Journal of Information Systems Engineering and Management
2025, 10(16s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Dynamic Data Stream Management: Real-Time Unique Frameset Generation in Unsupervised Residential Surveillance

Mintu Movi[1*], Abdul Jabbar P[2], Noufal K P[3], Bindu V R[4]
[1]*Research Scholar, School of Computer Sciences, Mahatma Gandhi University, Kottayam 686560, Kerala, India*
[2]*Assistant Professor, Graduate School, Stamford International University, Prawet, 10250, Bangkok, Thailand*
[3]*Deputy Director, Information Kerala Mission, Public Office, Thiruvananthapuram, 695033, Kerala, India*
[4]*Professor & Head, School of Computer Sciences, Mahatma Gandhi University, Kottayam 686560, Kerala, India*
[1]*mintu@mgu.ac.in ,[2]abduljabbar.perumbalath@stamford.edu, [3]noufal@ikm.gov.in, [4]binduvr@mgu.ac.in*
[1]0009-0002-0104-0875, [2]0000-0002-7374-6626, [3]0009-0007-5654-1200
[4]0000-0002-1447-8406 Corresponding Author*: Mintu Movi

ARTICLE INFO

Received: 01 Dec 2024

Revised: 25 Jan 2025

Accepted: 08 Feb 2025

ABSTRACT

Surveillance systems often struggle with managing dynamic spatial-temporal data streams due to information's vast volume and velocity. To address this, we introduce a novel self-acting framework tailored explicitly for generating unique framesets in an unsupervised video environment. Our approach leverages efficient data management techniques to extract and compress video data without losing critical details while filtering out irrelevant duplicated data, thereby enhancing real-time monitoring and analysis capabilities. We validate our methodology using a substantial dataset of one terabyte of surveillance footage, from which 26,454 unique frames were extracted, covering various activities and interactions within home premises, from daily chores to social interactions. The results demonstrate significant improvements in processing speed and data reduction, enabling better resource management. This research contributes to more effective home security solutions by providing a streamlined approach to managing dynamic data streams and identifying patterns often overlooked by traditional surveillance methods.

Keywords: Data stream management, Video data extraction, Surveillance system, Real-time analysis.

INTRODUCTION

Video data has become a pivotal resource in various sectors, particularly enhancing security and surveillance measures. Managing this data efficiently is crucial to extracting actionable insights [1] and ensuring timely responses to security threats. The data source in a surveillance system captures continuous footage that needs to be stored, processed, and analyzed [2]. Managing this video data cannot be overstated, as it directly impacts the ability to detect anomalies, identify suspicious activities, and provide real-time alerts [3]. Effective video data management involves organizing and storing the relevant data efficiently. This underscores the necessity of developing robust methodologies for handling unsupervised data's dynamic and extensive nature [4]. Dynamic data streams refer to continuous flows of data generated in real-time and can change rapidly as new information is captured. Managing dynamic data streams involves handling large volumes of incoming data [5], ensuring real-time processing, and making the data available for immediate or near real-time [6] analysis and interpretation [7]. Combining data from multiple cameras to create a cohesive view of events is complex. Extracting meaningful insights from spatiotemporal [8] data that correlate events across different times and locations involves sophisticated analytical techniques and tools [9]. This research investigates the necessity and significance of data management in surveillance systems by examining the intricacies of data storage, resource optimization, and analytical enhancements [10]. The main objectives of the proposed research work are,

- The study introduces a new methodology to manage dynamic data streams efficiently.

- This study provides a practical solution for managing large-scale video data by automating unique video data extraction, significantly improving resource management without compromising critical information.
- The research targets home security, where surveillance data is continuously generated, and the method has been validated using 1TB of primary surveillance data.

