no. 10, pp. 3875–3886, May 2016.
- [33] B. A. Mudassar, "Design and implementation of a content aware image processing module on FPGA," Thesis, Georgia Institute of Technology, Apr. 2015. [Online]. Available: [www.smartech.gatech.edu/handle/1853/53618](http://www.smartech.gatech.edu/handle/1853/53618)
- [34] D. Costa, L. Guedes, F. Vasques, and P. Portugal, "Research Trends in Wireless Visual Sensor Networks When Exploiting Prioritization," *Sensors*, vol. 15, no. 1, pp. 1760–1784, Jan. 2015. [Online]. Available: <http://www.mdpi.com/1424-8220/15/1/1760>
- [35] J. Ahmad, I. Mehmood, and S. W. Baik, "Efficient object-based surveillance image search using spatial pooling of convolutional features," *Journal of Visual Communication and Image Representation*, vol. 45, pp. 62–76, May 2017. [Online]. Available: <http://linkinghub.elsevier.com/retrieve/pii/S1047320317300470>
- [36] F. Daniyal, M. Taj, and A. Cavallaro, "Content and task-based view selection from multiple video streams," *Multimedia Tools and Applications*, vol. 46, no. 2-3, pp. 235–258, Jan. 2010. [Online]. Available: <http://link.springer.com/10.1007/s11042-009-0355-z>

- [37] M. Y. S. Uddin, H. Wang, F. Saremi, G.-J. Qi, T. Abdelzaher, and T. Huang, “PhotoNet: A Similarity-Aware Picture Delivery Service for Situation Awareness.” IEEE, Nov. 2011, pp. 317–326. [Online]. Available: <http://ieeexplore.ieee.org/document/6121449/>
- [38] Q. Fan, P. Gabbur, and S. Pankanti, “Relative Attributes for Large-Scale Abandoned Object Detection,” in *2013 IEEE International Conference on Computer Vision*. Sydney, Australia: IEEE, Dec. 2013, pp. 2736–2743. [Online]. Available: <http://ieeexplore.ieee.org/document/6751451/>
- [39] D. Figueira, M. Taiana, A. Nambiar, J. Nascimento, and A. Bernardino, “The HDA+ Data Set for Research on Fully Automated Re-identification Systems,” in *Computer Vision - ECCV 2014 Workshops*, ser. Lecture Notes in Computer Science. Springer, Cham, Sep. 2014, pp. 241–255. [Online]. Available: [https://link.springer.com/chapter/10.1007/978-3-319-16199-0\\_17](https://link.springer.com/chapter/10.1007/978-3-319-16199-0_17)
- [40] E. Ristani, F. Solera, R. Zou, R. Cucchiara, and C. Tomasi, “Performance Measures and a Data Set for Multi-Target, Multi-Camera Tracking,” in *European Conference on Computer Vision workshop on Benchmarking Multi-Target Tracking*, 2016.
- [41] T. Chavdarova, P. Baqu, S. Bouquet, A. Maksai, C. Jose, T. Bagautdinov, L. Lettry, P. Fua, L. Van Gool, and F. Fleuret, “WILDTRACK: A Multi-camera HD Dataset for Dense Unscripted Pedestrian Detection,” in *Proceedings of the IEEE international conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [42] G. Bradski, “The OpenCV Library,” *Dr. Dobb’s Journal of Software Tools*, 2000.

- [43] Facebook Research, “Caffe2.” [Online]. Available: <https://research.fb.com/downloads/caffe2>
- [44] —, “FAIR’s research platform for object detection research, implementing popular algorithms like Mask R-CNN and RetinaNet.: facebookresearch/Detectron,” Oct. 2018. [Online]. Available: <https://github.com/facebookresearch/Detectron>
- [45] —, “Caffe2 and PyTorch join forces to create a Research + Production platform PyTorch 1.0,” May 2018. [Online]. Available: [http://caffe2.ai/blog/2018/05/02/Caffe2\\_PyTorch\\_1.0.html](http://caffe2.ai/blog/2018/05/02/Caffe2_PyTorch_1.0.html)
- [46] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollr, “Microsoft COCO: Common Objects in Context,” *arXiv:1405.0312 [cs]*, May 2014.
- [47] Nvidia Corporation, “Nvidia Tesla P100 PCIe Datasheet,” 2016.
- [48] D. T. Ahmed and M. A. Hossain, “Dynamic prioritization of multi-sensor feeds for resource limited surveillance systems,” in *2012 IEEE International Instrumentation and Measurement Technology Conference Proceedings*, May 2012, pp. 412–416.
- [49] J. C. Neves and H. Proenca, “Dynamic camera scheduling for visual surveillance in crowded scenes using Markov random fields.” IEEE, Aug. 2015, pp. 1–6. [Online]. Available: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7301790>
- [50] K. Lin, S. Chen, C. Chen, D. Lin, and Y. Hung, “Abandoned Object Detection via Temporal Consistency Modeling and Back-Tracing Verification for Visual Surveillance,” *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 7, pp. 1359–1370, Jul. 2015.

