DE-41 (MTS) Abdullah

Single Camera-based Abnormal Activity Analysis for Surveillance Applications

COLLEGE OF
ELECTRICAL AND MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY
RAWALPINDI
2023

COLLEGE OF ELECTRICAL AND MECHANICAL ENGINEERING

DE-41 MTS

PROJECT REPORT

Single Camera-based Abnormal Activity Analysis for Surveillance Applications

Submitted to the Department of Mechatronics Engineering

in partial fulfillment of the requirements

for the degree of

Bachelor of Engineering

in

Mechatronics

2023

Sponsoring DS:

Dr. Tahir Habib Nawaz (Supervisor)

Dr. Umar Shahbaz Khan (Co-Supervisor)

Submitted By:

Abdullah

ACKNOWLEDGMENTS

First and foremost, I am deeply grateful to Allah, the almighty, for His blessings, guidance, and mercy.

I want to express my deep gratitude to my parents for their motivation and support.

I would also like to express profound appreciation to the supervisor, Dr. Tahir Habib Nawaz. His guidance, expertise, and valuable insights have been instrumental in shaping this research.

ABSTRACT

The project is to develop a framework for abnormal activity detection. This includes the collection of datasets, which were collected personally as well as from Internet platforms such as Twitter and YouTube. The datasets included different scenarios such as wall climbing, entrance into sensitive premises and intrusion through a window. For object detection the YOLO algorithm is used and only “person” class is detected. The algorithms provided us with accurate bounding boxes of detected objects and this data is further used to track people and analyze their activity. For object tracking the Deep SORT algorithm is used which retains the identities of detected objects in consecutive frames and helps to extract trajectories of detected person class. Trajectories are then analyzed to detect abnormal activity. The intrusion is considered an abnormal activity for this project. For greater accuracy of intrusion detection some features were introduced, which includes the translation of centroid to the bottom of the detected bounding box to avoid any intrusion remain undetected, since in majority of cases the area of restriction is highlighted on bottom surfaces. The distance between centroid and line of restricted area is calculated by using the formula of the distance between a point and line of restriction. This made the algorithm robust enough to detect intrusion from any side of the restricted zone. The packages required by the algorithm were installed and dependencies were resolved for the proper execution. The evaluation was performed on collected datasets as well as on the real time video feed and the algorithm was able to detect intrusion in all scenarios. The developed algorithm is then ported to an edge device Jetson Nano for real-time analyses. For optimization of algorithm different techniques were used such as down-sampling and the usage of light weight model. The input video feed was given by a single camera and after analyzing the input video Jetson Nano generates a warning if it detects an intrusion. For complete setup all the input and output devices such as power adapter, internet connection, display were managed for the smooth operation of project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

Latin Letters

v input feed

w motion trajectories

Acronyms

AI Artificial Intelligence

GHz Giga Hertz

IoU Intersection over Union

GB Giga Byte

mAP mean Average Precision

CPU Central Processing Unit

Particle Swarm Optimization

SPI Serial Peripheral Interface

GPU Graphics Processing Unit

I^2C Inter Integrated Circuit

UART Universal Asynchronous Receiver Transmitter

GPIO General Purpose Input Output

SDK Software Development Kit

L4T Linux for Tegra

IPM Inverse Perspective Mapping

MicroSD Micro Secure Digital

ROI Region of Interest

TPU Tensor Processing Unit

TDHA Trajectory Direction History Analysis

USB Universal Serial Bus

FPS Frames per Second

SORT Simple Online Realtime Tracking

I^2S Integrated Inter-IC Sound Bus

DC Direct Current

AWS Amazon Web Services

NMS Non-Maximum Suppression

Chapter 1 - INTRODUCTION

1.1. Motivation

The abnormal activity detection focused on intrusion detection has a major contribution in the surveillance of public spaces, residential and commercial areas. Currently there are high concerns to deal with the unauthorized access to sensitive premises or critical infrastructures, so for the growing demand the development of efficient intrusion systems can prevent potential security breaches.

Currently several surveillance systems require human operators to monitor cameras. Due to limitations of human performance to monitor multiple camera feeds and observe for a long duration with the same attention, there is a chance of errors in the security system. So, the development of a framework to detect an intrusion automatically would help the security personnel to effectively respond to any incident with reduced response time.

Such automated abnormal detection system would also be scalable and adaptable because for changing security requirements it is much easier to expand this system rather than following a resource intensive approach of hiring more video monitoring personnel.

Along with the security benefits, this automated system could also help us in post event analysis or investigations. The stored data could be used to analyze patterns and improve the systems to further enhance the security protocols.

1.2. Background Knowledge

The background knowledge for this project includes basic concepts of image processing such as the structure of images, how images are stored, how to manipulate image data and how videos work at different frames per second. It also needs to have some key concepts of computer vision, for example object detection, feature extraction and tracking.

It also involves some concepts of machine learning and deep learning. For example, YOLOv4 is based on darknet framework so one needs to have some prior knowledge of the

working of this framework, how the manipulation of parameters could improve the working of framework.

It is also important to keep an eye on the latest research and publications in the field of video surveillance, some features of this project are based on the ideas developed from available research. Along with this, awareness of the availability of publicly available datasets is also important.

The basic concepts of edge computing, which includes dealing with the edge devices (such as Jetson Nano for our project) is critically important. The requirements of peripherals to be attached with edge devices are also of great importance for example the Jetson Nano requires a power adapter of 5V 4A ratings, slight changes in the rating would result in malfunctioning of the device. Similarly, basic knowledge of Linux based operating system is also required.

1.3. Problem Formulation

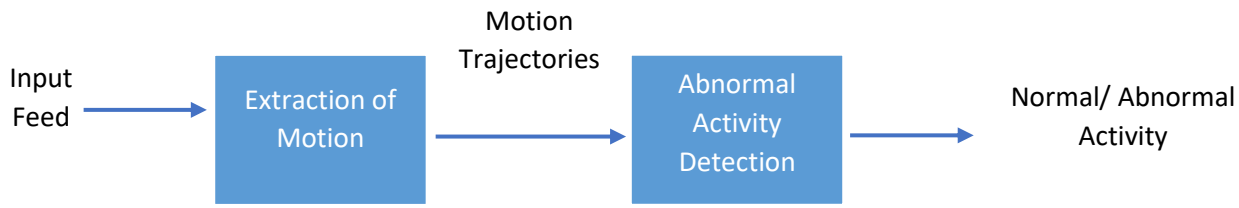

Figure 1. Problem formulation

The input feed v of video frames is fed into object detection and tracking algorithms which extracts the motion trajectories w. Based on predefined criteria, the motion is classified as either normal or abnormal. The input feed v could be either from already collected datasets or a real time video by a camera. The algorithms extract information from input frames to get motion trajectories w which is then used to detect motion to be normal or abnormal.

1.4. Contribution

For this project, first the datasets were collected. Some video clips of different scenarios are recorded personally. The scenarios include intrusion into sensitive premises, multiple people travelling on road, intrusion into space allocated for special people. Digital platforms such as YouTube and Twitter were also used to collect datasets. These includes a person trying to intrude through a window, and a person trying to get into a house by climbing a wall.

Then object detection and object tacking algorithms were used for detection and tracking respectively. For object detection YOLOv4 was used while for object tracking deep SORT was implemented. Since our project is focused on the intrusion of person class, the algorithm was restricted to detect only person. For the implementation of these algorithms, the environments were managed, and their conflicts were resolved. For example, NumPy was downgraded to 1.23.5 from its latest version to resolve the conflict. Due to computational limitations of local machine google colab was used to utilize its GPU.

There are several abnormal activities, but our project was focused on detecting intrusion as an abnormal activity. First, the centroid was extracted from the detected bounding boxes. Then it was translated to the bottom of bounding box because in most of the cases restricted area or line of restriction is defined on ground surfaces so there is a chance that our algorithm misses the intrusion. Trajectories were extracted and highlighted for each tracked id. The distance formula of distance between a point and a line was used to detect the intrusion which made our algorithm robust enough to detect intrusion from any side.

Then Jetson Nano was set up for porting algorithm into it. The setup included managing its hardware peripherals such as a power adapter of accurate ratings as prescribed by NVIDIA and other input/output devices. The internet connection was established for jetson nano by using a third-party software in a computer so that the ethernet port of a local computer could act as an internet provider. The Linux operating system was installed on Jetson Nano with the help of microSD card and a local computer. The libraries were installed in an environment and different conflicts were resolved among the packages for the proper execution of the algorithm. USB based camera was used to get live feed and detect an

intrusion on real-time. The algorithm was optimized by down sampling the input image and by using YOLOv4 Tiny wights.

The algorithm was tested and evaluated on the collected datasets and on a live feed from a camera. The algorithm was proved to be robust enough to detect intrusion from any side of the restricted area.

1.5. Organization of Thesis

There are five chapters in this thesis. The first chapter gives the introduction and overview of the complete project. The second chapter describes the framework of the project, detailed working, and the implementation of the project. The third chapter describes the experimental setup and results in detail. Finally, the last chapter gives a conclusion to the thesis. References are added to the Bibliography section. Appendix is added at the end for the list of packages installed in the environment.

Chapter 2 - LITERATURE REVIEW

2.1. Overview

This chapter reviews the research work in the field of abnormal activity detection going all the way from object detection to tracking and then review of detection of different abnormal activities. For object detection both single stage and two stage detectors were reviewed. Single stage detectors included SSD and YOLO while two stage detectors included variants of RCNNs. In object tracking, the working and performance of trackers such as deep SORT, Tracktor++ and MDNet was reviewed and in the last section before discussion different abnormal activities such as intrusion, loiter and fall detection are studied which includes different approaches of detection and some improvements in previous work.

2.2. Object Detection

Object detection is used for many applications such as automated vehicles, surveillance, machine inspection and many more. There are two categories, machine learning algorithms and deep learning algorithms. A decade back most of the object detection used classical machine learning (ML) techniques. After the deep learning (DL) has taken off, most of the object detection is done through deep learning-based approaches.

Machine learning algorithms have lower accuracy and shorter training while deep learning have higher accuracy and longer training this is because machine learning makes simple linear correlations while deep learning makes complex and non-linear correlations.

Machine learning can train on small datasets while deep learning requires large datasets, thus in this era of big data the deep learning algorithms are proved to be more productive. Along with this, the performance of deep learning is increased with the increased size of neural network. The comparison is illustrated as follows.

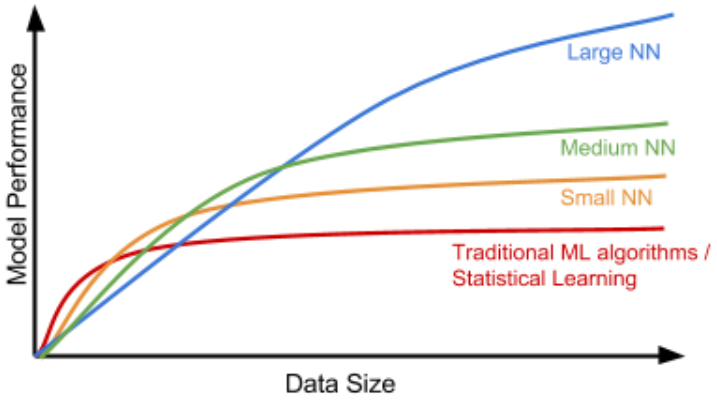

Figure 2. Comparison of data size and model performance. Image courtesy [31]

In deep learning object detection is categorized into two types:

i. Single stage object detectors

ii. Two stage object detectors

Some algorithms of both categories are shown in the following block diagram.