METHODS

The proposed methodology focuses on automating the management of dynamic data streams tailored explicitly for surveillance applications. The methodology systematically handles the ingestion, processing, and storage of video data, ensuring that critical information is captured and preserved without manual intervention from initial acquisition to final archiving. Algorithm 1 extracts frames from video files and saves them as individual images. For each video file in the input directory, Algorithm 1 checks if the file is in video format. If so, it creates a subfolder in the output directory to store the frames for that specific video. The VideoCapture function reads the video frame by frame. Each frame is then saved in the corresponding subfolder. The process continues until all frames from the video are extracted. Algorithm 2 is designed to identify and retain unique images by removing similar ones based on histogram comparisons. Initially, Algorithm 2 defines the paths to the relevant input-output directories. The folder path variable is set to the location within storage where the duplicate images are stored. Additionally, an empty list named unique images is initialized to keep track of the histograms of images identified as unique. This list is a reference for comparing new images against those already deemed unique. Algorithm 2 then iterates over every file. For each file, it checks whether the filename ends with the image extension. If so, the image is read into memory. To standardize the comparison process, each image is resized to a fixed dimension of 256x256 pixels, which helps maintain consistency. After resizing, the image is converted to a gray-scale or color [11]. This conversion simplifies the histogram calculation by reducing the image data to color channels, which is sufficient for comparing similarities. The methodology then calculates the histogram of the gray-scale or color image using Algorithm 3, focusing on the intensity distribution across 256 bins. The histogram is normalized and flattened into a one-dimensional array to facilitate an accurate comparison. This comparison uses the Bhattacharyya distance metric, which quantifies the similarity between two histograms, where a lower value indicates more remarkable similarity. A threshold of 0.07 is set; if the similarity score falls below this threshold, the image is considered similar [12]. If the image is determined to be unique, its histogram is appended to this list for future comparisons. Subsequently, Algorithm 2 ensures the output directory designated for unique images exists. The unique image is then saved to this directory with a modified filename. Algorithm 2 dynamically manages directories throughout the process, ensuring that input and output paths are correctly structured and accessible. The proposed methodology effectively filters out highly similar images by systematically resizing, converting, and comparing images based on their histograms, thereby maintaining a collection of unique visuals [13].

Algorithm 1 Extracting Frames from Videos

Require: input path (Folder containing videos), output path (Folder to store extracted frames)

Ensure: Extract frames from each video in the input directory and store them in the output directory.

1: Mount the storage

2: Ensure the output directory exists:

3: if output path does not exist then

4: Create the output directory output path

5: end if

6: Loop through all files in the input directory:

7: for each filename in input path do

8: if filename ends with .dav then

9: Define video path as the path to the video file

10: Open the video file: VideoCapture(video path)

11: Initialize count to 0

```
12:             Define video output  path as the subfolder to store frames for this video
13:             if video  output path does not exist then
14:                 Create the subfolder video output path
15:             end if
16:             Extract frames from the video:
17:             while video has more frames do
18:                 Read the next frame from the video
19:                 if frame read is successful then
20:                     Define frame  path as the path to save the current frame
21:                     Save the frame to frame path
22:                     Increment count
23:                 else
24:                     Break the loop
25:                 end if
26:             end while
27:     Print the message: print(f'Number of frames for {filename}:', count)
28:     end if
29: end for
30: function VideoCapture(video path)
31:     Initialize VideoCapture object: cap ← source
32:     if ¬ cap is Opened then
33:             Output: "Error: Could not open video source." return
34:     end if
35:     Loop:
36:     while True do
37:             Read a frame: ret,frame ← cap
38:             if ¬ret then                    ▷ No frame returned (end of video)
39:                 Break                       ▷ Exit the loop
40:             end if
41:             Process frame:
42:             Display the frame
43:             if waitKey(1)&0xFF == ord('q') then    ▷ Check for exit key
44:                 Break
45:             end if
46:     end while
47:     Release resources and
```

48: Close all windows

49: end function

Algorithm 2 Removing Duplicate Images and Extracting Unique Ones

Require: folder path (Path to the folder with images), save path (Path to store unique images)

Ensure: Detect and save unique images based on histogram comparison.

1: Mount the storage

2: Initialize list unique images to store histograms of unique images.

3: Loop through all image files in folder path:

4: for each filename in folder path do

5: if filename ends with .jpg then

6: Load image img from filename

7: Resize img to (256, 256)

8: Convert img to grayscale or color

9: Calculate histogram of the grayscale or color image using CalculateHistogram(image, channels, mask, histSize, range)

10: Normalize the histogram using NormalizeHistogram(hist, normType)

11: Flatten the histogram for comparison

12: Check similarity of the current image with previous unique images:

13: Initialize similar to False

14: for each histogram, hist ref in unique images do

15: Compare hist and hist ref using CompareHistograms(hist, hist-ref, method) with Bhattacharyya distance

16: if similarity < threshold (0.07) then

17: Set similar to True

18: Break the loop

19: end if

20: end for

21: if similar is False then

22: Add current histogram hist to unique images

23: Save the unique image to the output folder:

24: if save path does not exist then

25: Create save path

26: end if

27: Define the output file name: save name = filename + ' unique.jpg'