- [51] T. Bouwmans, L. Maddalena, and A. Petrosino, “Scene background initialization: A taxonomy,” *Pattern Recognition Letters*, vol. 96, pp. 3–11, Sep. 2017. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0167865516303798>
- [52] R. Gonzalez, *Digital Image Processing*, 3rd ed. Upper Saddle River, New Jersey: Pearson Prentice Hall, 2008.
- [53] K. Muhammad, T. Hussain, and S. W. Baik, “Efficient CNN based summarization of surveillance videos for resource-constrained devices,” *Pattern Recognition Letters*, Aug. 2018. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0167865518303842>
- [54] K. Irgan, C. Unsalan, and S. Baydere, “Priority Encoding of Image Data in Wireless Multimedia Sensor Networks for Border Surveillance,” in *Computer and Information Sciences*. Dordrecht: Springer Netherlands, 2011, vol. 62, pp. 191–194.
- [55] L. Akoglu, H. Tong, and D. Koutra, “Graph based anomaly detection and description: a survey,” *Data Mining and Knowledge Discovery*, vol. 29, no. 3, pp. 626–688, May 2015. [Online]. Available: <https://doi.org/10.1007/s10618-014-0365-y>
- [56] R. O. Duda and P. E. Hart, “Use of the Hough Transformation to Detect Lines and Curves in Pictures,” *Commun. ACM*, vol. 15, no. 1, pp. 11–15, Jan. 1972. [Online]. Available: <http://doi.acm.org/10.1145/361237.361242>
- [57] J. Canny, “A Computational Approach to Edge Detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, Nov. 1986.

- [58] X. Hou and L. Zhang, "Saliency Detection: A Spectral Residual Approach," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2007, pp. 1–8.
- [59] D. L. Hall and J. Llinas, "An introduction to multisensor data fusion," *Proceedings of the IEEE*, vol. 85, no. 1, pp. 6–23, Jan. 1997.
- [60] B. Khaleghi, A. Khamis, F. O. Karray, and S. N. Razavi, "Multisensor data fusion: A review of the state-of-the-art," *Information Fusion*, vol. 14, no. 1, pp. 28–44, Jan. 2013. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S1566253511000558>
- [61] N. S. Altman, "An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, Aug. 1992. [Online]. Available: <http://www.tandfonline.com/doi/abs/10.1080/00031305.1992.10475879>
- [62] 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*, vol. 12, pp. 2825–2830, 2011.
- [63] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. Seljebotn, and K. Smith, "Cython: The Best of Both Worlds," *Computing in Science Engineering*, vol. 13, no. 2, pp. 31–39, Apr. 2011.

# Appendix A

## Definitions

**Definition 1 (CNN).** *Convolutional Neural Networks (CNN) are a special subset of artificial neural networks which have proven very effective in the image processing space. Neurons, otherwise known as activations, are the basic building block of the neural net. Neurons are organized into layers. Any network with more than three layers is known as a deep neural network (DNN). The number of layers in a modern neural nets range from five to a thousand layers. The combination of layers, design of the activation functions, and number of dendrites change depending on the neural nets architecture.*

**Definition 2 (FPGA).** *A Field Programmable Gate Array (FPGA) is an array of configurable blocks and programmable logic that are used in many applications, ranging from communications to control. The FPGA marks a definitive change in the spectrum, where the hardware defines the functionality of the device, rather than the software. However, the defined functionality of the FPGA can be changed by using software to reconfigure the hardware. This software is know as a Hardware Description Language (HDL).*

**Definition 3 (FPS).** *Frames per second is a measure of speed often used to quantifies frame-rates. FPS is interchangeable with a unit of Hertz (Hz).*

**Definition 4 (FOV).** *The Field of View (FOV) is defined to be what can be seen by the camera.*

**Definition 5 (GPU).** *A Graphics Processing Unit (GPU) or General Purpose Graphics Processing Unit (GPGPU) was originally designed to handle processing for com-*

*puter graphics programs. A GPU is typically composed of multiple (in the order of 1000s) identical processors, broken into multiple multiprocessors. GPUs are designed for data parallelism and high throughput.*

# Appendix B

## Dataset Information

### B.1 HDA Dataset

#### B.1.1 Camera Topology

The HDA Dataset contains footage from 12 cameras spread out over three floors of the ISR-Lisbon building. Figure B.1, B.2, and B.3 show the camera locations and associated number in red. Camera 50 was included in the original dataset but only contained 15 minutes of footage compared to 25 minutes for the rest of the cameras; thus it was excluded from the data used.

#### B.1.2 Camera Information

The following Table B.1 outlines the camera information for the 12 camera streams originally from the HDA Dataset.

Table B.1: HDA Dataset Camera Information

<b>Camera Number</b>	<b>Frame Rate (FPS)</b>	<b>Resolution</b>
02	5.0	640x480
17	5.0	640x480
18	5.0	640x480
19	5.0	640x480
40	5.0	640x480
53	2.0	1280x800
54	2.0	1280x800
56	2.0	1280x800
57	2.0	1280x800
58	2.0	1280x800
59	2.0	1280x800
60	1.0	2560x1600