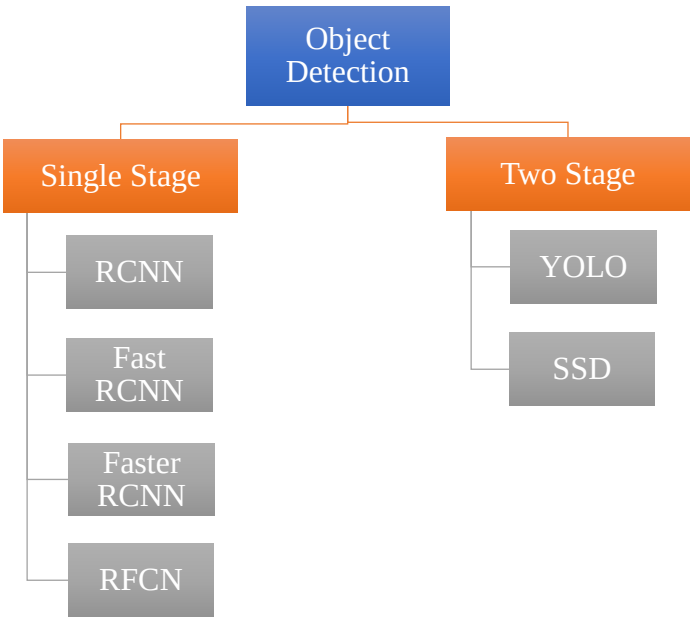

Figure 3. Block diagram of object detection algorithms. Image courtesy [32]

2.2.1. You Only Look Once (YOLO)

YOLO's first version (YOLOv1) [1] was published in 2015. It achieved mAP of 63.4% with 45 FPS on PASCAL VOC 2007 dataset. The CNN predicts bounding boxes and class probabilities in only one look at the full image (while in two staged algorithms first region proposals are generated and then further evaluation is completed).

First SxS sized grid cells are formed of an input image. Then bounding box generation and calculation of confidence score is performed. Along with this conditional probability P (class | object) for each grid cell is also created. Finally NMS is applied for the calculation of final predictions. This process is depicted in the diagram below.

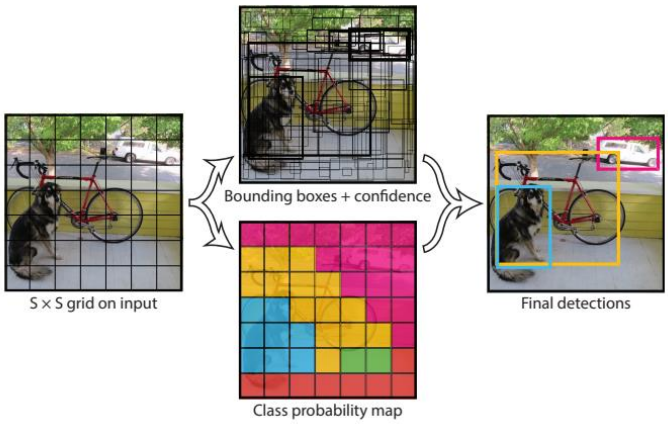

Figure 4. YOLO algorithm single stage process. Image courtesy [33]

Five parameters are generated for each bounding box which are width, height, center coordinates (x, y) and confidence score. It was found that version 1 has difficulty in detecting small objects present in groups and in detecting unconventional aspect ratios.

In version 2 [2] some improvements are introduced. Batch normalization is applied on each convolutional layer, as a result mAP improved by 2%. A higher resolution classifier is used which increased mAP by 4%. Anchor Boxes were also introduced which improved recall by a big margin.

Version 3 [3] also came up with some incremental improvements. Independent logistic classifiers were used for the prediction of classes rather than using softmax layer to enable multi-label classification. Detection on varying scales was also introduced in version

3. This version gave mAP of 57.9% on the COCO dataset with an IoU of 0.5 in comparison to mAP of 44% by version 2 with same dataset and IoU.

In version 4 [4] CSPDarknet53 convolutional neural network was introduced, and spatial pyramid pooling (SPP) is added to CSPDarknet53. mAP and the FPS were increased by 10% and 12% respectively in compared to version 3.

Version 5 improved its performance by using PyTorch training procedure while remaining of the model architecture remains similar to version4.

2.2.2. Single Shot Detector (SSD)

SSD [5] was published in 2016. It is also a single stage detector that outperforms other algorithms with its high speed and accuracy. On PASCAL VOC 2007 dataset it gives mAP of 74.3% at 59 FPS as compared to mAP of 73.2% at 7 FPS by Faster RCNN and mAP of 63.4% at 45 FPS by YOLO. SSD improved different things such as it used small convolutional filter for the prediction of object categories, sperate filters were used for the detection of different aspect ratios, multiple layers were used for the prediction of different scales.

2.2.3. Region-based Convolutional Neural Networks (R-CNN)

RCNN [6] was proffered in 2014. It gave mAP of 53.3% on the PASCAL VOC 2012 dataset which was 30% more than the previous highest score on the same dataset. This detector is divided into three parts. First is generation of category independent region proposals, around 2000 region proposals are produced. The second part is CNN which then extracts feature vector from each region. To give fixed size CNN input “affine image warping” technique is used. The third part consists of class specific linear support vector machines (SVMs) for the classification of each region. The following diagram depicts the steps involved in algorithm:

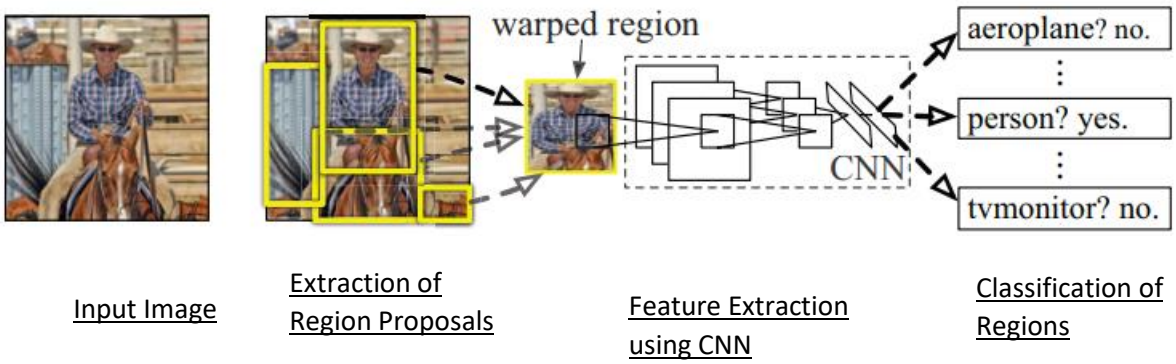

Figure 5. RCNN object detection system. Image courtesy [34]

2.2.4. Fast R-CNN

Fast RCNN [7] was proposed in 2015. The developers identified the following drawbacks in R-CNN

i. Multistage pipeline: First the tuning of CNN is done on object proposals (candidate object locations) then SVMs are used for object detection by using CNN features and finally bounding box regression.

ii. Expensive training: For bounding box regression and SVM the object proposals are extracted from every image which occupies hundreds of gigabytes disk space and training of networks. For example, VGG16 with 5k images of VOC07 dataset takes about 2.5 GPU days.

iii. Slow Object Detection: The algorithm extracts feature from each object proposal, and it takes 47 seconds per image with VGG16.

In Fast RCNN input is given to CNN for the creation of feature map rather than giving region proposals to CNN (this method is applied in RCNN) and then from feature map the region proposals are extracted. To feed these region proposals to fully connected layers the region proposals are warped with the help of ROI pooling layer. Softmax layer is used for prediction of classes and bounding box regressor is used for bounding-box regression offsets.

Since in Fast RCNN we don't have to feed 2000 regions for every image and only single convolution operation is required for each image thus Fast R-CNN is 9 times faster at

training time and 213 times faster at test time for VGG16 network. The following diagram illustrates the working of algorithm:

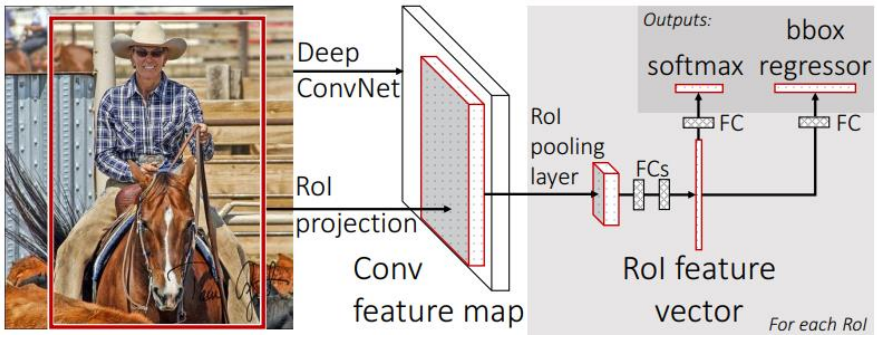

Figure 6. Fast RCNN object detection system. Image courtesy [35]

2.2.5. Faster RCNN

This algorithm [8] was also published in 2015 and developers find out that selective search region proposal algorithm is a bottleneck in Fast RCNN. So, to solve this problem region proposal network (RPN) was used instead of selective search algorithm.

Thus, Faster RCNN has two parts. The first part is RPN which is a deep fully convolutional network, its job is to predict bounds of objects and object presence scores simultaneously for every position. The second module is Fast RCNN, proposed regions are given as input to Fast RCNN detector.

Following diagram depicts the structure of algorithms:

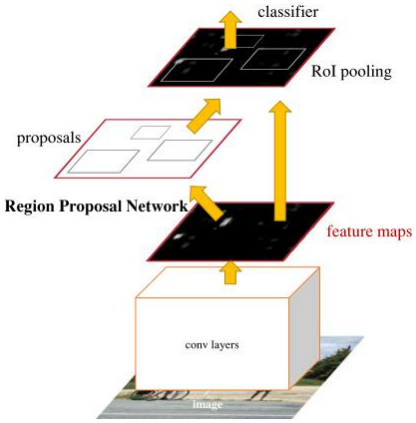

Figure 7. Faster RCNN object detection system. Image courtesy [36]

2.2.6. Region-based Fully Convolutional Networks(R-FCN)

It was published in 2016. It showed mAP of 83.6% on PASCAL 2007 dataset and 82% on 2012 dataset. This architecture is 2.5 to 20 times faster than Faster RCNN. In the previous algorithms fully connected layers for bounding box classification and regression are located after ROI layer. This architecture has drawbacks, first that the fully connected (FC) layers process is not shared with ROI pooling layer thus the process takes more time. Secondly, the FC layers located after ROI pooling layer increase number of connections which as a result increases complexity. So, in RFCN [9] the FC layers located after ROI layer are removed and placed before ROI layer. Now the score maps produced by FC layers will be utilized by ROI layer to perform average voting where average voting is a very simple calculation. This makes RFCN faster and more accurate. The following diagram depicts RFCN architecture.

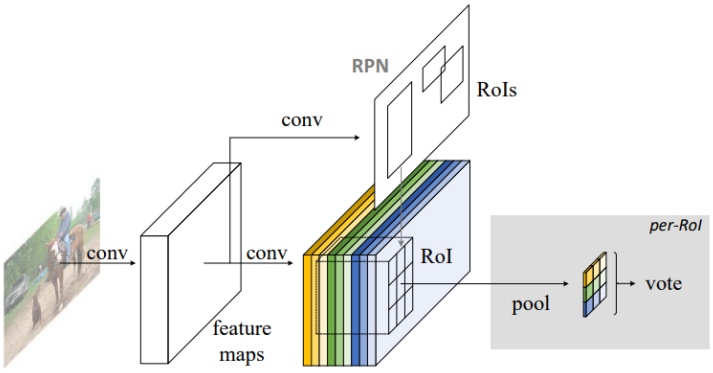

Figure 8. RFCN object detection dystem. Image courtesy [37]

2.3. Object Tracking

It is to predict or estimate the position and related information of moving objects. To track objects first they are detected using object detection algorithms and then object tracking algorithms are applied.

Object tracking consists of the following four stages:

i. Target initialization
ii. Appearance modeling
iii. Motion estimation
iv. Target positioning

Object tracking has two levels.

i. Single Object Tracking (SOT)
ii. Multiple Object Tracking (MOT)

The computer vision community is contributing a lot in research for multiple object tracking (MOT) for example “mot challenge” [30] is a platform where different researchers compete for the best tracking algorithm.

