A Real-time Analytics Pipeline for Scalable Smart Video Surveillance.

Joseph Honour - 130291538

Newcastle University

Author Note

Completed as part of my undergraduate BSc in Computer Science with Industrial Placement (G401), supervised by Matt Collison.

Abstract

Computer vision has been a large area of research in recent years, devising methodologies to understand and act on events seen within video streams. With the wide deployment of CCTV and IP cameras in the past years, the opportunity for smart video analysis is available, however smart video surveillance remains a feature reserved for new hardware systems. To enable the benefits of smart video analysis using existing hardware we need to overcome and standardize the process of combining analysis frameworks with existing hardware, while enabling scalability through the use of Cloud computing.

This paper proposes an extendable and scalable framework that is able to provide an end-to-end video processing pipeline for smart video surveillance. The framework is built on core technologies that are open source, reliable and scalable. Leveraging; OpenCV, Apache Kafka, Apache Flink, Apache Spark, Neo4J and Terraform, the framework provides object detection and tracking, scalability, activity detection, event classification, data representation and system deployment respectively.

The design approach adopted allows extensibility at every opportunity, so the framework can be adapted for a multitude of use cases, with a proposed use case based on the Abbey Road crossing in London shown within this paper. Finally, the framework will be hosted publicly allowing new avenues to be explored by the community, with avenues of exploration suggested at the end of this paper.

Declaration

“I declare that this dissertation represents my own work, except where otherwise stated.”

Acknowledgments

This is my acknowledgments.

Table of Contents

[Abstract 2](#_Toc509844365)

[Declaration 3](#_Toc509844366)

[Acknowledgments 4](#_Toc509844367)

[1. Introduction 7](#_Toc509844368)

[1.1 Motivation 7](#_Toc509844369)

[1.2 Aim 9](#_Toc509844370)

[1.3 Objectives 9](#_Toc509844371)

[1.4 Paper Structure 9](#_Toc509844372)

[2. Background and Literature Review 10](#_Toc509844373)

[2.1 Video Processing Methodologies and their Adoption 10](#_Toc509844374)

[2.1.1 Object Detection 11](#_Toc509844375)

[2.1.2 Object Tracking Techniques 13](#_Toc509844376)

[2.1.3 Behavior and Activity Analysis 14](#_Toc509844377)

[2.1.4 Event Classification 14](#_Toc509844378)

[2.3 Distributed Computing and the Cloud 15](#_Toc509844379)

[2.3.1 Parallel Computing 16](#_Toc509844380)

[2.3.2 Distributed Messaging 17](#_Toc509844381)

[2.3.3 Cloud Computing 18](#_Toc509844382)

[2.4 Existing Technologies and Approaches 19](#_Toc509844383)

[3. Proposed Framework 20](#_Toc509844384)

[3.1 System Architecture 20](#_Toc509844385)

[3.1.1 Pre-Processing 20](#_Toc509844386)

[3.1.2 Activity Analysis 21](#_Toc509844387)

[3.1.3 Event Analysis 21](#_Toc509844388)

[3.1.4 Data Storage 21](#_Toc509844389)

[3.1.5 Sub System Communication 21](#_Toc509844390)

[3.1.6 Cloud Based Architecture 22](#_Toc509844391)

[3.1.7 Limitations 22](#_Toc509844392)

[3.2 System Implementation 23](#_Toc509844393)

[3.2.1 Sub System Communication 24](#_Toc509844394)

[3.2.2 Video Pre-Processing 24](#_Toc509844395)

[3.2.3 Activity Analysis 24](#_Toc509844396)

[3.2.4 Event Classification 24](#_Toc509844397)

[3.2.5 Event Notifications 24](#_Toc509844398)

[3.2.6 Data Interfacing 24](#_Toc509844399)

[3.2.7 Infrastructure Deployments 24](#_Toc509844400)

[4. Use Case Evaluation 24](#_Toc509844401)

[5. Discussion and Conclusion 24](#_Toc509844402)

[5.1 Summary 25](#_Toc509844403)

[5.2 Future Development 25](#_Toc509844404)

[References 26](#_Toc509844405)

[Footnotes 30](#_Toc509844406)

[Tables 31](#_Toc509844407)

[Figures title: 32](#_Toc509844408)