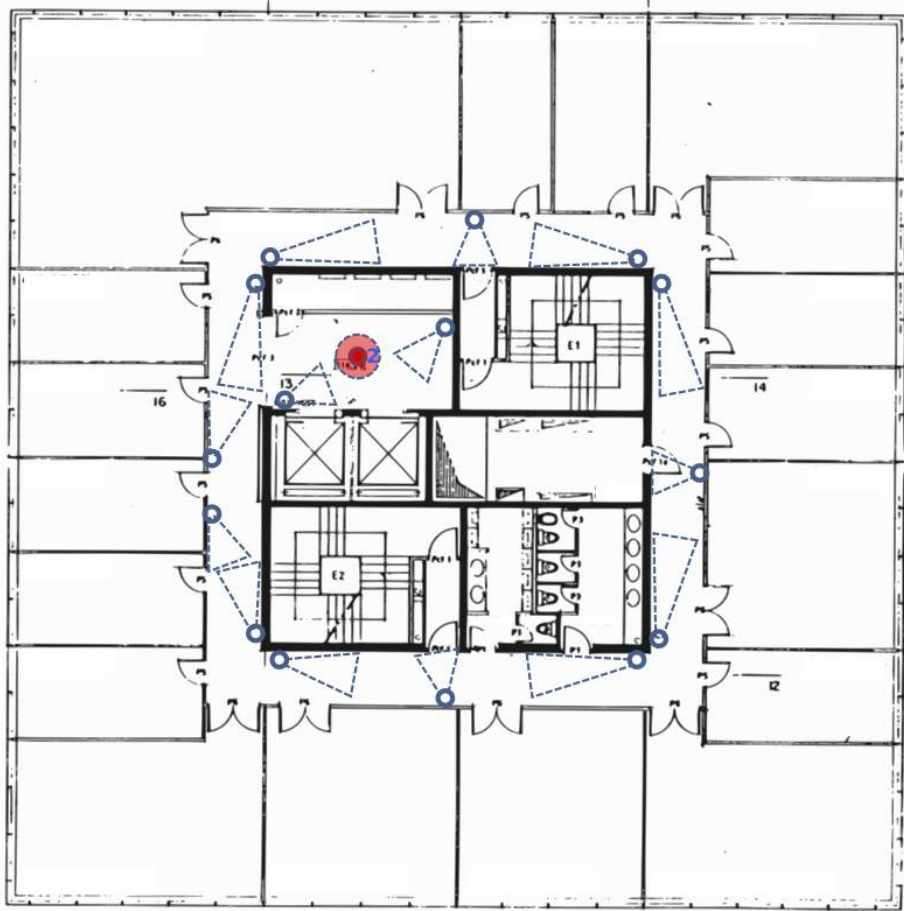


Figure B.1: Floor plan of Floor 6 at ISR-Lisbon [39]

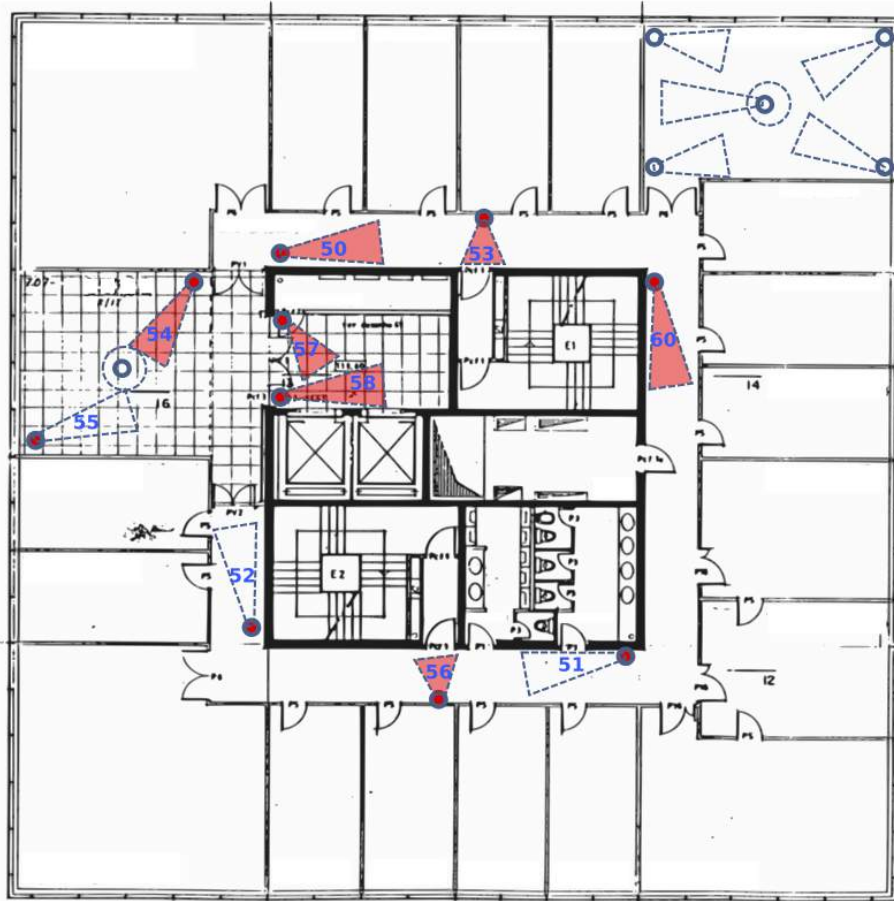


Figure B.2: Floor plan of Floor 7 at ISR-Lisbon [39]



Figure B.3: Floor plan of Floor 8 at ISR-Lisbon [39]

Table B.2: DukeMTMC Camera Information

Camera Number	Frame Rate (FPS)	Resolution
1	59.94	1080p
2	59.94	1080p
3	59.94	1080p
4	59.94	1080p
5	59.94	1080p
6	59.94	1080p
7	59.94	1080p
8	59.94	1080p

### B.1.3 Reference Frames

For some of the prioritization schema a reference frame is needed to compare the current frame against. This section contains the reference frames used for the 12 camera streams of the HDA dataset.