Since our project will use Multiple Object Tracking, only MOT algorithms will be reviewed.

2.3.1. SORT and Deep SORT

In the past object tracking algorithms (for example algorithms developed using Joint Probabilistic Data Association (JPDA) filter), it was identified that they incorporate complexities such as re-identification, which limits their use in real time applications. Thus, only bounding box position and size are utilized from objection detection data for motion estimation and data association.

The accuracy of algorithm is dependent on the type of detector, like the performance is better on Faster RCNN models than Autocorrelation Function (ACF) algorithm. The Multi-object tracking accuracy (MOTA) for these algorithms are as follows.

TABLE I. COMPARISON OF OBJECT DETECTION ALGORITHMS.

Detector	MOTA
ACF	15.1
Faster RCNN(ZF)	24.0
Faster RCNN(VGG16)	34.0

Thus, Faster RCNN detector is used.

The Kalman filter and Hungarian methods are utilized by SORT algorithm for object tracking. Some algorithms give greater accuracy at the cost of runtime processing, But the SORT algorithm gives higher accuracy and fast runtime processing simultaneously which makes it useful for surveillance applications.

The developers of Deep SORT [10] found that SORT [11] is not able to perform well in occlusions lasting for longer time which increases identity switches. So, they proposed deep association metric for person re-identification (which is basically to incorporate deep neural networks for the estimation of object location). As a result, identity switches decreased by 45%.

To keep the algorithm fast this computationally complex learning is placed in offline pre-training stage.

2.3.2. MDNet

The founders of MDNet [12] observed that it is difficult to learn unified representation from video sequences which have different characteristics. Each sequence has different targets which have different moving patterns, appearances, and class labels. So, they proposed CNN architecture named Multi-Domain Networks which take sequence independent information. The algorithm performs online visual tracking.

This architecture has a small number of layers as compared to VGG and Alex network. The MDNet architecture consists majorly of three parts as shown in the following diagram. First shared layers then come domain specific layers and finally classification layers.

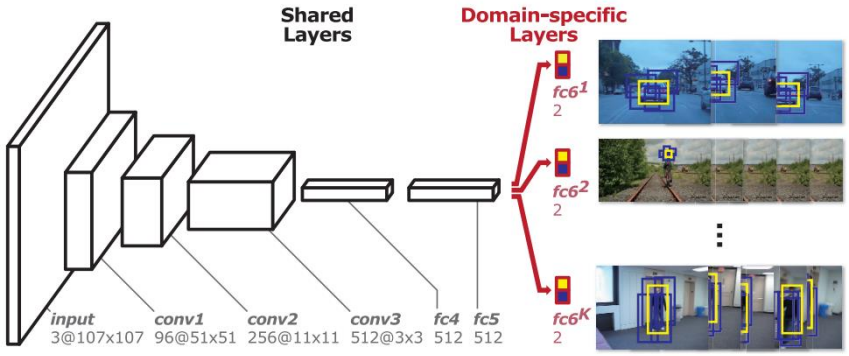

Figure 9. MDNet object tracking system. Image courtesy [38]

2.3.3. Tracktor++

This algorithm introduced a new paradigm. Previously developed algorithms specifically focus on tasks such as object re-identification, dealing with occlusion and motion prediction. But this makes algorithm computationally complex. So Tracktor++ [13] came up with a solution to this problem. It used bounding box regression of the detector for motion prediction and no optimization or training was performed on tracking data. Thus, transforming a Detector into Tracktor.

Two extensions are also added to Tracktor named motion model and re-identification Siamese network. The first one is used to deal with the issue of slow frame rate and moving camera. Re-identification Siamese network is to verify that the location of target in frame t-1 is matched with the estimated location in next frame t. Otherwise, the tracklet is considered to be dead because of its deactivation. Due to these two extensions Tracktor is named Tracktor++.

2.4. Abnormal Activity Analysis

The object tracking algorithms as we discussed before are used to produce trajectories which are then used to analyze activities in different scenarios, for example traffic management, video analysis in sports, video surveillance.

Many challenges are faced during trajectory analysis. Diverse surveillance scenes, for example the scene could be either indoor or outdoor, many object classes observed in a single scene, big datasets such as surveillance scene recordings.

There are different datasets [17] available for different scenarios. The following table classifies these datasets with respect to each scenario.

TABLE II. DIFFERENT SCENARIOS AND THEIR RESPECTIVE AVAILABLE DATASETS

Scene	**Datasets**
Indoor activity and event detection	UMN, ISE Lab, VAVIAR
Outdoor activity and interaction analysis	UCF101, PETS2001
Traffic monitoring and pattern understanding	MIT, QMUL Junction
Crowd analysis	CUHK, GCS
Human robot interaction	MHHIRI
University events	ERCe

Trajectory extraction involves different techniques such as heat maps and qualitative trajectory calculus (QTC). For visualization of movement patterns and abnormalities trajectory clustering is performed.

Defining abnormal activities is context dependent and subjective. There are several research covering different types of abnormal activities such as illegal U-turn, stopped or slow vehicle, restricted movement of traffic, crowd changes, loitering, falling on floor, off road driving, intrusion and many more.

Since our project is related to abnormal activity analysis for surveillance, only the following activities will be reviewed further.

2.4.1. Intrusion

The intrusion detection systems developed are designed to detect unauthorized entry in sensitive areas. Different techniques are used and some of them are discussed as follows.

Background subtraction and frame difference methods are combined to detect pedestrians. Then the vertex of pedestrian and vertex of sensitive areas are analyzed for intrusion detection [18]. The frame difference method is divided into two methods, the double frame difference method and triple frame difference method. The triple frame difference method suppresses noise and is more accurate. To detect the intrusion in sensitive area first the fixed area of pre-defined sensitive ROI is calculated, if any object intrudes that specified ROI the change in area of indicates intrusion. The following diagram describes the situation.

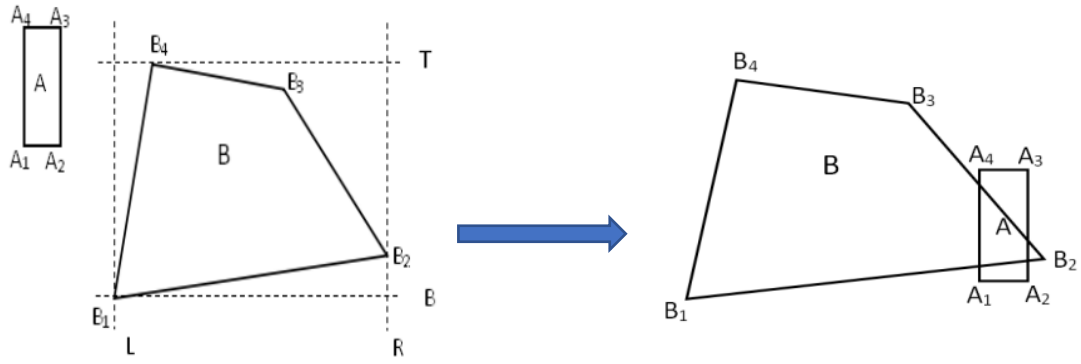

Figure 10. Intrusion detection using changes in area. Image courtesy [39]

Another way to detect intrusion is to track the centroid of intruder [19] and when the centroid of an object enters a defined ROI an intrusion warning is generated, this method decreases computational complexity. Since the cameras are located at certain height and a t some inclination as a result the algorithm sometimes miss the intrusion even if an object passes through an ROI, to solve this issue, centroid of object was shifted to the bottom of object which is depicted in the following diagram.

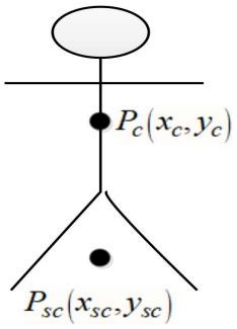

Figure 11. Shifting centroid of an object to bottom. Image courtesy [40]

2.4.2. Crowd Analysis

Crowds are analyzed from microscopic, mesoscopic, and macroscopic perspectives [20]. In macroscopic approach the crowd is treated by its overall performance in which crowds have same movement pattern. Individual movement is neglected in this approach. Different techniques are proposed for this approach such as unsupervised feature extraction and optical flow which are used to detect emergency event in large crowds, spatio-temporal motion model for the detection abnormal behavior in large crowds, Bernoulli statistical shape model is utilized to find the number of people in a busy area. The microscopic approach focuses on each individual. Techniques for this approach include Bayesian clustering, which is used to analyze abnormal behavior of individuals, AdaBoost classifier for the detection of fight scenes, shape and motion template to identify an individual's movement in crowd. Mesoscopic approach is used for medium sized crowds in which people don't have same movement patterns. Small groups are identified by using agglomerative clustering.

Some researchers model crowd motion to fluid flow motion. For which crowd is represented as a flow field and "optical flow estimation" is utilized to transform crowd into a vector field. Now methods for fluid flow analysis are implemented on crowd motion.

Force based models are also used to analyze crowd activities. This includes social force model in which three forces are considered: influence force from people, driving force towards destination, repulsive force from other objects, for example a barrier. This model is used to monitor escape and panic situations. This model was further modified by assigning

weights to people for the calculation of their relative speeds. PSO is also used to further increase the efficiency of social force model.

2.4.3. Loiter

To determine the anomalous behavior of waiting, standing, or walking apparently with no purpose, different methods have been proposed such as TDHA method [21]. In this method first the moving object is detected and then a trajectory consisting of centroids of moving object is generated. If the distance between consecutive points in trajectory increases by specified threshold than the trajectory is regenerated by using Representative Points (RP) which could be written as:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

If d > threshold, then the trajectory is regenerated by using RPs

$$p_n = (x, y)\,, \quad T_n^1 = \{p_1, p_2, p_3, \dots \dots \dots p_n\}$$

Adjacent points $p_1(x_1, y_1), p2(x_2, y_2)$ are calculated by the following equations to analyze the direction history of trajectory:

$$v = \left(v_x, v_y\right), v_x = (x_1 - x_2), v_y = (y_1 - y_2)$$

$$T_n^2 = \{v_1, v_2, v_3, \dots \dots \dots v_{n-1}\}$$

Normal trajectory will show small variation in directions between the two vectors while in abnormal trajectory large or irregular variation in direction is observed. Another technique known as IPM is also proposed to remove distortion in direction due to perspective effect.

2.4.4. Fall Detection

According to WHO 28-35% people of age 65 and above fall each year. There are different fall types and features which are used to detect them. They are discussed in the following table.

TABLE III. FEATURES USED TO DETECT CERTAIN FALL TYPES.

Features used for detection	Fall types
Geometrical orientation and state of silhouette	Lying down in stretched position & tucked position
Vertica and horizontal gradient distribution	Forward, sideways, backward
Near-field imaging floor	Onto knees, rotate right to left and vice versa
Floor pressure angle	Forward, backward, and sideways
Silhouette and center of mass	Forward, backward, and lateral
Height, width, and depth of human posture	//
Deformation and change in height	Fall from standing position or from chair

2.4.5. Trajectory Clustering

Information is analyzed across multiple frames to get object trajectories, which are then clustered to get motion patterns to study different behaviors. Frequency domain features and clustering spatio temporal features are used [23-26]. In clustering there are multiple techniques such as generating subclusters for different object trajectories [27], fusing clusters [25]. For aerial videos it is very challenging to extract motion patterns since they require camera motion compensation [28]. While applying clustering discrete wavelet transform coefficients are used.

To aid clustering of trajectories and motion pattern extraction, deep trajectory representation is used [29]. This utilizes the result from the hidden layer of the smallest size for the encapsulation of trajectory. To get dominant trajectory clusters mean shift clustering framework is used. Then to extract motion pattern, distance minimization from centroids technique is used.