# Introduction

## 1.1 Motivation

In the United Kingdom, an estimated that over 1.85 million surveillance cameras record video streams every day, with each person being caught on camera an average of 68 times per day [1]. Fueling the mass deployment of surveillance equipment is its capability to deter criminal activity [2] coupled with enablement of event causality [3]. As a result of mass deployment, CCTV is estimated to have prevented hundreds of thousands of crimes per year [4], providing evidence for thousands of criminal investigations, including being available in 96% of homicide investigations [5]. However, with an estimated 5000 years of footage produced every day it has also been estimated that CCTV is only used in 0.5% of crimes that are recorded [2], [6], and while physical CCTV cameras are an active deterrent to criminal activities, there remains a limited approach to responding to the data produced from CCTV cameras.

Video surveillance is traditionally monitored in one of two ways; passively where it is only investigated in response to an event, or actively where it is viewed live by a single or team of people. Active monitoring and video investigation are time consuming processes, which enable a multi-million pound industry to form around them [7]. As the chance of an event of interest occurring within a video stream decreases, users tend to adopt a more passive approach, which has led to developments that bridge the gap between passive and active monitoring. Sensor driven active monitoring has produced a more efficient approach to video analysis, where movement sensors or external stimuli, trigger a prompt for active monitoring of footage. Within an industrial setting, dedicated hardware has been integrated with sensor-driven active monitoring with increasingly more sophisticated computer vision being used to trigger active monitoring.

With the development of Smart Cities, the capacity of smart video stream analysis needs to be able to scale to data volumes never encountered before within this domain. Large scale applications being with them new considerations; minimizing data transfer, distributing work to enable greater processing throughput, while providing analysis results in a quick enough time to be valuable. To support this capacity and existing video surveillance deployments, a base framework is proposed. It provides common functionality for video processing by default, while allowing extensions to be developed, enabling its adoption in the widest variety of use cases. Furthermore, distributed technologies provide a base to meet the demands of even the largest surveillance networks.

## 1.2 Aim

Propose an extendable and scalable framework for video stream analysis with object, activity and event classification.

## 1.3 Objectives

1. To research existing video processing techniques and available software packages in order to support current computer vision techniques.
2. To research machine learning techniques with the capability to identify events in time series data.
3. Develop a framework that provides a minimum viable product of object, activity and event detection, while being scalable and extensible.
4. Develop a use case that will allow the evaluation of the proposed real-time video processing framework.
5. Using the use case defined in objective four, evaluate the frameworks ability to support existing video processing techniques, while meeting bespoke user requirements.
6. Compare and contrast the performance of the proposed framework against existing CCTV technologies and approaches.

## Paper Structure

This will be my paper structure.

# Background and Literature Review

Computer vision applications are often deployed with a variety of desired outcomes, including facial recognition, optical character recognition (OCR), medical image analysis, automatic number plate recognition (AMPR), traffic analysis and footfall analysis. Within computer vision domains, analysis is usually implemented with a pipelined architecture, with a focus on pattern recognition.

## 2.1 Video Processing Methodologies and their Adoption

The exploration of theoretical stages within generic video processing are well documented. Ko et al [8], provides an industrial case setting, with the goal of providing surveillance information to government agencies, that can be used as a basis for common core functionality required of smart surveillance systems. Within this setting functionality can be seen to include background modelling, object segmentation, object classification, object tracking, with the outcome of person identification with behavior and activity analysis (Figure 1). This provides an avenue of interrogation within the realm of smart surveillance systems. An alternative approach is to use black box models for pattern recognition, such as deep learning techniques, that are able to integrate multiple phases of the pipeline described, within the layers of the classification model [9]. Typically, modern video analysis pipelines adopt a combination approach that uses black box models for object detection, object tracking, behavior analysis and event classification but allows for limited interrogation of the intermediate steps.



Figure 1: Adaptation of video processing within a general framework for automated visual surveillance system [8]. This shows common stages involved in providing computer vision applications in the context of automated surveillance systems.

2.1.1 Object Detection is the base of all computer vision applications, providing the ability to accurately identify objects within a frame. Work within this field has relied upon being able to interpret the combinations of pixels correctly to identify an object. The most common method of doing this is using a Haar feature-based cascade classifier [10]. This machine learning approach works by showing a classifier a multitude of images, with some containing the object you wish to detect. Then, by applying features to the image that allow accurate pattern recognition of the object, we can train the classifier to identify the unique signature of the desired object when it is present. From here, given a trained classifier, we extract the features from the images that identify the object, where each feature is a single value obtained by subtracting the sum of pixels under the white rectangle from the sum of pixels under the black rectangle (Figure 2). The composition of the identifying features of an object enable the final model to identify if, and where, an object is present to a degree of accuracy.