28: Save image to save path

29: end if

30: end if

31: end for

Algorithm 3 Histogram Calculation, Normalization, and Comparison

1: function CalculateHistogram(image, channels, mask, histSize, range)
2: Input:
3: image: Input image
4: channels: Channels to calculate histogram for (e.g., grayscale or color)
5: mask: Region of interest (optional, can be None)
6: histSize: Number of bins (e.g., 256)
7: range: Pixel intensity range (e.g., 0 to 255)
8: 1. Split the image into specified channels: ch ← split(image)
9: 2. Count the number of pixels for each intensity value in channels
10: 3. Create an array hist of size histSize, where each element stores the count of pixels for the corresponding intensity value
11: 4. If a mask is provided, only consider pixels inside the mask when counting
12: 5. Return the computed hist for the image
13: return hist
14: end function
15: function NormalizeHistogram(hist, normType)
16: Input:
17: hist: The histogram to normalize
18: normType: Type of normalization (e.g., cv2.NORM MINMAX)
19: 1. Calculate the minimum and maximum values in the hist:
20: min_val←min(hist)
21: max_val←max(hist)
22: 2. If using NORM MINMAX, scale all values to be between 0 and 1:
23: For all elements i in hist:
24: hist[i]←hist[i]-min_val / max_val-min_val
25: 3. Return the normalized hist
26: return hist
27: end function
28: function CompareHistograms(hist1, hist2, method)
29: Input:
30: hist1: The first histogram
31: hist2: The second histogram
32: method: The comparison method (e.g., Bhattacharyya distance)
33: 1. Compute the sum of products of corresponding bins from hist1 and hist2:

34: s ← sum of √ hist1[i] · hist2[i]

35: 2. Use the formula for Bhattacharyya distance:

36: d← √1 – ((1 / h¯1 · h¯2 · n2) s)

37: where h¯1 and h¯2 are the mean values of the two histograms and n is the number of bins.

38: 3. Return d as the Bhattacharyya distance between hist1 and hist2

39: return d

40: end function

Residential Activity Capture Dataset: A detailed and diverse collection of CCTV footage datasets is required to enable effective data stream management tailored to home environments. RACD, comprising 26,454 unique frames extracted from 1TB of primary data collected over a month, has been curated to facilitate this. The dataset was collected from six distinct channels labeled A1 to A6. Channels A1, A2, and A3 capture footage from the front area of the home premises, while channels A4, A5, and A6 cover the restricted area. The dataset encompasses various categories, ranging from routine activities to specialized visits. RACD consists of 12 classes. A total of 672 hours of CCTV footage was collected. From this, 63 videos were selected to cover various activities within the premises. Each video, with a duration of 1 hour, generates an average of 50,000 frames using Algorithm 1. After processing, 26,454 unique frames were extracted using Algorithms 2 and 3.

Running Platform: The research study utilized the NVIDIA T4 GPU, a versatile and efficient GPU designed for machine learning, deep learning, and other high-performance computing tasks. It is part of the NVIDIA Tesla series and is based on the Turing architecture, which offers substantial improvements in performance and efficiency over previous generations [14]. The T4 GPU includes 2,560 CUDA cores and 320 Tensor Cores, specifically engineered to accelerate deep learning workloads by performing mixed-precision matrix computations much faster than general-purpose cores [15].

RESULTS AND DISCUSSION

Table 1 provides a detailed comparison between the original Data Stream (raw footage) and the extracted data stream (processed footage) across various activity classes, such as waste collection, postal and utility services, guest visits, and restricted area monitoring, highlighting significant reductions in frame count, run time, and size. The original data is large and resource-intensive, with lengthy runtimes and high storage demands. After processing, the extracted data retains only the most relevant frames, resulting in substantial reductions. This optimization makes data more manageable and efficient for anomaly detection or rare event identification in CCTV footage.

Table 1 Comparison of Original and Processed Data Streams

Classes	Original Data Stream			Extracted Data Stream		
	Frame Count	Run Time (Minutes)	Size (MB)	Frame Count	Run Time (Minutes)	Size (MB)
Waste Collection	150494	180	1400	1273	30	660
Postal and Utility Services	148393	180	1340	1238	27	651
Guest Visits	293075	360	2680	2487	60	1280
Family Interactions	301350	360	2700	2606	62	1300
Household Services	296149	360	2690	2534	61	1290
Vehicle Services	148751	180	1340	1232	27	648
Financial Transactions	298396	360	2690	2568	61	1290
Newspaper Delivery	150534	180	1400	1288	30	658
Delivery Services	149125	180	1400	1293	30	660

Security Incidents	146420	180	1340	1229	27	648
Restricted Area	146420	1,260	9500	8706	210	4610

CONCLUSION

The proposed methodology and primary dataset aim to enhance the reliability and accuracy of surveillance solutions, providing a robust foundation for data stream management. Across all classes, the processed data streams exhibit a notable reduction in size and processing time, reflecting the algorithm's robust ability to manage and condense large volumes of data efficiently. The successful extraction and compression of data streams without compromising the integrity of critical information demonstrate the method's practical applicability in real-time systems. The data reduction highlights the effectiveness of duplicate removal and focusing on unique elements. This can be crucial for applications that require immediate response and decision-making.