2.5. Discussion

The research papers provided great insight to the current trends in object detection, object tracking algorithms and abnormal activity detection techniques. Both single stage and two stage detectors were studied, and it was found that single stage detectors would be more suitable for our project because our algorithm is to be ported on an edge device which has limited computational capacity. Among object tracking algorithms deep SORT found to be an advanced algorithm that deals with occlusions and retains identities with greater accuracy. In the research for abnormal activity analysis, a very important point of translating centroid

to the bottom of the bounding box for intrusion detection was mentioned which I used in my project.

Chapter 3 - FRAMEWORK

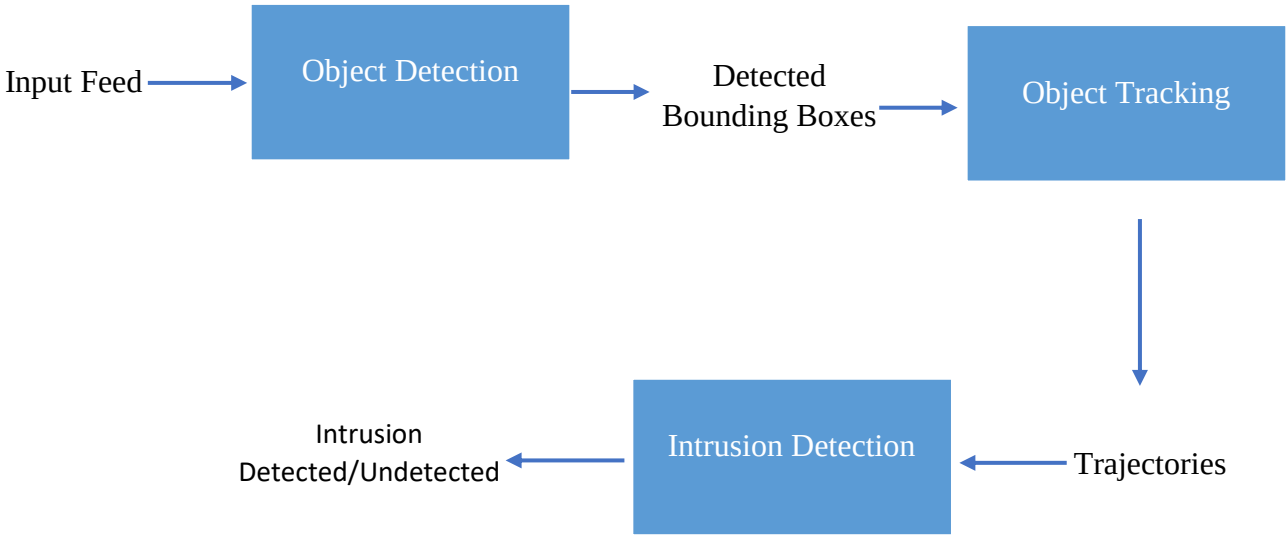

Figure 12. Block diagram of project framework.

3.1. Overview

This chapter gives an overview to all of the components of the framework for our project. Starting from the implementation of YOLOv4 for object detection and extracting the parameters of bounding boxes of the person class. Then the implementation of object tracking using deep SORT by utilizing the information of bounding boxes gathered from detection algorithm. For the improvement in the accuracy of abnormal activity of detection translation of centroid to the bottom of bounding boxes. Extracting trajectories of the motion of people and then utilizing all this information to detect intrusion into a specified restricted zone. Further to implement the concept of edge computing the algorithm is ported to an edge device the jetson nano. Then for the smooth execution of algorithm the creation and maintenance of the environment along with conflict resolutions. Finally for relatively fast detections of real time input feed, the optimization by using different approaches such as using light weight models and down sampling of input feed.

3.2. Object Detection

This is used to classify and localize objects within a given frame. The localization includes providing a very precise location of the object within the frame either in the form of bounding box or a region enclosing classified object. Usually input feed is given in the form of images or video frames. The images are represented as a matrix of pixels and each pixel represents the color intensity within its region. Usually, pretrained object detection algorithms have predefined classes such as person, cars, bottles etc. For classification the algorithms extract features based on color, shape, or texture to distinguish from other objects and background. Generally, there are two main approaches for object detection: deep learning-based approach and traditional approach. The first method includes convolutional neural networks while traditional methods include cascade classifiers, sliding window method etc.

3.2.1. YOLOv4

For the project YOLOv4 object detection algorithm was used. The pretrained model was used to detect only “person” class. Its training is performed on COCO dataset. The dataset contains 80 classes. The structure of object detector is given as follows.

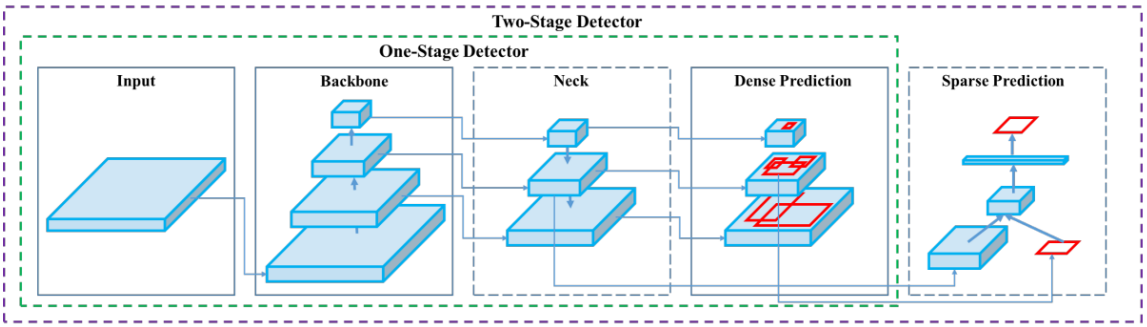

Figure 13. Anatomy of YOLOv4. Image courtesy [4]

The YOLOv4 takes images from input feed and compress features by using a convolutional neural network-based backbone. CSPDarknet53 is implemented as a backbone for YOLOv4 which consists of 53 layers. Feature aggregation is performed at neck where the features formed in backbone are mixed and combined to prepare for the detection.

PANet is used for feature aggregation in YOLOv4. A spatial pyramid pooling block is also placed next to the backbone for the increase in receptive field and to extract important features. Head is considered as a detection step where the authors of YOLOv4 used same head of YOLOv3. YOLO is one stage detector because it predicts classification and localization simultaneously as also obvious from the name You Only Look Once.

3.3. Object Tracking

It follows and monitors the motion of an object within the sequence of images. The major objective is to retain the identity of moving object, track the exact position and other related features. In general, the object of interest is initialized in a frame using object detection algorithms. Then important features are extracted, and the motion is estimated on the basis of these extracted features.

3.3.1. Deep SORT

Deep SORT is the upgraded version of SORT algorithm. It involves deep learning in SORT by including appearance descriptor for the reduction of identity switches. In SORT there are 4 key stages, First comes detection, and in our project, detections were passed by YOLOv4. The second step is estimation in which Kalman filter is used to estimate the position of object as it moves from one frame to another. The third step is data association, in which cost matrix is calculated as IOU for the detected bounding box and the target bounding box. Hungarian algorithm is used in this step. The final step is to create and delete track identities. If IOU is less, then a predefined threshold, the track is terminated otherwise retained. Since SORT fails if occlusion occurs and the identities are lost. So deep SORT comes up with a better association matric in which appearance descriptors and motion are combined. In other words, we can also say that deep SORT does not only considers velocity and motion but also incorporates the appearance of the moving object.

3.3.2. Centroid Extraction & Translation

The centroid of a frame detected for an object is very useful in object tracking. It has a key role in object tracking techniques such as mean-shift and Kalman filtering. It helps us to get an estimate of motion, for example features such as speed, trajectory and direction could be examined. It also helps to localize an object within a frame as now we can relate the coordinates of centroid with the coordinates of complete image. They can also help us for classification purposes by identifying the features around the centroid.

To analyze the abnormal activity of an object it is necessary to have some reference point on an object. For our project centroid was considered as an important feature to help us in abnormal activity detection. The tracked bounding boxes which were detected by YOLOv4 algorithm contains the following four parameters.

$$x, y, w, h$$

where x, y are coordinates of top left corner of the bounding box and w, h are the width and height of the bounding box. So, the centroid could be calculated as.

$$x_{centroid} = x + \frac{w}{2}$$

$$y_{centroid} = y + \frac{h}{2}$$

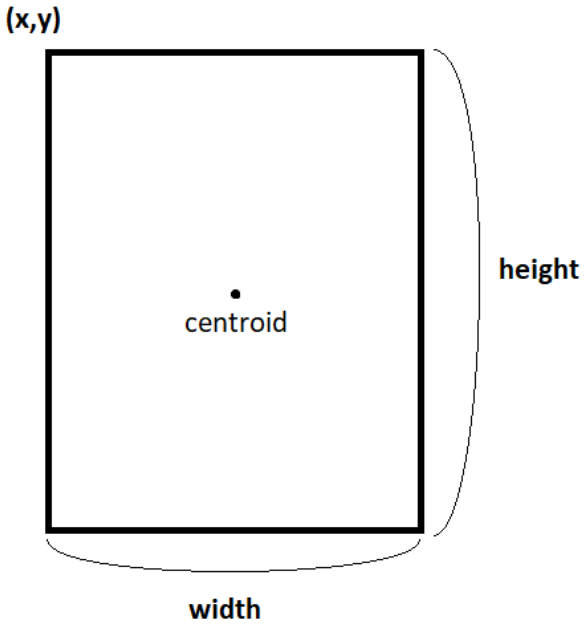

Figure 14. Bounding box with centroid and its parameters for YOLO

Although the centroid is extracted but for intrusion detection there is a chance that it is missed by the algorithm because boundaries of restricted zone or border lines are usually drawn on ground surface so there is a chance that a person intrudes but the centroid misses the line of restricted area.

To solve this issue the centroid was translated to the bottom of the bounding box, which proved to be very effective technique and now the algorithm was able to detect all intrusions successfully. To implement this, the formula of centroid was modified as follows.

$$x_{translated\_centroid} = x + \frac{w}{2}$$

$$y_{translated\_centroid} = y + h$$

The graphical representation of translated centroid is as follows.

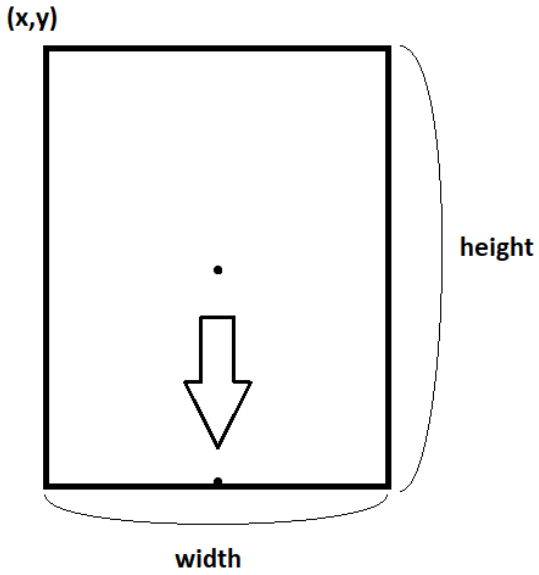

Figure 15. Depiction of the translation of centroid

3.3.3. Trajectories

Trajectories are the path traced by the detected objects. Since detected objects are represented by bounding boxes, the path followed by consecutive frames of same identity are represented by trajectories. For the trajectories to be drawn different references can be used either with respect to the corner points of bounding boxes or by following the centroids.

Trajectories can be used to observe different features and scene dynamics. For example, features include speed and acceleration and scene dynamics such as vehicle movements and motion pattern of people. This helps to identify anomalies in the scene by comparing the expected trajectories with the current trajectory of object. Trajectories can also be used to predict future positions of objects with the help of previous data of moving object.

For the project we chose centroid to be the reference for the extraction of trajectories. For each object, the centroids of consecutive frames are stored in a dictionary and a trajectory is drawn for the complete path history of an object. For each identity a sperate trajectory is drawn and if a different identity appears in the frame a separate trajectory appears. The trajectories drawn on a video of self-collected dataset are shown in the following image.