Figure 2: Left, an example of Haar cascade features. Right, the adoption of Haar cascade features in detecting a face [11].

Further to this, we are able to improve the performance of detections using an algorithm called Adaboost [12], which allows ranking of object detection features based on their error rate when attempting to successfully identify the object.

In more recent years, with the improvement of Graphics Processing Units (GPU) hardware, advanced detection techniques have been created upon Deep Learning models [9]. Further to this, object detection techniques often utilize motion and background separation methods, enabling them to provide insight into events and activities occurring [13]. Utilizing the foundation of accurate object detection, objects can be given identity between frames.

2.1.2 Object Tracking Techniques enable the observation of object movement vectors, along with the monitoring of object interaction patterns. Work within the object tracking domain build predictive models with online data, meaning they adapt at runtime to provide improved predictions. Algorithms within this domain often model an objects location as a set of positions that each could contain the objects location based on its previous known location [14], [15], [16], with the most common tracking techniques described in Table 1. Given successful employment of object tracking, we can now observe the behaviors of individual objects through a video stream.

|  |  |  |  |
| --- | --- | --- | --- |
| Approach | Overview | Pros | Cons |
| Multiple Instance Learning | Tracks an object by treating its location as a set of positions (‘bags’) that each could contain the objects location based on its previous location. This gives the learning algorithm the responsibility of removing the ambiguity of the exact object location and predicting which instance in each bag is most correct [14]. | Works well when object partially occluded due to its bag representation. | The tracker cannot handle full occlusion of objects well. |
| Median Flow | Tracks an object by selecting a variety of points within the object space and then computes the trajectory for this object, both forwards and backwards in time. This means the tracker is able to compare both trajectories and make accurate predictions of the objects final location [15]. | Works well when an objects trajectory is predictable. | The tracker becomes less reliable under fast or unpredictable motion. |
| Tracking-Learning-Detection | Breaks the tracking problem up into: Tracking, learning and detection. The tracker then follows an object between frames with the detector correcting the tracker based on all observed appearances. The learning estimates the error of the detector and updates it to avoid these errors in future [16]. | Works well under long periods of occlusion. | The tracker does not perform well under large full rotations and can create many false positives. |

Table 1: A comparison of three common approaches to object tracking. This aims to show the diversity in object tracking, while illuminating the costs and benefits to each approach.

2.1.3 Behavior and Activity Analysis allow the assignment of context to object movements, enabling more advanced processing techniques down-stream in the video processing pipeline, such as event classification. Within computer surveillance it is often desirable to identify endangering or suspicious activities [8]. Active work within this field is varied, with successful modelling of behaviors being produced using Markov Models [17], [18], along with the more computationally intense techniques architected with Neural Networks [8]. Markov Model techniques have identified behaviors to precision rates above the 90th percentile [19], however they struggle with noise in the data which can cause accuracy to drop. Other popular methods of behavior analysis include [8]:

* Dynamic Time Warping: a technique for comparing the similarity between two sequences of events. This allows the calculation of the probability that a shown behavior corresponds to a previously known behavior pattern.
* Finite State Machines: models’ behavior as a finite set of states, with transitions between related states. This allows for analysis on real-time video, watching for movements that trigger a tracked object to progress to a different state. A comparison between the transition path through the state machine and known behavior traits enables the identifying of witnessed behaviors.

The understanding of activities within a video stream provides a deep insight into what is transpiring, allowing the modelling of the video context through time.

2.1.4 Event Classification builds upon activity analysis, giving a method for identifying unusual data points within the context of the video stream. Work in this field focuses on building a model to represent the current state of the data, and then compares new data points to the model calculating how far the point deviates from the existing data set. Multiple models are able to provide this, frequently built around clustering techniques [20].

Further to this, the adaptation of One Class Support Vector Machines have provided anomaly detection for short term observed behaviors within video streams [19]. They are able to detect whether a short-term series of points is outside of normal observed behaviors, and provide an abnormal threshold capability that can distinguish which data points require more in-depth anomaly analysis.