## B.2 DukeMTMC Dataset

### B.2.1 Camera Topology

The DukeMTMC Dataset contains footage from cameras setup in the surrounding vicinity to Duke University’s chapel. Figure B.5 show the camera locations and associated number.

### B.2.2 Camera Information

Table B.2 outlines the camera information for the 8 camera streams originally from the DukeMTMC Dataset.

### B.2.3 Reference Frames

For some of the prioritization schema a reference frame is needed to compare the current frame against. This section contains the reference frames used for the 8 camera streams of the Duke MTMC dataset.



(a) Reference Frame used for Cam 2 of the HDA Dataset



(b) Reference Frame used for Cam 17 of the HDA Dataset



(c) Reference Frame used for Cam 18 of the HDA Dataset



(d) Reference Frame used for Cam 19 of the HDA Dataset



(e) Reference Frame used for Cam 40 of the HDA Dataset



(f) Reference Frame used for Cam 53 of the HDA Dataset



(g) Reference Frame used for Cam 54 of the HDA Dataset



(h) Reference Frame used for Cam 56 of the HDA Dataset



(i) Reference Frame used for Cam 57 of the HDA Dataset



(j) Reference Frame used for Cam 58 of the HDA Dataset



(k) Reference Frame used for Cam 59 of the HDA Dataset



(l) Reference Frame used for Cam 60 of the HDA Dataset

Figure B.4: Reference Frames used for the HDA Dataset

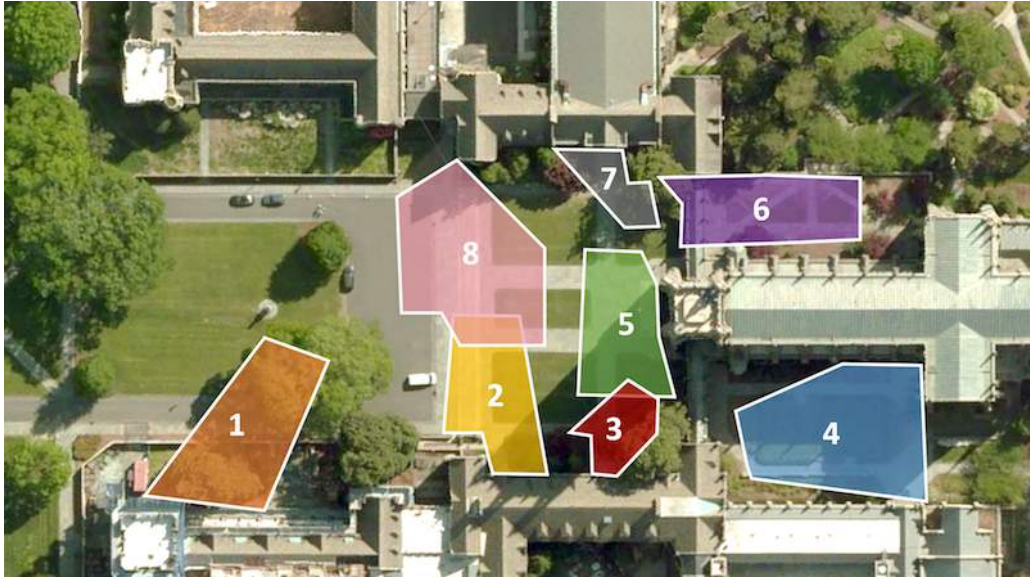


Figure B.5: Camera Topology of the Duke MTMC dataset [40]

Table B.3: WILDTRACK Camera Information

Camera Number	Frame Rate (FPS)	Resolution
1	60	1080p
2	60	1080p
3	60	1080p
4	60	1080p
5	60	1080p
6	60	1080p
7	60	1080p

## B.3 WILDTRACK Dataset

### B.3.1 Camera Topology

The WILDTRACK Dataset contains footage from cameras setup facing a busy courtyard located in Zürich, Switzerland. Figure B.5 show the camera locations and associated number.

### B.3.2 Camera Information

The following Table B.3 outlines the camera information for the 7 camera streams originally from the WILDTRACK Dataset.



