Study on Design and Implementation of Distributed Multiple Camera Surveillance and Tracking System

Sureshkumar Jha [1*], *Rohan Sawant*[1], and *Parth Shinde*[1] *Rakhi Kalantri*[2] *Shagufta Rajguru*[2]

[1]*BE Students, Department of Computer Engineering, Fr Conceicao Rodrigues Institute of Technology, Vashi Navi Mumbai, India.*
[2]*Assistant Professor, Department of Computer Engineering, Fr Conceicao Rodrigues Institute of Technology, Vashi Navi Mumbai, India.*

Abstract. This work proposes a system to successfully track, identify and tag target objects or individuals in a real time environment. Consider a sequence of cameras installed in an environment or a facility. Through these cameras user can track those intruders who are aware of how the surveillance system operates and are actively trying to avoid getting seen by the surveillance system. This system tracks the movement of any person's movement and record and maintain that data throughout any facility in which such a solution is deployed. Movement logging of any individual person can also be done if it does not infringe his/her privacy. Design and analysis a distributed algorithm for the optimization of the recognition and mapping of the given subject/subjects on a User Interface. Finally, the performance and robustness of the distributed system is further analyzed through continuously training the algorithm or maybe real time demo.
Keywords— Movement Logging, Recognition, Distributed Systems, Real-Time Introduction.

1. INTRODUCTION

This Paper guides a stepwise walkthrough of an intelligent system that detects and captures motion information of moving targets for accurate object tracking in real-time. The classified object is being tracked for high-level analysis. This paper provides a detailed description on how to detect humans, recognize their faces, inference the unrecognized and recognized people and through that detect their movement. The human detection in a smart surveillance system aims at making distinctions among moving objects (i.e., specifically humans in this case) in a video sequence. Human detection may be a difficult task from a computer vision perspective because it is influenced by a good range of possible appearance thanks to changing articulated pose, clothing, lighting and background. Combined assumptions were made on the environment to be monitored, the cameras, and thus the targets. The environment of the facility is assumed to be divided into multiple layers (i.e., floors) and assume that each floor is a corridor and is often completely observed by a sequence of cameras. The matter of perimeter surveillance could also be a special case of this framework. The cameras are assumed to be subject to physical constraints, e.g., limited field of view and to be equipped with a low-level routine to detect intruders/targets that fall within the field of vision of a camera. Humans are assumed to be randomly appearing in the video sequences, within the sense that they have no idea of the cameras' configuration at any instant, and prediction of their trajectory is not possible. The gate camera must have proper vision of the person and possibly his face in an entry video sequence.

1.1. Problem Description and Formulation

This project deals with the implementation of a real-time dynamic closed-circuit television, which tracks both the known and unknown targets and classifies them accordingly. This classification and detection of such targets is visualized using a 2d Map or blueprint of the whole area. This helps in tracking of the targets within the facility.

The whole system relies upon the synchronization of the cameras to trace a specific target. This results in easy capture of any "red flag" activity of any target. 'RED FLAG' activities can constitute entry of the person through other means, rather than the designated entry, this may be immediately raised if the given target is an unknown entity. This flag is also raised when any valued object goes missing. The trajectory of the camera is additionally considered during the analysis and processing of the varied video streams coming from different cameras.

The facial data of any unknown person entering the power is merely relevant till a selected period of time, then its erased, only frequent visitors are logged into the database with their approval. This is done to make sure the privacy of the target is additionally not compromised

The visualization contains a map of the corridor or floor, or by making use of green or red bounding boxes, through which the inference of the algorithm is shown. This visualization concept also can expand vastly, depending on the sort of study or level of analysis the user requires.

1.2. Related Work

Some examples of the already existing software systems for the two are following: -

1.2.1. Categorization of Target Tracking Systems

First of all, target tracking systems can be classified into on-line real-time and off-line batch systems. Real time tracking system can be classified into different types. For the differentiation of the classes; Firstly, the video stream is provided to the software. Once the object is detected in the frame, the object is later classified on the basis of shape-based

method, motion-based method and texture-based method, after running these modules it can determined whether the object in the frame is an object or a human.

1.2.2. Design and Implementation of Human Tracking using CCTV Cam

Closed Circuit Television (CCTV) technology made a big impact on how crimes were solved. Suppose a wanted person is in the vicinity, the authorities only have to give a dataset of the person's face. Once the person's face is seen in an ip camera node, it passes the data to the neighboring node thus tracking the person.

1.2.3. Intrusion Detection Using Image Processing Techniques

Through this monitoring system, user can easily detect the intrusion happening in a particular area. The software will be given a mapped data of the area and the people authorized in the area. If a person recognized by the software enters the area it would be safe, but if unauthorized personnel enter the area the system raises the intruder alert and stores the unknown face in the dataset and tracks the person in the overall premises.