Recently, with the wide adoption of Neural Networks, many methods have emerged built around this core architecture that can provide high accuracy detections, at the cost of requiring high performance hardware [21]. Based around learning the distribution of features within the data, the models can detect when a data point falls outside of the normal distribution and classify it as anomalous. A Restricted Boltzmann Machine is a Neural Network model for the learning of distributions and, when multiple are stacked together, they build a Deep Belief Network [21]. However, Deep Belief Networks bring with them a large computational overhead and therefore may not be appropriate for a generic video processing framework.

## 2.3 Distributed Computing and the Cloud

Video stream data is rich and complex and therefore raises significant challenges in data handling and processing when attempting to retain real-time requirements. In order to provide actionable intelligence, analysis not only has to have the functional requirement of being correct, but it must also meet the non-functional requirement of being produced within a time frame that allows actions to be taken with significance.

2.3.1 Parallel Computing allows the concurrent processing of data streams, enabling analysis on large quantities of data, also known as throughput. Within the realm of video processing, there is a need to perform large amounts of calculations on individual frames, in order to provide services ranging from initial object detection to tracking and analysis. As more video streams are added to a system for analysis, there is a need for these expensive operations to scale, therefore we must be able perform these operations in parallel over a distributed set of machines. Current openly available Frameworks, enable the distribution of work within a cluster of machines, vastly increasing the throughput of the overall system (Table 2).

|  |  |
| --- | --- |
| Framework | Functional Details |
| Apache Storm | Apache Storm runs on a distributed cluster of machines, where a topology is defined representing the series of processing stages to apply to each event. A topology is comprised of bolts, with a bolt representing an individual processing stage on an event. This architecture gives Apache Storm power to process billions of events across a distributed set of machines in a single day [22]. |
| Apache Flink | Apache Flink works by expressing data processing as a series of tasks to apply to each event as a directed graph. the sending of messages between stages in the dataflow pipeline make use of buffers, with the backpressure from these buffers being used to control the throughput of the pipeline. This allows Apache Flink to provide throughputs of up to 80 million events per second [23]. |
| Apache Spark | Apache Spark was originally designed for distributed batch processing, but has since been extended for stream processing. It processes events in small micro-batches in order to run them in real-time. It represents data as a resilient distributed dataset that can have operations applied to it [24]. |

Table 2: An overview of popular available frameworks for distributed computing, showing a brief insight into the architecture of each system.

As these technologies offer similar services, performance comparisons between them exist. Under high workloads, it has been found that Apache Flink out performs other systems with regards to latency, however Apache Spark is able to provide the largest processing throughput [25]. Nevertheless, these systems are in their infancy of development and therefore full conclusion about the final performance of each system are hard to make.

2.3.2 Distributed Messaging provides the ability to decouple the different stages of processing, allowing for the addition of new features without effecting existing stages of a processing pipeline. As well as decoupling, video processing analysis relies on creating an immutable log of objects, behaviors and anomalies detected within a given video stream. This immutable log enables the understanding of the causality of events; which is a core component of video processing analytics. Further to this, a distributed messaging layer should be able to scale to multiple video stream inputs without effecting real-time performance requirements. Providing these capabilities is Apache Kafka [26].

Apache Kafka is a distributed streaming platform, which lets applications publish and subscribe to topics of messages. It stores the messages published in a fault tolerant way, partitioning information across multiple machines within a configured cluster. To communicate with Apache Kafka a client performs a high-performance, language agnostic, TCP protocol, meaning any system can speak with it. Further to its core functionality, Apache Kafka is also extremely fast, outperforming traditional messaging systems when handling the producing and consuming of messages between decoupled systems (Figure 5).

Figure 5: A comparison of Apache Kafka’s producer performance (Left), and its consumer performance (Right) against competitor messaging systems [27].

2.3.3 Cloud Computing enables a user to rent the computational power they require, scaling up when demand is high, and reducing when systems become idle. This elasticity is a highly desirable feature when proposing a framework that needs to cater to a large variety of projects and requirements. This enables the distributed processing systems discussed to have resources allocated to them at runtime, when demand is high on an individual stage of the pipeline more machines can be created and added to meet non-functional requirements. Further to this, machines can be destroyed if they are not required, reducing running costs of the system.

