Real-time Scalable Video Stream Analysis with Object, Activity and Anomaly Detection.

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. A major application of computer vision is to detect anomalies autonomously, and alert users to when they occur. Although industrial technologies exist that are able to do this to a basic standard, they often rely on expensive and exclusive hardware.

This paper proposes an extendable and scalable framework that is able to provide accurate anomaly detection, in real-time, without complex hardware requirements. The framework will show how the adoption of distributed computing and machine learning enable real-time anomaly detections, without requiring specialized hardware. My design approach is to allow extensibility at every opportunity, so the framework can be adapted for a multitude of use cases, some of which I propose within this paper. Furthermore, the framework will allow horizontal scaling enabling it to handle large volumes of data, while keeping its real-time requirements intact. 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](#_Toc506893809)

[Declaration 3](#_Toc506893810)

[Acknowledgments 4](#_Toc506893811)

[Introduction 7](#_Toc506893812)

[Motivation 7](#_Toc506893813)

[Aim 8](#_Toc506893814)

[Objectives 8](#_Toc506893815)

[Paper Structure 9](#_Toc506893816)

[Background and Literature Review 9](#_Toc506893817)

[Video Processing Methodologies and their Adoption 9](#_Toc506893818)

[Object Detection Techniques 10](#_Toc506893819)

[Object Tracking Techniques 12](#_Toc506893820)

[Behavior and Activity Analysis 13](#_Toc506893821)

[Anomaly Detection with Machine Learning 13](#_Toc506893822)

[Anomaly Detection Models 13](#_Toc506893823)

[The Impact of Human Behavior 13](#_Toc506893824)

[Distributed Computing and the Cloud 13](#_Toc506893825)

[Parallel Computing 13](#_Toc506893826)

[Distributed Messaging 16](#_Toc506893827)

[Cloud Computing 17](#_Toc506893828)

[Existing Technologies and Approaches 18](#_Toc506893829)

[References 20](#_Toc506893830)

[Footnotes 23](#_Toc506893831)

[Tables 24](#_Toc506893832)

[Figures title: 25](#_Toc506893833)

# Introduction

## Motivation

In the United Kingdom, an estimated 4.2 million surveillance cameras watch us every day (Norris and McCahill, 2006). With CCTV’s deployment to businesses, homes, shops and high streets we have become one of the most watched nations in the world. Fueling the mass deployment of surveillance equipment is its capability to deter criminals coupled with the recording of evidence of perpetrators if a crime is committed. This can be seen through CCTV’s 96% availability in homicide investigations (Yard, 2010). However, for it to be used an individual must manually watch all relevant footage, highlighting points of interest, which is extremely time consuming and inefficient. A more proactive approach to policing could evolve if the real-time detection of these points of interest within video-streams became available through automation.

This is the key problem with current CCTV; it does not provide real-time analysis or understanding of what is happening within the video stream. This means its adoption can currently only provide a post-event detection capacity, given a large time input from users. It also means, due to its simple nature, all video footage must be stored for long periods of time, when a user may only be interested in the periods within the video stream containing points of interest. To solve this a system needs to be developed that can work with existing CCTV systems providing understanding of events occurring within a video, detecting and alerting users when points of interest occur.

Systems in the current market that could provide this functionality, only either provide sub-parts of the full desired behavior, or require specialized hardware requiring the replacement of currently deployed CCTV cameras (Table 2).

To enable the demands of intelligent video stream analysis, the development of a base framework is necessary. The framework should provide common functionality for video processing by default, while allowing easy development extension to cater to the broadest range of use cases. Furthermore, horizontally scaling technologies should provide the framework base to meet the demands of even the largest video stream development projects.

## Aim

Propose an extendable framework for object detection and tracking, behavior and activity analysis, and anomaly detection, making use of distributed computing and machine learning.