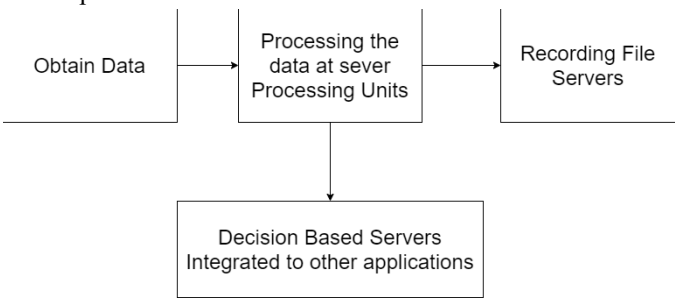

fig 1: Data collection and comparison for intrusion detection

1.3. Contribution

The above listed data are the various modules which are building blocks of the product that are going to be developed. The close circuit television (CCTV) technology is referred while developing the module to track the person's motion in the vicinity. Then the intruder alert system is referred to, which detects an unknown face in the vicinity. In the system, user is tracking a person in a particular vicinity and storing the data of the person in the server. This data is passed from one node to another node and tracks the person more efficiently, it also considers a person entering the area where the system is active. The person and the motion of the person is tracked by the yolov4 framing the bounding boxes around the body of the person. Then the identity of the person is determined using the deep sort and the mtcnn which stores the different characteristics of a person's face and creates bounding boxes around the face. Through this, it is determined whether the person is an intruder or regular by processing the face and streaming it simultaneously, this is achieved with the help of the multi-threading system where each work is considered as threads and different types of threads are executed simultaneously thus, reducing the system upgrades and speed of processing, and streaming the video. An unknown person will be red flagged and will be tracked and the face will be stored on the server for tracking the person through the vicinity.

2. SYSTEM OVERVIEW

The system is mainly divided into a Subordinate, Superordinate Systems, and an external input (mobile app). The Superordinate system interacts with the actual raw data stream coming into the surveillance software, while the subordinate system generally deals with the internal flow and interaction between the various components of the superordinate system.

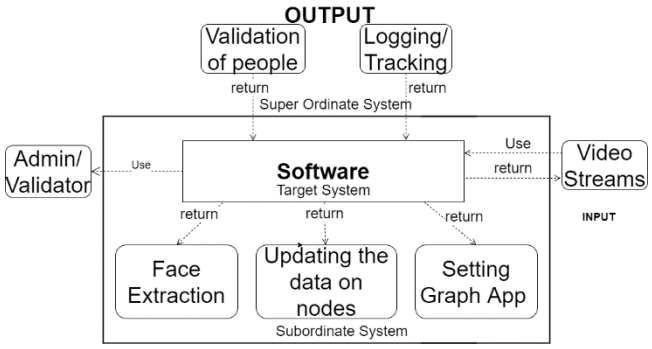

fig 2: basic architecture of the system where data is collected stored and processed.

The Superordinate System is a central object which receives the input video streams and the graph generated through the mobile app, it is responsible for preprocessing this data and passing it through various processing blocks such as feature extractor (can be both face and person, or either face only or person only), Feature Comparer, Tracker and Visualizer.it also handles the message/data passing between the Gate camera its children's nodes. Additionally, it performs regular checks if the graph needs setting or resetting whenever a new camera is added. It also checks whether a new face is added to the data pool and effectively pipelines all the processes with the respect to the newly updated data pool. It also sends alerts for validation to the User/Admin (i.e., Security Personnel of the facility). The Subordinate System is an object which handles the flow and interaction between all the different processes. It is responsible for maintaining proper flow of the system between its sub-modules. It thus helps the Superordinate maintain the proper pipeline for processing data and inferencing it to the output display

This system consists of the following processes:

1. Feature extractor
2.Feature Comparer
3.Visualizer

These also use sub modules such as Tracker and Parrallelizers

2.1. Flow of the System

To understand the Practical application of the project it is necessary to understand the flow of the system. The IP camera has its own Motion Detector and Face Detector.

This is used by the other successive processes to perform proper face recognition and human tracking.