The benefits of Cloud computing do not come without risks however; with virtualized hardware exact performance metrics may be unknown until runtime. As a client of a Cloud provider, you are also at the mercy of their availability; if their service becomes unavailable, you are incapable of fixing it yourself. Further considerations to make can include, data confidentiality, software licensing costs and runtime renting costs [28].

## 2.4 Existing Technologies and Approaches

Computer vision is often most successful in batch analysis with extensive post processing, however smart video surveillance requires real-time event classification. This relies upon the computer assuming the active monitoring role rather than to prompt a human to analyses the footage, which is beyond the scope of most existing applications. Table 3 provides a summary of existing technologies that use computer vision applications within the video processing pipeline. It is seen however, that these are often designed for an individual use case or are developed under closed licensing and therefore cannot be built upon.

|  |  |
| --- | --- |
| Product | Functionality Details |
| Nest [29] | Alerts users in real-time, via an app, to event and motion detection events seen on camera. Provides event and anomaly detection and real-time alerts when these occur. However, it requires specialized hardware, is not extendable for development. |
| A Video Analysis Framework for Surveillance System [30] | Gives a novel framework approach to online video analysis using .NET 2.0. Provides object detection, event detection with extensibility to insert new functionality within the application. It is limited by its lack of distributed computing; therefore, it will not be able to scale to all user requirements. Further to this, the product is developer in .NET, requiring a Microsoft workstation to run. |
| DiVA [31] | Gives a distributed video processing framework that uses a database as a message source allowing components to communicate agnostic of technologies adopted. Provides object detection with the goal of detecting object abandonment and removal. Extensibility is provided through the communication of modules/algorithms occurring at a database level. However, it uses a singular database to communicate, rather than a distributed messaging system. It also requires fixed hardware meaning it currently does not make use of Cloud Computing. |

Table 3: A list of current technologies in the market that aim to provide smart CCTV.

# Proposed Framework

## 3.1 System Architecture

Proposed is a scalable framework that supports common video processing techniques by default, while being open for extension to enable domain specific modifications (Figure 2). It adopts a streaming architecture to provide real-time analytics on data generated from the raw video input. This enables decisions to be made as video footage is processed, while still supporting traditional batch processing applications.



Figure 6: The proposed video processing architecture, showing the avenues of communication in the distributed system.

3.1.1 Pre-Processing of the raw video input allows for a fine grained control of the volume of data entering the downstream processing stages of the pipeline. Using a technique known as Edge Computing [32], applying filtering and detection on the raw video feed to control the amount of data entering the downstream processing stages, the networks capacity is maximized. Thus, this increases the throughput of the pipeline as less unnecessary data arrives at the messaging system.

3.1.2 Activity Analysis is then performed on the annotated data published to the distributed messaging system. This is done over a distributed processing framework, allowing for parallel identification of activities. Making us of pattern matching techniques, it reads the log of annotations and compares the processed events with its internal model of activity signatures. When an activity is identified it can then be published onto the messaging service for downstream services to analyze.

3.1.3 Event Analysis can then engage with the information produced from upstream stages to identify clusters of behavior, looking for events that deviate from the normal. If an event of interest occurs it then has the capacity to send notifications to the appropriate user, allowing for further investigation.

3.1.4 Data Storage occurs as the data is streamed between services. It consumes all messages being sent and persists them to a database. This enables offline analysis of the data by users, along with querying and exploration of data produced by the pipeline, agnostic of data producing services.

3.1.5 Sub System Communication is complete through a distributed messaging layer, decoupling systems and maintaining a flexible and extensible application. Communication can then be achieved by using a common Application Program Interface (API) between systems and the messaging layer. The decoupled nature of the sub systems means they can be deployed independently, allowing the most appropriate tool to be used for each area of processing, with individual resource allocations for different parts of the system. This enables users to deploy infrastructure on a per-service basis, giving a fine-grained level of control to avoid over or under allocation of resources to a task. A messaging layer adds complexity to the framework, as systems have the added overhead of indirectly communicating with each other, which is a noticeable cost when considering this approach.

3.1.6 Cloud Based Architecture provides a service that allows clients to rent hardware rather than buying it upfront, as this gives a flexible way of managing infrastructure depending on specific performance requirements, and the processing framework looks to make use of this. However, the deployment can be made to a local environment, and considerations can be made as to the most appropriate production infrastructure on a per use case basis [28]. Once deployed, the framework makes use of distributed technologies to manage throughput and latency during processing.