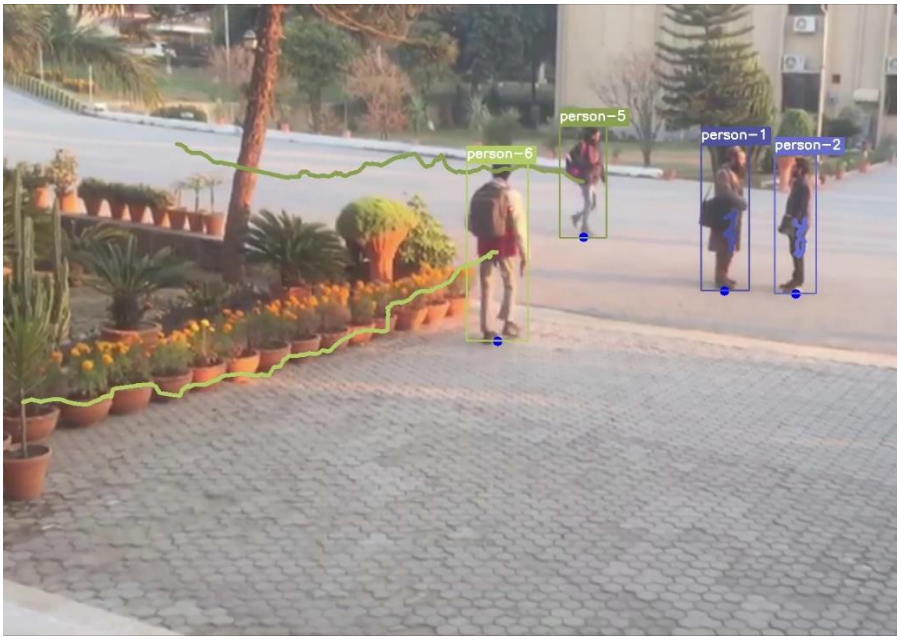

Figure 16. Trajectories for multiple tracked objects.

3.4. Abnormal Activity Detection

Abnormal activity detection is to identify behaviors that are deviating from the normal patterns. To implement this, various techniques are used. Generally, normal behavior activities are modeled by using different techniques such as statistical modeling or deep learning, which then are used as a reference to compare with other scenes. In the model, features are extracted to identify different events. The examples of features include object trajectories, motion patterns and spatial-temporal relationships. The algorithms which are generally used to compare scene with pre-built model includes Gaussian models and recurrent neural networks.

Although abnormal activities include a number of activities that are considered to be abnormal, but our project is defined for intrusion to an abnormal activity.

3.4.1. Intrusion Detection

The intrusion detection for our project is defined to identify unauthorized entrance into a defined zone. The restricted zone could be an ROI, or a simple border line. Whenever a person tries to intrude, the system generates a warning.

The criterion that is defined in our algorithm for the detection of intrusion is to determine whether the perpendicular distance between the centroid and the line of restricted zone is less than the defined threshold or not. The technique is illustrated in the following diagram.

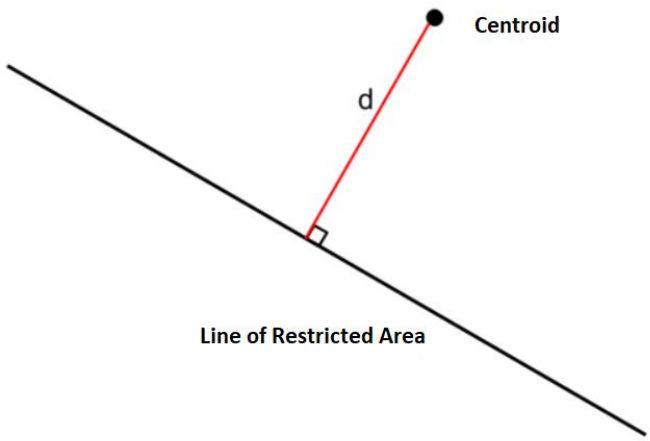

Figure 17. Distance between centroid and the line of restricted area.

The formula used for the calculation of distance between centroid and the line is given as.

$$d = \frac{|ax_0 + by_0 + c|}{\sqrt{(a^2 + b^2)}}$$

Where x_0 and y_0 are coordinates of the centroid and a,b,c are parameters determined by the straight-line equation.

This technique makes our algorithm robust enough to identify intrusion from any side of restricted zone. The threshold depends on the resolution of image used. Frames with higher resolution require greater threshold and vice versa because the distance increases with a smaller number of pixels in the same area.

3.4.2. Google Collaboratory GPU Utilization

Google colab is a development environment which allows the editing and execution of python codes and files. The major benefit of using google colab is that that they provide us with a GPU or TPU for fast computations. It's a free platform but has some limits on the duration of usage and some features. It has optimized pre-installed packages which eliminates the need to build and maintain an environment.

For our project we used google drive as a container to store files and then execute using the Jupyter notebook development environment managed by google colab, which makes it useful to get the computational advantage by utilizing their GPU or TPU instead of using local machine with lower computing power.

3.5. Jetson Nano

Jetson Nano is a small single board computational device that is used for edge computing applications. It consists of an integrated GPU which helps to execute neural networks in parallel, thus making it more useful for image processing applications. It is a part of NVIDIA's Jetson series. It supports different AI frameworks such as caffe, TensorFlow and MXNet.

Figure 18. Jetson Nano developer kit. Image courtesy [41]

The specifications of board are described in the following table:

TABLE IV. TECHNICAL SPECIFICATIONS

Properties	Description
Power	5W
CPU	Quad core ARM A57 with the clock speed of 1.43GHz
GPU	128 core Maxwell
RAM	4 GB
Memory card	MicroSD (64 GB for the project)
Camera communication protocol	2x MIPI CSI-2 DPHY lanes
Data communication protocols	I^2C, UART, I^2S, SPI, GPIO, USB 2.0, USB 3.0, Micro-B
Display Ports	Display port, HDMI
Dimensions	69mm x 45mm

3.5.1. Operating System Installation

The Jetson Nano was installed with a Linux based operating system, which is provided with JetPack SDK. It is basically a highly optimized version of Ubuntu thus provides us with the basic pre-installed libraries and optimized execution of tasks. The provided operating system is known as NVIDIA L4T.

Since the Jetson Nano does not have storage, so to install this OS we have to write the image of JetPack SDK on a microSD card with the help of a separate computer with an internet connection and the computer should also be able to read and write data on microSD cards either with the help of USB based SD card reader or with the built-in SD card reader slot. The Jetpack is provided and maintained by the official website of NVIDIA. It was downloaded in zip format. There are different procedures of installation for the computer having Windows. macOS or Linux based operating system. Since our computer was based on Windows OS the image was flashed accordingly.

First, the microSD card was formatted. NVIDIA recommends using the formatter provided by SD Association. From the formatting options of “Overwrite format, Quick format and CHS format” the “Overwrite format” was selected. The box for “Volume label” was left blank. Now the “Format” button was pressed and if a warning dialog appears, click the “Yes” button.

After the formatting was completed, microSD card was flashed with the provided OS. NVIDIA recommends using “Etcher” software for this purpose. In the software, first “Select Image” option was selected, which prompted a window to choose the zip file already downloaded from the website. Then “Select drive” option was clicked which then prompted to select the drive thus microSD was selected. Now the final option “Flash” was selected which took approximately 10 minutes to flash the image. It is also highly recommended to use USB3.0 slot if you are using USB based SD card reader. Now our microSD card was ready for the setup of Jetson Nano.

The microSD card was inserted into the slot beneath the surface on which heat sink is mounted. Then the Jetson Nano was powered, and it started its boot automatically. The first-time boot required us to follow the following steps. Acceptance and review of end user license agreement. Selection of time zone, keyboard configurations, language settings,

computer name, username and password. Finally, it asked about the size of APP partition. Maximum size is chosen for better performance. After successful procedure the Jetson Nano's user interface was visible.

3.5.2. Configuration of Environment

To get an isolated space for our algorithm an environment is created by using "Anaconda" as an environment management tool. This increased the flexibility and reproducibility of our algorithm as now we can run the same algorithm on any machine simply by recreating the environment. It also becomes very easy to change the versions of packages and their dependencies.

The Jupyter Notebook is used as a development environment to manage and execute the files. The algorithm is primarily based on TensorFlow framework which is executed with python 3.6 for our project.

During the installation of libraries, a lot of conflicts were managed especially between the dependencies. For example, NumPy was downgraded to version=1.23.5 from its latest version. Similarly, Jupyter notebook was only able to execute when root privileges are granted.

Another issue that was faced was the utilization of GPU. The TensorFlow 2.10.0 and onwards for ARM64 architecture are maintained by third party AWS. So, by default it installs "tensorflow-cpu-aws" which does not initiate GPU utilization. The TensorFlow version used by our algorithm is 2.12.0 and since Jetson Nano is based on ARM64 based architecture so the version of TensorFlow is incompatible for GPU utilization.

To overcome this issue the version of TensorFlow was downgraded to 2.7.0 which also required to downgrade its dependencies. Although The GPU was successfully initiated, but the algorithm caused the conflicts with the downgraded dependencies. Such as the "setuptools" package version 65.5.0 was not satisfied and version compatibility issue with keras.

For details of libraries installed in the environment see Appendix.

3.6. Optimization

There are multiple ways to optimize algorithms, some methods are discussed as follows. For example, PyTorch framework is better than TensorFlow on Jetson Nano. Multi-threading also helps to improve FPS, this could be implemented by executing detection and tracking on separate threads. Overclocking is a technique to increase computational performance of Jetson Nano, but it also increases heat generation and requires a cooling fan to be mounted on heat sink. Among object detection (using YOLOv4) and object tracking (using deepSORT), object detection is more computationally extensive. So, the major emphasis is on the optimization of YOLOv4.

3.6.1. YOLOv4 Tiny

YOLOv4 Tiny was used to increase the efficiency of algorithms for real-time detections. It is a lighter version of YOLOv4 with reduced computational complexity. It has a small network which has less layers and parameters, thus it results in higher inference speed but at the cost of some accuracy. Another limitation to the tiny version is that that it only accepts a fixed resolution of input image as the tiny version has specific requirements for its network. It only accepts the input resolution of 416 x 416 pixels. The tiny version helped us to improve the FPS from around 0.5 to 2 on the Jetson Nano.

3.6.2. Down-sampling

By reducing the resolution of input image, the algorithm can perform fast computations. The balance between accuracy and optimization must be maintained. The input resolution should be decreased to a level where the essential characteristics are preserved. The down-sampling is not only applicable for YOLOv4 Tiny because the tiny version is only designed for specific resolution as mentioned in the previous section. For our project different combinations were tried, for example resolution of 640 x 480 pixels.

3.7. Summary

The intrusion detection of a person in specified zone required us to first detect the object using YOLOv4 and extract the centroid from the detected bounding boxes. To track classified objects, Deep SORT is used which tracks the objects in consecutive frames and retains the identities. The trajectories are then extracted and highlighted in the output video. This information is then used to detect intrusion of people into a specified zone. Improvements were implemented in the features that are used to detect intrusion with higher accuracy includes the translation of centroid to the bottom of bounding box and using distance between line and point formula to calculate distance between centroid and the line of restricted area. For fast inference different optimization methods were used such as using light weight model and down sampling. Next chapter includes all details about the experimental setup of this framework along with results of our algorithm implemented on datasets of different scenarios.

Chapter 4 - EXPERIMENTAL RESULTS AND SETUP

4.1. Overview

This chapter demonstrates the practical implementation of this project. Starting from hardware setup of jetson nano, the detailed explanation of all input and output peripherals required and their working. Then the collection of datasets from different sources which includes different social media platforms and personal collection as well. The evaluation criterion is also explained. Then comes the results section which includes visuals of intrusion detection on all datasets of different scenarios. Both qualitative and quantitative aspects are also evaluated to give deep insights into the working of this project.