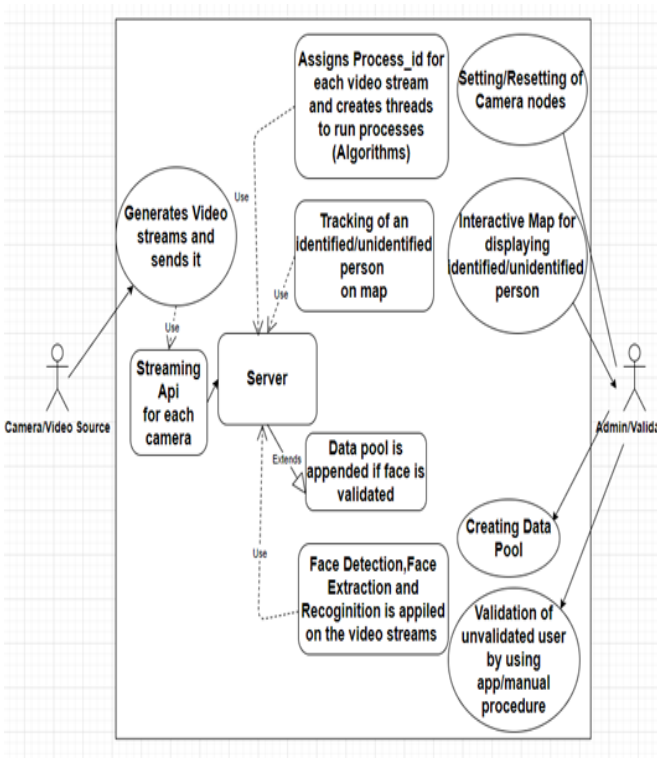

fig 3: interaction between user and admin

In an environment as specified in the Introduction, various video streams are given to System (Server). The Superordinate object initiates a camera object for each video stream, prioritizing the gate video stream object. First, since it is where the original data collection needs to start. It then catches each n (parameter for making batches) frames from the data stream and resizes it in proper format and batches it so it can be easily.

Be sent to the subordinate system for processing. The subordinate system enables the Tracker, Tracker Manager and the Parrallelizers and pipelines them accordingly before processing the frame. *

*Note-
The Tracker Object is generated exclusively for the gate camera and is only updated at the gate after every frame, this object is shared with its neighboring nodes using the Tracker Manager. The Tracker object traverses across the Graph which is created beforehand in the Graph Creation process.

The IP camera object stream the frames which goes through an object detector, it collects face and body data on the entry/point and updates the data pool and its indexes. If this frame is not from an entry point goes through the face/person detector (feature extractor) and then searches for the face/person in the data pool* using the feature comparer, once the face/person identified it can be tracked through the whole facility (Note: -The user can be enabling tracking of only the unidentified persons as well only identified targets or both, though only unidentified setting is recommended). The resultant data from both the Tracker and the Comparer are now fed to the Visualizer which in turn shows the output. The unidentified face/person is tracked throughout the facility for a specific time period. If the person is not validated it alerts the admin/security personnel about the person using his unique id. The unidentified person can also be validated by the admin in this time frame, if this is done, the face/person will be appended to the global data pool and given the identified status immediately.

Different algorithms are applied and explored for each process.

*Note-
The data pool mentioned here is a temporary one, it updates after each frame

2.2. Hardware and Software Requirements

Each process uses different resources namely CPU and GPU of the computer and therefore the software is heavily dependent upon the hardware system its running on. A pc or server with x86-64 (64-bit) compatible processor:3 GHz or a better processor with 6 cores is recommended. Graphical processing unit (GPU): GTX 1080 GPU or a better is recommended at least 6 GB of VRAM is recommended. Compute Capability 3.5 or better should be supported by the GPU. At least 8 GB of RAM is required. Digital camera(s): Camera resolution may vary depending on the actual application. The recommended resolution is about 2 Megapixel.

CUDA 10.1 toolkit or newer is required. cuDNN 7.5 library is required.

2.3. Graph Creation

To create a Graph of cameras latitude, longitude, altitude of the camera is required. Cameras can be considered as the nodes in 3D space and by using distance vector algorithm the nearest neighbor graph is created. Purpose of this NN graph is to minimize the global search.

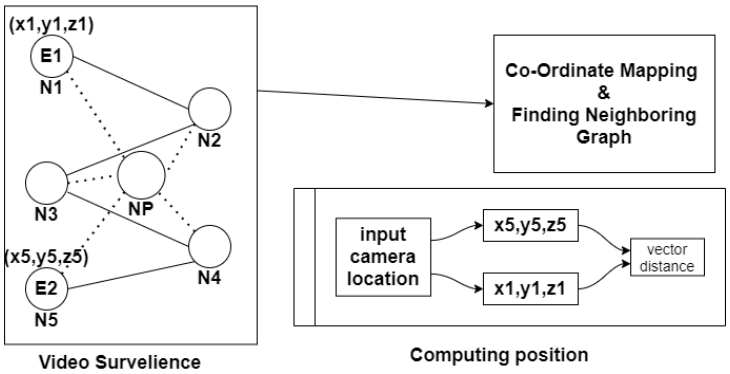

fig 4: camera interacting with each other passing the features from one node to another.

Whenever a camera node detects a person, it will extract the features and face of that person. Every node will have a local database in order to re-identify the person.

For re-identification purpose, Features and the faces of the person tracked will be stored and sent to neighboring nodes to look for that person with similar features and compare the face.