Taking successful aspects from previously seen work [30] [31], giving a modular approach to design, while extending the ability to distribute work between machines. The framework improves upon these designs by making use of Cloud Computing infrastructure, while enabling clusters of machines to distribute work for a single task, rather than just distributing work between different stages of the processing pipeline. This approach hopes to overcome the challenges of scaling, while maintaining the success of modular design seen in previous work, enabling large scale computer vision applications to become possible.

3.1.7 Limitations can be found with a distributed based approach, the network latency of the infrastructure heavily affects distributed processing, as each node in a cluster must communicate with its counterparts to organize and distribute work. This can drastically reduce the performance of the proposed framework if deployed onto a degraded network environment. Mitigating this, the frameworks deployment to Cloud infrastructure should allow for dynamic network configuration to meet individual requirements, coupled with intense processing happening at the edge of the Cloud. Accompanying this feature, Cloud deployments should be made smooth and approachable enabling users with little infrastructure experience to deploy applications to production.

Further to this, Cloud infrastructure can become expensive as network usage is charged to the user along with the rented computing power of the machines. To combat this, the framework is not linked to a single Cloud provider, or to the Cloud at all, giving the freedom of choice to the adopting user.

## 3.2 System Implementation

The proposed system architecture is implemented as shown in Figure 7, producing the data processing pipeline in Figure 8. The sub systems are written independently and do not share dependencies, allowing them to be modified and extended without the need for modifications to other systems. Accompanying this, support projects have been created allowing the automated deployment of all required distributed processing frameworks to Amazon Web Services (AWS).



Figure 7: The proposed video processing framework implementation, distributed computing technologies (Apache Kafka, Apache Flink, Apache Spark) are adopted to provide high throughput, low latency, processing within a Cloud environment.



Figure 8: The data flow through the application, showing how each processing stage makes use of Apache Kafka to read input and produce outputs.

3.2.1 Video Pre-Processing is implemented as a Python package offering configurable video processing techniques for object detection and tracking. The module is built around a processing pipeline interface, allowing the user to choose which processing techniques are applied to each frame of the video input, and where the results of the processing stages are sent. Video inputs can be accepted from file and from a directly connected USB camera (Figure 9). Further to this, person and car detection is provided using the OpenCV library, with tracking of all detected objects enabled through a Kernalized Correlation Filters model. This dynamic configuration allows for the package to be extended and configured easily to individual domains. The output of the pre-processing stage is sent to Apache Kafka by default, with the annotated video shown on screen. This gives the user the choice to see the annotation information being generated and sent to downstream processes, which aids in visualization of generated data.



Figure 9: An example configuration of the video pre-processing package. This displays how the use of a builder pattern can be adopted to enable easy building of custom processing stages.

3.2.2 Activity Analysis is delivered as an Apache Flink task, that can be executed on any deployed Apache Flink cluster. It offers distributed activity detections for standing, walking and running people, along with parked and moving cars. Reading the data produced from the pre-processing service via Apache Kafka, the service adopts Apache Flink’s advanced pattern matching technology to look for simultaneous movement events typical of known activity behaviors. The service is tuned through configuration files read in at task runtime, allowing for activity identification to be tuned at runtime to provide precise results for particular camera installations and viewing angles. Furthermore, the service is made extendable through the Flink pattern matching API, and new patterns can be developed to allow the identification of new behaviors.

3.2.3 Event Classification adopts the core libraries provided by the Apache Spark ML library to provide unsupervised event classification. The service reads data produced from the activity analysis service and then uses an unsupervised K-Means clustering model to assign it a cluster of origin and calculate how far the instance deviates from the cluster center. This model is adaptive, as the data is streamed through the system it is able to adjust cluster locations to provide accurate classifications even if data patterns change. The service is tunable through command line parameters, that enable the configuration of the number of clusters that the model should attempt to identify. Once it calculates the events cluster and distance from the cluster center, it published this information to Apache Kafka.

3.2.4 Event Notifications are enabled as a task running on Apache Flink, reading data produced from the event classification service via Apache Kafka, notifications can be sent for events assigned to a specific cluster or for events that deviate from their assigned cluster by a set amount. These rules can be configured at runtime through configuration files, allowing for dynamic business domain rules to be created and changed as more insight into data is achieved. If an event is deemed to require a notification, this is currently supported through the sending of an email to a configured user account.