4.2. Experimental Setup

For real time intrusion detection Jetson Nano was used as an edge device. The input feed was given by using a USB camera. Internet was provided by an ethernet cable. For storage a 64 GB microSD card was used. The display was given to an LED by using HDMI port. The device was powered by a 5V 4A adapter through a DC barrel jack. Other input devices such as keyboard and mouse were connected to through USB ports.

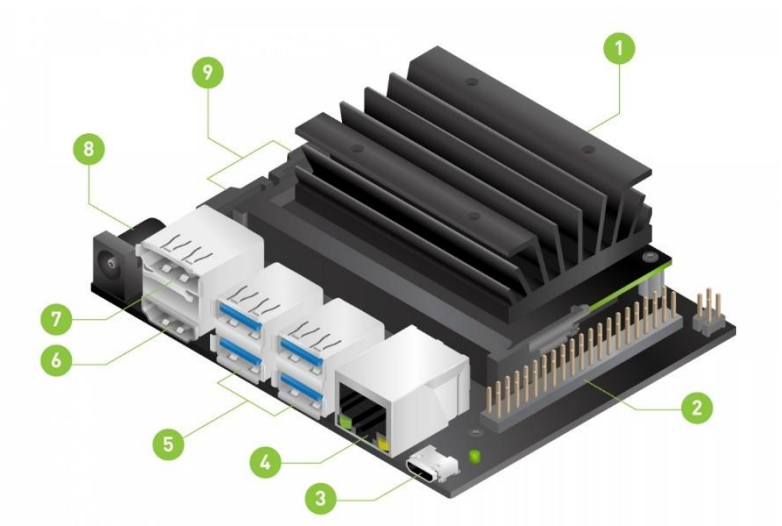

Figure 19. Input/output ports for jetson nano. Image courtesy [42]

1. microSD card
2. expansion header (not used in project)
3. micro-USB port (not used in project)
4. Ethernet port
5. 4x USB ports
6. HDMI port
7. Display port (not used in project)
8. DC Barrel Jack
9. MIPI CSI-2 camera connectors (not used in project)

4.3. Datasets

Datasets were collected both personally and using digital platforms such as Twitter and YouTube. The view of sensitive premises, entrance and ramp for special people were collected personally while the view of wall and window were calculated from digital platforms. The field of view of collected datasets is shown as follows.

.

Sensitive Premises

Entrance

Wall

Window

Ramp for Special People

Figure 20. Datasets

4.4. Evaluation Criteria

To evaluate the developed algorithm, datasets were collected from different scenarios both personally and from online platforms. The scenarios included wall climbing, window intrusion, premises intrusion and area specified for special people. The criterion was whether a warning is generated or not in the case of intrusion. After testing all dataset videos and real time video feed it was proved that the algorithm is working accurately. For wall climbing dataset the red signal was generated when the person crosses the line of restriction. Similarly for window intrusion clip the intrusion was detected despite the fact that only the top view of person was visible, but the algorithm was smart enough to detect it. Also, for the case of specified zone of special people a warning of intrusion was generated. For real time video feed both the intrusion of single object and multiple objects was tested, and both were successfully detected.

4.5. Results and Evaluation

The algorithm was evaluated on datasets collected for different scenarios and on the real time video feed from the USB camera. The results proved the algorithm to be accurate and robust enough to detect every intrusion. The datasets included wall intrusion, window intrusion, ramp intrusion and sensitive premises intrusion. In wall intrusion datasets the algorithm smoothly tracks the person climbing the wall and warns of an intrusion occurs, despite a lot of changes in posture of climbing person there is only one identity switch. For window intrusion, despite the fact that the camera is located at an inclination, and it is tough to detect a person, the algorithm detects that person right after his appearance in camera throughout the end of his appearance. A smooth trajectory is generated without any identity switch and an intrusion warning is generated. In ramp intrusion video two people appear in the frame one near the camera and the other one in background occluded with vegetations. Both people are detected and tracked. A warning is also generated for the person intruding into the restricted area. The final dataset is of sensitive premises intrusion. There are multiple people in that video sequence. The object detection and tracking algorithms are successfully able to get trajectories of all people accurately with only two identity switches and right after a person intrudes at the entrance of premises a warning signal is generated.

4.6. Qualitative Evaluation

The developed algorithm was implemented on different datasets, both personally collected datasets and from digital platforms as mentioned in the datasets section. The algorithm was able to detect intrusion successfully in all scenarios. First, simple object detection and tracking algorithms were tested on dataset then intrusion was evaluated. The algorithm generated a warning and displayed it on the screen whenever intrusion was detected. Similar was the case for all datasets of wall climbing, window intrusion, ramp intrusion and sensitive premises intrusion. The object detection and object tracking are illustrated as.

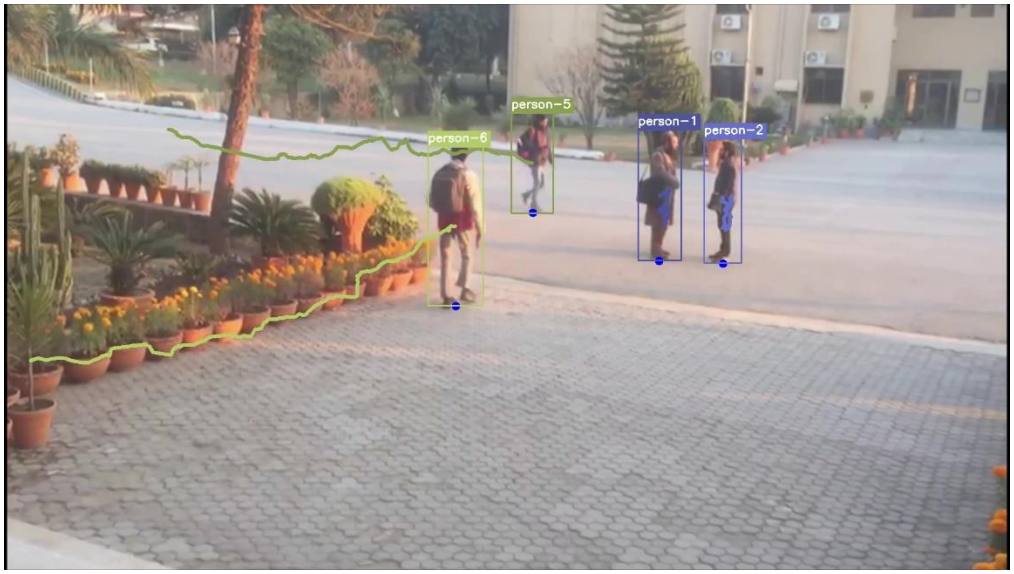

Figure 21. Evaluation of object detection and object tracking.

The intrusion detection by our algorithm on different datasets is illustrated as follows.

I. Sensitive premises intrusion detection.

Figure 22. Sensitive premises intrusion detection.

II. Wall intrusion detection.

Figure 23. Wall climbing intrusion detection. Image courtesy [43]

III. Window Intrusion detection.

Figure 24. Window intrusion detection. Image courtesy [44]

IV. Ramp for special people intrusion detection.

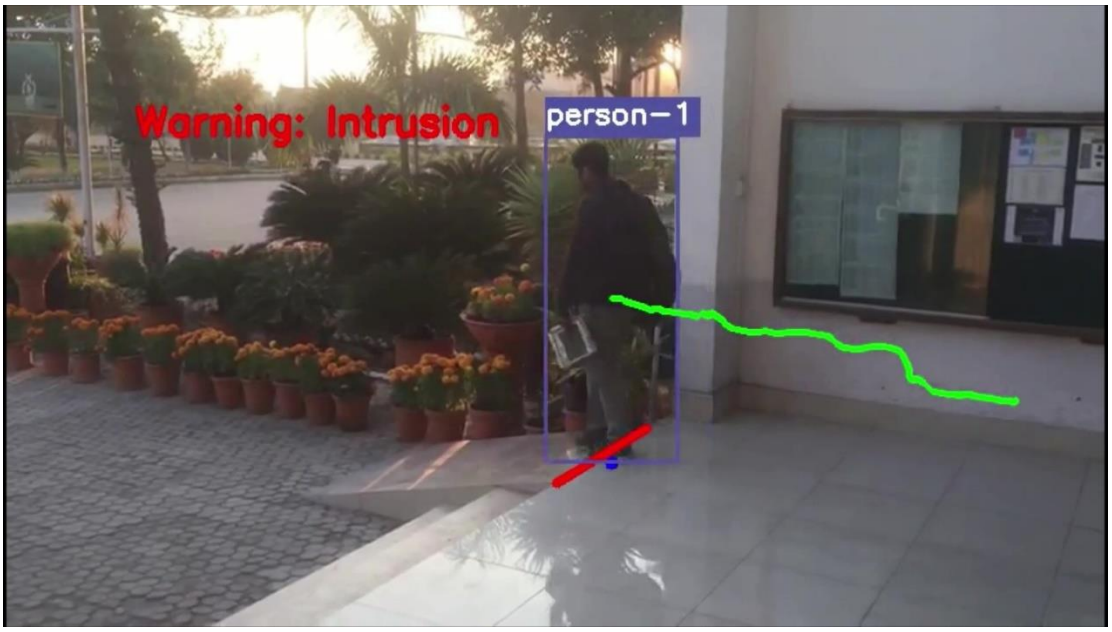

Figure 25. Ramp intrusion detection.

4.7. Quantitative Evaluation

The intrusion detection algorithm is computationally complex algorithm because it goes through all the processes of detection, tracking and finally the intrusion detection. Before the optimization of algorithms, the real time inference occurred at around 0.5 FPS but after the optimization of algorithm by using different techniques (as mentioned in framework section) the FPS improved to 2 FPS approximately.

The further quantitative detail of evaluation on datasets is as follows.

TABLE V. QUANTITATIVE EVALUATION.

Evaluation Criteria	Sensitive Premises	Wall Intrusion	Window Intrusion	Ramp Intrusion
Number of people	6	1	1	2
Successfully Tracked	6	1	1	2
Identity Switches	2	1	0	0
Intrusion Detected/ Undetected	Detected	Detected	Detected	Detected
Number of people intruded	1	1	1	1

4.8. Discussion

The results of this project were found to be very insightful that how the algorithm detects every person with accurate bounding boxes and then the tracking algorithm works accurately even in occlusions, especially for sensitive premises dataset. The identities are retained despite significant changes in the posters as observed in the wall climbing video, the person in video completely bends his body but still a perfect trajectory is traced. The tracker also works fine in the scenarios when there are multiple objects, and it does not compromise on the accuracy of tacking as observed in sensitive premises scenario. The detector and tracker are also very sensitive to background motion, as in the ramp intrusion scene it also detects a person walking far away behind vegetation. In all four datasets the intrusion is detected successfully, and the algorithm is proved to be robust enough to detect intrusion in any case. The improved intrusion detection algorithms could have a great impact on the robustness of video surveillance systems.

Chapter 5 - CONCLUSIONS

This chapter concludes this thesis by reiterating the major achievements and some recommendations for the future work.

5.1. Summary of Achievements

The final year project named “single camera based abnormal activity analysis for surveillance applications” focused on intrusion detection as an abnormal activity was implemented. The literature was reviewed to get an insight into the status of work already accomplished in this field and the improvements that are considered nowadays. Different techniques for object detection, object tracking, and abnormal activity detection were reviewed. The object detectors included both single stage and two stage detectors and for our project it was decided to use single stage object detectors because they provide us with higher computational speeds rather than two stage detectors. Among object trackers different trackers were reviewed which were based on methods such as Hungarian algorithm or Kalman filters and for our project it was decided to use deep SORT due to its better speed and higher accuracy. After research the object detection and tracking were performed by integrating YOLOv4 and Deep SORT algorithms. The YOLOv4 provided us with the parameters of detected bounding boxes of person class. These important features were then used to track the objects and retain their identities in consecutive frames. To detect intrusion the formula of distance between a point and a line was used and if the distance falls below a predefined threshold the intrusion is detected. To further enhance the accuracy of algorithm the centroid of bounding box was translated to the bottom of bounding box because in most of the cases the line of restricted zone is located on ground surfaces and there is a chance that intrusion remains undetected. After intrusion is detected, the system generates a warning on the display. The project was initially developed on a local computer and then ported to Jetson Nano. The environment containing required libraries for the algorithm was created and maintained along with resolving dependency issues and conflicts. The algorithm was tested on different datasets. For fast inference, optimization was performed by using light

weights of object detector and by downsampling the input video feed. Finally, the experimental setup was established for real time intrusion detection on a real time camera feed.