2.4. Feature Extractor

Feature extraction is particularly important in face recognition. Modules like FaceNet and TorchReid for face extraction and person feature extraction.

2.4.1. FaceNet

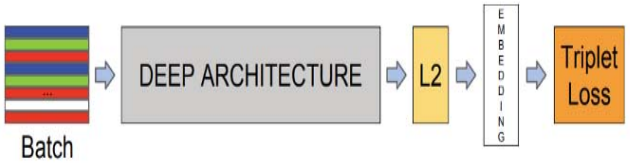

Fig5: FaceNet Architecture

Faces are stored in the form of 128 feature embeddings. Face recognition systems for face identification and verification using the VGGFace2 deep learning model are used to get 128 feature embeddings. The VGGFace refers to a series of models developed for face recognition and demonstrated on benchmark computer vision datasets. Face Identification involves calculating a face embedding for a new given face and comparing the embedding to the embedding for the single example of the face known to the system.

A one-to-many mapping for a given face against a database of known faces is called face Identification. For every batch of frames, faces are extracted and the face comparer is used to compare extracted faces with the already known faces from the global database. This data pool is populated by an entry flagged camera node and this global database is used by comparer to fetch the faces.

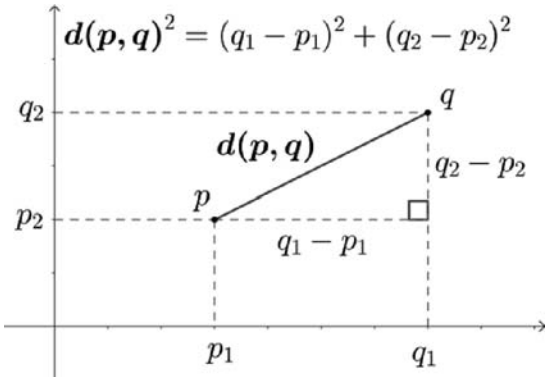

fig 6: Euclidean distance Formula

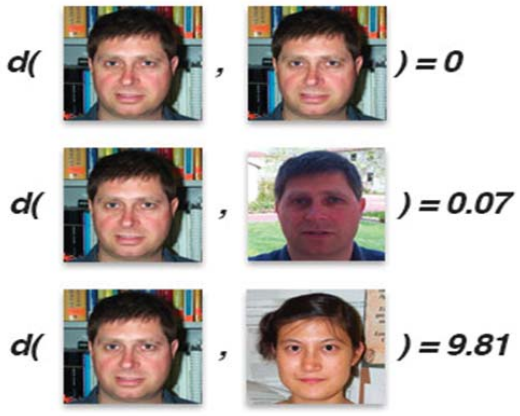

fig 7: Comparing the Euclidean distance determine same face

2.4.2. TorchReid

Torch Reid is a library for person identification written in PyTorch. provides a unified interface for all the ReID operations. The Pipelining of this framework is also quite simple and easy to use. It also provides easy access to various complicated tasks of person reidentification. In the context of a feature extractor for the system the TorchReid can be used to extract the features of a person from singular or multiple frames.

2.5. Face Comparer

A face embedding is a vector that represents the features extracted from the face. This can then be compared with the vectors generated for other faces. For example, another vector that is close (by some measure) may be the same person, whereas another vector that is far (by some measure) may be a different person. Typical measures such as Euclidean distance and Cosine distance are calculated between two embeddings and faces are said to match or verify if the distance is below a predefined threshold, often tuned for a specific dataset or application.

Here MTCNN is used to extract faces in the video stream. MTCNN (Multi-task Cascaded Convolutional Neural Networks) is an algorithm consisting of 3 stages, which detects the bounding boxes of faces in an image along with their 5 Point Face Landmarks (link to the paper). Each stage gradually improves the detection results by passing its inputs through a CNN, which returns candidate bounding boxes with their scores, followed by non max suppression.

Benchmark:

The specifications for the software to work are Intel i7-3621QM , TensorFlow 1.4.1.

The Pictures containing a single face:

Image Size	Total pixels	Process Time	FPS
460 x 259	119,140	0.118 seconds	8.5
561 x 561	314,721	0.227 seconds	4.5
667 x 1000	667,000	0.456 seconds	2.2
1920 x 1200	2,304,000	1.093 seconds	0.9
4799 x 3599	17,271,601	8.798 seconds	0.1

Fig 8: Benchmark of MTCNN

In this above context we use these modules to perform two mechanisms. Any of these methods may be used as a face comparison mechanism.

2.5.1. Iterative Method

In this method the face extracted from a batch of frames is compared to a global database of known faces one by

one. Every vector of faces stored in the database is compared with the face to be identified. One-to-many comparisons are made by finding the Euclidean distance less than a particular threshold. If the Euclidean distance is less than a particular threshold then the face is identified as a known face and a green flag is given to that tracker object. Iterative method has to sequentially scan each entry in the database and find the Euclidean distance and compare it with the threshold. This is brute force technique which runs in linear time complexity of O(n) where n is the number of faces in the database. As the number of faces in the database increases the performance will reduce.