3.2.5 Data Storage is achieved through the reading of Apache Kafka topics.

3.2.6 Data Interfacing is given to the user through the built in Neo4J dashboard.

3.2.7 Infrastructure Deployments are available using Terraform scripts.

# Use Case Evaluation

Speak about the use case we will evaluate the pipeline under.

# Discussion and Conclusion

This will be my conclusion.

## 5.1 Summary

This will be a summary of work completed and its success.

## 5.2 Future Development

This will be a look into areas of future development.

References

[1] T. Reeve, “How many cameras in the UK? Only 1.85 million, claims ACPO lead on CCTV,” *CCTV Image magazine*, 2011. [Online]. Available: http://www.securitynewsdesk.com/2011/03/01/how-many-cctv-cameras-in-the-uk/.

[2] B. C. Welsh and D. P. Farrington, “Public area CCTV and crime prevention: An updated systematic review and meta-analysis,” *Justice Quarterly*, vol. 26, no. 4. pp. 716–745, 2009.

[3] J. M. Caplan, L. W. Kennedy, and G. Petrossian, “Police-monitored CCTV cameras in Newark, NJ: A quasi-experimental test of crime deterrence,” *J. Exp. Criminol.*, vol. 7, no. 3, pp. 255–274, 2011.

[4] S. J. McLean, R. E. Worden, and M. S. Kim, “Here’s Looking at You: An Evaluation of Public CCTV Cameras and Their Effects on Crime and Disorder,” *Crim. Justice Rev.*, vol. 38, no. 3, pp. 303–334, 2013.

[5] S. Yard, “CCTV in Homicide Investigations,” 2010. [Online]. Available: https://goo.gl/oS5Tgn. [Accessed: 15-Nov-2017].

[6] E. L. Piza, J. M. Caplan, and L. W. Kennedy, “Analyzing the Influence of Micro-Level Factors on CCTV Camera Effect,” *J. Quant. Criminol.*, vol. 30, no. 2, pp. 237–264, 2014.

[7] S. Germain, “A prosperous ‘business’: The success of CCTV through the eyes of international literature,” *Surveill. Soc.*, vol. 11, no. 1–2, pp. 134–147, 2013.

[8] T. Ko, “A survey on behavior analysis in video surveillance for homeland security applications,” *Appl. Imag. Pattern Recognit. Work. 2008. AIPR ’08. 37th IEEE*, pp. 1–8, 2008.

[9] Y. Lecun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553. pp. 436–444, 2015.

[10] P. Viola and M. Jones, “Robust real-time object detection,” *Int. J. Comput. Vis.*, vol. 57, no. 2, pp. 137–154, 2001.

[11] OpenCV, “Facial Detection with Haar Cascades in OpenCV,” 2018. [Online]. Available: https://docs.opencv.org/3.3.0/d7/d8b/tutorial\_py\_face\_detection.html. [Accessed: 30-Jan-2018].

[12] P. a Viola and M. J. Jones, “Fast and Robust Classification using Asymmetric AdaBoost and a Detector Cascade,” *Adv. Neural Inf. Process. Syst.*, no. December, pp. 1311–1318, 2001.

[13] Y. Zhu, N. M. Nayak, and a K. Roy-Chowdhury, “Context-Aware Activity Recognition and Anomaly Detection in Video,” *Sel. Top. Signal Process. IEEE J.*, vol. 7, no. 1, pp. 91–101, 2013.

[14] B. Baben and S. Belongie, “Visual tracking with online Multiple Instance Learning,” *2009 IEEE Conf. Comput. Vis. Pattern Recognit.*, pp. 983–990, 2009.

[15] Z. Kalal, K. Mikolajczyk, and J. Matas, “Forward-backward error: Automatic detection of tracking failures,” in *Proceedings - International Conference on Pattern Recognition*, 2010, pp. 2756–2759.

[16] Z. Kalal, K. Mikolajczyk, and J. Matas, “Tracking-learning-detection,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 7, pp. 1409–1422, 2012.

[17] J. Kröckel and F. Bodendorf, “Intelligent Processing of Video Streams for Visual Customer Behavior Analysis,” *ICONS 2012, Seventh Int. Conf. Syst.*, no. c, pp. 163–168, 2012.