## Objectives

1. To research existing video processing techniques and available software packages in order to detect objects and activities within a video stream.
2. To research machine learning techniques with the capability to detect anomalies in time series data produced from objective one.
3. Develop testing scenarios that will allow the evaluation of a proposed real-time video processing framework.
4. Develop a framework that provides a minimum viable product of object, activity and anomaly detection, while being scalable and extensible.
5. Using the test scenarios defined in objective three, evaluate the frameworks ability to analyze real-time video streams in order to detect anomalies and alert users in real-time.
6. Compare and contrast the performance and storage requirements of the proposed framework against existing CCTV technologies and approaches.

## Paper Structure

I will describe my paper structure here.

# Background and Literature Review

When developing an extendable framework for video processing, we must consider both software and hardware designs that will allow the framework to be suitable for the largest variety of video stream analysis scenarios. To do this, we must first look at common video processing methodologies and the software techniques adopted in providing them. Further to this, we need to decide how best to scale and extend the framework, making it a contender for high frequency scenarios. This will involve the investigation and adoption of Cloud Computing. The combination of suitable software and hardware decisions will provide the base for implementing a successful framework.

## Video Processing Methodologies and their Adoption

In order to develop an effective video processing framework that enables accurate autonomous decisions, we must be able to understand what is occurring within a series of sequential video frames. Existing Architecture designs presented (Figure 1) show theoretically the different stages you must build upon to enable successful video processing. These stages build upon each other, until we reach a point of desired intelligent understanding of the processed video stream.

The base of these stages is the ability to detect objects within a frame that are of interest to us. From this, tracking can then occur from frame to frame, giving objects persistent identity. Detection and tracking gives our first intelligent understanding of what is occurring in the video, and provides the basis for more advanced techniques. Advanced techniques often include facial recognition and person identification, along with behavior and activity analysis, all of which require the location of objects and their previous movements through the video. Given we are able to provide this intelligence over the video stream we can then build an autonomous way of providing anomaly detection on the seen behaviors, allowing real-time alerts to be sent to users.



Figure 1: Adaptation of video processing within a general framework for automated visual surveillance system (Ko, 2008).

### Object Detection Techniques

The first stage of the video processing pipeline gives the ability to detect objects accurately. Detecting objects in images relies on being able to interpret the combinations of pixels correctly to identify the particular object you are looking for. The most common method of doing this is using a Haar feature-based cascade classifier (P. Viola and Jones, 2001). This machine learning approach works by showing a classifier a multitude of images, with some containing the object you wish to detect. The classifier then attempts to apply features to the image that allow accurate detection of the desired object. From here, given a trained classifier, we extract the Haar features from the images, 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).

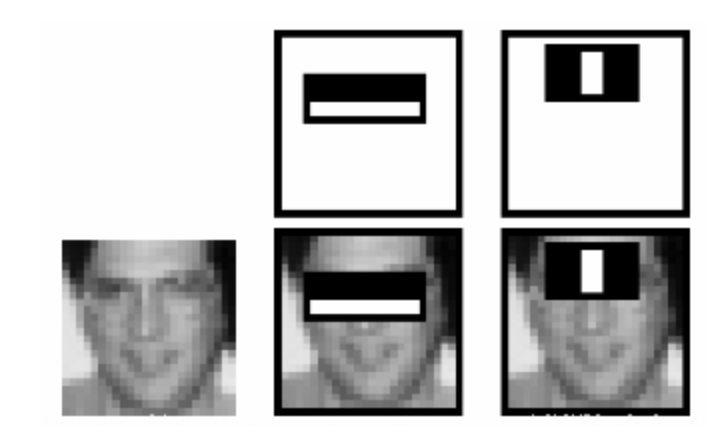
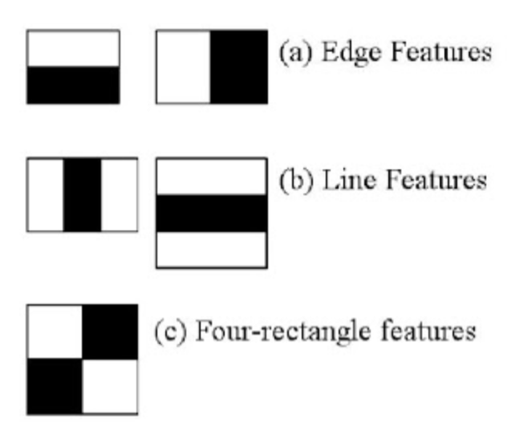


Figure 2: Left, an example of Haar cascade features. Right, the adoption of Haar cascade features in detecting a face (OpenCV, 2018).