(a) Reference Frame used for Cam 1 of the DukeMTMC Dataset



(b) Reference Frame used for Cam 2 of the DukeMTMC Dataset



(c) Reference Frame used for Cam 3 of the DukeMTMC Dataset



(d) Reference Frame used for Cam 4 of the DukeMTMC Dataset



(e) Reference Frame used for Cam 5 of the DukeMTMC Dataset



(f) Reference Frame used for Cam 6 of the DukeMTMC Dataset



(g) Reference Frame used for Cam 7 of the DukeMTMC Dataset



(h) Reference Frame used for Cam 8 of the DukeMTMC Dataset

Figure B.6: Reference Frames used for the DukeMTMC Dataset

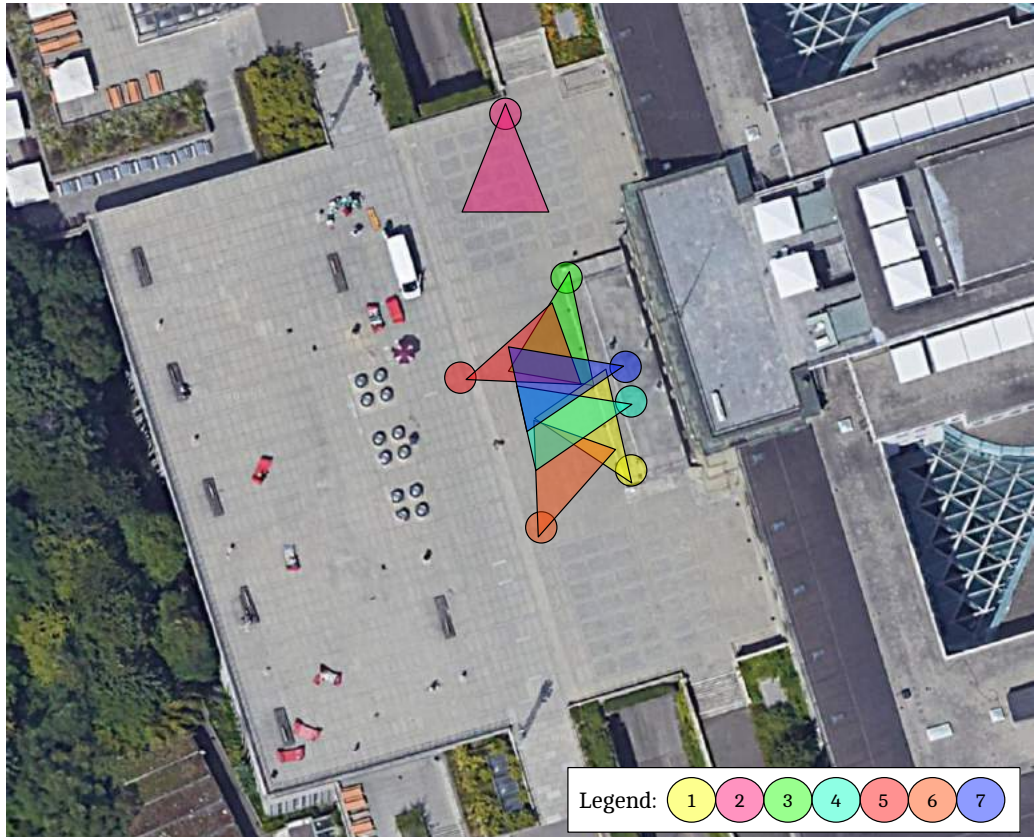


Figure B.7: Camera Topology of the WILDTRACK dataset [41]

### B.3.3 Reference Frames

For some of the prioritization schema a reference frame is needed to compare the current frame against. This section contains the reference frames used for the 7 camera streams of the WILDTRACK dataset.