2.5.2. Approximate nearest neighbors

Nearest neighbor search of embeddings is the fastest and optimal way for searching through the database. Vectorized cosine distance is notably faster than the iterative method but still may be too slow. This is where approximate nearest neighbors' shines: returning results quickly by using annoy indexes. It reduces the time complexity to O (log n) where n is the number of faces in the database. If your use case requires creating and building indexes often, then search time is the key performance. On the other hand, if your use case requires rebuilding indexes often, then adding embeddings and index building times are important. The following table demonstrates the times I spent for those ann packages on 1M vectors. Performance of packages.

	Adding Embeddings	Index Building	Search	Total
Spotify Annoy	3.106772	7.398274	0.000715	10.505761
Facebook Faiss	0.225953	0.000000	0.244324	0.470277
Nmslib	0.805913	263.954751	0.003436	266.764100

fig 9: Benchmarks for annoy

We have used a better version called hnswlib which dynamically updates the given indexes for any features that are extracted.

2.6. Tracker:

A Tracker is used for the following reasons: -It is faster than Detection, it can help when detection fails, it preserves identity:

The tracker object consists of a person tracker and Person Reid module encapsulated together in a single class. We have used different trackers, some of them are: -

2.6.1. OpenCV inbuilt Tracker and Person Reid

In this module we use cv2.inbuilt tracker api for tracking a person/face in a singular video stream, and torch reid person re identification engine for re-identifying the person across the surveillance system. These two are bundled together in a Tracker class. The tracker we used in our system was: -

2.6.1.1. GOTURN tracker: -

This is a CNN based offline tracker. First, the tracker is trained using thousands of videos for general person/face tracking. Now, this tracker can be used to track most of the objects without any problem even if those objects were not part of the training set. The goturn algorithm is best for detecting already trained objects such as persons, cars etc. It works fast if it is trained to recognize a single object.

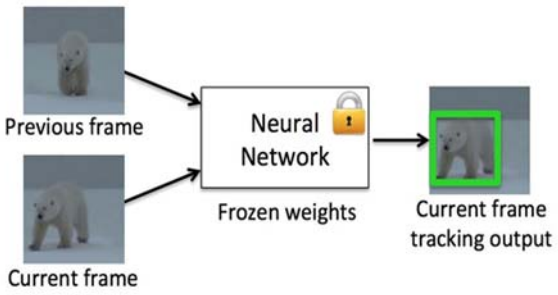

Fig10: Working of GOTURN

2.6.2. Tracker using deep sort algorithm through yolov4 and Person Reid

We use deep sort class and algorithm with the yolov4 for person tracking, this is bundled with torch reid person re identification. This object tracker coupled with person re identification engine of TorchReid can easily track human targets in multi camera multi target environment (limited only to humans).The TorchReid engine is uses the earlier extracted embeddings to recognize the person in the frame.

2.6.2.1. DeepSort

This is a great application for tracking, it contains a script to generate features for re-identification, which are appropriate to compare the visual appearance of the binding boxes for pedestrians using cosine simulations. It can also be used in standalone format without TorchReid.

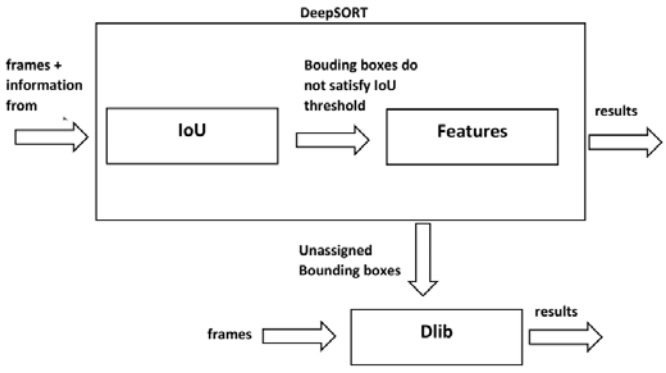

Fig11: DeepSort Architecture

2.6.2.2. YOLOv4

YOLO, as stated, stands for You Only Look Once, it is an object detection system in real-time that recognizes various objects in a single enclosure. Moreover, it identifies objects more rapidly and more precisely than other recognition systems. YOLOv4 outruns the existing methods significantly in both terms "detection performance" and "superior. The person tracker uses YOLO-v4 to perform person detection, which is used extensively for tracking.