From this point we are able to improve the performance of detections using an algorithm called Adaboost (P. a Viola and Jones, 2001), which allows ranking of features based on their error rate when attempting to successfully identify the object.

Further work has been proposed in this field using a deep learning based approach (Lecun, Bengio and Hinton, 2015). However, to make this approach feasible within real-time constraints specialized GPU (Graphics Processing Unit) hardware must be used, which would limit the extensibility of any proposed framework making use of these techniques.

### Object Tracking Techniques

Once we have accurate object detection we then need to be able to identify the same object seen through multiple frames of the video stream, employing tracking techniques. Tracking works on an online basis, meaning as new sequential data becomes available to the model, it can adapt at runtime to provide improved predictions on future data. Usually, this incorporates identifying an objects appearance and motion pattern allowing the accurate predictions of an objects location as it moves. When providing object tracking capabilities a variety of algorithms have been developed and made available (Table 1).

|  |  |  |  |
| --- | --- | --- | --- |
| 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 (Baben and Belongie, 2009). | 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 (Kalal, Mikolajczyk and Matas, 2010). | 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 (Kalal, Mikolajczyk and Matas, 2012). | Works well under long periods of occlusion. | The tracker does not perform well under large full rotations and can create lots of false positives. |

Table 1: A comparison of three common approaches to object tracking.

### Behavior and Activity Analysis

Detection techniques.

## Anomaly Detection with Machine Learning

This will be a talk on anomaly detection and machine learning.

### Anomaly Detection Models

Talk about the models specifically and research done into them.

### The Impact of Human Behavior

Talk about how humans may affect ability to detect anomalies.

## Distributed Computing and the Cloud

In order to provide video stream analysis at scale a large amount of computing power is required. Processing video with real-time requirements requires keeping the latency of any system under a desired threshold, even when presented with high volumes of data. In recent years, the availability of processing power has become more widely available, and cheap to access, through Cloud computing suppliers. This has enabled the adoption of distributed computing technologies that allow the parallel completion of tasks, enabling video processing to be possible on a large scale. Tools developed in recent years utilize distributed computing, and in this section, we will discuss their use and application in the field of computer vision.

### Parallel Computing

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. Two of the most popular frameworks that supply the capability of horizontally scalable distributed computing are Apache Storm (Apache, 2018) and Apache Flink (Apache, 2018a).

Apache Storm is a real-time distributed stream data processing engine that, when given a source of data events, known as a spout, is able to perform a desired set of transformations to each event in parallel. 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. The execution of bolts happens within a worker process, with the management of its state controlled by a Zookeeper service. In order to track an events progress through a topology Apache Storm monitors the progress of the bolts for an event, allowing Apache Storm to provide ‘at least once’ processing of events. Apache Storm then uses a service known as Nimbus to schedule and maintain the flow of data through a topology. This architecture gives Apache Storm power to process billions of events across a distributed set of machines in a single day (Toshniwal *et al.*, 2014).