5.2. Future Recommendations

There could be multiple improvements to this project which are listed below.

- The latest version of YOLO (currently YOLOv8) could be integrated with the deep SORT for greater efficiency and accuracy.
- The TensorFlow Lite or TensorFlow-TensorRT can be used for inference optimization on Jetson Nano.
- Docker container can be used to avoid conflicts among packages and for the optimized versions of packages.
- Batch size of YOLOv4 can be adjusted for fast inference.
- PyTorch framework can be used instead of TensorFlow since several platforms recommend that PyTorch provides better performance on Jetson Nano.
- Multi-threading can also be used for fast performance, for example detection and tracking could be processed on different threads.
- A custom board could be developed instead of using Jetson Nano because there are some extra features which are not utilized in our project for example an extra USB port, a display port. This could be useful for commercialization purposes of this project.
- A casing could be developed to hold embedded systems such as Jetson Nano or any customized board, camera, and power supply so that this project can be considered as a completely marketable product.
- An IP camera can also be interfaced with Jetson Nano, this would enable the device to process input feed remotely.
- An android/iOS app could be developed for the users of this project to monitor the activity remotely.
- An indicator could also be integrated which alarms the user in case of any intrusion.

BIBLIOGRAPHY

[1] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "*You Only Look Once: Unified, Real-Time Object Detection*," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.

[2] J. Redmon and A. Farhadi, "*YOLO9000: Better, Faster, Stronger," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6517-6525, doi: 10.1109/CVPR.2017.690.

[3] Redmon, J., & Farhadi, A. (2018). *YOLOv3: An Incremental Improvement*. *arXiv*. https://doi.org/10.48550/arXiv.1804.02767

[4] Bochkovskiy, A., Wang, C., & Liao, H. (2020). YOLOv4*: Optimal Speed and Accuracy of Object Detection*. *arXiv*. https://doi.org/10.48550/arXiv.2004.10934

[5] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C., & Berg, A. C. (2015). *SSD: Single Shot MultiBox Detector*. *arXiv*. https://doi.org/10.1007/978-3-319-46448-0_2

[6] R. Girshick, J. Donahue, T. Darrell and J. Malik, "*Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation*," 2014 IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 580-587, doi: 10.1109/CVPR.2014.81.

[7] R. Girshick, "*Fast R-CNN*," 2015 IEEE International Conference on Computer Vision (ICCV), 2015, pp. 1440-1448, doi: 10.1109/ICCV.2015.169.

[8] S. Ren, K. He, R. Girshick and J. Sun, "*Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 6, pp. 1137-1149, 1 June 2017, doi: 10.1109/TPAMI.2016.2577031.

[9] Dai, J., Li, Y., He, K., & Sun, J. (2016). *R-FCN: Object Detection via Region-based Fully ConvolutionalNetworks*. *arXiv*. https://doi.org/10.48550/arXiv.1605.06409

[10] N. Wojke, A. Bewley and D. Paulus, "*Simple online and realtime tracking with a deep association metric*," 2017 IEEE International Conference on Image Processing (ICIP), 2017, pp. 3645-3649, doi: 10.1109/ICIP.2017.8296962.

[11] A. Bewley, Z. Ge, L. Ott, F. Ramos and B. Upcroft, "*Simple online and realtime tracking*," 2016 IEEE International Conference on Image Processing (ICIP), 2016, pp. 3464-3468, doi: 10.1109/ICIP.2016.7533003.

[12] H. Nam and B. Han, "*Learning Multi-domain Convolutional Neural Networks for Visual Tracking*," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 4293-4302, doi: 10.1109/CVPR.2016.465.

[13] P. Bergmann, T. Meinhardt and L. Leal-Taixé, "*Tracking Without Bells and Whistles*," 2019 IEEE/CVF International Conference on Computer Vision (ICCV), 2019, pp. 941-951, doi: 10.1109/ICCV.2019.00103.

[14] R. Nayak, M. M. Behera, U. C. Pati and S. K. Das, "*Video-based Real-time Intrusion Detection System using Deep-Learning for Smart City Applications*," 2019 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), 2019, pp. 1-6, doi: 10.1109/ANTS47819.2019.9117960.

[15] S. A. Ahmed, D. P. Dogra, S. Kar and P. P. Roy, "*Trajectory-Based Surveillance Analysis: A Survey*," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 29, no. 7, pp. 1985-1997, July 2019, doi: 10.1109/TCSVT.2018.2857489.

[17] S. A. Ahmed, D. P. Dogra, S. Kar and P. P. Roy, "*Trajectory-Based Surveillance Analysis: A Survey*," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 29, no. 7, pp. 1985-1997, July 2019, doi: 10.1109/TCSVT.2018.2857489.

[18] J. -x. Wang, "*Research and implementation of intrusion detection algorithm in video surveillance*," 2016 International Conference on Audio, Language and Image Processing (ICALIP), 2016, pp. 345-348, doi: 10.1109/ICALIP.2016.7846572.

[19] R. Nayak, M. M. Behera, U. C. Pati and S. K. Das, "*Video-based Real-time Intrusion Detection System using Deep-Learning for Smart City Applications*," 2019 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), 2019, pp. 1-6, doi: 10.1109/ANTS47819.2019.9117960.

[20] X. Zhang, Q. Yu and H. Yu, "*Physics Inspired Methods for Crowd Video Surveillance and Analysis: A Survey*," in IEEE Access, vol. 6, pp. 66816-66830, 2018, doi: 10.1109/ACCESS.2018.2878733.

[21] J. -G. Ko and J. -H. Yoo, "*Rectified Trajectory Analysis Based Abnormal Loitering Detection for Video Surveillance*," 2013 1st International Conference on Artificial Intelligence, Modelling and Simulation, 2013, pp. 289-293, doi: 10.1109/AIMS.2013.53.

[22] Igual, R., Medrano, C. & Plaza, I. *Challenges, issues and trends in fall detection systems*. *BioMed Eng OnLine* **12**, 66 (2013). https://doi.org/10.1186/1475-925X-12-66

[23] W. Hu, X. Xiao, Z. Fu, D. Xie, T. Tan, and S. Maybank, "*A system for learning statistical motion patterns*," IEEE Trans. on PAMI, vol. 28, no. 9, pp. 1450–1464, September 2006.

[24] T. Zhang, H. Lu, and S. Z. Li, "*Learning semantic scene models by object classification and trajectory clustering*," in Proc. of IEEE CVPR, Miami, 2009.

[25] N. Anjum and A. Cavallaro, "*Multi-feature object trajectory clustering for video analysis*," IEEE Trans. on CSVT, vol. 18, no. 11, pp. 1555 – 1564, 2008.

[26] X. Wang, K. T. Ma, G.-W. Ng, and W. E. L. Grimson, "*Trajectory analysis and semantic region modeling using nonparametric hierarchical bayesian models*," IJCV, vol. 95, no. 3, pp. 287–312, December 2011.

[27] W. Hu, X. Xiao, Z. Fu, D. Xie, T. Tan, and S. Maybank, "*A system for learning statistical motion patterns*," IEEE Trans. on PAMI, vol. 28, no. 9, pp. 1450–1464, September 2006.

[28] T. Nawaz, A. Cavallaro and B. Rinner, "*Trajectory clustering for motion pattern extraction in aerial videos*," 2014 IEEE International Conference on Image Processing (ICIP), 2014, pp. 1016-1020, doi: 10.1109/ICIP.2014.7025203.

[29] J. Boyle, T. Nawaz and J. Ferryman, "*Deep trajectory representation-based clustering for motion pattern extraction in videos*," 2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), 2017, pp. 1-6, doi: 10.1109/AVSS.2017.8078509.

[30] https://motchallenge.net/ dated 13.12.2022

[31] https://ilr.law.uiowa.edu/print/volume-106-issue-2/clearing-opacity-through-machine-learning/ dated 13.12.2022

[32] https://www.v7labs.com/blog/object-detection-guide dated 13.12.2022 dated 13.12.2022

[33] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "*You Only Look Once: Unified, Real-Time Object Detection*," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91. dated 13.12.2022

[34] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," 2014 IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 580-587, doi: 10.1109/CVPR.2014.81.dated 13.12.2022

[35] R. Girshick, "Fast R-CNN," 2015 IEEE International Conference on Computer Vision (ICCV), 2015, pp. 1440-1448, doi: 10.1109/ICCV.2015.169.dated 13.12.2022

[36] S. Ren, K. He, R. Girshick and J. Sun, "*Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 6, pp. 1137-1149, 1 June 2017, doi: 10.1109/TPAMI.2016.2577031.dated 13.12.2022

[37] Dai, J., Li, Y., He, K., & Sun, J. (2016*). R-FCN: Object Detection via Region-based Fully Convolutional Networks*. *arXiv*. https://doi.org/10.48550/arXiv.1605.06409 dated 13.12.2022

[38] H. Nam and B. Han, "Learning Multi-domain Convolutional Neural Networks for Visual Tracking," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 4293-4302, doi: 10.1109/CVPR.2016.465. dated 13.12.2022

[39] Wang, J.X., 2016, July. Research and implementation of intrusion detection algorithm in video surveillance. In *2016 International Conference on Audio, Language and Image Processing (ICALIP)* (pp. 345-348). IEEE. dated 13.12.2022

[40] R. Nayak, M. M. Behera, U. C. Pati and S. K. Das, "Video-based Real-time Intrusion Detection System using Deep-Learning for Smart City Applications," 2019 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), 2019, pp. 1-6, doi: 10.1109/ANTS47819.2019.9117960. dated 13.12.2022

[41] https://developer.nvidia.com/embedded/jetson-nano-developer-kit. Dated 14.5.2023

[42] https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit#intro Dated 24.5.2023