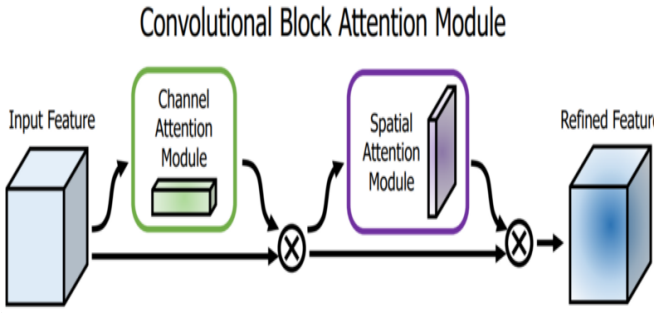

Fig12: How yolov4 framework processes images from video

There is a pre-trained YOLOv4 acquisition model that can accommodate up to 80 classes. To perform the tracking of an object using YOLOv4, we first convert the weights into a compatible TensorFlow model that will be stored in the test locations folder. After that all we need to do is run the script_ject_tracker.py script to use our object tracker with YOLOv4, Deep Sort and TensorFlow.

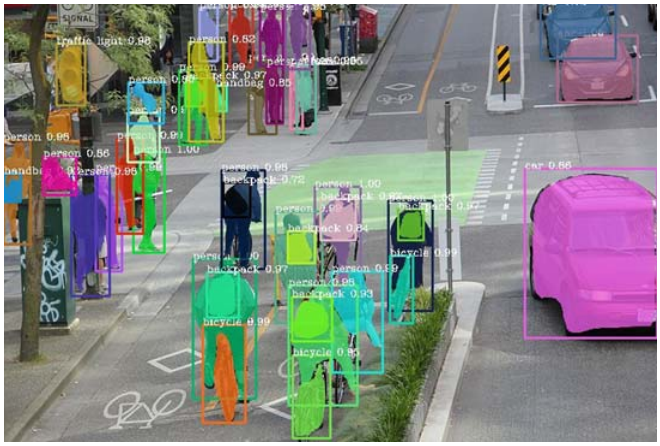

Fig13: creating bounding boxes around the humans in the frame

DeepSort implementation using Yolv4 is using the following parts: -

The Yolov4 gives us the Object detections and the original bounding boxes.

Kalman Filter: -It gives the bounding boxes of each person while tracking them using 8 parameters such as aspect ratio, height of the image etc. It helps us factor in the noise in detection and uses prior state in predicting a good fit for bounding boxes.

To verify our detections with the Kalman filter bounding boxes we have an association algorithm which can be quantified using a distance metric. The distance metric used here is the Hungarian algorithm. Deep learning is used by getting the appearance features of the person in a particular frame, since in a real-world situation these above methods are still not enough as they face problems such as occlusions and different viewpoints etc. This algorithm is used for multi camera multi target environments using TorchReid which helps in person reidentification in different videos.

2.7. Parrallelizers

These are the objects which help to optimize the performance of the system. There are various methods for such processes.

2.7.1. Multi-Threading

In this type of parallelism, there is a single thread for each of the camera video streams, it does all the jobs that are pipelined in the design flow of the system. This type of parallelism includes concurrently running threads, which have context switching and sharing of data enabled between them.

This type of parallelism ensures fast I/O work and less spawning time and resources for each thread. Since CPU resources are comparatively less, the overall process gets slowed down if multiple updates occur in the video processing part of the software.

2.7.2. Multi-Processing

In this type of parallelism, there is a single process for each of the camera video streams, it does all the jobs that are pipelined in the design flow of the system. This type of parallelism includes parallelly running processes on CPU/GPU cores, which have sharing of data enabled between them. This type of parallelism ensures less processing time for video processing, and since each process has comparatively. large amount CPU resources allocated; this helps in completing the video processing mechanisms such matrix multiplication inferences faster. Since CPU resources are comparatively more, this increases the spawn time required for each process and the I/O mechanism is also slowed down due to overall increased resource usage.

2.7.3. Multithreading coupled with Multiprocessing

In this Type of parallelism each video stream is assigned a thread for I/O operations (i.e., it opens and closes the stream) and each video processing jobs such as trackers, comparer, etc. are assigned individual processes for completing their jobs. Sharing is enabled among the internal processes for each stream and for among threads for different streams. This helps in smooth execution of both I/O as well as video processing operations. Message passing overhead increases this type of parallelism. Since a large number of execution units are created this may require more powerful hardware. These Parrallelizers help in scaling of the system for large facilities.

3. Main Results

We have implemented our system on a webapp and visualized its inference on the ip camera stream itself. The tracker tracks a person across different cameras and consistently persists the id of the person as well as the tag of recognized or unrecognized individual in the proximity of the multi-ca4mera system. Some results of the different combinations of the modules used are specified in the table below. The fps and the accuracy of the whole system is considered as an evaluation metric. We have used the following hardware specs for benchmarking purposes.