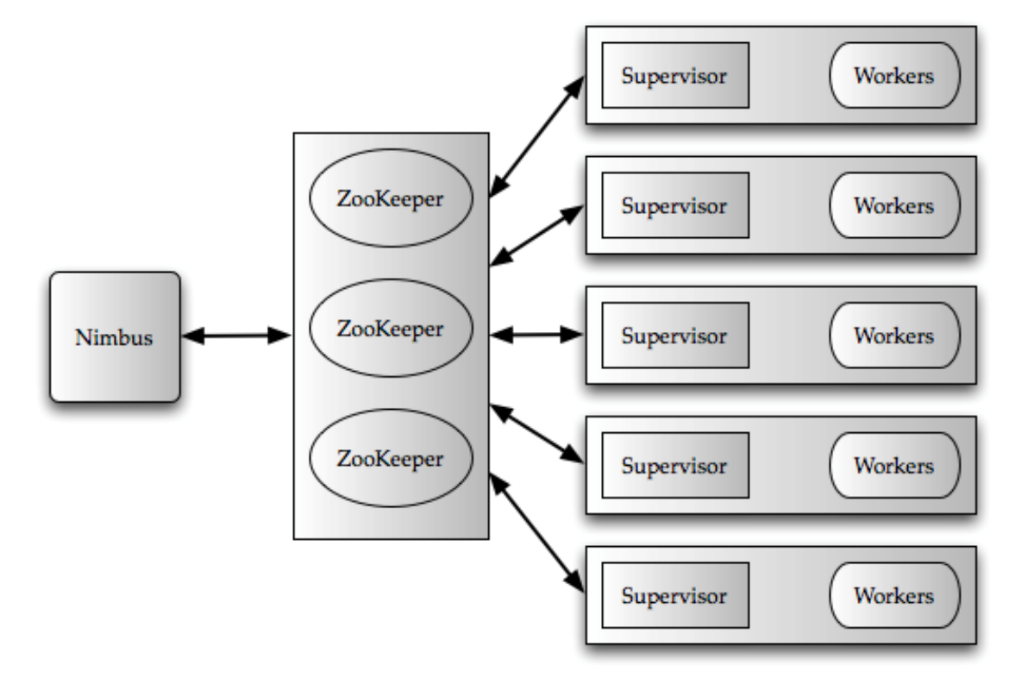


Figure 3: A high level architecture overview of Apache Storm (Toshniwal *et al.*, 2014).

A competitor to Apache Storm is the Apache Flink framework. Apache Flink is a system for processing streaming and batch data. Similar to Apache Storm, it works by expressing a data processing pipeline as a series of tasks to apply to each event as a directed graph. An Apache Flink architecture contains the core components of; a client, a Job Manager and at least one Task Manager (Figure 4). The client takes user code and transforms it into a dataflow graph, similar to Apache Storm’s topology. The Job Manager then coordinates the distributed execution of the dataflow, tracking state and progress of each operation within the stream. This then leaves the actual data processing execution to the Task Managers, reporting the result of any operation to the Job Manager. Differing from Apache Storm, 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 (Carbone *et al.*, 2015).

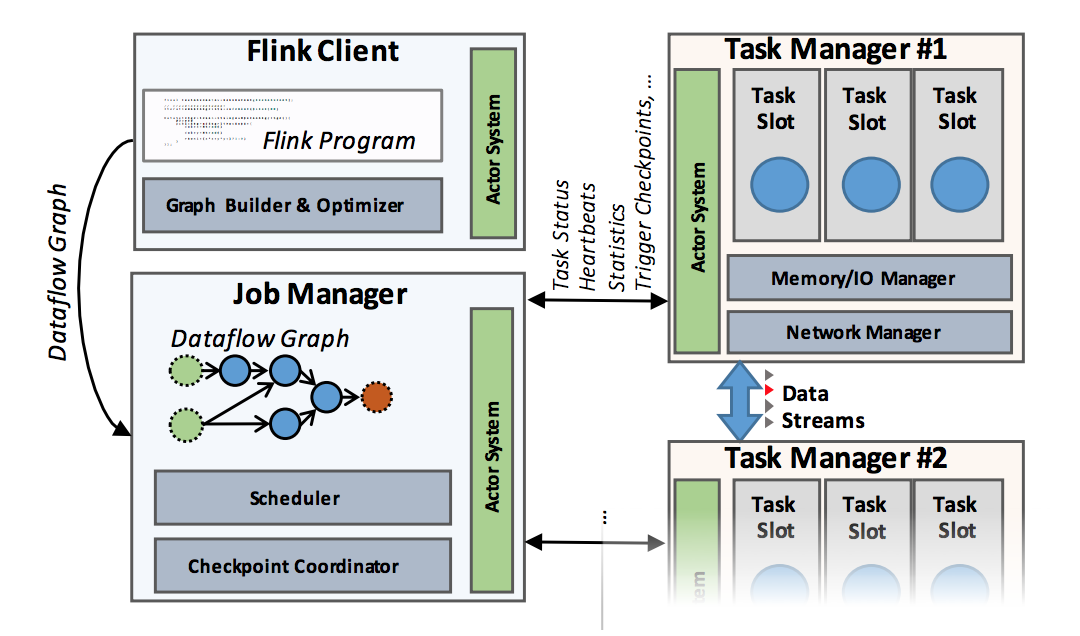


Figure 4: The Apache Flink processing model, showing the architecture adopted from an initial program definition to execution within an Apache Flink cluster (Carbone *et al.*, 2015).