[18] T. T. Z. T. T. Zin, P. T. P. Tin, T. Toriu, and H. Hama, “A Markov Random Walk Model for Loitering People Detection,” *Intell. Inf. Hiding Multimed. Signal Process. (IIH-MSP), 2010 Sixth Int. Conf.*, 2010.

[19] P. Antonakaki, D. Kosmopoulos, and S. J. Perantonis, “Detecting abnormal human behaviour using multiple cameras,” *Signal Processing*, vol. 89, no. 9, pp. 1723–1738, 2009.

[20] M. Goldstein and S. Uchida, “A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data,” *PLoS One*, vol. 11, no. 4, 2016.

[21] M. Längkvist, L. Karlsson, and A. Loutfi, “A review of unsupervised feature learning and deep learning for time-series modeling,” *Pattern Recognit. Lett.*, vol. 42, no. 1, pp. 11–24, 2014.

[22] A. Toshniwal *et al.*, “Storm@twitter,” in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data - SIGMOD ’14*, 2014, pp. 147–156.

[23] P. Carbone, S. Ewen, S. Haridi, A. Katsifodimos, V. Markl, and K. Tzoumas, “Apache Flink: Unified Stream and Batch Processing in a Single Engine,” *Data Eng.*, vol. 36, pp. 28–38, 2015.

[24] M. Zaharia *et al.*, “Apache Spark: a unified engine for big data processing,” *Commun. ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[25] S. Chintapalli *et al.*, “Benchmarking streaming computation engines: Storm, flink and spark streaming,” in *Proceedings - 2016 IEEE 30th International Parallel and Distributed Processing Symposium, IPDPS 2016*, 2016, pp. 1789–1792.

[26] Apache, “Apache Kafka,” 2018.

[27] J. Kreps, N. Narkhede, and J. Rao, “Kafka: a Distributed Messaging System for Log Processing,” 2011.

[28] M. Armbrust, A. Fox, R. Griffith, A. Joseph, and RH, “Above the clouds: A Berkeley view of cloud computing,” *Univ. California, Berkeley, Tech. Rep. UCB* , pp. 07–013, 2009.

[29] Nest, “No Title,” 2017. [Online]. Available: https://nest.com/uk/cameras/nest-cam-indoor/overview/. [Accessed: 24-Nov-2017].

[30] N. Suvonvorn, “A video analysis framework for surveillance system,” *2008 IEEE 10th Work. Multimed. Signal Process.*, pp. 867–871, 2008.

[31] J. C. SanMiguel, J. Bescós, J. M. Martínez, and Á. García, “DiVA: A Distributed Video Analysis Framework Applied to Video-Surveillance Systems,” in *International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS)*, 2008, pp. 207–210.

[32] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge Computing: Vision and Challenges,” *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, 2016.

Footnotes

1[Add footnotes, if any, on their own page following references. For APA formatting requirements, it’s easy to just type your own footnote references and notes. To format a footnote reference, select the number and then, on the Home tab, in the Styles gallery, click Footnote Reference. The body of a footnote, such as this example, uses the Normal text style. (Note: If you delete this sample footnote, don’t forget to delete its in-text reference as well. That’s at the end of the sample Heading 2 paragraph on the first page of body content in this template.)]

Tables

Table 1

[Table Title]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Column Head | Column Head | Column Head | Column Head | Column Head |
| Row Head | 123 | 123 | 123 | 123 |
| Row Head | 456 | 456 | 456 | 456 |
| Row Head | 789 | 789 | 789 | 789 |
| Row Head | 123 | 123 | 123 | 123 |
| Row Head | 456 | 456 | 456 | 456 |
| Row Head | 789 | 789 | 789 | 789 |

Note: [Place all tables for your paper in a tables section, following references (and, if applicable, footnotes). Start a new page for each table, include a table number and table title for each, as shown on this page. All explanatory text appears in a table note that follows the table, such as this one. Use the Table/Figure style, available on the Home tab, in the Styles gallery, to get the spacing between table and note. Tables in APA format can use single or 1.5 line spacing. Include a heading for every row and column, even if the content seems obvious. A default table style has been setup for this template that fits APA guidelines. To insert a table, on the Insert tab, click Table.]

Figures title:

Figure 1. [Include all figures in their own section, following references (and footnotes and tables, if applicable). Include a numbered caption for each figure. Use the Table/Figure style for easy spacing between figure and caption.]

For more information about all elements of APA formatting, please consult the APA Style Manual, 6th Edition.