(a) Reference Frame used for Cam 1 of the WILDTRACK Dataset



(b) Reference Frame used for Cam 2 of the WILDTRACK Dataset



(c) Reference Frame used for Cam 3 of the WILDTRACK Dataset



(d) Reference Frame used for Cam 4 of the WILDTRACK Dataset



(e) Reference Frame used for Cam 5 of the WILDTRACK Dataset



(f) Reference Frame used for Cam 6 of the WILDTRACK Dataset



(g) Reference Frame used for Cam 7 of the WILDTRACK Dataset

Figure B.8: Reference Frames used for the WILDTRACK Dataset

# Appendix C

## Additional Tables and Plots of Complete Results

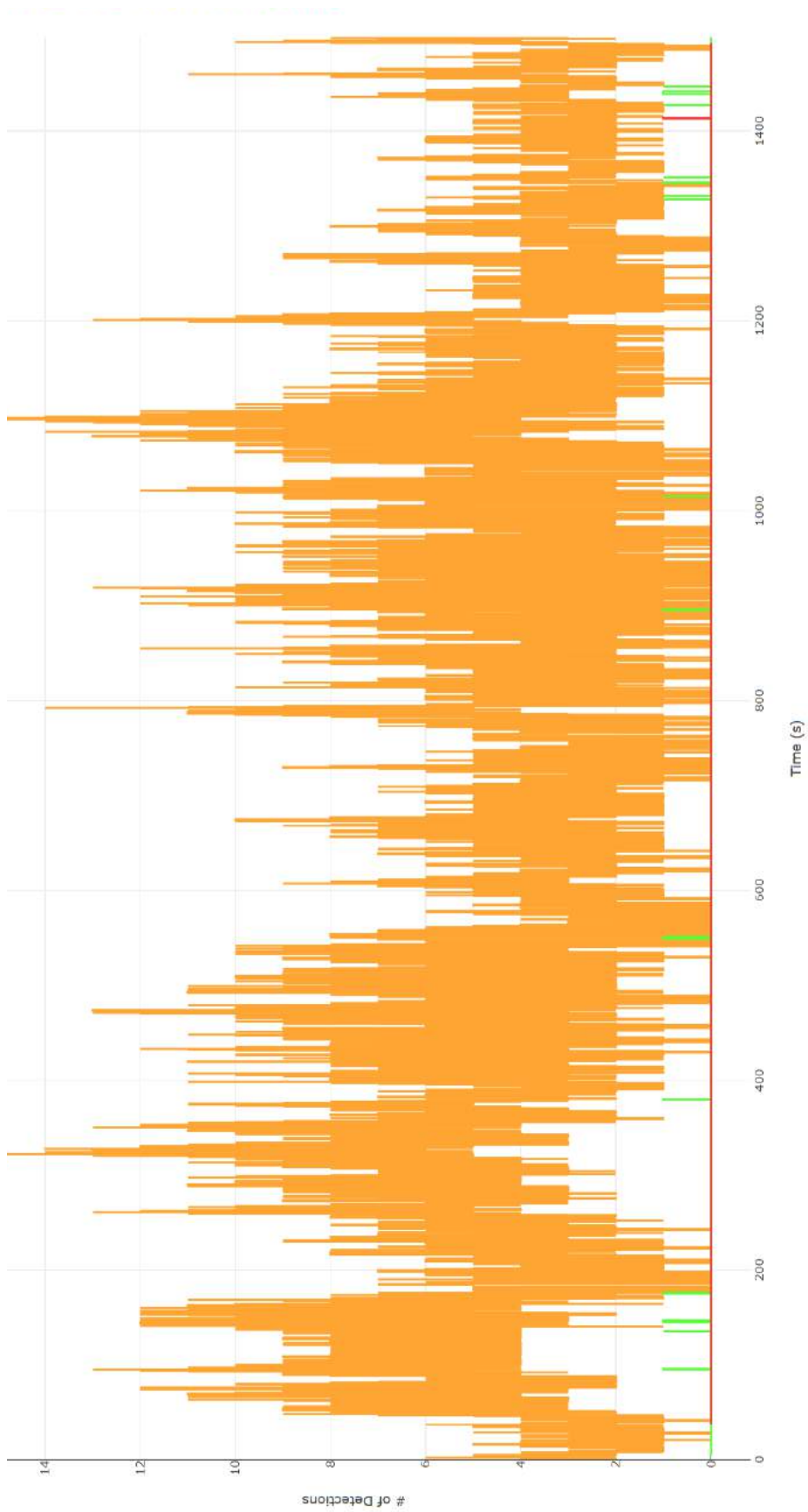
### C.1 Dataset Detection Information

#### C.1.1 Class Detection Count Line Plots

Figures C.1, C.2, and C.3 show the number of objects detected over time in a dataset.

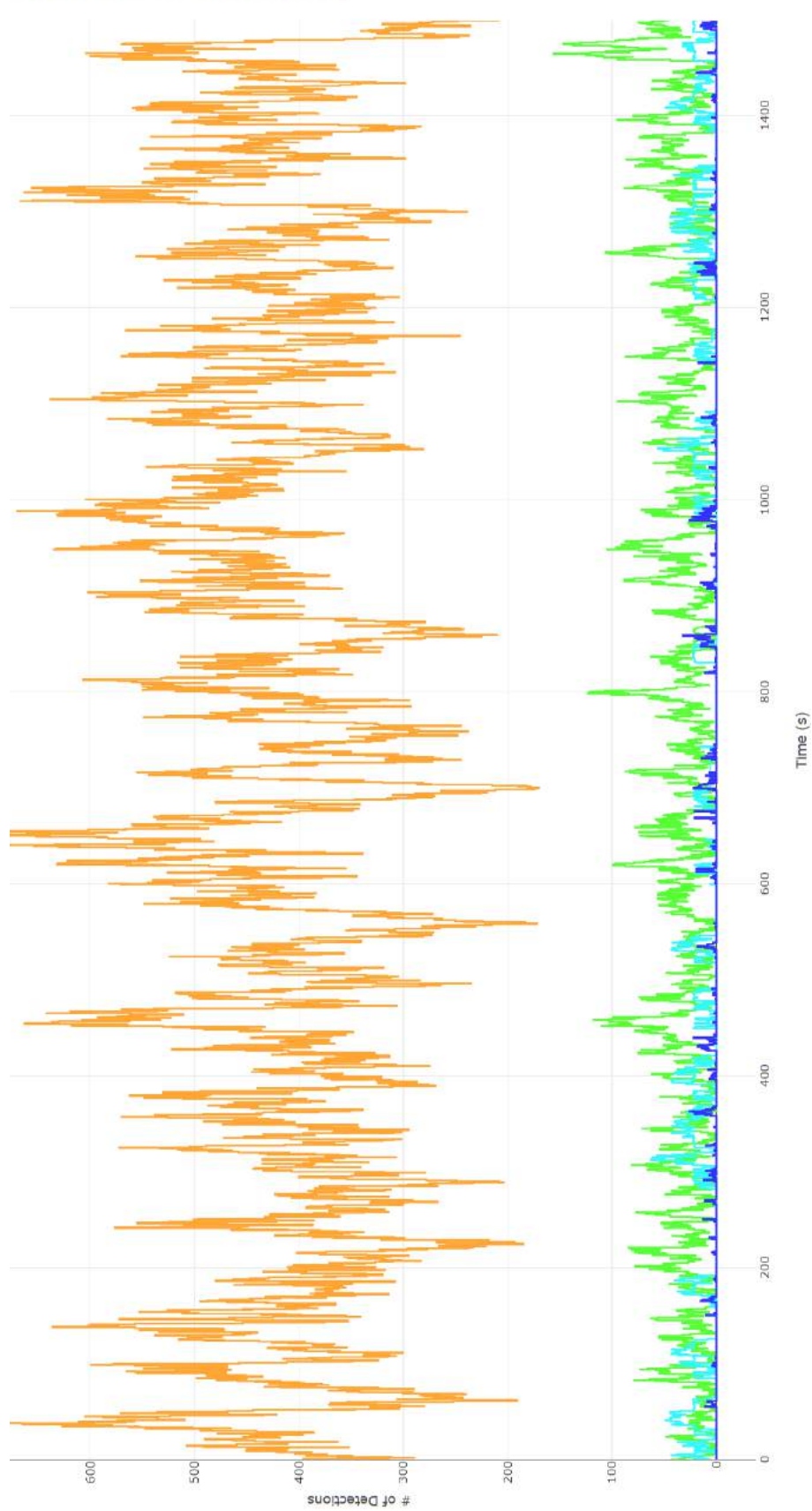
#### C.1.2 Total Detection Count Per Camera