Performance comparisons between Apache Storm and Apache Flink exist as they offer the same stream processing service. Under high workloads, it has been found that Apache Flink out performs Apache Storm marginally with regards to latency, with both systems producing similar throughputs (Chintapalli *et al.*, 2016). However, both systems are in their infancy of development, therefore full conclusion about the final performance of each system are hard to make.

### Distributed Messaging

Within any video processing framework, a key success criterion is the ability to decouple the different stages of processing, allowing for new features to be added without effecting other stages of the 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 allows for causality of events to be tracked and understood, which is a key area of analysis. Further to this, this functionality should be able to scale to multiple video stream inputs without effecting real-time performance requirements. Providing these capabilities is Apache Kafka (Apache, 2018b).

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, known as brokers, within an Apache Kafka cluster. Communication with Apache Kafka is done through a high-performance language agnostic TCP protocol, meaning any system can speak with it. Message queues, known as topics, are an immutable log of messages stored over multiple brokers. Clients can subscribe to topics, allowing for real-time processing of messages as they occur. Apache Kafka’s fault tolerant storage makes it suitable as a storage system as well as a messaging system, meaning real-time and traditional off-line processing can be performed. This enables a large variety of operations to be performed on the data held within Apache Kafka, making it a suitable choice for an extensible video processing framework. 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).

The adoption of Apache Kafka will allow the widest variety of applications to be supported by a proposed video processing framework, and therefore is the optimum technology for providing extensibility and scalability.



Figure 5: A comparison of Apache Kafka’s producer performance (Left), and its consumer performance (Right) against competitor messaging systems (Kreps, Narkhede and Rao, 2011).

### Cloud Computing

To enable a horizontally scaling framework for video processing, a range of computing power is required. For applications working with a single video stream a smaller amount of computational power is required than applications performing analysis on hundreds of concurrent video streams. To enable this elasticity within a proposed framework, Cloud Computing can be adopted.

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.

Cloud computing isn’t without it risks however, with rented virtualized hardware performance of individual systems may be unknown. As a client of a Cloud provider, you are also at the mercy of their availability, if their service experiences a fault you are incapable of fixing it yourself. Further considerations to make can include, data confidentiality, software licensing costs and runtime renting costs (Armbrust *et al.*, 2009).

Even with these considerations, Cloud Computing offers such elasticity and extensibility that supporting the deployment of a proposed framework to a Cloud Provider can offer great benefits. As Cloud Computing becomes more highly adopted in the future the service offered will improve, unlike the physical purchase of hardware to run a service, bringing greater benefit to users of a proposed framework. These benefits therefore make Cloud Computing a staple architecture design within a scalable and extendable video processing framework, enabling users the option to deploy as much or little computing power required to meet their requirements.

## Existing Technologies and Approaches

|  |  |  |  |
| --- | --- | --- | --- |
| Product | Overview | Functionality | Limitations |
| Nest (Nest, 2017) | 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. | Requires specialized hardware, is not extendable for development. |
| A Video Analysis Framework for Surveillance System (Suvonvorn, 2008) | 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. | Does not make use of distributed computing so will be unable to scale to all user requirements. Further to this, the product is developer in .NET, requiring a Microsoft workstation to run. |
| DiVA (SanMiguel *et al.*, 2008) | 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. | 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 2: A list of current technologies in the market that aim to provide smart CCTV.

References

Apache (2018) *Apache Flink*. Available at: https://flink.apache.org/ (Accessed: 19 February 2018).

Apache (2018) ‘Apache Kafka’. Available at: https://kafka.apache.org/.

Apache (2018) *Apache Storm*. Available at: http://storm.apache.org/ (Accessed: 19 February 2018).