[43] https://www.youtube.com/

[44] https://www.twitter.com/

APPENDIX

TABLE VI. PACKAGES MAINTAINED IN THE ENVIRONMENT

Package Name	Version	Build	Channel
openmp_mutex	4.5	2_gnu	conda-forge
absl-py	1.3.0	pyhd8ed1ab_0	conda-forge
absl-py	1.4.0	<pip>	
anyio	3.6.2	pyhd8ed1ab_0	conda-forge
argon2-cffi	21.3.0	pyhd8ed1ab_0	conda-forge
argon2-cffi-bindings	21.2.0	<pip>	
argon2-cffi-bindings	21.2.0	py310h761cc84_3	conda-forge
asttokens	2.1.0	pyhd8ed1ab_0	conda-forge
astunparse	1.6.3	<pip>	
attrs	22.1.0	pyh71513ae_1	conda-forge
backcall	0.2.0	pyh9f0ad1d_0	conda-forge
backports	1.0	py_2	conda-forge
backports.functools_lru_cache	1.6.4	pyhd8ed1ab_0	conda-forge
beautifulsoup4	4.11.1	pyha770c72_0	conda-forge
bleach	5.0.1	pyhd8ed1ab_0	conda-forge
brotli	1.0.9	h4e544f5_8	conda-forge
brotli-bin	1.0.9	h4e544f5_8	conda-forge
bzip2	1.0.8	hf897c2e_4	conda-forge
ca-certificates	2022.9.24	h4fd8a4c_0	conda-forge
cachetools	5.3.0	<pip>	
certifi	2022.12.7	<pip>	
certifi	2022.9.24	pyhd8ed1ab_0	conda-forge
cffi	1.15.1	py310hf0c4615_2	conda-forge
cffi	1.15.1	<pip>	
charset-normalizer	3.1.0	<pip>	
colorama	0.4.6	pyhd8ed1ab_0	conda-forge
contourpy	1.0.6	py310hb15e014_0	conda-forge
contourpy	1.0.6	<pip>	
cycler	0.11.0	pyhd8ed1ab_0	conda-forge
debugpy	1.6.3	py310h130cc07_1	conda-forge
debugpy	1.6.3	<pip>	

decorator	5.1.1	pyhd8ed1ab_0	conda-forge
defusedxml	0.7.1	pyhd8ed1ab_0	conda-forge
easydict	1.9	py_0	conda-forge
entrypoints	0.4	pyhd8ed1ab_0	conda-forge
executing	1.2.0	pyhd8ed1ab_0	conda-forge
flatbuffers	23.3.3	<pip>	
flit-core	3.8.0	pyhd8ed1ab_0	conda-forge
fonttools	4.38.0	<pip>	
fonttools	4.38.0	py310h761cc84_1	conda-forge
freetype	2.12.1	hbbbf32d_0	conda-forge
gast	0.4.0	<pip>	
google-auth	2.17.3	<pip>	
google-auth-oauthlib	1.0.0	<pip>	
google-pasta	0.2.0	<pip>	
grpcio	1.53.0	<pip>	
h5py	3.8.0	<pip>	
icu	70.1	ha18d298_0	conda-forge
idna	3.4	pyhd8ed1ab_0	conda-forge
importlib-metadata	5.0.0	pyha770c72_1	conda-forge
importlib_resources	5.10.0	pyhd8ed1ab_0	conda-forge
ipykernel	6.14.0	<pip>	
ipykernel	6.14.0	py310h7d5ade6_0	conda-forge
ipython	8.4.0	<pip>	
ipython	8.4.0	py310h4c7bcd0_0	conda-forge
ipython_genutils	0.2.0	py_1	conda-forge
ipywidgets	8.0.2	pyhd8ed1ab_1	conda-forge
jax	0.4.8	<pip>	
jedi	0.18.1	pyhd8ed1ab_2	conda-forge
jinja2	3.1.2	pyhd8ed1ab_1	conda-forge
jpeg	9e	h9cdd2b7_2	conda-forge
jsonschema	4.17.0	pyhd8ed1ab_0	conda-forge
jupyter	1.0.0	py310h4c7bcd0_7	conda-forge
jupyter	1.0.0	<pip>	
jupyter_client	7.4.7	pyhd8ed1ab_0	conda-forge
jupyter_console	6.4.4	pyhd8ed1ab_0	conda-forge
jupyter_core	5.0.0	py310h4c7bcd0_0	conda-forge

jupyter_core	5.0.0	<pip>	
jupyter_server	1.23.2	pyhd8ed1ab_0	conda-forge
jupyterlab_pygments	0.2.2	pyhd8ed1ab_0	conda-forge
jupyterlab_widgets	3.0.3	pyhd8ed1ab_0	conda-forge
keras	2.12.0	<pip>	
kiwisolver	1.4.4	py310h9ceb0a0_1	conda-forge
kiwisolver	1.4.4	<pip>	
lcms2	2.14	h5246980_0	conda-forge
ld_impl_linux-aarch64	2.39	ha75b1e8_0	conda-forge
lerc	4.0.0	h4de3ea5_0	conda-forge
libblas	3.9.0	16_linuxaarch64_openblas	conda-forge
libbrotlicommon	1.0.9	h4e544f5_8	conda-forge
libbrotlidec	1.0.9	h4e544f5_8	conda-forge
libbrotlienc	1.0.9	h4e544f5_8	conda-forge
libcblas	3.9.0	16_linuxaarch64_openblas	conda-forge
libclang	16.0.0	<pip>	
libdeflate	1.14	h4e544f5_0	conda-forge
libffi	3.4.2	h3557bc0_5	conda-forge
libgcc-ng	12.2.0	h607ecd0_19	conda-forge
libgfortran-ng	12.2.0	he9431aa_19	conda-forge
libgfortran5	12.2.0	hf695500_19	conda-forge
libgomp	12.2.0	h607ecd0_19	conda-forge
libiconv	1.17	h9cdd2b7_0	conda-forge
liblapack	3.9.0	16_linuxaarch64_openblas	conda-forge
libnsl	2.0.0	hf897c2e_0	conda-forge
libopenblas	0.3.21	pthreads_h6cb6f83_3	conda-forge
libpng	1.6.38	hf9034f9_0	conda-forge
libsodium	1.0.18	hb9de7d4_1	conda-forge
libsqlite	3.40.0	hf9034f9_0	conda-forge
libstdcxx-ng	12.2.0	hc13a102_19	conda-forge
libtiff	4.4.0	hacef7f3_4	conda-forge
libuuid	2.32.1	hf897c2e_1000	conda-forge
libwebp-base	1.2.4	h4e544f5_0	conda-forge

libxcb	1.13	h3557bc0_1004	conda-forge
libxml2	2.10.3	h249b6dd_0	conda-forge
libxslt	1.1.37	h4871090_0	conda-forge
libzlib	1.2.13	h4e544f5_4	conda-forge
lxml	4.9.1	<pip>	
lxml	4.9.1	py310h141c14b_1	conda-forge
Markdown	3.4.3	<pip>	
MarkupSafe	2.1.1	<pip>	
markupsafe	2.1.1	py310hdc54845_2	conda-forge
MarkupSafe	2.1.2	<pip>	
matplotlib	3.6.2	<pip>	
matplotlib	3.6.2	py310hbbe02a8_0	conda-forge
matplotlib-base	3.6.2	py310hf92b850_0	conda-forge
matplotlib-inline	0.1.6	pyhd8ed1ab_0	conda-forge
mistune	2.0.4	pyhd8ed1ab_0	conda-forge
ml-dtypes	0.1.0	<pip>	
munkres	1.1.4	pyh9f0ad1d_0	conda-forge
nbclassic	0.4.8	pyhd8ed1ab_0	conda-forge
nbclient	0.7.0	pyhd8ed1ab_0	conda-forge
nbconvert	7.2.5	pyhd8ed1ab_0	conda-forge
nbconvert-core	7.2.5	pyhd8ed1ab_0	conda-forge
nbconvert-pandoc	7.2.5	pyhd8ed1ab_0	conda-forge
nbformat	5.7.0	pyhd8ed1ab_0	conda-forge
ncurses	6.3	headf329_1	conda-forge
nest-asyncio	1.5.6	pyhd8ed1ab_0	conda-forge
notebook	6.5.2	pyha770c72_1	conda-forge
notebook-shim	0.2.2	pyhd8ed1ab_0	conda-forge
numpy	1.23.4	<pip>	
numpy	1.23.5	<pip>	
numpy	1.23.4	py310he617cf3_1	conda-forge
oauthlib	3.2.2	<pip>	
opencv-python	4.7.0.72	<pip>	
openjpeg	2.5.0	h9b6de37_1	conda-forge
openssl	3.0.7	h4e544f5_0	conda-forge
opt-einsum	3.3.0	<pip>	
packaging	23.1	<pip>	

packaging	21.3	pyhd8ed1ab_0	conda-forge
pandoc	2.19.2	h8af1aa0_1	conda-forge
pandocfilters	1.5.0	pyhd8ed1ab_0	conda-forge
parso	0.8.3	pyhd8ed1ab_0	conda-forge
pexpect	4.8.0	pyh1a96a4e_2	conda-forge
pickleshare	0.7.5	py_1003	conda-forge
pillow	9.2.0	py310hd6d4ca1_3	conda-forge
Pillow	9.2.0	<pip>	
pip	22.3.1	pyhd8ed1ab_0	conda-forge
pkgutil-resolve-name	1.3.10	pyhd8ed1ab_0	conda-forge
platformdirs	2.5.2	pyhd8ed1ab_1	conda-forge
prometheus_client	0.15.0	pyhd8ed1ab_0	conda-forge
prompt-toolkit	3.0.32	pyha770c72_0	conda-forge
prompt_toolkit	3.0.32	hd8ed1ab_0	conda-forge
protobuf	4.22.3	<pip>	
psutil	5.9.4	<pip>	
psutil	5.9.4	py310h761cc84_0	conda-forge
pthread-stubs	0.4	hb9de7d4_1001	conda-forge
ptyprocess	0.7.0	pyhd3deb0d_0	conda-forge
pure_eval	0.2.2	pyhd8ed1ab_0	conda-forge
pyasn1	0.4.8	<pip>	
pyasn1-modules	0.2.8	<pip>	
pycparser	2.21	pyhd8ed1ab_0	conda-forge
pygments	2.13.0	pyhd8ed1ab_0	conda-forge
pyparsing	3.0.9	pyhd8ed1ab_0	conda-forge
pyrsistent	0.19.2	<pip>	
pyrsistent	0.19.2	py310hdc54845_0	conda-forge
python	3.10.6	h92ab765_0_cpython	conda-forge
python-dateutil	2.8.2	pyhd8ed1ab_0	conda-forge
python-fastjsonschema	2.16.2	pyhd8ed1ab_0	conda-forge
python_abi	3.10	2_cp310	conda-forge
pyzmq	24.0.1	py310h7c6bb8d_1	conda-forge
pyzmq	24.0.1	<pip>	
readline	8.1.2	h38e3740_0	conda-forge
requests	2.28.2	<pip>	
requests-oauthlib	1.3.1	<pip>	

rsa	4.9	<pip>	
scipy	1.10.1	<pip>	
send2trash	1.8.0	pyhd8ed1ab_0	conda-forge
setuptools	65.5.1	pyhd8ed1ab_0	conda-forge
six	1.16.0	pyh6c4a22f_0	conda-forge
sniffio	1.3.0	pyhd8ed1ab_0	conda-forge
soupsieve	2.3.2.post1	pyhd8ed1ab_0	conda-forge
stack_data	0.6.1	pyhd8ed1ab_0	conda-forge
tensorboard	2.12.2	<pip>	
tensorboard-data-server	0.7.0	<pip>	
tensorboard-plugin-wit	1.8.1	<pip>	
tensorflow	2.12.0	<pip>	
tensorflow-cpu-aws	2.12.0	<pip>	
tensorflow-estimator	2.12.0	<pip>	
tensorflow-io-gcs-filesystem	0.32.0	<pip>	
termcolor	2.2.0	<pip>	
terminado	0.15.0	<pip>	
terminado	0.15.0	py310hbbe02a8_0	conda-forge
tinycss2	1.2.1	pyhd8ed1ab_0	conda-forge
tk	8.6.12	hd8af866_0	conda-forge
tornado	6.2	<pip>	
tornado	6.2	py310hdc54845_1	conda-forge
tqdm	4.64.1	pyhd8ed1ab_0	conda-forge
traitlets	5.5.0	pyhd8ed1ab_0	conda-forge
typing_extensions	4.5.0	<pip>	
typing_extensions	4.4.0	pyha770c72_0	conda-forge
tzdata	2022f	h191b570_0	conda-forge
unicodedata2	15.0.0	<pip>	
unicodedata2	15.0.0	py310h761cc84_0	conda-forge
urllib3	1.26.15	<pip>	
wcwidth	0.2.5	pyh9f0ad1d_2	conda-forge
webencodings	0.5.1	py_1	conda-forge
websocket-client	1.4.2	pyhd8ed1ab_0	conda-forge
Werkzeug	2.2.3	<pip>	

wheel	0.38.4	pyhd8ed1ab_0	conda-forge
widgetsnbextension	4.0.3	pyhd8ed1ab_0	conda-forge
wrapt	1.14.1	<pip>	
xorg-libxau	1.0.9	h3557bc0_0	conda-forge
xorg-libxdmcp	1.1.3	h3557bc0_0	conda-forge
xz	5.2.6	h9cdd2b7_0	conda-forge
zeromq	4.3.4	h01db608_1	conda-forge
zipp	3.10.0	pyhd8ed1ab_0	conda-forge
zstd	1.5.2	hc1e27d5_4	conda-forge