Tables C.1, C.2, and C.3 show the total number of detections for each camera in a dataset.



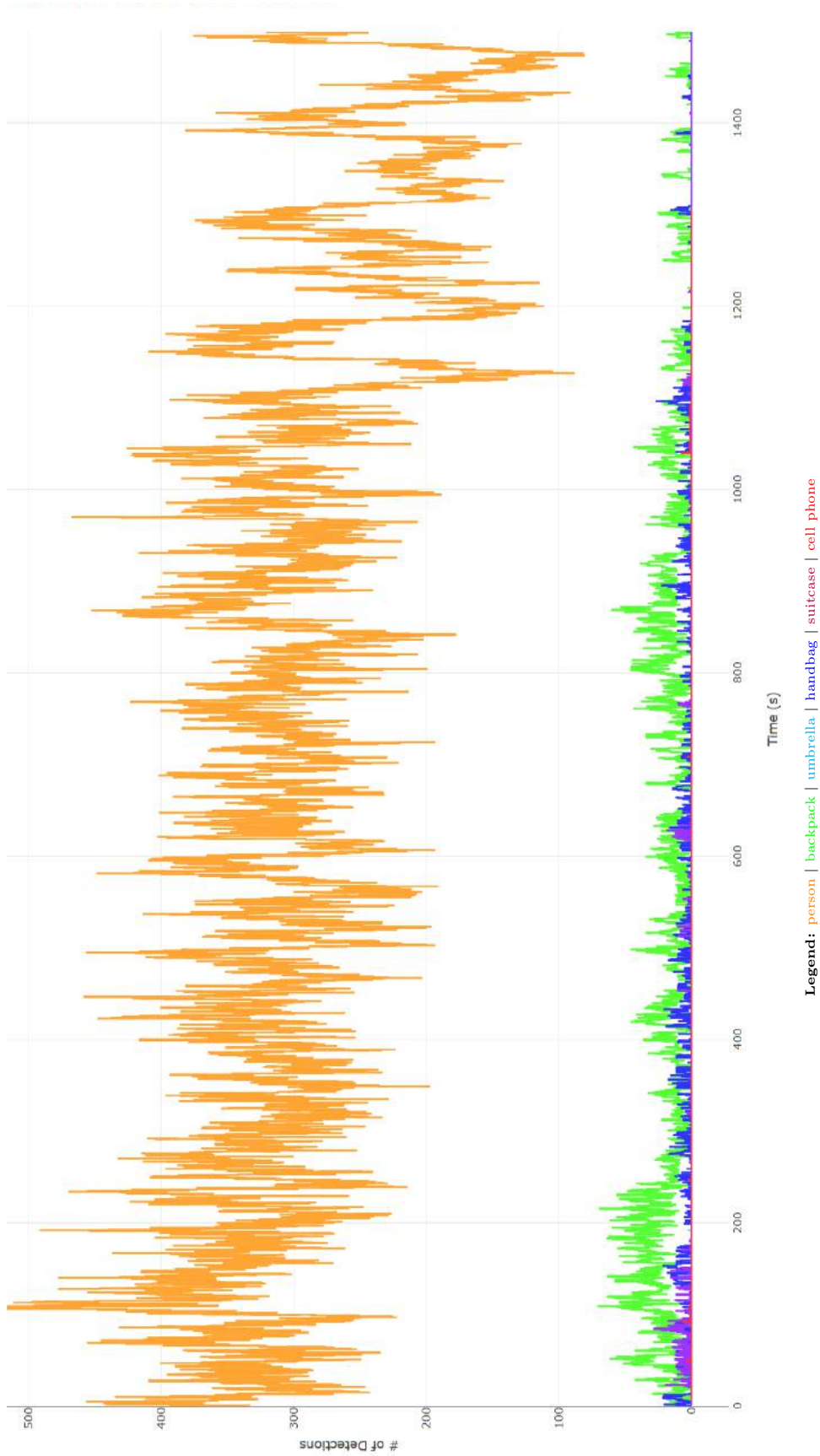
**Legend:** person | backpack | umbrella | handbag | suitcase | cell phone