Armbrust, M. *et al.* (2009) ‘Above the clouds: A Berkeley view of cloud computing’,  *University of California, Berkeley, Tech. Rep. UCB* , pp. 07–013. doi: 10.1145/1721654.1721672.

Baben, B. and Belongie, S. (2009) ‘Visual tracking with online Multiple Instance Learning’, *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 983–990. doi: 10.1109/CVPR.2009.5206737.

Carbone, P. *et al.* (2015) ‘Apache Flink: Unified Stream and Batch Processing in a Single Engine’, *Data Engineering*, 36, pp. 28–38. doi: 10.1109/IC2EW.2016.56.

Chintapalli, S. *et al.* (2016) ‘Benchmarking streaming computation engines: Storm, flink and spark streaming’, in *Proceedings - 2016 IEEE 30th International Parallel and Distributed Processing Symposium, IPDPS 2016*, pp. 1789–1792. doi: 10.1109/IPDPSW.2016.138.

Kalal, Z., Mikolajczyk, K. and Matas, J. (2010) ‘Forward-backward error: Automatic detection of tracking failures’, in *Proceedings - International Conference on Pattern Recognition*, pp. 2756–2759. doi: 10.1109/ICPR.2010.675.

Kalal, Z., Mikolajczyk, K. and Matas, J. (2012) ‘Tracking-learning-detection’, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7), pp. 1409–1422. doi: 10.1109/TPAMI.2011.239.

Ko, T. (2008) ‘A survey on behavior analysis in video surveillance for homeland security applications’, *Applied Imagery Pattern Recognition Workshop, 2008. AIPR ’08. 37th IEEE*, pp. 1–8. doi: 10.1109/AIPR.2008.4906450.

Kreps, J., Narkhede, N. and Rao, J. (2011) ‘Kafka: a Distributed Messaging System for Log Processing’. Available at: http://notes.stephenholiday.com/Kafka.pdf.

Lecun, Y., Bengio, Y. and Hinton, G. (2015) ‘Deep learning’, *Nature*, pp. 436–444. doi: 10.1038/nature14539.

Nest (2017) *No Title*. Available at: https://nest.com/uk/cameras/nest-cam-indoor/overview/ (Accessed: 24 November 2017).

Norris, C. and McCahill, M. (2006) ‘CCTV: Beyond penal modernism?’, *British Journal of Criminology*, 46(1), pp. 97–118. doi: 10.1093/bjc/azi047.

OpenCV (2018) *Facial Detection with Haar Cascades in OpenCV*. Available at: https://docs.opencv.org/3.3.0/d7/d8b/tutorial\_py\_face\_detection.html (Accessed: 30 January 2018).

SanMiguel, J. C. *et al.* (2008) ‘DiVA: A Distributed Video Analysis Framework Applied to Video-Surveillance Systems’, in *International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS)*, pp. 207–210. doi: 10.1109/WIAMIS.2008.29.

Suvonvorn, N. (2008) ‘A video analysis framework for surveillance system’, *2008 IEEE 10th Workshop on Multimedia Signal Processing*, pp. 867–871. doi: 10.1109/MMSP.2008.4665195.

Toshniwal, A. *et al.* (2014) ‘Storm@twitter’, in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data - SIGMOD ’14*, pp. 147–156. doi: 10.1145/2588555.2595641.

Viola, P. a and Jones, M. J. (2001) ‘Fast and Robust Classification using Asymmetric AdaBoost and a Detector Cascade’, *Advances in Neural Information Processing System*, (December), pp. 1311–1318. Available at: https://papers.nips.cc/paper/2091-fast-and-robust-classification-using-asymmetric-adaboost-and-a-detector-cascade.pdf.

Viola, P. and Jones, M. (2001) ‘Robust real-time object detection’, *International Journal of Computer Vision*, 57(2), pp. 137–154. doi: http://dx.doi.org/10.1023/B:VISI.0000013087.49260.fb.

Yard, S. (2010) *CCTV in Homicide Investigations*. Available at: https://goo.gl/oS5Tgn (Accessed: 15 November 2017).

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.