REFERENCES

[1] Wu, Y., Wang, X., Chen, T., Dou, Y.: Da-resnet: dual-stream resnet with attention mechanism for classroom video summary. Pattern Analysis and Applications 27, 32 (2024) https://doi.org/10.1007/s10044-024-01256-1.

[2] Liu, N.: Cctv cameras at home: Temporality experience of surveillance technology in family life. New Media and Society (2024) https://doi.org/10.1177/14614448241229175. University of Leicester, UK

[3] Agarwal, S., Reddy, C.R.K.: A smart intelligent approach based on hybrid group search and pelican optimization algorithm for data stream clustering. Knowledge and Information Systems 66, 2467–2500 (2024) https://doi.org/10.1007/s10115-023-02002-5.

[4] Kaur, N., Rani, S., Kaur, S.: Real-time video surveillance based human fall detection system using hybrid haar cascade classifier. Multimedia Tools and Applications (2024) https://doi.org/10.1007/s11042-024-18305-w.

[5] Malarvizhi, C., Dass, P., Karthikeyani, P., Sudha, V., Iniyan, S.: Machine Learning and Advanced Technology Based Fire Detection. IEEE, Bengaluru, Karnataka, India. International Conference on Intelligent and Innovative Technologies in Computing, Electrical and Electronics (IITCEE) (2023). https://doi.org/10.1109/IITCEE57236.2023.10090923.

[6] Singh, A., Tiwari, R.K.: AIGuard: Criminal Tracking in CCTV Footage using MTCNN and ResNet. IEEE, Noida, Uttar Pradesh, India. 14th International Conference on Cloud Computing, Data Science and Engineering (Confluence) (2024). https://doi.org/10.1109/CONFLUENCE60223.2024.10463292.

[7] Darwante, N.K., Sonawane, A., Surlakar, A., Prahlad, R.V., William, P.: Hybrid Model for Robotic Surveillance using Advance Computing Techniques with IoT. IEEE, Pune, India. 3rd International Conference on Pervasive Computing and Social Networking (ICPCSN) (2023). https://doi.org/10.1109/ICPCSN58272.2023.00219.

[8] Shien, G.Y., Shanmugam, K., Rana, M.E.: Automated Face Mask Detection using Artificial Intelligence and Video Surveillance Management. IEEE, Baghdad, Anbar, Iraq. 15th International Conference on Developments in eSystems Engineering (DeSE) (2023). https://doi.org/10.1109/DESE58274.2023.10099878.

[9] Samuda, P., K, S., N.G., P., C., N., D., K., C., L.: Low-cost Prototype for IoT-based Smart Monitoring through Telegram. IEEE, Tirunelveli, India. 5th International Conference on Smart Systems and Inventive Technology (ICSSIT) (2023). https://doi.org/10.1109/1CSSIT.55814.2023.10061155.

[10] Mathew, D., G, K., K, V., I, E.B., R, S.K.: System for Detecting Intrusions using Raspberry PI. IEEE, Coimbatore, India. International Conference on Computer Communication and Informatics (ICCCI) (2023). https://doi.org/10.1109/ICCCI56745.2023.10128487.

[11] Markad, A., Phadke, S., Shah, P., Pawar, R.: Real-Time CCTV Face Recognition Model. IEEE, Pune, India. 7th International Conference on Computing, Communication, Control And Automation (ICCUBEA) (2023). https://doi.org/10.1109/ICCUBEA58933.2023.10392020.

[12] Kumar, D.S., K, S.A., J, V.: Advanced Home Surveillance Using OpenCV and ML. IEEE, Chennai, Tamil Nadu, India. Intelligent Computing and Control for Engineering and Business Systems (ICCEBS) (2023). https://doi.org/10.1109/09/ICCEBS58601. 2023.10448826

[13] Guesmi, A., Hanif, M.A., Ouni, B., Shafique, M.: Physical adversarial attacks for camera-based smart systems: Current trends, categorization, applications, research challenges, and future outlook. IEEE Access (2023) https://doi.org/10.1109/ACCESS.2023.3321118.

[14] Singh, A., Sharma, K., Mehta, P.: Novel algorithms for compressing video data without losing critical details. In: 14th International Conference on Cloud Computing, Data Science and Engineering (Confluence) (2024)
[15] Li, Y., Wu, C., Li, W., Tsung, F., Guo, J.: Dynamic modeling and online monitoring of tensor data streams with application to passenger flow surveillance. Annals of Applied Statistics 18(3), 1789–1814 (2024) https://doi.org/10.1214/23-AOAS1845.