Figure C.1: Line plot of the number of class detections of the HDA dataset



**Legend:** person | backpack | umbrella | handbag | suitcase | cell phone

Figure C.2: Line plot of the number of class detections of the DukeMTMC dataset



**Legend:** person | backpack | umbrella | handbag | suitcase | cell phone

Figure C.3: Line plot of the number of class detections of the WILDTRACK dataset

Table C.1: Detection Count Per Camera for the HDA Dataset

Prioritizer	Cam 2	Cam 17	Cam 18	Cam 19	Cam 40	Cam 53	Cam 54	Cam 56	Cam 57	Cam 58	Cam 59	Cam 60	Total
Optimal	390	2209	5668	3010	2978	242	8758	210	1640	1287	471	750	27613
$S_1$	378	1414	5702	2797	2177	196	5600	212	617	1121	386	494	21094
$S_2$	353	2065	5699	2442	2752	208	8732	171	1235	824	452	733	25606
$S_3$	373	2054	1513	2745	2755	249	6480	229	1519	1076	449	736	20178
$S_4$	404	2179	4472	2766	2277	240	6026	144	1556	1231	363	493	22151
$S_5$	280	1805	4255	2279	2422	182	6060	166	1228	986	325	200	20188
$S_6$	390	739	5705	3095	2803	128	8764	95	1171	1089	336	240	24555
$S_7$	381	2165	5106	2979	2830	251	8244	224	1733	1301	485	630	26329
$S_8$	337	2006	5562	2745	2792	236	8515	209	1647	1252	519	551	26371
$S_9$	381	2181	5082	2979	2877	251	7826	229	1715	1274	496	665	25956
$S_{10}$	385	2209	4106	2978	3015	248	1770	207	1808	1263	422	214	18625
$S_{11}$	298	2086	3395	2560	2582	239	2595	216	1346	1050	461	439	17267
$S_{12}$	339	2131	5527	2800	2786	236	8277	208	1666	1226	518	614	26328
$S_{13}$	380	2152	5077	2981	2899	251	7802	229	1713	1271	495	672	25922

Table C.2: Detection Count Per Camera for the DukeMTMC Dataset

Prioritizer	Cam 1	Cam 2	Cam 3	Cam 4	Cam 5	Cam 6	Cam 7	Cam 8	Total
Optimal	262931	323383	120468	126118	185153	337844	168786	261713	1786396
$S_1$	408	227007	133785	133924	165956	339651	177249	178970	1356950
$S_2$	267044	297690	89311	116774	183125	285513	121725	258417	1619599
$S_3$	210407	249532	103409	104346	149600	253381	133401	204449	1408525
$S_4$	203908	216523	108356	113987	168233	306166	79978	256227	1453378
$S_5$	231659	267022	90570	109329	164958	260971	104683	205036	1434228
$S_6$	283026	13941	139466	140706	13	341250	179617	247241	1345260
$S_7$	280591	319987	125168	81907	179773	317771	168439	248584	1722220
$S_8$	282187	305670	76581	88246	190273	334773	161686	260496	1699912
$S_9$	279339	319533	128720	75748	180605	312452	170314	246594	1713305
$S_{10}$	210680	270408	91150	93043	141359	308233	100338	257769	1472980
$S_{11}$	269799	308447	59442	103059	172525	301918	124214	167908	1507312
$S_{12}$	282212	307387	75938	88958	190526	333695	160374	261042	1700132
$S_{13}$	279347	319559	128744	75729	180654	312316	170324	246613	1713286

Table C.3: Detection Count Per Camera for the WILDTRACK Dataset

	Cam 1	Cam 2	Cam 3	Cam 4	Cam 5	Cam 6	Cam 7	Total
<b>Optimal</b>	277662	397290	328016	261931	404632	312866	372442	2354839
$S_1$	274660	287818	317781	257743	373786	257432	368316	2137536
$S_2$	285101	314197	341775	251873	329487	256111	384787	2163331
$S_3$	257100	346816	290922	250757	379074	282855	325033	2132557
$S_4$	296895	168477	332180	280442	405445	275412	375138	2133989
$S_5$	267356	340095	304992	263905	345084	280190	318274	2119896
$S_6$	309772	135940	347905	301542	418744	331353	271866	2117122
$S_7$	285648	296918	321599	254142	350742	286633	374375	2170057
$S_8$	307392	266580	343399	251244	335892	267410	390286	2162203
$S_9$	284657	299553	321234	261899	359474	272388	372840	2172045
$S_{10}$	277833	262925	322389	216558	384314	328791	387084	2179894
$S_{11}$	251737	343264	314178	253342	341838	309980	311429	2125768
$S_{12}$	307178	274129	345345	251775	335663	259109	390307	2163506
$S_{13}$	284564	300001	321536	262118	359139	271808	372724	2171890

# Curriculum Vitae

**Candidate's full name:** James Allan Douglas Cameron

**University attended:**

- University of New Brunswick, 2017, Bachelors of Science in Engineering

**Publications:**

- X. St-Onge, J. A. D. Cameron, S. A. M. Saleh, and E. J. Scheme, A Symmetrical Component Feature Extraction Method for Fault Detection in Induction Machines, IEEE Transactions on Industrial Electronics, pp. 11, 2018.