Graphics Card/GPU: NVIDIA GeForce GTX 1060

Processor: Ryzen 1700x

Motherboard: GIGABYTE AX 370

We have considered 4 video streams together currently; this can be easily extended to 6 with some negligible loss in fps and with a higher CPU and GPU can be extended to 10 video streams (the fps is expected to increase slightly)

Feature Extractor	**Compar-er**	**Tracke-r**	**Pers-on Per Fra-me**	**Parallel-izer**	**FP-S**	**Effici-ency**
Facenet/ MTCNN	Iterative	None	1-2	Multi thread	10-15	Good
Facenet/ MTCNN	Iterative	None	3-5	Multi thread	8-10	Medium
Facenet/ MTCNN	Iterative	None	3-5	Multi process	4-6	Bad
Facenet/ MTCNN	Annoy	None	3-5	Multi process	8-12	Medium
Facenet /MTCNN	Annoy	Deep Sort	3-5	Multi process	6-10	Medium
Facenet/ MTCNN	Hnswlib/ Frame Skipper	Deep Sort	3-5	Multi thread	10-20	Medium
TorchReid	Hnswlib/ Frame Skipper	Deep Sort	3-5	Multi thread	10-20	Good
TorchReid /Facenet MTCNN	Hnswlib/ Frame Skipper	Deep Sort	3-5	Multi thread	10-20	Best

Fig14: Benchmark of our system

Note*- The TorchReid module is relatively newly being added for experimentation in the system, more testing is required to give accurate analysis.

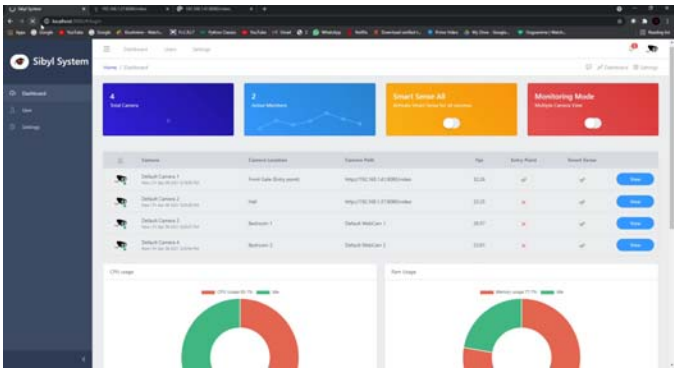

Fig15: Initial Setup screen

We have tested the results in public as well as private/Home environments, with a single camera on the top left corner of fig19 as the entry camera and all the others set as normal cameras. We have taken two positive targets, and two negative targets, which appear in the camera streams in regular intervals. They showed consistent results in the private environments and fluctuated in public spaces. We noticed uniform results according to the table in Fig17, though generally depending upon the consistent lighting of the given environment.

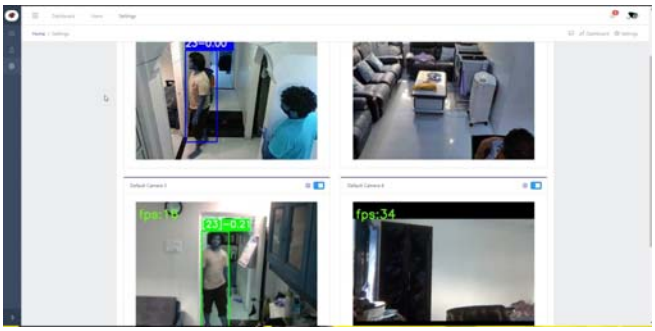

Fig16: Initialization of the system

Fig17: Green box and persistence of the person on multiple cameras

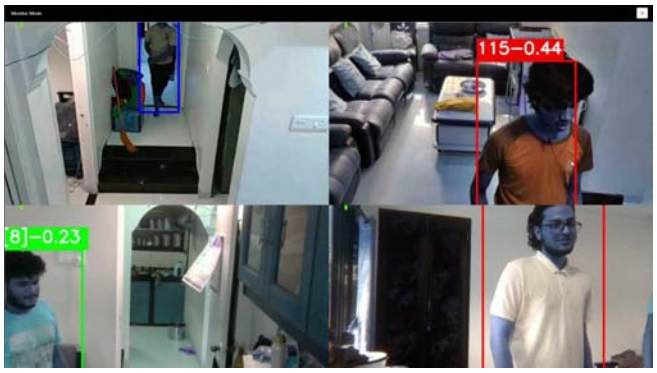

Fig18: Red Box and green box detection of negative as well as positive targets

4. Future Prospects

Since the results are experimental in nature, we continue to explore the combinations between different modules in our subsystem. The resultant table can be updated in the future. Our visualizing software could also be changed in the future (i.e., 2d map format), this would also have a significant impact on the fps, as it would generally increase in the absence of any extra creation of bounding boxes. In addition, we could easily deploy such a system on any cloud platform for a cheap subscription fee, so this can also be easily used in any practical situation in the future. Many applications can be derived using our proposed system such as attendance manager using computer vision, accurate footfall measurements in a distributed system environment, detection of crime scenes in multiple video streams, analysis of such crime scenes etc.

5. Conclusion

Detecting human beings accurately in a surveillance video is one of the major topics of vision research due to its wide range of applications. It is challenging to process the image obtained from a surveillance video as it has low resolution. A review of the available systems is also provided.

A proposed system and its design, flow and submodules are discussed, and various ways to increase the efficiency of the system are given to improve the human detection and tracking process in surveillance videos. These include adopting a new multi-view approach based on concentrating on developing an optimized system by experimenting on different combinations of the modules discussed.

6. Acknowledgement

Special thanks to Professor Ms. Rakhi Kalantari, Department of Computer Engineering, Fr. C. Rodrigues Institute of Technology, Vashi for her support and guidance.

7. References

This paper is citated from the below references:

[1] Design and Implementation of a Human Tracking CCTV System using IP-Cameras: https://www.researchgate.net/publication/328980042_Design_and_Implementation_of_a_Human_Tracking_CCTV_System_using_IP-Cameras

[2] Distributed multi-camera synchronization for smart-intruder detection: https: //ieeexplore.ieee.org/abstract/document/6314665.

[3] Paul, M., Haque, S.M.E. & Chakraborty, S. Human detection in surveillance videos and its applications - a review. EURASIP J. Adv. Signal Process. 2013, 176 (2013). https://doi.org/10.1186/1687-6180- 2013-176

[4] J. Celine and S. A. A, "Face Recognition in CCTV Systems," 2019 International Conference on Smart Systems and Inventive Technology (ICSSIT), Tirunelveli, India, 2019, pp. 111-116, doi: 10.1109/ICSSIT46314.2019.8987961.

[5] Todi, Manish. "Real-Time Machine Learning." Medium, Analytics Vidhya, 18 Oct. 2019, medium.com/analytics-vidhya/real-time-machine-learning-7aa55dafb2b.

[6] Wojke, N., Bewley, A., & Paulus, D. (2017). Simple Online and Realtime Tracking with a Deep Association Metric. In 2017 IEEE International Conference on Image Processing (ICIP) (pp. 3645–3649).

[7] Wojke, N., & Bewley, A. (2018). Deep Cosine Metric Learning for Person Re-identification. In 2018 IEEE Winter Conference on Applications of Computer Vision (WACV) (pp. 748–756).

[8] Facial recognition https: //github.com/BrandonJoffe/home_surveillance/tree/d5fd430dbc149b4 8a1424f44fc748c79ef971511

[9] Yolo motion detection: https://github.com/hunglc007/tensorflow-yolov4-tflite

[10] Xu Zhang, Xiangyang Hao, Songlin Liu, Junqiang Wang, Jiwei Xu, & Jun Hu (2019). Multi-target tracking of surveillance video with differential YOLO and DeepSort. In Eleventh International Conference on Digital Image Processing (ICDIP 2019) (pp. 701 – 710). SPIE.

[11] Zhou, K., & Xiang, T. (2019). Torchreid: A Library for Deep Learning Person Re-Identification in Pytorch. arXiv preprint arXiv:1910.10093.

[12] Zhou, K., Yang, Y., Cavallaro, A., & Xiang, T. (2019). Omni-Scale Feature Learning for Person Re-Identification. In ICCV.

[13] Zhou, K., Yang, Y., Cavallaro, A., & Xiang, T. (2019). Learning Generalisable Omni-Scale Representations for Person Re-Identification. arXiv preprint arXiv:1910.06827.

[14] The AIGuy. (2020) . yolov4-deepsort. https://github.com/theAIGuysCode/yolov4-deepsort.

[15] arunmandal53. (2020). facematch. https://github.com/arunmandal53/facematch.

[16] Mandal, Arun, et al. "MTCNN Face Detection and Matching Using Facenet Tensorflow." Pytorials.com, 16 Feb. 2018, www.pytorials.com/face-detection-matching-using-facenet/.

[17] Drake, Victoria. "Multithreaded Python: Slithering Through an I/O Bottleneck?" FreeCodeCamp.org, FreeCodeCamp.org, 15 Mar. 2021, www.freecodecamp.org/news/multithreaded-python/.

B. Amos, B. Ludwiczuk, M. Satyanarayanan,"Openface: A general-purpose face recognition library with mobile applications,"CMU-CS-16-118, CMU School of Computer Science, Tech. Rep., 2016

8. AUTHORS

Sureshkumar Jha –B.E. computers, Fr. C. Rodrigues Institute of Technology.

Rohan Sawant– B.E. computers, Fr. C. Rodrigues Institute of Technology.

Parth Shinde– B.E. computers, Fr. C. Rodrigues Institute of Technology